Spring 2022

Stochastic Model Predictive Control via Fixed Structure Policies

Elias Wilson
Embry-Riddle Aeronautical University, wilsone8@my.erau.edu

STOCHASTIC MODEL PREDICTIVE CONTROL VIA FIXED STRUCTURE POLICIES

By

Elias Wilson

A Dissertation Submitted to the Faculty of Embry-Riddle Aeronautical University

In Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy in Aerospace Engineering

May 2022

Embry-Riddle Aeronautical University

Daytona Beach, Florida

STOCHASTIC MODEL PREDICTIVE CONTROL VIA

FIXED STRUCTURE POLICIES

By

Elias Wilson

This Dissertation was prepared under the direction of the candidate's Dissertation Committee Chair, Dr. Richard Prazenica, Department of Aerospace Engineering, and has been approved by the members of the Dissertation Committee. It was submitted to the Office of the Senior Vice President for Academic Affairs and Provost, and was accepted in the partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Aerospace Engineering.

DISSERTATION COMMITTEE

Richard J. Prazenica Digitally signed by Richard J. Prazenica Date: 2022.04.22 13:17:54 -04'00'

Chair, Dr. Richard Prazenica

Morad Nazari Digitally signed by Morad Nazari Date: 2022.04.22 14:21:00 -04'00'

Member, Dr. Morad Nazari

Troy A. Henderson Digitally signed by Troy A. Henderson Date: 2022.04.22 16:53:12 -04'00'

Member, Dr. Troy Henderson

K. Merve Dogan Digitally signed by K. Merve Dogan Date: 2022.04.22 23:04:01 -04'00'

Member, Dr. K. Merve Dogan

Sergey V. Drakunov Digitally signed by Sergey V. Drakunov Date: 2022.04.23 10:42:09 -05'00'

Member, Dr. Sergey Drakunov

Sirish Namilae Digitally signed by Sirish Namilae Date: 2022.04.26 12:10:46 -04'00'

Graduate Program Coordinator,
Dr. Sirish Namilae

Date

Dean of the College of Engineering,
Dr. James W. Gregory

Date

Senior Vice President for Academic Affairs and Provost,
Dr. Lon Moeller

Date

ACKNOWLEDGMENTS

I would like to thank every member of my committee as each of you have helped me, in some way, learn and grow. I cannot express enough praise for my advisor Dr. Prazenica who gave me an opportunity to find my own path and do what I wanted. You were helpful along the way and always offered useful advice or at least some dry humor. I was always excited to meet and discuss 20 minutes of actual work over 2 hours. I look forward to staying connected and working with you more in future. Dr. Henderson will forever go down as the professor who could beat Dr. Prazenica in never returning graded homework. Your brief yet quick replies to emails at any odd hour were always appreciated. Dr. Nazari convinced me that making simulations within scripts instead of Simulink was cool. I desire to be cool, so I started to model systems in this way. I now realize that creating my own work flows, integrators, and functions not only makes me cool but also results in me learning what is actually happening. Dr. Drakunov revolutionized the way I think about randomness in our lives and he did so through a web cam and a board only slightly larger than a piece of paper. Unfortunately, I was not able to take any classes with Dr. Dogan. I was, however, fortunate enough to see your rise within the university as I watched the presentation you gave prior to be hired. You put me in my place in terms of how much work I have got done as a Ph.D. student. I need to thank Dr. Alvaro Velasquez who is not on my committee but he gave me the opportunity to work with him at the Air Force Research Laboratory, and without that experience, I would have never ended up going down this path. Lastly, Dr. Mark Gummin, of small town Oregon fame, is the real reason why I am in this position. Without you, I would have not attended graduate school.

ABSTRACT

In this work, the model predictive control problem is extended to include not only open-loop control sequences but also state-feedback control laws by directly optimizing parameters of a control policy. Additionally, continuous cost functions are developed to allow training of the control policy in making discrete decisions, which is typically done with model-free learning algorithms. This general control policy encompasses a wide class of functions and allows the optimization to occur both online and offline while adding robustness to unmodelled dynamics and outside disturbances. General formulations regarding nonlinear discrete-time dynamics and abstract cost functions are formed for both deterministic and stochastic problems. Analytical solutions are derived for linear cases and compared to existing theory, such as the classical linear quadratic regulator. It is shown that, given some assumptions hold, there exists a finite horizon in which a constant linear state-feedback control law will stabilize a nonlinear system around the origin. Several control policy architectures are used to regulate the cart-pole system in deterministic and stochastic settings, and neural network-based policies are trained to analyze and intercept bodies following stochastic projectile motion.

TABLE OF CONTENTS

LIST OF FIGURES

Figure **Page**

LIST OF TABLES

Table **Page**

1 Introduction

Model predictive control (MPC), introduced by Richalet et al., has been extensively studied over the past few decades and has proven to be a powerful tool for controlling large complex systems. The MPC algorithm works by predicting a plant's trajectory, over a finite time horizon, given an open-loop control sequence (OCS), then improving upon those inputs until an optimal set is found with respect to a user defined cost function. In general, cost functions follow a similar quadratic form and can be easily designed to perform varying tasks from regulation to tracking. No assumptions are made on the plant dynamics, other than being differentiable, so the algorithm can be applied to both linear and nonlinear systems. Lastly, functional constraints can be placed on both the system states and control inputs to ensure the problem does not give physically infeasible solutions. Most notably, including a terminal constraint on the system states dictates that the states must be equal to or within a region around a specified value at the end of the time horizon.

Similar to MPC, dynamic programming (DP) works by predicting the system response over a finite time horizon. Dynamic programming involves finding optimal value and policy functions over the entire state space. For this reason, DP solutions come in the form of feedback control laws, which are robust to unmodelled dynamics and disturbances, while MPC solutions give an OCS. Model predictive control simplifies the overall problem compared to DP, in some way, by reducing the underlying optimization to just a single trajectory. For systems with large or continuous state spaces, MPC proves more computationally realizable as the solution does not concern the entire space. This issue is known as the curse of dimensionality and, by itself, opened another realm of research regarding DP.

The goal of this work is to explore MPC's ability to compute feedback control policies. It is easy to imagine that, given an optimal state trajectory and set of control inputs, some arbitrary function could be formed to map those states to the control inputs. If a function with universal approximation capability, such as an infinite width single layer neural network, were used, then any set of states could be mapped to any set of control inputs. That system

could then operate using the feedback function and follow the same trajectory as the OCS. More interestingly, if the optimal trajectories and inputs were found for a variety of initial conditions and covered a larger portion of the state space, a more powerful version of the feedback function, which could produce more than one optimal trajectory, could be found. Essentially, this would be pushing the MPC algorithm towards DP as more and more of the total state space is explored.

In this work, a version of MPC, which is denoted parameterized policy model predictive control (PPMPC), is presented. The goal is to find feedback control policies, not by solving conventional MPC problems and then performing least squares regressions, but by directly computing the optimal policy parameters. By taking the derivative of the cost function with respect to the policy parameters, an iterative optimization scheme can be used to solve the problem. This does not necessarily preclude solving for an OCS as the policy is taken to be a general user-defined function. The policy parameters could take the values of the OCS, a linear feedback gain, an arbitrary polynomial, a piecewise defined function, or even a neural network. The main benefits to solving the optimal control problem in this way is a potential decrease in computational effort, if there is a decrease in the total number of parameters to solve for compared to conventional MPC, and increased robustness to unmodelled dynamics and disturbances.

As computation power becomes increasingly accessible and easier to include within system designs, "brute force" control strategies become more appealing. One of these strategies is Monte Carlo simulation, which has shown to be extremely powerful given enough computational effort. Within the realm of reinforcement learning, Monte Carlo simulation is used to perform massive statistical regressions within highly nonlinear discontinuous systems. Silver et al. showed how Monte Carlo Tree Search (MCTS) could be used to not only play the ancient Chinese game Go, a difficult task, but also beat the world champion in a real time sanctioned match. By "simply" running thousands or millions of simulations, statistical information regarding a system can be gauged and utilized to form advantageous

control policies.

Within traditional control theory, stochastic systems are generally handled by assuming random variables within the dynamics and measurements come from predefined probability distributions. Most commonly, a Gaussian distribution is assumed, which enables estimators, such as the various forms of the Kalman filter, to provide information on the true value of the system state. Within complex nonlinear systems, even if the random variable or variables come from some well behaved probability distribution, like the Gaussian one, the characteristics of that distribution are distorted by the dynamics. This implies that the probability distribution is actually changing throughout time. By performing the policy optimization on a bundle of trajectories, the underlying statistical effects can be accounted for along with a wider berth of the state space.

There are three main technical objectives to be addressed throughout this work: developing a deterministic PPMPC method, developing a stochastic PPMPC method, and including decision making capabilities within these methods. For both the deterministic and stochastic cases, the proposed formulation will be compared, within a linear environment, to well known theory. The linear quadratic regulator and linear quadratic Gaussian will both be derived using the presented formulas. Outside of analytical solutions for linear systems, the algorithm will be tested in nonlinear environments using a numerical optimization approach. Existing work regarding closed-loop regulation stability for general nonlinear policies is presented along with specific focus on constant linear state feedback policies. It is shown that, given a large enough optimization horizon, a constant linear state feedback policy produces will stabilize the system around the origin. Additionally, a continuous cost function that allows a policy to be trained to make high level decisions and control a system is developed.

This dissertation is organized as follows. In the next section, a simple motivating problem is presented to help further explain the goal and potential power of solving optimal control problems by optimizing policy parameters within nonlinear environments compared to using linear approximations. In Chapter 2, an overview of related work is given along with

comparisons to the algorithm proposed here. Chapter 3 briefly introduces a general control policy definition, thoroughly describes the value function, presents several cost function architectures, and gives several forms of the optimizations problems to be solved. Methods for solving optimization problems are given in Chapter 4, including a generic derivative of the value function. Chapter 5 thoroughly describes how control policies are used within the context of the this work and gives not only several examples to be tested but also the required derivatives of those for use within the value function derivative. Regulation stability is addressed in Chapter 6. Existing stability proofs regarding MPC are presented and leveraged to show optimal linear state feedback policies will stabilize nonlinear systems, given that certain assumptions on the cost function, dynamics, and initial condition are met. Linear quadratic regulator problems are analyzed in Chapter 7. The classical finite horizon linear quadratic regulator is derived and that solution is used show how the linear quadratic Gaussian can be derived. Additionally, constant linear state feedback gains are explored for use within finite horizon problems. Chapter 8 presents the cart-pole and dot intercept environments, which will be used to test the presented control strategy. Simulation results are provided in Chapter 9, conclusions are made in Chapter 10

1.1 Simple Motivating Example

Model predictive control can be used to find not only a numerically identical feedback gain for a linear system compared to the linear quadratic regulator (LQR) but also a better performing gain for a system with nonlinearities. To show the efficacy of this approach, a simple nonlinear oscillator is considered:

$$f(\underline{x}, u) = \begin{bmatrix} x_2 \\ -\kappa x_1 - c x_2 + b(3 \sin x_1) + u \end{bmatrix} \tag{1.1}$$

where $\underline{x} = [y \;\; \dot{y}]^T$ is the state vector, $y \in \mathbb{R}$ is the displacement from zero, $\dot{y} \in \mathbb{R}$ is the velocity, $\kappa \in \mathbb{R}$ is the spring stiffness, $c \in \mathbb{R}$ is the damping, and $b \in \{0, 1\}$ acts as a switch to enable or disable the sinusoid. This work focuses on discrete-time dynamics, so

this continuous time model is discretized using the first order Euler method:

$$\underline{x}_{k+1} = \underline{x}_k + \Delta t f(\underline{x}_k, u_k) \tag{1.2}$$

where $k \in \mathbb{N}$ is the time step and $\Delta t \in \mathbb{R}_{>0}$ is the amount of time between steps. The goal is to find a feedback gain K for the control policy $u_k = -K\underline{x}_k$ that minimizes the cost function

$$J = \frac{1}{2}\sum_{k=0}^{\infty}\left(\underline{x}_k{}^T Q \underline{x}_k + R u_k{}^2\right) \tag{1.3}$$

where $Q \in \mathbb{R}^{2\times 2}_{\geq 0}$ is a symmetric state weighting matrix and $R \in \mathbb{R}_{>0}$ is a scalar weight on the control inputs.

Given a completely linear system, the solution to this problem is well known: the infinite horizon LQR, which, for the scope of this example, will be referred to simply as the LQR [5]. The LQR feedback gain is given by solving the algebraic Riccati equation (ARE)

$$0^{2\times 2} = S_\infty - A^T S_\infty (I + BR^{-1}B^T S_\infty)^{-1} A - Q \tag{1.4}$$

where $0^{2\times 2}$ is a matrix of zeros, $A \in \mathbb{R}^{2\times 2}$ is the linear dynamics state matrix, $B \in \mathbb{R}^{2\times 1}$ is the control input matrix, and $S_\infty \in \mathbb{R}^{2\times 2}$ is the solution to the ARE. The sizes of these matrices are set by the number of states and control inputs within the dynamic system. Once the solution to the ARE is found, the feedback gain is given by:

$$K_\infty = R^{-1}B^T S_\infty \left(I^{2\times 2} + BR^{-1}B^T S_\infty\right)^{-1} \tag{1.5}$$

In a simplified sense, the LQR computes a sequence of control inputs that satisfies the condition

$$\frac{dJ}{du_k} = 0 \;\forall\; k \in \mathbb{N}_{[0,\infty)} \tag{1.6}$$

and it just happens to be that the solution to this problem is a linear feedback control law.

Using MPC, this problem can be solved by making the initial assumption that a linear feedback control law is going to be used and finding the gain $K_{MPC} \in \mathbb{R}^{1\times 2}$ that satisfies

$$\frac{dJ}{dK_{MPC}} = 0^{1\times 2} \tag{1.7}$$

Some approximation must be made since solving an infinite horizon optimal problem numerically is not tractable; however, given a long enough horizon, the MPC solution will converge towards the LQR solution. This problem can easily be solved by numerical optimization software without even requiring the first order derivatives be input.

First, consider the case where $b = 0$ and the system is linear. Let $\Delta t = 0.01s$, $\kappa = -1$, and $c = 1$. Then the discrete dynamics matrices can be written as

$$A = \begin{bmatrix} 1 & 0.01 \\ 0.01 & 0.99 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 0.01 \end{bmatrix} \tag{1.8}$$

and the weighting matrices can be taken to be: $Q = I_2$ and $R = 0.1$. Additionally, all simulations are ran for 10 seconds, which sets the horizon of the MPC cost, and all initial conditions are taken to be $\underline{x}_0 = [2 \;\; 0]^T$. The LQR gain can be found by solving the ARE and substituting that solution into Eq. (1.5), which yields $K_\infty = [4.260 \;\; 3.386]$ and a corresponding cost of $J_\infty = 316.0$. Letting the MATLAB interior-point solver compute the MPC gain yields $K_{MPC} = [4.260 \;\; 3.386]$ with the corresponding cost $J_{MPC} = 316.0$. Clearly, the MPC approach will numerically reproduce the analytical solution.

Next, consider the case where $b = 1$ with κ, c, Q, and R remaining the same. Linearizing the dynamics around $\underline{x} = 0^{2\times 1}$, for use within the LQR algorithm, gives

$$A = \begin{bmatrix} 1 & 0.01 \\ 0.04 & 0.99 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 0.01 \end{bmatrix} \tag{1.9}$$

which leads to the feedback gain $K_\infty = [8.988 \;\; 4.335]$ with the resulting cost of $J_\infty =$

509.4. Solving this nonlinear problem with MPC yields $K_{MPC} = [5.890 \ \ 2.015]$ with the cost $J_{MPC} = 486.3$. Since the MPC algorithm bases its solution on the true nonlinear dynamics, a more optimal feedback gain is found. The spring constant begins to decrease as the displacement increases, which likely necessitates the smaller gain.

In this simple example, a performance benefit is shown when solving for a linear feedback gain using MPC within a nonlinear dynamic setting. More importantly, this example suggests that MPC can be used to determine more powerful control policies rather than just linear feedback gains. A side effect of the nonlinear dynamics is that the optimal feedback gains are now functions of the initial condition, a topic that will be explored more later in this disseration. While linear control theory works well for systems like this one, within some range of $\underline{x}_0$, performing complex maneuvers with nonlinear systems can prove to be quite difficult. Given that MPC takes into account nonlinearities within the system, both powerful nonlinear feedback policies and improved simpler linear policies can be found and shown to perform better than the linear control theory counterpart.

2 Current Technology

2.1 Optimal Control Methods

Using full-state feedback and Euler-Lagrange equations, the linear quadratic regulator (LQR) can be derived in a straightforward manner. No a priori assumption is made about the form of the control law, yet the derivation eventually shows that a linear time-varying or constant feedback control law is the optimal choice for finite or infinite horizons, respectively. In both cases, a Ricatti equation is used to solve for the feedback gain, which does not depend on the system states, but rather just the linear dynamic matrices. By linearizing nonlinear dynamics around a given trajectory or within a given range of the state space, linear optimal control techniques like this can be applied to nonlinear systems to produce state feedback control laws (SFCLs) [6]. [7, 8] give closed-form solutions for regulation of nonlinear systems using time and state dependent feedback gains found via a state dependent Riccati equation.

The derivation of the LQR for output feedback is a different process [5]. For the output feedback LQR, in continuous time, the assumption $\underline{u}(t) = -K\underline{y}(t)$, where $\underline{y}(t) = C\underline{x}_(t)$ is the state measurement and $C \in \mathbb{R}^{r\times n}$ is the output matrix, is made from the start, and the problem is posed as finding an optimal value of K not the control sequence $\underline{u}(t)$. This process yields a system of equations to solve for the optimal feedback gain. Misra and Bai used an assumed feedback gain to control stochastic systems with output feedback. Instead of just using the current state measurement, a time history of information was used to aid in noise filtering and state estimation. This method is somewhat related to how convolutional neural networks (CNNs) were leveraged by Wilson and Prazenica, where a time history of data was used to estimate hidden states within a rotor system.

Explicit MPC entails finding an explicit control law offline and then implementing it, a generally piecewise affine function, online. This helps reduce the online computational burden since only a single function needs to be evaluated as opposed to solving an optimization problem. Alessio and Bemporad give a survey of explicit MPC methods and Rawlings and Mayne provide technical derivations. Both show that most applications treat linear sys-

tems with either quadratic or linear cost functions, which result in either quadratic or linear multiparametric programs. Given linear constraints on the states and controls, the optimal control law is found as a piecewise function whose domain is specified by the constraints. Hovland et al. use explicit MPC to control large systems with fast dynamics, which generally provide a challenge for MPC algorithms. Within a large state space, the computational requirements for optimization are large, especially if that optimizations needs to be performed quick enough to control a naturally unstable system with fast dynamics. Using explicit MPC moves the computational burden offline as the optimal SFCL found is significantly easier to evaluate.

Least-squares (LS) regression can be used to find approximations of optimal control policies. Chen et al. used explicit MPC to derive a piecewise control law for high frequency power electronics and then trained a neural network (NN) to reproduce the outputs of the function. The NN helped increase computation speed as a large search through the domain did not need to be performed. In a similar sense, Suykens et al. solved for optimal radial basis function (RBF) weights by simultaneously solving a base MPC problem to find the optimal inputs, and a regression problem, to minimize the error between the RBF output and those optimal inputs. Sánchez-Sánchez and Izzo used a LS approach to train a NN based policy to land spacecraft. By training a NN to reproduce optimal feedback functions or inputs, these methods resemble the problem being presented in this work. Using a single function as the control policy, all of the optimal solutions over some portion of the state space are captured, in a least squared error sense.

Chen et al. approximated an explicit MPC law by directly finding the optimal parameters for a NN based function. The poilcy parameters are optimized using the policy gradient theorem [18] with an advantage function. The advantage function essentially provides a way to greedily update the policy based on the system dynamics without taking a gradient of those dynamics. By performing a large number of simulations with a stochastic control policy, the state-control space is explored and any control inputs that provide an advantage

over others are weighted heavier while updating the policy parameters.

Eisenberg and Sage introduced Specific Optimal Control (SOC), which considers solutions to general nonlinear problems using predefined parameterized control policies. While the presentation of SOC was general, the implementation was limited as computer power was lacking. Kleinman and Athans solved linear quadratic variants using the SOC method as the problems do not rely heavily on simulations and data. Colombo et al. use optimize the weights of a time-based basis function within an MPC algorithm to control a robot with an attached arm. At the end of the 20th century and moving into the 21st, when computer power began to grow exponentially, Parisini, Zoppoli, et al. applied the SOC method to train NN based policies [22–28]. While the problems are presented using the SOC method, an emphasis is placed on LS regression approaches as NNs have large approximation power. It is shown that, if an optimal policy approximation can reproduce the true optimal solution to within some defined error, then the system will remain stable [26].

2.2 Dynamic Programming

Dynamic programming (DP) was introduced by Bellman and has been heavily studied since [29]. Solving optimal control problems in the form of state feedback policies, DP provides some attractive benefits over MPC but is not without some drawbacks. First developed for discrete action and state spaces, the Bellman equation is as simple as it is powerful and is written as:

$$V^*(\underline{x}_k) = min_{\underline{u}_k} \left(L(\underline{x}_k, \underline{u}_k) + V^*(\underline{x}_{k+1}) \right) \tag{2.1}$$

where V^* is the optimal value of being in a particular state, L is the observed cost of going to state $\underline{x}_{k+1}$ by using control input $\underline{u}_k$ in state $\underline{x}_k$. In words, the equation states that the optimal cost from time step k occurs by taking the optimal control input over one time step and then following the optimal trajectory from that new state. This sets up a method to perform backward recursion from the goal state to determine the optimal trajectory. By

optimizing the control input at all of the possible states along the horizon, a feedback law is determined in the form of a table. Given a discrete action space, computing the control input that gives a minimal cost is at least somewhat tractable. However, given a continuous action space, solving for this optimal control input at every time step becomes quite difficult as implementation requires exploring all of the possible trajectories throughout the horizon, and since the action space is continuous, there are an infinite number of possible trajectories.

Within discrete action spaces, powerful policy determination methods have been developed and are known as policy and value iteration. These two methods are quite similar and can be grouped under one idea called generalized policy iteration (GPI) [18]. The scheme is composed of two phases: policy evaluation and policy improvement. During the policy evaluation phase, the state value function is determined for the current policy at all of the reachable states. In traditional policy iteration, the state value function is completely determined for the current policy before moving to the policy improvement phase. In value iteration, a single iteration on the value function is performed and then policy improvement begins. Zhou et al. used an extension of DPI known as approximate (or adaptive) dynamic programming (ADP) to control systems without complete observability. A large breadth of information regarding DP, DPI, and ADP is covered by Bertsekas [31, 32].

2.3 Reinforcement Learning

Reinforcement learning (RL) methods of controlling all types of dynamic systems have become incredibly popular as research continues to advance their effectiveness and accessibility grows with not only increasing computer power but also easy to use code libraries that can reduce a complicated RL problem to a single script. At a base level, RL is trying to solve two problems, which have been reoccurring throughout this literature section, a policy optimization and a least squares regression. The largest assumption of RL is that the plant dynamics are completely unknown and, in turn, can be highly nonlinear, discontinuous, and/or stochastic. The regression problem, most commonly, considers a critic learning a Q-function that incorporates these dynamics along with the underlying reward structure

[33]. Again, this reward signal can be any abstract function, which allows sparse rewards to be given. Generally, RL problems concern maximizing a reward as opposed to minimizing a cost. The policy optimization problem then involves an actor finding a policy based on that learned Q-function. Instead of directly optimizing the Q-function to find the optimal inputs for a given state, which can prove to be quite difficult, the parameters of a feedback function are optimized to produce desired control inputs. As an example, Balakrishnan and Biega used an actor-critic based algorithm to find optimal regulators for the longitudinal plane of an aircraft.

Reinforcement learning offers a large advantage over MPC by simplifying the assumptions regarding the cost (reward) function. Model predictive control relies on derivatives of the cost function and determining these can be quite difficult for discrete tasks (i.e. a reward is observed by entering a state, but zero reward is given otherwise). These kinds of rewards work well for robotics tasks that involve decision making. Making a certain decision may be desirable and result in a positive reward, while making another may be undesirable and not give any reward. Farshidian et al. present a method of combining the reward learning portion of RL with the dynamic optimization of MPC. Using an actor-critic structure where the critic comes from RL techniques and the actor uses the system dynamics like MPC, the algorithm finds a blend of the two architectures' powers. If the dynamics of the system are known, using them within the optimization is clearly helpful, and learning a cost function allows the system to not only operate with a sparse reward signal but also adapt to unmodelled dynamics and/or disturbances. Similarly, Saerens and Soquet present a method of learning plant dynamics while also solving optimal control problems using those dynamics.

3 Parameterized Policy Model Predictive Control

Conventional MPC problems entail finding open-loop control sequences (OCSs) over some finite horizon, of length $N_h \in \mathbb{N}_{>0}$, that minimize a cost function $J : \mathbb{R}^{n\times N_h+1} \times \mathbb{R}^{m\times N_h} \mapsto \mathbb{R}_{\geq 0}$. The discrete system dynamics are given by $f : \mathcal{X} \times \mathcal{U} \mapsto \mathbb{R}^n$ as $\underline{x}_{k+1} = f(\underline{x}_k, \underline{u}_k)$, where $\mathcal{X} \subseteq \mathbb{R}^n$ and $\mathcal{U} \subseteq \mathbb{R}^m$ are sets of admissible states and inputs, respectively. Here, MPC is extended to include not only open-loop control but also feedback control.

3.1 Control Policy

The inputs of an OCS are not constrained to take the form of some function like a typical feedback control law: $\underline{u} = -K\underline{x}$. However, the MPC problem can be generalized to encompass both the conventional OCS and solutions where the control is assumed to be given by a specific type of function with unknown parameters. Let, the control input be given by the generic function $\mu : \mathcal{X} \times \Theta \times \mathbb{N}_{[0,N_h-1]} \mapsto \mathcal{U}$, where $\theta \in \Theta$ parameterizes the function. Not all of the inputs will be used for every architecture, and the set $\Theta \subset \mathbb{R}^p$, where $p \in \mathbb{N}_{>0}$ is the number of parameters, is determined by the assumed structure of the policy and the desired performance of the controller.

If the control inputs were assumed to come from a linear feedback control law $\underline{u}_k = -K\underline{x}_k$, θ would simply be the control gain $K \in \mathbb{R}^{m\times n}$ reshaped to lie within $\Theta \subseteq \mathbb{R}^{mn}$ via the operator

$$\theta = \text{vec}(K) \tag{3.1}$$

which can be inverted with

$$K = \text{vec}^{-1}_{m\times n}(\theta) \tag{3.2}$$

Note, the subscript of the inverse reshaping function will be dropped when dimensions have been clearly defined. In the constant linear feedback case, the state input is used but the time step is not. Considering the conventional MPC case, the state input is no longer used and the parameter $\theta \in \Theta \subseteq \mathbb{R}^{mN_h}$ is just the OCS $U \in \mathbb{R}^{mN_h}$, which does not need to be reshaped. Using this assumption of a general control law introduces the ability to derive

optimal controllers of varying forms: open-loop control inputs, linear feedback laws, or other abstract nonlinear functions.

Any input constraints can be intrinsically handled by the control policy given that $\mu(\underline{x}_k, \theta, k) \in \mathcal{U} \; \forall \; \underline{x}_k \in \mathcal{X}, \theta \in \Theta, k \in \mathbb{N}_{[0,N_h-1]}$. If the control constraints are simple upper and lower bounds, a saturating function like a sigmoid or hyperbolic tangent can be used to constrain outputs while also being differentiable. In practice, discontinuous functions like $min(\cdot,\cdot)$ and $max(\cdot,\cdot)$ can be used to achieve the same outcome. Some consideration must be taken with regard to the value of the derivatives at or near the constraint boundary. Using a discontinuous function with a derivative taking values of 0 or 1 can result in no update occurring if the constraint is active, and even in the continuous case, the derivatives of the sigmoid and hyperbolic tangent functions quickly approach zero, which can scale the derivative down significantly. This effect is especially apparent in time varying policies.

3.2 Cost and Value Function

Cost function design drives optimal controllers to perform in different ways. Commonly cost functions include quadratic terms as these not only are easy to work with but also satisfy the assumptions needed for optimization. More complex cost functions can be used as well, although optimization becomes much more difficult if there are multiple local minima. In this section, the base assumptions for a cost function with a single globally optimal solution are presented along with some example quadratic cost functions commonly used. Then, the system dynamics and control policy are used to derive the value function, which will be used as the objective function in this work.

3.2.1 Cost Function

The cost function is assumed to be composed of a terminal cost $\phi : \mathcal{X} \mapsto \mathbb{R}_{\geq 0}$ and a running cost $L : \mathcal{X} \times \mathcal{U} \mapsto \mathbb{R}_{\geq 0}$ and is given by:

$$J(X, U) = \phi(\underline{x}_{N_h}) + \sum_{k=0}^{N_h-1} L(\underline{x}_k, \underline{u}_k) \tag{3.3}$$

where the state trajectory $X \in \mathcal{X}^{N_h+1}$ and control trajectory $U \in \mathcal{U}^{N_h}$ are defined as:

$$X = \begin{bmatrix} \underline{x}_0{}^T & \underline{x}_1{}^T & \cdots & \underline{x}_{N_h-1}{}^T & \underline{x}_{N_h}{}^T \end{bmatrix}^T \tag{3.4}$$

$$U = \begin{bmatrix} \underline{u}_0{}^T & \underline{u}_1{}^T & \cdots & \underline{u}_{N_h-2}{}^T & \underline{u}_{N_h-1}{}^T \end{bmatrix}^T \tag{3.5}$$

The following assumptions hold for the terminal and running cost functions:

Assumption 1. $\exists \frac{\partial L(\underline{x}_k, \underline{u}_k)}{\partial \underline{x}_k}, \frac{\partial L(\underline{x}_k, \underline{u}_k)}{\partial \underline{u}_k} \; \forall \; k \in \mathbb{N}_{[0,N_h-1]}, \underline{x}_k \in \mathcal{X}, \underline{u}_k \in \mathcal{U}$

Assumption 2. $\exists \frac{\partial \Phi(\underline{x}_{N_h})}{\partial \underline{x}_{N_h}} \; \forall \; k \in \mathbb{N}_{[0,N_h-1]}, \underline{x}_k \in \mathcal{X}$

Assumption 3. $L(\underline{x}_k, \underline{u}_k) \geq 0 \; \forall \; \underline{x}_k \in \mathcal{X}, \underline{u}_k \in \mathcal{U}$ *and* $L(0^n, 0^m) = 0$

Assumption 4. $L(\underline{x}_k, \underline{u}_k) > 0 \; \forall \; \underline{x}_k \in \mathcal{X}$ *when* $\underline{u}_k \neq 0^m$

Assumption 5. $\phi(\underline{x}_{N_h}) \geq 0 \; \forall \; \underline{x}_{N_h} \in \mathcal{X}$ *and* $\phi(0^n) = 0$

Assumption 6. $L(\underline{x}_k, \underline{u}_k) \to \infty$ *as* $\underline{x}_k \to \infty$ *or* $\underline{u}_k \to \infty$

Assumption 7. $\phi(\underline{x}_{N_h}) \to \infty$ *as* $\underline{x}_{N_h} \to \infty$

There are three main attributes being forced by Assumptions 1-7: continuity, positive (semi)definiteness, and radial unboundedness. To assess first order optimally and perform gradient descent, a derivative (the transpose of the gradient) of the cost function will be used, which means the cost function must be at least $\mathcal{C}^1$ with respect to its inputs. The cost function must be positive semidefinite with respect to the system states and positive definite with respect to the control inputs. These requirements lead to the existence of minima and observability of the dynamics through the cost function. Within a linear quadratic context, observability is addressed via the Gramian of the pair $(A, Q^{1/2})$, where $Q \in \mathbb{R}^{n \times n}_{\geq 0}$ is a state weighting matrix. In a nonlinear context, Assumptions 3 and 5 assert that a nonzero state value returns a nonzero cost, and when combined with the radial unboundedness Assumptions 6 and 7, result in an observable system.

A non-decreasing radially unbounded cost function helps prove regulation stability of optimally controlled systems as unstable systems return infinite cost for large horizons and lower costs directly correlate to states that closer to the origin; class $\mathcal{K}$ and $\mathcal{K}_\infty$ functions are used to show a cost function is non-decreasing. First, a function $\alpha : \mathbb{R}_{\geq 0} \mapsto \mathbb{R}_{\geq 0}$ is class $\mathcal{K}$ if it is strictly increasing and returns zero given a zero input ($\alpha(0) = 0$). Second, a function $\alpha_\infty : \mathbb{R}_{\geq 0} \mapsto \mathbb{R}_{\geq 0}$ is a class $\mathcal{K}_\infty$ if it is a class $\mathcal{K}$ function that satisfies $\lim_{z\to\infty} \alpha_\infty(z) = \infty$. So, a non-decreasing cost function is one such that there exists four $\mathcal{K}_\infty$ functions $\alpha_1, \alpha_2, \alpha_3, \alpha_4 : \mathcal{X} \mapsto \mathbb{R}_{\geq 0}$ where

$$\alpha_1(||\underline{x}_k||) < L(\underline{x}_k, \underline{u}_k) < \alpha_2(||\underline{x}_k||) \tag{3.6}$$

for a given $\underline{u}_k$ and

$$\alpha_3(||\underline{x}_k||) < \phi(\underline{x}_k) < \alpha_4(||\underline{x}_k||) \tag{3.7}$$

The same property can be applied with respect to the control inputs. Given two more $\mathcal{K}_\infty$ functions $\alpha_5, \alpha_6 : \mathcal{U} \mapsto \mathbb{R}_{\geq 0}$ and a particular state $\underline{x}_k$, the cost of taking certain actions is bounded as

$$\alpha_5(||\underline{u}_k||) < L(\underline{x}_k, \underline{u}_k) < \alpha_6(||\underline{u}_k||) \tag{3.8}$$

The non-decreasing properties are used within Lyapunov based stability assessments. Not all systems considered in this work satisfy these properties.

The most commonly used cost function in both linear and nonlinear control is the quadratic function:

$$J(X, U) = \frac{1}{2}\underline{x}_{N_h}{}^T S \underline{x}_{N_h} + \frac{1}{2}\sum_{k=0}^{N_h-1} \frac{1}{2}\underline{x}_k{}^T Q \underline{x}_k + \underline{u}_k{}^T R \underline{u}_k \tag{3.9}$$

where $S, Q \in \mathbb{R}^{n\times n}_{\geq 0}$ and $R \in \mathbb{R}^{m\times m}_{>0}$. Along with the ease of implementation, a quadratic cost also satisfies all the necessary assumptions and non-decreasing properties required to assess regulation stability. Note, in the context of matrices, the subscripts "≥ 0" and "> 0" denote positive semidefinite and positive definite matrices, respectively. Modifying the cost to take

in error inputs

$$J(X,U) = \frac{1}{2}\underline{e}_{N_h}{}^T S \underline{e}_{N_h} + \frac{1}{2}\sum_{k=0}^{N_h-1} \frac{1}{2}\underline{e}_k{}^T Q \underline{e}_k + \underline{u}_k{}^T R \underline{u}_k \tag{3.10}$$

where $\underline{e}_k = \underline{x}_k - \underline{x}_{r,k}$ with $\underline{x}_{r,k} \in \mathcal{X}_r$ begin a reference state, enables trajectory tracking.

Another commonly used cost function architecture, which is further explored in this dissertation, is one where the inputs are minimized with a soft terminal constraint. In this case

$$J(X,U) = \phi(\underline{x}_{N_h}) + \frac{1}{2}\sum_{k=0}^{N_h-1} \underline{u}_k{}^T R \underline{u}_k \tag{3.11}$$

where ϕ could take on the quadratic form in either Eq. (3.9) or (3.10) or it can be some other appropriate function. This case is useful in that the outcome, or terminal state, is the most important component as opposed to the entire trajectory. Similar problems are solved by placing a terminal constraint on the problem and removing ϕ from the cost, but by just increasing the weight of ϕ, a soft constraint is imposed on the problem. Soft constrained problems are significantly easier to solve numerically as adding constraints not only increases the overall computational complexity but also significantly restricts the solution space.

3.2.2 Value Function

Within optimal control it is common to find control inputs by minimizing (or maximizing) the cost function with the dynamics included in the problem as a functional constraint. In this case, Lagrange multipliers are typically used to solve the problem, resulting in the Euler-Lagrange equations. Here, a value function, which is commonly found within RL algorithms, is used as the objective function. The value function returns the cost of starting at the initial state $\underline{x}_0$ and following the control policy μ. This is done by incorporating the system dynamics and control policy into the cost function and effectively constraining the cost function. Given the system dynamics, $\underline{x}_0$, μ, and θ, the state and control trajectories

can be completely defined as:

$$X^{\mu}(\underline{x}_0) = \begin{bmatrix} \underline{x}_0 \\ f\left(\underline{x}_0, \mu(\underline{x}_0, \theta, 1)\right) \\ \vdots \\ f\left(\underline{x}_{N_h-2}, \mu(\underline{x}_{N_h-2}, \theta, N_h-2)\right) \\ f\left(\underline{x}_{N_h-1}, \mu(\underline{x}_{N_h-1}, \theta, N_h-1)\right) \end{bmatrix} \tag{3.12}$$

$$U^{\mu}(\underline{x}_0) = \begin{bmatrix} \mu(\underline{x}_0, \theta, 0) \\ \mu(\underline{x}_1, \theta, 1) \\ \vdots \\ \mu(\underline{x}_{N_h-2}, \theta, N_h-2) \\ \mu(\underline{x}_{N_h-1}, \theta, N_h-1) \end{bmatrix} \tag{3.13}$$

Evaluating the cost function along these trajectories then produces the value function $V^{\mu} : \mathcal{X} \times \Theta \mapsto \mathbb{R}_{\geq 0}$:

$$V^{\mu}(\underline{x}_0, \theta) = J(X^{\mu}(\underline{x}_0), U^{\mu}(\underline{x}_0)) \tag{3.14}$$

Since both the dynamics and control policy are constrained to map to $\mathcal{X}$ and $\mathcal{U}$, respectively, no functional constraints need to be placed on the value function based optimization problem. Constraints can be added, however, to achieve desirable performance if necessary. For instance, the parameter space Θ could be constrained to some area that returns stabilizing control policies. Without any additional functional constraints on the problem, the optimization becomes much simpler to perform as basic gradient descent or stochastic search algorithms can be used to find solutions.

3.3 Unconstrained Deterministic Problem

Typically, in optimal control, a problem is constrained by the system dynamics and solved using Lagrange multipliers. For this work, the system dynamics are embedded into the cost function, which gives rise to the value function in Eq 3.14. All of the problems considered in

this work will be constrained by the system dynamics, control policy, initial state, admissible state set, and admissible control input set. Problems that are not subject to any additional constraints (e.g. terminal constraints or state boundaries) will be considered unconstrained.

The unconstrained problem is given by

$$\begin{gathered} \min_{\theta} \quad J(X,U) \\ s.t. \\ \underline{x}_{k+1} = f(\underline{x}_k, \underline{u}_k), \quad \underline{u}_k = \mu_\theta(\underline{x}_k, \theta, k), \quad \underline{x}_k \in \mathcal{X}, \quad \underline{u}_k \in \mathcal{U}, \quad \underline{x}_0 = x_0 \end{gathered} \tag{3.15}$$

The value function, acting as a constrained cost function, gives the cost incurred by being in state x_0 and following policy μ. The problem can be rewritten in terms of a value function as:

Problem 1.

$$\begin{gathered} \min_{\theta} \quad V^{\mu}(\underline{x}_0, \theta) \\ s.t. \\ \underline{x}_k \in \mathcal{X}, \quad \underline{u}_k \in \mathcal{U}, \quad \underline{x}_0 = x_0 \end{gathered}$$

3.4 Unconstrained Stochastic Problem

Adding uncertainty to the problem is done for two reasons: to handle a stochastic system or to create a more robust control policy. In both cases, the problem is similar to the deterministic problem, but instead of handling the system states directly, the controller works with expected values over the state distribution. No assumptions are made about the probability distribution associated with the system state; instead, using Monte Carlo simulations, the expected values will be determined using appropriate amounts of data. In the most general form, the unconstrained stochastic problem is given as

Problem 2.

$$\min_{\theta} \quad \mathbb{E}\left[V^{\mu}(\underline{x}_0, \theta)\right]$$

$$s.t.$$

$$\underline{x}_k \in \mathcal{X}, \quad \underline{u}_k \in \mathcal{U}, \quad \underline{x}_0 \sim \mathcal{D}_{x_0}$$

where $\mathbb{E}$ is the expectation and $\mathcal{D}$ is used to denote a general probability distribution with the subscript describing what it applies to. In this problem, the dynamics embedded within the value function are given by the stochastic nonlinear dynamics

$$\underline{x}_{k+1} = f(\underline{x}_k, \underline{u}_k, \underline{\omega}_k) \tag{3.16}$$

where $\underline{\omega}_k \in \mathbb{R}^n \sim \mathcal{D}_{\omega}$ is the process noise, which is assumed to be uncorrelated.

In the most obvious case, uncertainty in the problem is a direct result of either the initial state or system dynamics being stochastic. Another use for uncertainty is creating robust control policies. Adding some artificial stochastic terms to a deterministic system will cause more of the state space to be explored, and if the control policy is trained on that expanded view of the state space, a more robust control policy will be found. The simplest way to explore a wider area of the state space is by adding some noise to the initial state. In this case, the dynamics are still deterministic, so the policy will learn to control the true system but over a range of initial conditions, which give a range of state trajectories. This process is similar to RL algorithms as they rely on stochastic exploration of the state space, but in this case, the width of the search is easily controlled by the added noise.

3.5 Constrained Deterministic Problem

Functional constraints placed on the optimization in the form of equalities or inequalities ensure the system behaves is some desirable way. Most notably, constraining the terminal system state to the origin is used to prove the closed-loop stability. Equality constraints like the one used to constrain the system state to a particular point, dictate exact values for states or control inputs along the horizon and are given as $C_{eq} : \mathcal{X} \times \Theta \mapsto \mathbb{R}^{c_1}$, where

$c_1 \in \mathbb{N}$ is the number of equality constraints. Both the system dynamics and control policy, which constrain the cost function into the value function, are equality constraints. Inequality constraints are useful for placing bounds on the admissible states and control inputs. The $c_2 \in \mathbb{N}$ inequality constraints are given as $C_{ineq} : \mathcal{X} \times \Theta \mapsto \mathbb{R}^{c_2}$. If some region of the state space is undesirable, an inequality constraint can be used to ensure the system does not visit that region. The constrained problem is given by

$$\begin{gathered} \min_{\theta} \quad J(X, U) \\ s.t. \\ \underline{x}_{k+1} = f(\underline{x}_k, \underline{u}_k), \;\; \underline{u}_k = \mu_\theta(\underline{x}_k, \theta, k), \;\; \underline{x}_k \in \mathcal{X}, \;\; \underline{u}_k \in \mathcal{U}, \;\; \underline{x}_0 = x_0, \\ C_{eq}(\underline{x}_0, \theta) = 0^{c_1}, \;\; C_{ineq}(\underline{x}_0, \theta) \geq 0^{c_2} \end{gathered} \tag{3.17}$$

Again, the problem can be rewritten in terms of a value function as:

Problem 3.

$$\begin{gathered} \min_{\theta} \quad V^{\mu}(\underline{x}_0, \theta) \\ s.t. \\ X \in \mathcal{X}, \;\; U \in \mathcal{U}, \;\; \underline{x}_0 = x_0, \;\; C_{eq}(\underline{x}_0, \theta) = 0^{c_1}, \;\; C_{ineq}(\underline{x}_0, \theta) \geq 0^{c_2} \end{gathered}$$

3.6 Least-Squares Approximation

Least squares approximation can be used to determine a control policy, though this policy is not necessarily optimal in terms of Problem 1. The LS policy is found by first solving Problem 1 with an OCS control policy and then performing a LS regression using the optimal state $\underline{x}_k{}^*$ and control trajectories $\underline{u}_k{}^*$ as given by Problem 4.

Problem 4.

$$\min_{\theta} \quad \sum_{k=0}^{N_h-1} ||\underline{u}_k{}^* - \mu(\underline{x}_k{}^*, \theta, k)||$$

$$s.t.$$

$$\underline{u}_k \in \mathcal{U}$$

While using LS approximation in this way seems like a natural path given that solving the OCS problem is common, doing so neglects interactions between the LS approximation error and optimal solution. In the following section, the value function derivative shows the control trajectory relies on not only the policy parameters but also the state trajectory. In Problem 4, the state trajectory is fixed, so the control inputs are solely optimized by the policy parameters. In Problem 1, the control inputs are optimized directly with the policy parameters and indirectly through the state trajectory, assuming some state feedback policy is used.

If there is some residual in the LS approximation (i.e. Problem 4 returns a cost greater than zero) or the control trajectory found in Problem 1, using the same state feedback policy, is not the optimal OCS, the optimal state trajectory using the feedback policy differs from the OCS state trajectory, as considered in [27]. Since Problem 4 results in a function mapping the optimal OCS trajectory X^* to the LS approximation $\hat{U}$, if there is some deviation from X^* in the resulting trajectory when implementing the LS policy, which exists since $\hat{U} \neq U^*$, a compounding error effect occurs and drives the resulting control trajectory away from U^*. For this reason, incorporating the system dynamics into the optimization improves the result. However, given sufficiently small LS approximation error, the results will be similar.

4 Optimization Methods

Along with properly constructing a problem, performing the optimization can prove to be time-consuming and difficult. Various optimization methods are available, and depending on the characteristics of the problem, some methods will outperform others. With linear or quadratic programming problems, optimal policies can largely be found analytically. These simpler problems generally rely on systems with linear dynamics. Given some general non-linear dynamics function, analytical solutions are likely impossible to find, so numerical approaches are used. All commonly used scripting software/programming languages have numerical optimization packages readily available or in some cases, already built-in. These optimization packages include a variety of different schemes including both gradient-based and gradient-free methods.

Gradient-based methods, such as vanilla gradient descent, are simple to perform and have proven stability, given some assumptions about the problem. In artificial intelligence research, the most common optimization strategy used is a gradient-based method termed ADAM [37]. Incorporating momentum, ADAM works well for systems with highly nonlinear stochastic objective functions as the momentum stabilizes the optimization process. The most time consuming part of gradient based methods is calculating the gradient. If a policy is defined by a large convolutional neural network, the number of parameters could easily reach into the millions. Both analytically computing the derivative offline and computing its value online can take significant computational effort. Built in optimization software commonly includes numerical differentiation, which eases the implementation but increases the computational effort.

Gradient-free methods rely on large amounts of data and some "brute-force" computations. Essentially, by simulating a massive number of choices, the optimal solution can be found simply by picking the choice with the best performance. If the search space is relatively small and discretized, performing a grid-search, which involves evaluating every possible option, can be tractable. In a continuous space, a grid-search is clearly not an op-

tion as there are an infinite number of choices. Methods like the genetic algorithm (GA) or particle swarm add some heuristics to a "brute-force" search. Using performance data from one generation of choices, the next generation can be chosen around areas that performed the best. Naturally, gradient-free search methods provide little guarantee of finding true optimal solutions, but in practice, they work well.

Three methods applied to policy optimization, are presented here. First, a derivative of the value function is taken with respect the the policy parameters, which is used for both vanilla gradient descent and ADAM. Secondly, the GA is explained with some special consideration for neural networks.

4.1 Gradient Descent

4.1.1 Vanilla Gradient Descent

Vanilla gradient descent is a simple yet powerful numerical optimization technique. Let $i \in \mathbb{N}_{\geq 0}$ be the iteration, then the gradient descent method is given by the discretized dynamic system:

$$\theta_{i+1} = \theta_i - \alpha \frac{dV_i^{\mu}(\underline{x}_0)}{d\theta_i} \tag{4.1}$$

where $\alpha \in \mathbb{R}_{>0}$ is the learning rate. Most commonly, $\alpha \ll 1$ but given a flat space, α can be close to 1 or potentially even larger. Varying the learning rate throughout the optimization can be useful as in the beginning, gradients are normally quite large, while near the end, the value function flattens significantly. The parameter θ can be initialized randomly, or in some cases, by a gradient-free method. Convergence is evaluated using

$$\frac{\left|V_{i+1}^{\mu}(\underline{x}_0) - V_i^{\mu}(\underline{x}_0)\right|}{V_i^{\mu}(\underline{x}_0)} \geq \epsilon \tag{4.2}$$

where $\epsilon \ll 1$ is the convergence tolerance. While Eq. (4.2) is true, the system in Eq. (4.1) continues iterating.

4.1.2 ADAM

ADAM, introduced by Kingma and Ba, has become the prominent optimization strategy for RL algorithms. ADAM is a gradient descent algorithm that includes estimates of the first and second moments of the value function, which help stabilize the system. Within this work, element-wise operations will be used along with matrix products. First, the operation $\cdot$ denotes the element-wise product:

$$\underline{a} \cdot \underline{b} = \begin{bmatrix} a_0 b_0 & \cdots & a_n b_n \end{bmatrix}^T \tag{4.3}$$

given $a, b \in \mathbb{R}^n$. Second, standard notion for division is used under the context of element-wise operations:

$$\frac{\underline{a}}{\underline{b}} = \begin{bmatrix} \frac{a_0}{b_0} & \cdots & \frac{a_n}{b_n} \end{bmatrix}^T, \quad b_i \neq 0 \; \forall \; i \in \mathbb{N}_{[0,n]} \tag{4.4}$$

Note, if a matrix inverse is to be used within a computation, a superscript will be used to denote the inverse. The ADAM algorithm is given by the following system

$$\bar{g}_i = \frac{dV_i^{\mu}(\underline{x}_0)}{d\theta_i} \tag{4.5}$$

$$\bar{m}_{i+1} = \beta_1 \cdot \bar{m}_i + (1 - \beta_1) \cdot \bar{g}_i \tag{4.6}$$

$$\bar{v}_{i+1} = \beta_2 \cdot \bar{v}_i + (1 - \beta_2) \cdot \bar{g}_i{}^2 \tag{4.7}$$

$$\hat{m}_i = \frac{\bar{m}_i}{1 - \beta_1{}^{i+1}} \tag{4.8}$$

$$\hat{v}_i = \frac{\bar{v}_i}{1 - \beta_2{}^{i+1}} \tag{4.9}$$

$$\theta_{i+1} = \theta_i - \alpha \frac{\hat{m}_i}{\sqrt{\hat{v}_i + \hat{\epsilon}}} \tag{4.10}$$

where $\bar{g}_i \in \mathbb{R}^p$ is a dummy variable that holds the value function derivative; $\bar{m}_i, \bar{v}_i \in \mathbb{R}^p$ are the biased first and second moment estimates, respectively; $\beta_1, \beta_2 \in \mathbb{R}_{[0,1)}$ are exponential decay rates for the moment estimates; $\hat{m}_i, \hat{v}_i \in \mathbb{R}^p$ are the bias-corrected first and second moment estimates, respectively, and $\hat{\epsilon} \ll 1$ is a dummy variable that helps avoid division

by zero. Convergence is evaluated by Eq. (4.2). Note, outside of the hyperparameters introduced by the ADAM algorithm, no additional information about the system is required as compared to vanilla gradient descent.

4.2 Value Function Derivative

Here, the derivative (transpose of the gradient) of the value function is taken to use in both the gradient based optimization methods presented and analytical solutions presented in the following sections. The goal is to find the value of θ such that the first order condition of optimality is satisfied:

$$\frac{dV^{\mu}(\underline{x}_0)}{d\theta} = 0 \tag{4.11}$$

In traditional MPC, the control inputs are implicitly defined by the states, which simplifies the first order derivative of the value function. Since open-loop control inputs are not explicitly defined by a feedback law, some differential change in an input near the beginning of the horizon does not directly affect a control input near the end. As the whole optimization process plays out, these two control inputs will exhibit some implicit coupling. However, given a feedback control law, some change in a control input near the beginning of the horizon will have a direct effect on a control input near the end since the inputs are explicitly defined by a function of the state. This is to say, a differential change in the policy parameters will not only affect the control inputs by changing the function mapping from states to inputs but also by changing the inputs to that function.

A differential change in the value function can be written as a component due to the two differentials dX and dU:

$$dV^{\mu}(\underline{x}_0) = \frac{\partial V^{\mu}(\underline{x}_0)}{\partial X} dX + \frac{\partial V^{\mu}(\underline{x}_0)}{\partial U} dU \tag{4.12}$$

These two partial derivatives, $\frac{\partial V^{\mu}(\underline{x}_0)}{\partial X}$ and $\frac{\partial V^{\mu}(\underline{x}_0)}{\partial U}$, are of dimension $1 \times n(N_h + 1)$ and $1 \times mN_h$, respectively, while the differentials, dX and dU are of dimension $n(N_h + 1)$ and

mN_h, respectively, the same dimensions as X and U.

$$dX = \begin{bmatrix} d\underline{x}_0{}^T & d\underline{x}_1{}^T & \cdots & d\underline{x}_{N_h-1}{}^T & d\underline{x}_{N_h}{}^T \end{bmatrix}^T \tag{4.13}$$

$$dU = \begin{bmatrix} d\underline{u}_0{}^T & d\underline{u}_1{}^T & \cdots & d\underline{u}_{N_h-2}{}^T & d\underline{u}_{N_h-1}{}^T \end{bmatrix}^T \tag{4.14}$$

The partial derivatives of the value function are formed as

$$\frac{\partial V^\mu(\underline{x}_0)}{\partial X} = \begin{bmatrix} \frac{\partial V^\mu(\underline{x}_0)}{\partial \underline{x}_0}^T & \frac{\partial V^\mu(\underline{x}_0)}{\partial \underline{x}_1}^T & \cdots & \frac{\partial V^\mu(\underline{x}_0)}{\partial \underline{x}_{N_h-1}}^T & \frac{\partial V^\mu(\underline{x}_0)}{\partial \underline{x}_{N_h}}^T \end{bmatrix} \tag{4.15}$$

$$\frac{\partial V^\mu(\underline{x}_0)}{\partial U} = \begin{bmatrix} \frac{\partial V^\mu(\underline{x}_0)}{\partial \underline{u}_0}^T & \frac{\partial V^\mu(\underline{x}_0)}{\partial \underline{u}_1}^T & \cdots & \frac{\partial V^\mu(\underline{x}_0)}{\partial \underline{u}_{N_h-2}}^T & \frac{\partial V^\mu(\underline{x}_0)}{\partial \underline{u}_{N_h-1}}^T \end{bmatrix} \tag{4.16}$$

which leads to

$$dV^\mu(\underline{x}_0) = \frac{\partial V^\mu(\underline{x}_0)}{\partial \underline{x}_0}^T d\underline{x}_0 + \cdots + \frac{\partial V^\mu(\underline{x}_0)}{\partial \underline{x}_{N_h}}^T d\underline{x}_{N_h} + \frac{\partial V^\mu(\underline{x}_0)}{\partial \underline{u}_0}^T d\underline{u}_0 + \cdots + \frac{\partial V^\mu(\underline{x}_0)}{\partial \underline{u}_{N_h-1}}^T d\underline{u}_{N_h-1} \tag{4.17}$$

Any differential change in the state trajectory will result entirely from some change in the policy parameter:

$$dX = \frac{\partial X}{\partial \theta} d\theta \tag{4.18}$$

where $\frac{\partial X}{\partial \theta} \in \mathbb{R}^{n(N_h+1)\times p}$ is the block vector

$$\frac{\partial X}{\partial \theta} = \begin{bmatrix} \frac{\partial \underline{x}_0}{\partial \theta} \\ \vdots \\ \frac{\partial \underline{x}_{N_h}}{\partial \theta} \end{bmatrix} \tag{4.19}$$

that incorporates the system dynamics within the problem, and can be formed with:

$$\frac{\partial \underline{x}_{k+1}}{\partial \theta} = \left(\frac{\partial f(\underline{x}_k, \underline{u}_k)}{\partial \underline{x}_k} + \frac{\partial f(\underline{x}_k, \underline{u}_k)}{\partial \underline{u}_k} \frac{\partial \mu(\underline{x}_k)}{\partial \underline{x}_k} \right) \frac{\partial \underline{x}_k}{\partial \theta} + \frac{\partial f(\underline{x}_k, \underline{u}_k)}{\partial \underline{u}_k} \frac{\partial \mu(\underline{x}_k)}{\partial \theta} \tag{4.20}$$

for rows $k \in \mathbb{N}_{[0,N_h]}$ with the initial condition $\frac{\partial \underline{x}_0}{\partial \theta} = 0^{n\times p}$. A detailed derivation of this

equation is given in Appendix A. Now, the differential dU can be expanded as two terms:

$$dU = \frac{\partial U}{\partial \theta} d\theta + \frac{\partial U}{\partial X} dX \tag{4.21}$$

where the two partial derivative terms $\frac{\partial U}{\partial \theta} \in \mathbb{R}^{mN_h \times p}$ and $\frac{\partial U}{\partial X} \in \mathbb{R}^{mN_h \times n(N_h+1)}$ are given by:

$$\frac{\partial U}{\partial \theta} = \begin{bmatrix} \frac{\partial \mu(\underline{x}_0)}{\partial \theta} \\ \vdots \\ \frac{\partial \mu(\underline{x}_{N_h-1})}{\partial \theta} \end{bmatrix} \tag{4.22}$$

and

$$\frac{\partial U}{\partial X} = \begin{bmatrix} \frac{\partial \mu(\underline{x}_0)}{\partial \underline{x}_0} & 0 & \cdots & 0 & 0 \\ 0 & \frac{\partial \mu(\underline{x}_1)}{\partial \underline{x}_1} & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \frac{\partial \mu(\underline{x}_{N_h-1})}{\partial \underline{x}_{N_h-1}} & 0 \end{bmatrix} \tag{4.23}$$

where $0 := 0^{m \times p}$. Note, the last column of zeros in Eq. (4.23) shows that $\underline{x}_{N_h}$ has no direct affect on the control trajectory. Substituting Eqs. (4.18) and (4.21) back into Eq. (4.12) and factoring $d\theta$ yields:

$$dV^{\mu}(\underline{x}_0) = \left(\frac{\partial V^{\mu}(\underline{x}_0)}{\partial X} \frac{\partial X}{\partial \theta} + \frac{\partial V^{\mu}(\underline{x}_0)}{\partial U} \frac{\partial U}{\partial \theta} + \frac{\partial V^{\mu}(\underline{x}_0)}{\partial U} \frac{\partial U}{\partial X} \frac{\partial X}{\partial \theta} \right) d\theta \tag{4.24}$$

Then, the total derivative of the state value can be written as:

$$\frac{dV^{\mu}(\underline{x}_0)}{d\theta} = \left(\frac{\partial V^{\mu}(\underline{x}_0)}{\partial X} + \frac{\partial V^{\mu}(\underline{x}_0)}{\partial U} \frac{\partial U}{\partial X} \right) \frac{\partial X}{\partial \theta} + \frac{\partial V^{\mu}(\underline{x}_0)}{\partial U} \frac{\partial U}{\partial \theta} \tag{4.25}$$

This gives the required information to perform gradient descent within the parameter space and find a solution to Eq (4.11).

4.2.1 Stochastic Value Function Derivative Approximation

Given a stochastic problem, an approximation can be made to calculate the derivative of the objective function. The law of large numbers can be used to show that the average value of a selection of randomly distributed points approaches the expected value of their probability distribution function (PDF) as the number of sampled points increases, given that the sampled points are uncorrelated. Consider a sample size $N_b \in \mathbb{N}_{>0}$:

$$\mathbb{E}\left[Z\right] \approx \frac{1}{N_b} \sum_{i=1}^{N_b} z_i \tag{4.26}$$

where $z_i \sim \mathcal{D}_Z$ is a sample point and D_Z is the PDF of the random variable Z. Both the expectation and derivative operators are linear, so

$$\frac{\partial \mathbb{E}\left[V^\mu \underline{x}_0\right]}{\partial \theta} = \mathbb{E}\left[\frac{\partial V^\mu \underline{x}_0}{\partial \theta}\right] \tag{4.27}$$

Using Eq. (4.26) and (4.27), the expected derivative of the value function can be approximated as

$$\frac{\partial \mathbb{E}\left[V^\mu \underline{x}_0\right]}{\partial \theta} = \frac{1}{N_b} \sum_{i=1}^{N_b} \frac{\partial V^\mu \underline{x}_0^i}{\partial \theta} \tag{4.28}$$

where $\underline{x}_0^i \sim \mathcal{D}_{\underline{x}_0}$ is a sampled initial condition. If the initial condition is deterministic and the problem is stochastic solely due to the dynamics, $\underline{x}_0^i = \underline{x}_0 \; \forall \; i \in \mathbb{N}_{[1,N_b]}$. In implementation, this requires N_b simulations be run in parallel with the derivative of each individual trajectory evaluated using Eq. (4.25) The mean of the derivatives can then be used within a gradient descent algorithm. ADAM includes a momentum component, which aids in filtering any noise caused by fluctuation of the objective function from approximation error.

4.3 Genetic Algorithm

The GA started with Holland who was attempting to simultaneously explore adaptation in biological systems and produce methods for solving optimization problems. Affenzeller et al. explain the origins and inner workings of the GA along with the parallel research in

evolutionary strategies, which differs slightly. Based on the evolution of organisms, the GA works by evaluating the performance of clusters- or generations- of policy parameters that are kept at a fixed population size $N_p \in \mathbb{N}$. The parameters in a generation that performed the best are kept and used to create "children" that are tested in the next generation. Any parameters in a generation that do not perform well enough to reproduce are discarded and replaced by these children. As the generations continue to pass, the parameters start to converge in an area of high performance.

There are three main operators used in the GA, which makes implementations fairly simple. The first, and most apparent, is reproduction, where the best $N_r \in \mathbb{N}$ performing parameters of a generation are passed onto the next generation. Original implementations leave reproduction up to chance by directly correlating the probability of reproduction to performance, but in this work, a set number of best performing parameters are reproduced. Crossover is the main operator and is what produces children. Two of the N_r high performing parameters in a generation are combined randomly to produce a new child parameter:

$$\theta_{i+1}^{j} = \begin{bmatrix} \theta_{1,i}^{p_1} & \cdots & \theta_{c_p,i}^{p_1} & \theta_{c_p+1,i}^{p_2} & \cdots & \theta_{p,i}^{p_2} \end{bmatrix}^T \tag{4.29}$$

where $j \in \mathbb{N}_{[1,N_p]}$ is the generational parameter index, $p_1, p_2 \in \mathbb{N}_{[1,N_p]}$ are the indexes for the two parents at the i-th generation, $c_p \sim U_d(1,p) \in \mathbb{N}_{[1,p]}$ is the crossover point, $U_d(a,b)$ is the discrete uniform distribution over range $[a,b]$, and $p \in \mathbb{N}$ is the size of θ. The last operator is mutation. Each member in the population has some set probability of mutating at the beginning of a new generation, which can be achieved with

$$\theta_i^j = \begin{cases} \theta_i^j + \nu_m & \text{if } \tilde{\epsilon}^j < \tilde{\omega} \\ \theta_i^j & \text{otherwise} \end{cases} \tag{4.30}$$

where $\nu_m \sim \mathcal{N}(0, \sigma_m) \in \mathbb{R}^p$ is the mutation, $\mathcal{N}(a,b)$ is a normal distribution centered at a with standard deviation b, $\sigma_m \in \mathbb{R}^p$ is the mutation standard deviation, $\tilde{\epsilon}^j \in \mathbb{R}_{[0,1)}$ is

probability of the j-th parameter mutating, and $\tilde{\omega} \sim U(0,1) \in \mathbb{R}$ is a dummy uniformly distributed random variable that causes mutation.

Some special consideration can be taken to increase exploration and decrease the training time for certain policy architectures while using the GA. Note, this work is not dedicated to optimization methods, so this claim is made without providing sufficient comparative evidence. Some policy architectures have distinct components (e.g. NNs or time-varying linear feedback gains), so crossover can be done on a per component basis. Looking at the time-varying linear feedback gain, this could entail performing a crossover between two gains from the same time instance. So, for a sequence of N_h feedback gains, there would be N_h crossovers occurring. Similarly, NNs have obvious components, namely the various weighting matrices and biases. Every weight matrix and associated bias from a particular member of a population can be combined with other applicable parts from other members.

The GA can be used to solve Problem 2, which means a stochastic objective function will be minimized. Ideally, enough simulations would be performed to perfectly describe the state distributions and the mean value would approach the expected value:

$$\mathbb{E}\left[V^{\mu}(\underline{x}_0)\right] \approx \frac{1}{N_b} \sum_{i=1}^{N_b} V^{\mu}(\underline{x}_0^i) \tag{4.31}$$

In practice, this approximation may be erroneous as there are computational limitations. Since the GA analyzes these mean values to determine which members of the population are performing well, it is important to properly compare the members. A bundle of trajectories is used to approximate the expected value, and these trajectories each have their own randomly generated initial condition. Some initial conditions inherently return a lower cost than others, regardless of the controller performance. So, using the same initial conditions for every member of the population is important for proper comparison.

5 Control Policy Architectures

Varying control policy architectures have varying characteristics, training techniques, and capabilities. Here, a select number of architectures are explored including: the OCS, constant linear feedback gain (CG), time-varying linear feedback gain (TG), multi-layer perceptron (MLP) networks, and convolutional neural networks (CNN). Along with presenting the policy architectures, the derivatives of each architecture with respect to their inputs, $\frac{\partial U}{\partial X}$ and $\frac{\partial U}{\partial \theta}$, are presented for use within Eq. (4.25) and upper and lower bound control input constraint handling is explored.

Time-varying policies like the OCS and TG are commonly found within optimal control. Open-loop control sequences are used within conventional MPC algorithms as they can produce abstract control trajectories, while TGs are found to be optimal solutions to both deterministic and stochastic LQ problems. Time-varying gains are also shown to work for nonlinear systems in a suboptimal manner [6]. Constant linear feedback gains are widely studied for control of linear and nonlinear systems. In this context, the CG provides not only a robust control method for applicable initial conditions but also the fewest number of parameters to optimize (except for some specific cases like an OCS over an extremely short horizon).

Neural network based policies are extremely popular in RL and can be very powerful. A single CNN was trained using a Monte Carlo tree search algorithm to play the game Go and in an real world sanctioned competition, beat the best player in the world in a multi-round game [4]. Given a large enough network, the approximation capabilities of NNs are immense and lead to their use within complex systems with large state spaces. In the case of Go, even though it is a purely discrete state-action space, which might lend itself to DP, there are around 10^{360} possible moves in an average game, so searching the entire state-action space to determine optimal policies for each case is intractable. Hence, NNs can be used to find approximations of the the true optimal policies over some reasonable amount of data, though in this specific Go case, the amount of data and computational power is still immense.

5.1 Control Bound Constraint

In most systems, the control inputs are physically constrained to lie within some range (e.g. the ailerons on a wing have some finite maximum deflection). For some cases, letting the physical system saturate itself may lead to acceptable results, but for this work, the control bounds are used within the optimization to find solutions that work best given those limitations. This section focuses on control bounds that do not change as a function of the state, i.e. Problem 1. By incorporating saturating functions into the the control policy, the control bounds will be satisfied and incorporated into the optimization.

An output layer is commonly used to map the hidden layer outputs to the desired space within a NN. In some cases, the output layer is just a simple matrix multiplication, which would be applicable for cases where $\mathcal{U} = \mathbb{R}^m$. If $\mathcal{U} \subset \mathbb{R}^m$, then continuous activation functions (AFs) like the sigmoid and hyperbolic tangent or discontinuous AFs like $\min(\cdot, \cdot)$ and $\max(\cdot, \cdot)$ can be used in combination with a matrix multiplication to constrain the outputs. Continuous AFs aid in mathematical proofs and suffer less from dropout, which is when the gradient becomes zero and stops the learning process, when compared to discontinuous AFs. However, discontinuous AFs are simple and efficient to implement and, in practice, perform better given proper training.

One of the first successful AFs, the sigmoid, gives outputs over the range $(0, 1)$ and is defined as

$$(\underline{w}_k)^+_{sigmoid} = \frac{1}{1 + e^{-\underline{w}_k}} \tag{5.1}$$

where $\underline{w} \in \mathbb{R}^m$ is the hidden layer output. Note, hidden layer is a term commonly used for the layers of neurons in a NN but in this context, the hidden layer is taken to be the portion of the policy that comes before the output clamping and scaling. In the case of linear state feedback, the hidden layer output is $\underline{w}_k = \text{vec}^{-1}(\theta)\underline{x}_k$. The output of the sigmoid can be interpolated to cover the range of the action space using the linear method

$$\underline{u}_k = (\underline{u}_{max} - \underline{u}_{min}) \cdot (\underline{w}_k)^+_{sigmoid} + \underline{u}_{min} \tag{5.2}$$

where $\underline{u}_{max}, \underline{u}_{min} \in \mathcal{U}$ are the maximum and minimum admissible control inputs (i.e. they lie at the edges of the control set $\mathcal{U}$). With similar characteristics, the hyperbolic tangent gives values over the range $(-1, 1)$:

$$(\underline{w}_k)^+_{tanh} = \tanh(\underline{w}_k) = \frac{e^{\underline{w}_k} - e^{-\underline{w}_k}}{e^{\underline{w}_k} + e^{-\underline{w}_k}} \tag{5.3}$$

which works well for cases where $\underline{u}_{max} = -\underline{u}_{min}$ and the output can be mapped to the action space with

$$\underline{u}_k = \underline{u}_{max} \cdot (\underline{w}_k)^+_{tanh} \tag{5.4}$$

Figure 5.1 compares the sigmoid and hyperbolic tangent and clearly shows the two output ranges. The discontinuous activation functions $\min(\cdot, \cdot)$ and $\max(\cdot, \cdot)$ return the element-wise

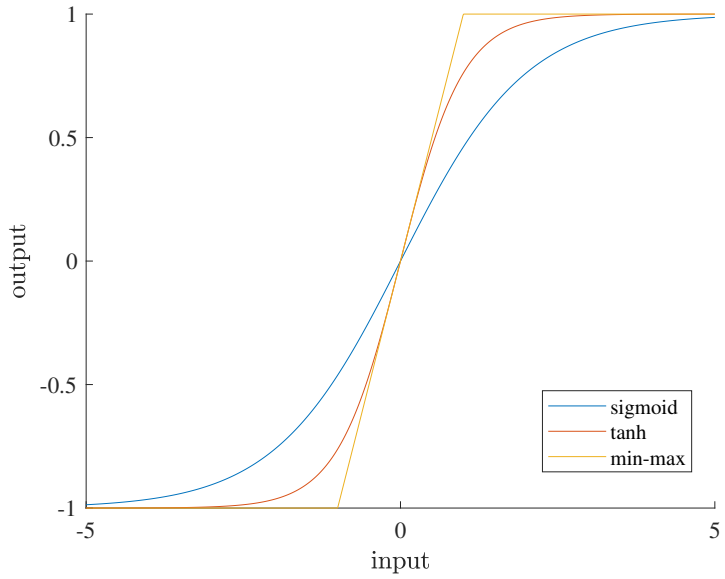

Figure 5.1 Sigmoid, hyperbolic tangent, and min-max activation function outputs given $u \in \mathbb{R}_{[-1,1]}$.

minimum and maximum of the two inputs, respectively. These can be used directly with the hidden layer output and control bounds as

$$\underline{u}_k = \min(\max(\underline{w}_k, \underline{u}_{min}), \underline{u}_{max}) \tag{5.5}$$

which will be referred to as the min-max AF.

Table 5.1 gives a comparison of the computational speed for these three output layer clamping and scaling techniques. Clearly, the discontinuous case is the quickest, not only because the two functions are simple but also because they require no additional scaling step. The hyperbolic tangent function becomes suboptimal when $\underline{u}_{max} \neq -\underline{u}_{min}$ since two scaling steps would need to be performed: one to map the AF output from $(-1, 1)$ to $(0, 1)$ and a second to map that scaled output to the control space. Note, Table 5.1 is presented solely for comparative purposes as the timing being performed is not rigorous.

Table 5.1 Comparative average computational time of output layer activation functions and scaling over 10^6 iterations where the input at each iteration is taken to be $\underline{w} \in \mathbb{R}^{1000\times 4} \sim \mathcal{N}(0, 1)$ and the control bounds are $\underline{u}_{max} = -\underline{u}_{min} = 1^4$.

	min-max	tanh	sigmoid
time $(10^{-5}s)$	7.685	11.71	13.47
difference (%)	-	52.4	75.2

5.1.1 Output Layer Derivative and Dropout

The derivative of each AF and scaling function is given here for use in determining $\frac{\partial U}{\partial X}$ and $\frac{\partial U}{\partial \theta}$ and will be applicable to any control policy architecture with control bound constraints. First, the derivative of the sigmoid is taken as

$$\frac{\partial (\underline{w}_k)^+_{sigmoid}}{\partial \underline{w}_k} = \frac{e^{-\underline{w}_k}}{(1 + e^{-\underline{w}_k})^2} \tag{5.6}$$

which can be combined with the scaling function derivative

$$\frac{\partial \underline{u}_k}{\partial (\underline{w}_k)^+_{sigmoid}} = (\underline{u}_{max} - \underline{u}_{min}) \tag{5.7}$$

to produce the block matrix:

$$\frac{\partial U}{\partial W} = \begin{bmatrix} (\underline{u}_{max} - \underline{u}_{min}) \cdot \frac{e^{-\underline{w}_0}}{(1+e^{-\underline{w}_0})^2} & \cdots & 0^m \\ \vdots & \ddots & \vdots \\ 0^m & \cdots & (\underline{u}_{max} - \underline{u}_{min}) \cdot \frac{e^{-\underline{w}_{N_h-1}}}{(1+e^{-\underline{w}_{N_h-1}})^2} \end{bmatrix} \tag{5.8}$$

where $W \in \mathbb{R}^{mN_h}$ is the trajectory of hidden layer outputs:

$$W := \begin{bmatrix} \underline{w}_0{}^T & \cdots & \underline{w}_{N_h-1}{}^T \end{bmatrix}^T \tag{5.9}$$

Next, the hyperbolic tangent derivative is taken as

$$\frac{\partial (\underline{w}_k)^+_{tanh}}{\partial \underline{w}_k} = 1 - \tanh^2(\underline{w}_k) \tag{5.10}$$

$$= 1 - \left(\frac{e^{\underline{w}_k} - e^{-\underline{w}_k}}{e^{\underline{w}_k} + e^{-\underline{w}_k}} \right)^2 \tag{5.11}$$

with the simple scaling factor derivative

$$\frac{\partial \underline{u}_k}{\partial (\underline{w}_k)^+_{tanh}} = \underline{u}_{max} \tag{5.12}$$

The block matrix derivative for the hyperbolic tangent output layer is then formed as:

$$\frac{\partial U}{\partial W} = \begin{bmatrix} \underline{u}_{max} \cdot \left(1 - \left(\frac{e^{\underline{w}_0} - e^{-\underline{w}_0}}{e^{\underline{w}_0} + e^{-\underline{w}_0}} \right)^2 \right) & \cdots & 0^m \\ \vdots & \ddots & \vdots \\ 0^m & \cdots & \underline{u}_{max} \cdot \left(1 - \left(\frac{e^{\underline{w}_{N_h-1}} - e^{-\underline{w}_{N_h-1}}}{e^{\underline{w}_{N_h-1}} + e^{-\underline{w}_{N_h-1}}} \right)^2 \right) \end{bmatrix} \tag{5.13}$$

Lastly, the min-max AF derivative can be written for the i-th element of the k-th time step control input as

$$\frac{\partial \underline{u}_{k,i}}{\partial \underline{w}_{k,i}} = \begin{cases} 1 & \text{if } \underline{u}_{min,i} \leq \underline{w}_i \leq \underline{u}_{max,i} \\ 0 & \text{otherwise} \end{cases} \tag{5.14}$$

and used to assemble the block matrix:

$$\frac{\partial U}{\partial W} = \begin{bmatrix} \frac{\partial \underline{u}_0}{\partial \underline{w}_0} & \cdots & 0^m \\ \vdots & \ddots & \vdots \\ 0^m & \cdots & \frac{\partial \underline{u}_{N_h-1}}{\partial \underline{w}_{N_h-1}} \end{bmatrix} \tag{5.15}$$

Figure 5.2 shows the derivatives of the sigmoid, hyperbolic tangent, and min-max AFs over the range $[-5, 5]$ given that $u \in \mathbb{R}_{[-1,1]}$. This figure explains how a poor initial guess can severely limit the controller's performance, especially when using the min-max output layer. Suppose an OCS control policy were used, then the control inputs do not rely on information from the state trajectory. Since $\frac{\partial U}{\partial X} = 0^{mN_h \times n(N_h+1)}$, the control trajectory is only updated via the derivative $\frac{\partial U}{\partial \theta}$. Furthermore, suppose an initial guess was given where $w_k > 1 \; \forall \; k \in \mathbb{N}_{[0,N_h-1]}$ and the min-max output layer is used. Then, $\frac{\partial U}{\partial \theta} = 0^{mN_h \times mN_h}$, and no updates will occur within a gradient based optimization. Since both the sigmoid and hyperbolic tangent AFs are based on the exponential function, their derivatives only asymptotically approach zero as $w \to \infty$. However, as seen in Fig. 5.2, the output layer derivative of the hyperbolic tangent AF quickly approaches zero, which can slow the optimization process.

The issue of dropout largely plagues time varying control policies like the OCS and TG, but some adjustments can be made to work past the issue. Stationary policies, like the NNs implemented here and the CG feedback, are susceptible to dropout as well but their updates rely much more on the interaction of the states and control inputs $\frac{\partial U}{\partial X}$, which reduces the issue somewhat. Most obviously, using a continuous output layer AF would keep the derivative from completely dropping out and fix most issues. However, given the potential online performance increase of operating the min-max AF, other methods can be used to keep dropout form occurring during training.

In the OCS case, the search space can simply be constrained to $\mathcal{U}$. In general, $\theta \in \mathbb{R}^p$ since the output layer clamping relegates the inputs to the desirable set, but if $\theta \in \mathcal{U}$, then no dropout will occur as $\underline{u}_{min} \leq \underline{w} \leq \underline{u}_{max}$ and the derivative of the min-max AF is defined

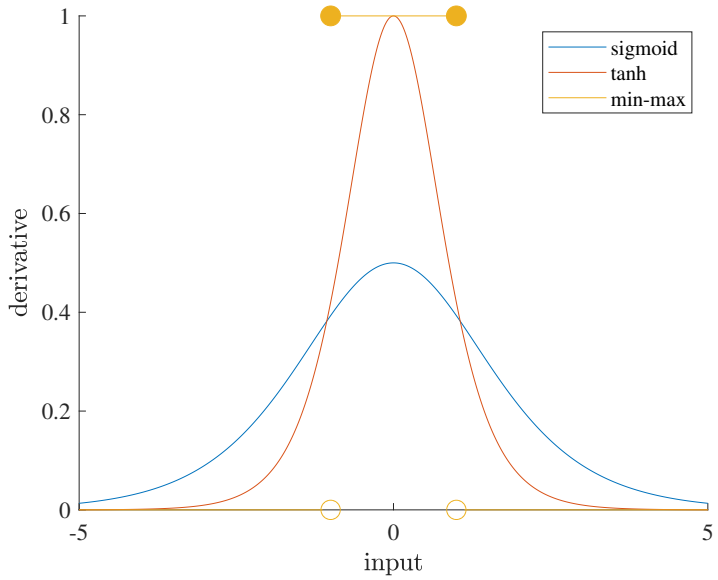

Figure 5.2 Sigmoid, hyperbolic tangent, and min-max activation function derivatives given $u \in \mathbb{R}_{[-1,1]}$

to be 1 at the boundary. In implementation, this is as easy as saturating θ by replacing offending entries with the applicable maximum or minimum value.

In the TG case, the hidden layer output $\underline{w}_k$ is a function of the system state, so limiting the parameter space does not fix the problem. An approximation of the min-max output layer derivative can be used in this case, which is similar to an AF known as leaky ReLU:

$$\frac{\partial \underline{u}_{k,i}}{\partial \underline{w}_{k,i}} = \begin{cases} 1 & \text{if } \underline{u}_{min,i} \leq \underline{w}_i \leq \underline{u}_{max,i} \\ \epsilon & \text{otherwise} \end{cases} \tag{5.16}$$

where $0 < \epsilon << 1$. By replacing the zero derivative with a small number, information will still be able to flow into the policy parameter via the small gradient. The original output layer remains unaffected, so the control bounds are still satisfied. Using this approximation simply allows the policy parameter to eventually learn to produce $\underline{u}_k$ within the interior of $\mathcal{U}$ if the boundary is not the optimal solution.

Table 5.2 Comparative average computational time of output layer activation function derivatives and scaling over 10^6 iterations where the input at each iteration is taken to be $\underline{w} \in \mathbb{R}^{1000\times 4} \sim \mathcal{N}(0,1)$ and the control bounds are $\underline{u}_{max} = -\underline{u}_{min} = 1^4$.

	min-max	tanh	sigmoid
time ($10^{-5}s$)	14.37	16.73	10.93
difference (%)	-	16.3	-23.9

5.2 Open-Loop Control Sequences

An OCS is a function that maps time steps to control inputs

$$\mu_{OCS} : \Theta \times \mathbb{N}_{[0,N_h-1]} \mapsto \mathcal{U} \tag{5.17}$$

and is given by

$$\underline{u}_k = \left(\theta_{[mk+1,m(k+1)]}\right)^+_{OL} \tag{5.18}$$

where $\Theta = \mathbb{R}^{mN_h}$ and $(\cdot)^+ : \mathbb{R}^m \mapsto \mathcal{U}$. Given there are no additional constraints placed on the control inputs and $\mathcal{U} = \mathbb{R}^m$, the parameter space is simply an extension of the control space, i.e. $\Theta = \mathcal{U}^{N_h}$ and $(\cdot)^+$ is an identity function. Since every input within an OCS is an independent event, control trajectories can take on any arbitrary shape, which means the OCS is the way to produce the true optimal solution to Problem 1 and 3. Similarly, in Problem 4 a control policy is trained to reproduce an OCS for a single initial condition or a range of OCSs for multiple initial conditions, since the OCS produces optimal solutions.

5.2.1 Open-Loop Control Sequence Derivative

Implementing an OCS policy architecture is straightforward as the policy derivatives are simple. The control inputs do not depend on the states and are equal to the policy parameter, so

$$\frac{\partial W}{\partial X} = 0^{mN_h \times n(N_h+1)} \tag{5.19}$$

and

$$\frac{\partial W}{\partial \theta} = I^{mN_h \times mN_h} \tag{5.20}$$

Using the chain rule with the output layer derivatives will give the derivatives required for Eq. (4.25).

5.3 Linear Feedback Gain Based Policies

Linear feedback gains are widely used to control both linear and nonlinear systems. A large amount of theory has been produced that shows linear feedback gains can create closed-loop linear systems with any desired characteristics, given that the system is controllable. Optimal controllers for linear systems often include linear feedback components, e.g. the finite and infinite horizon LQRs. Along with the linear system theory, localized nonlinear systems can be controlled using linear feedback gains by linearizing the system and applying linear control theory. In some cases, a time-varying linear feedback gain can even control the system globally. Similar to the finite horizon LQR, time-varying feedback gains can be found with a closed form solution involving a state-dependent Riccati equation [6]. In this section, both constant gain feedback and time-varying gain feedback are discussed.

5.3.1 Constant Gain

Constant gain feedback provides limited approximation capability and restricts the state space in which the initial condition can reside for the system to be stabilizable but simultaneously simplifies the optimization by reducing the size of θ and inherently adds robustness to the system. The CG policy maps states to control inputs

$$\mu_{CG} : \mathcal{X} \times \Theta \mapsto \mathcal{U} \tag{5.21}$$

and is given by:

$$\underline{u}_k = \left(\text{vec}^{-1}_{m\times n}(\theta)\underline{x}_k\right)^+_{OL} \tag{5.22}$$

Constant Gain Derivative

The hidden layer derivatives of the CG policy are given as

$$\frac{\partial W}{\partial X} = \begin{bmatrix} \text{vec}^{-1}_{m\times n}(\theta) & \cdots & 0^{m\times n} \\ \vdots & \ddots & \vdots \\ 0^{m\times n} & \cdots & \text{vec}^{-1}_{m\times n}(\theta) \end{bmatrix} \tag{5.23}$$

and

$$\frac{\partial W}{\partial \theta} = \begin{bmatrix} \underline{x}_0{}^T & \cdots & 0^{1\times n} \\ \vdots & \ddots & \vdots \\ 0^{1\times n} & \cdots & \underline{x}_0{}^T \\ \vdots & & \vdots \\ \underline{x}_{N_h-1}{}^T & \cdots & 0^{1\times n} \\ \vdots & \ddots & \vdots \\ 0^{1\times n} & \cdots & \underline{x}_{N_h-1}{}^T \end{bmatrix} \tag{5.24}$$

Equation (5.24) is a block matrix that can be rewritten as:

$$\frac{\partial W}{\partial \theta} = \begin{bmatrix} \frac{\partial \underline{w}_0}{\partial \theta} \\ \vdots \\ \frac{\partial \underline{w}_{N_h-1}}{\partial \theta} \end{bmatrix} \tag{5.25}$$

where each block $\frac{\partial \underline{w}_k}{\partial \theta} \in \mathbb{R}^{m\times nm}$.

5.4 Time-Varying Gain

A TG policy maps the current state and time step to a control input

$$\mu_{TG} : \mathcal{X} \times \Theta \times \mathbb{N}_{[0,N_h-1]} \mapsto \mathcal{U} \tag{5.26}$$

and is given by

$$\underline{u}_k = (K_k \underline{x}_k)^+_{OL} \,. \tag{5.27}$$

where $K_k = \text{vec}^{-1}_{m\times n}(\theta_{[nmk+1,nm(k+1)]})$ and $\theta \in \mathbb{R}^{nmN_h}$. Time-varying gains have the potential to globally control some nonlinear systems. Given $m \leq n$, any $\underline{u}_k$ can be produced from any $\underline{x}_k$, and the TG policy has universal approximation capability and can perfectly reproduce the optimal OCS.

Time-Varying Gain Derivative

The TG policy derivatives are quite similar to the CG policy derivatives, except now at every time step there is a different feedback gain being used and the increased dimension of θ increases the size of the derivative with respect to the policy parameter. They are given as

$$\frac{\partial W}{\partial X} = \begin{bmatrix} K_0 & \cdots & 0^{m\times n} \\ \vdots & \ddots & \vdots \\ 0^{m\times n} & \cdots & K_{N_h-1} \end{bmatrix} \tag{5.28}$$

and

$$\frac{\partial W}{\partial \theta} = \begin{bmatrix} \underline{x}_0{}^T & \cdots & 0^{1\times n} & \cdots & 0^{1\times n} & \cdots & 0^{1\times n} \\ \vdots & \ddots & \vdots & & \vdots & \ddots & \vdots \\ 0^{1\times n} & \cdots & \underline{x}_0{}^T & \cdots & 0^{1\times n} & \cdots & 0^{1\times n} \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0^{1\times n} & \cdots & 0^{1\times n} & \cdots & \underline{x}_{N_h+1}{}^T & \cdots & 0^{1\times n} \\ \vdots & \ddots & \vdots & & \vdots & \ddots & \vdots \\ 0^{1\times n} & \cdots & 0^{1\times n} & \cdots & 0^{1\times n} & \cdots & \underline{x}_{N_h+1}{}^T \end{bmatrix} \tag{5.29}$$

Again, this can be rewritten as

$$\frac{\partial W}{\partial \theta} = \begin{bmatrix} \frac{\partial \underline{w}_0}{\partial \text{vec}(K_0)} & \cdots & 0^{m\times n} \\ \vdots & \ddots & \vdots \\ 0^{m\times n} & \cdots & \frac{\partial \underline{w}_{N_h-1}}{\partial \text{vec}(K_{N_h-1})} \end{bmatrix} \tag{5.30}$$

where each block $\frac{\partial \underline{w}_k}{\partial \text{vec}(K_k)} \in \mathbb{R}^{m\times n}$.

5.5 Neural Network-Based Policies

Neural networks have become widely popular for their generalized usability and power. As everyday computers become more powerful and RL libraries become more available, the accessibility of NN based estimation and control increases, which fuels more research. Neural networks can approximate most functions, given the NN's size is adjusted accordingly, and are used not only in least squares fitting tasks but also in direct function optimization. Within modern actor-critic RL algorithms, two NNs are used; one, the critic, is trained using a least squares regression, while the other, the actor, is updated using the gradient of the critic. The actor NN is trying to predict the optimal inputs to the critic and in turn, the optimal actions for the environment. In this work, the actor NN will be used along with analytical dynamics of the environment, as opposed to learning them via a least squares regression.

Most of a NN is composed of hidden nodes. These hidden nodes, each equipped with an activation function (AF), exist within some number of hidden layers. At a minimum, a NN can have a single hidden layer, while the maximum number of layers, theoretically, is infinite. Deep NNs have a high number of hidden layers and an issue termed the vanishing gradient, where by means of the chain rule, the magnitude of the gradient decreases as it backpropagates from the output layer to the input layer [40]. Single layer NNs have one hidden layer but still have uniform approximation capability if they become infinitely wide [41]. A layer's width, or dimension, refers to the number of nodes in that layer. Designing a NN requires appropriate selection of both the number of layers and the size of each one.

There are three main types of NNs: multi-layer perceptron (MLP), recurrent (RNN), and convolutional (CNN). Multi-layer perceptron networks are the simplest and are used in both recurrent and convolutional networks. Using simple matrix multiplication and nonlinear activation functions, feeding data through a MLP network is straightforward:

$$\underline{w}^{j+1} = vec^{-1}_{h_{j+1}\times h_j}(\theta_{[i,k]})\left(\underline{w}^{j}\right)^{+} + \theta_{[a,b]} \tag{5.31}$$

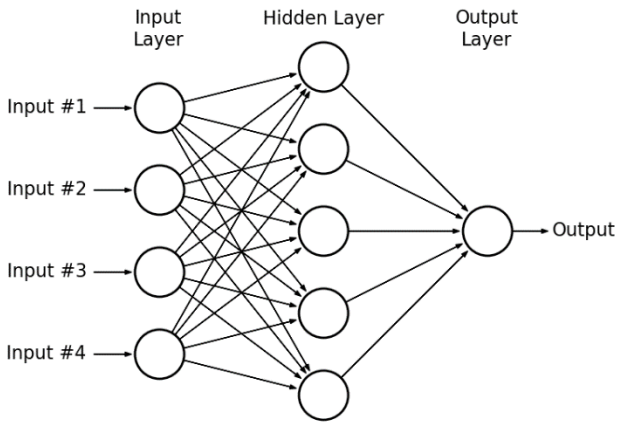

Figure 5.3 An example multi-layer perceptron neural network [1]

$\underline{w}^j \in \mathbb{R}^{h_j}$ is the value at the j-th layer, $h_j \in \mathbb{N}$ is the dimension of the j-th layer, the range subscript $[i, k]$ denotes the values in θ that correspond to the j-th weights of the NN, the range $[a, b]$ corresponds to the j-th bias value, and $(\cdot)^+ : \mathbb{R}^{h_j} \mapsto \mathbb{R}^{h_j}$ is the AF.

5.5.1 Activation Functions

Activation functions are what give NNs their name, nonlinear characteristics, and universal approximation capabilities. Inspired by neuroscience, McCulloch and Pitts explained the concept of an excitation threshold that must be passed for a neuron to generate an impulse. Activation functions simulate this phenomenon, to some extent. Neurons in a brain act like binary operators; once a neuron has received enough of an electrical signal, it sends a signal to the other neurons it is attached to. Activation functions work similarly. A large enough value will cause an AF to start producing a non-zero output, and in some cases, that output will saturate at or asymptotically approach a maximum value. Different AFs can be leveraged for applicable roles.

The three main roles for AFs, within this work, are separated by whether the AF is in a input layer, hidden layer, or output layer. Output layers were previously discussed but have been proven to be important for derivative scaling; hence, the sigmoid or hyperbolic tangent AFs are generally used because the NN outputs are constrained to the regions $(0, 1)$ or $(-1, 1)$ [43]. Similar to the output layer's purpose of scaling the outputs of the NN,

the input layer scales the inputs to a range of $(0, 1)$. This is done using a simple linear interpolation scheme, give the boundaries of $\mathcal{X}$. The hidden layers are the main "brain" of the NN and perform most of the function approximation. Unlike the output layer AFs, hidden layer AFs are commonly discontinuous and have no restrictions on their output range. However, continuous AFs are required for mathematical proofs, so both will be presented.

Input Layer

The input layer has a simple purpose of scaling the NN inputs to the range $(0, 1)$, which helps normalize the gradients within the neurons. Given the boundaries of the admissible state space $\underline{x}_{max}, \underline{x}_{min} \in \mathcal{X}$, the NN input is scaled using

$$\underline{w}_k^0 = \frac{\underline{x}_k - \underline{x}_{min}}{\underline{x}_{max} - \underline{x}_{min}} \tag{5.32}$$

Input Layer Derivative

The input layer is only affected by the system state, and its derivative is given by:

$$\frac{\partial \underline{w}_k^0}{\partial \underline{x}_k} = \frac{1}{\underline{x}_{max} - \underline{x}_{min}} \tag{5.33}$$

Hidden Layer

The rectified linear unit (ReLU) has seen wide success in the realm of RL. The ReLU is simple to implement and is given by

$$(\underline{w}_i)_{ReLU}^+ = \begin{cases} \underline{w}_i & \text{if } \underline{w}_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.34}$$

Note, implementing ReLU is done with an element-wise maximum: $(\underline{w})_{ReLU}^+ = max(\underline{w}, 0)$. The ReLU AF is not a continuous operator. In practice, the undefined derivative at $\underline{w}_i = 0$ is not a problem as numerical approaches will just define $\frac{\partial (\underline{w}_i)_{ReLU}^+}{\partial \underline{w}_i} = 0$ at $\underline{w}_i = 0$. In terms of mathematical proofs, this discontinuity does have an effect as most proofs regarding

optimization problems require that derivatives of the objective function exist over the entire admissible state space. An AF termed the swish, introduced by Ramachandran et al.,

$$(\underline{w})^{+}_{swish} = \frac{\underline{w}}{1 + e^{-\underline{w}}} \tag{5.35}$$

has similar characteristics to the ReLU but is based on the exponential function, so the swish is smooth. Figure 5.4 compares the ReLU and swish AFs, and Table 5.3 shows a comparison of computation times. The ReLU AF is clearly much faster since one simple function is called as opposed to several mathematical operations.

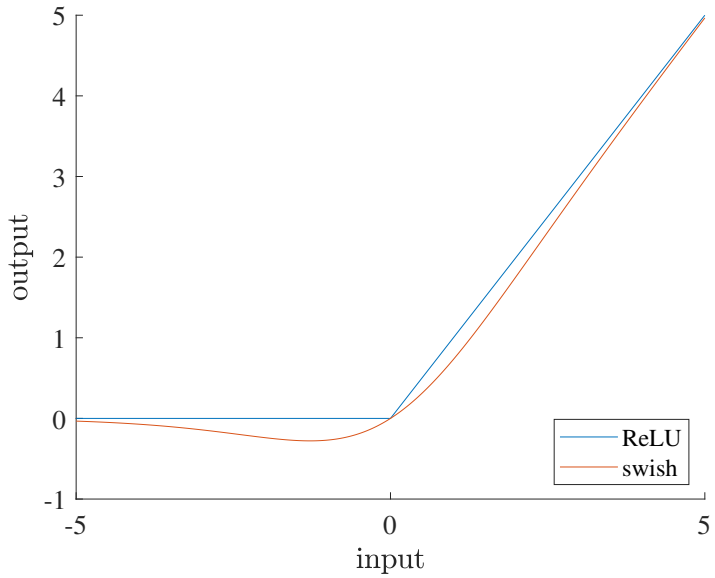

Figure 5.4 Rectified linear unit (ReLU), which is widely used and discontinuous, compared to the swish activation function, which has similar properties and is continuous.

Table 5.3 Comparative average computational time of hidden layer activation functions over 10^6 iterations where the input at each iteration is taken to be $\underline{w} \in \mathbb{R}^{1000\times 4} \sim \mathcal{N}(0,1)$.

	ReLU	swish
time ($10^{-5}s$)	3.305	7.351
difference (%)	-	122

Hidden Layer Activation Function Derivatives

The hidden layer AF derivatives are given by

$$\frac{\partial(\underline{w}_i)^+_{ReLU}}{\partial \underline{w}_i} = \begin{cases} 1 & \text{if } \underline{w}_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{5.36}$$

and

$$\frac{\partial(\underline{w}_i)^+_{swish}}{\partial \underline{w}_i} = \frac{1 + (1 + \underline{w}_i)e^{\underline{w}_i}}{(1 + e^{-\underline{w}_i})^2} \tag{5.37}$$

Figure 5.5 shows a comparison of the ReLU and swish derivatives. Similar to the dropout issue within the output layer, the ReLU AF is susceptible to effectively turning a neuron off. If the weights in the NN are such that any input causes the signal to a single neuron to be zero or less, then that neuron will never produce a signal or be trained to do so. A previously mentioned AF, known as leaky ReLU can be used, where the neuron returns some small value instead of zero and, therefore, never completely drops out; however, in most cases this is not a sigificant issue. Table 5.4 shows a comparison of the computational time for the

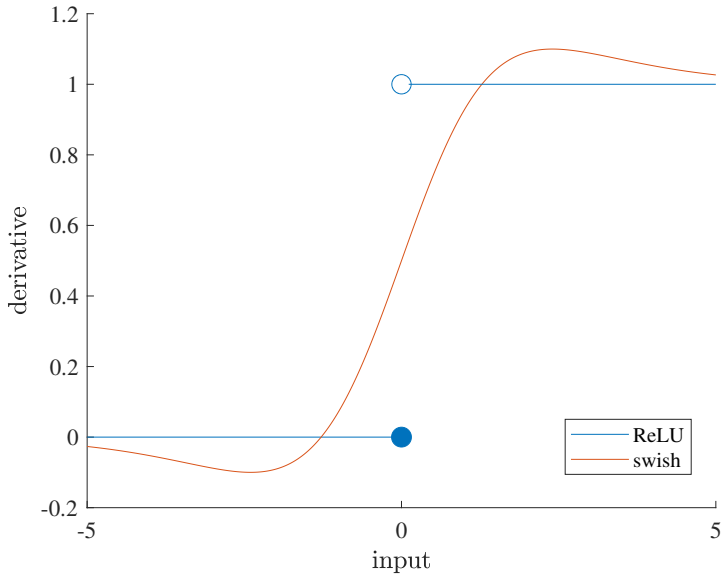

Figure 5.5 Derivative of the rectified linear unit (ReLU) and swish activation functions.

derivatives of the ReLU and swish AFs. Again, the ReLU AF is much faster to compute.

Table 5.4 Comparative average computational time of hidden layer activation function derivatives over 10^6 iterations where the input at each iteration is taken to be $\underline{w} \in \mathbb{R}^{1000\times 4} \sim \mathcal{N}(0,1)$.

	ReLU	swish
time $(10^{-5}s)$	8.113	14.82
difference (%)	-	76.7

5.5.2 Single-Layer Perceptron

The single-layer perceptron (SLP) is the simplest version of a MLP network and is used in this work to control systems as it not only has universal approximation capability but also is simple to implement in comparison to a multi-layer network. A SLP policy could take in the current time step, however a stationary version is considered here, so the policy simply maps states to control inputs

$$\mu_{SLP} : \mathcal{X} \times \Theta \mapsto \mathcal{U} \tag{5.38}$$

and is given by

$$\underline{u}_k = \left(A_2 \left(A_1 \left(\underline{x}_k\right)_{IL}^{+} + b_1\right)_{HL}^{+} + b_2\right)_{OL}^{+} \tag{5.39}$$

where $A_2 := \text{vec}_{m\times h}^{-1}(\theta_{[h(n+1)+2,h(m+n+1)+2]})$, $A_1 := \text{vec}_{m\times h}^{-1}(\theta_{[1,nh]})$, $b_1 := \theta_{[nh+1,h(n+1)+1]}$, and $b_2 := \theta_{[h(m+n+1)+3,h(m+n+1)+3+m]}$ are weights and biases formed from the policy parameter. The hidden node size $h \in \mathbb{N}$ is the width of the SLP. To forgo the large subscripts, define $\theta_{A_1} \in \mathbb{R}^{hn}$, $\theta_{b_1} \in \mathbb{R}^{h}$, $\theta_{A_2} \in \mathbb{R}^{mh}$, and $\theta_{b_2} \in \mathbb{R}^{m}$ as

$$\theta = \begin{bmatrix} \text{vec}(A_1) \\ b_1 \\ \text{vec}(A_2) \\ b_2 \end{bmatrix} := \begin{bmatrix} \theta_{A_1} \\ \theta_{b_1} \\ \theta_{A_2} \\ \theta_{b_2} \end{bmatrix} \tag{5.40}$$

Single-Layer Perceptron Derivative

The SLP policy derivatives are the most involved to compute as there are three separate AFs and two matrix multiplications to consider. First, define the intermediate values $\underline{w}_k^0 \in$

$\mathbb{R}^n_{(0,1)}$, $\underline{w}^1_k \in \mathbb{R}^h$, and $\underline{w}^2_k \in \mathbb{R}^m$ as

$$\underline{w}^0_k := (\underline{x}_k)^+_{IL} \tag{5.41}$$

$$\underline{w}^1_k := A_1 \underline{w}^0_k + b_1 \tag{5.42}$$

$$\underline{w}^2_k := A_2 \left(\underline{w}^1_k\right)^+_{HL} + b_2 \tag{5.43}$$

$$\underline{u}_k = \left(\underline{w}^2_k\right)^+_{OL} \tag{5.44}$$

This definition breaks the convention given in Eq. (5.31), but is useful for the derivative presented in this section. The trajectories W^0, W^1, and W^2 are taken to be time sequences of their respective hidden layer values, i.e.

$$W^0 = \begin{bmatrix} \underline{w}^0_0 \\ \vdots \\ \underline{w}^0_{N_h-1} \end{bmatrix} \quad W^1 = \begin{bmatrix} \underline{w}^1_0 \\ \vdots \\ \underline{w}^1_{N_h-1} \end{bmatrix} \quad W^2 = \begin{bmatrix} \underline{w}^2_0 \\ \vdots \\ \underline{w}^2_{N_h-1} \end{bmatrix} \tag{5.45}$$

First, the derivative with respect to the normalized inputs can be taken as

$$\frac{\partial W^2}{\partial W^0} = \begin{bmatrix} A_2 \frac{\partial (\underline{w}^1_0)^+_{HL}}{\partial \underline{w}^1_0} A_1 & \cdots & 0^{m\times n} \\ \vdots & \ddots & \vdots \\ 0^{m\times n} & \cdots & A_2 \frac{\partial (\underline{w}^1_{N_h-1})^+_{HL}}{\partial \underline{w}^1_{N_h-1}} A_1 \end{bmatrix} \tag{5.46}$$

and combining with the input and output layer derivatives with the chain rule, produces the control derivative

$$\frac{\partial U}{\partial X} = \frac{\partial U}{\partial W^2} \frac{\partial W^2}{\partial W^0} \frac{\partial W^0}{\partial X} \tag{5.47}$$

The derivative with respect to the policy parameter is more involved as their are sever components in the NN to handle. The derivative of the hidden layer output can be written

as the block matrix

$$\frac{\partial W^2}{\partial \theta} = \begin{bmatrix} \frac{\partial \underline{w}_0^2}{\partial \theta} & \cdots & 0^{m\times p} \\ \vdots & \ddots & \vdots \\ 0^{m\times p} & \cdots & \frac{\partial \underline{w}_{N_h-1}^2}{\partial \theta} \end{bmatrix} \tag{5.48}$$

where each block $\frac{w_k^2}{\partial \theta} \in \mathbb{R}^{m\times p}$. Now, the derivative of a single time step will be presented, which can be used to fill these blocks. The reshaping function $\text{vec}(\cdot)$ and its inverse $\text{vec}^{-1}(\cdot)$ are both linear operators; the output information is not altered in any way. In other words, some differential change in the policy parameter can simply be reshaped into a differential change in the respective component, for example

$$dA_1 = \text{vec}_{h\times n}^{-1}(d\theta_{A_1}) \tag{5.49}$$

Taking the derivative of vectors with respect to matrices is not straightforward, so vectorized versions of the NN weighting matrices will be considered. This also satisfies the dimensions required for updating θ. So, the goal is to find:

$$\frac{\partial \underline{w}_k^2}{\partial \theta} = \begin{bmatrix} \frac{\partial \underline{w}_k^2}{\partial \theta_{A_1}} & \frac{\partial \underline{w}_k^2}{\partial \theta_{b_1}} & \frac{\partial \underline{w}_k^2}{\partial \theta_{A_2}} & \frac{\partial \underline{w}_k^2}{\partial \theta_{A_2}} \end{bmatrix} \tag{5.50}$$

which can be scaled using the output layer derivatives to find $\frac{\partial U}{\partial \theta}$. The derivative of the value at the hidden layer is simple as it is a matrix multiplication:

$$\frac{\partial \underline{w}_k^1}{\partial \theta_{A_1}} = \begin{bmatrix} (\underline{w}_k^0)^T & \cdots & 0^{1\times h} \\ \vdots & \ddots & \vdots \\ 0^{1\times h} & \cdots & (\underline{w}_k^0)^T \end{bmatrix} \tag{5.51}$$

To get the hidden layer output, the previous value must be scaled by the AF derivative, which is done using an element-wise product

$$\frac{\partial \underline{w}_k^2}{\partial \theta_{A_1}} = A_2 \left(\frac{\partial (\underline{w}_k^1)_{HL}^+}{\partial \underline{w}_k^1} \cdot \frac{\partial \underline{w}_k^1}{\partial \theta_{A_1}} \right) \tag{5.52}$$

where the matrix AF derivative is formed as

$$\frac{\partial (\underline{w}_k^1)_{HL}^+}{\partial \underline{w}_k^1} := \begin{bmatrix} \frac{\partial (\underline{w}_{0,k}^1)_{HL}^+}{\partial \underline{w}_{0,k}^1} & \cdots & \frac{\partial (\underline{w}_{0,k}^1)_{HL}^+}{\partial \underline{w}_{0,k}^1} \\ \vdots & & \vdots \\ \frac{\partial (\underline{w}_{h,k}^1)_{HL}^+}{\partial \underline{w}_{h,k}^1} & \cdots & \frac{\partial (\underline{w}_{h,k}^1)_{HL}^+}{\partial \underline{w}_{h,k}^1} \end{bmatrix} \tag{5.53}$$

Similarly, the derivative with respect to the first bias can be handled in two steps. The bias a simple constant so the derivative at the hidden layer is the identity

$$\frac{\partial \underline{w}_k^1}{\partial \theta_{b_1}} = I^{h \times h} \tag{5.54}$$

Then, the hidden layer output can be found using

$$\frac{\partial \underline{w}_k^2}{\partial \theta_{b_1}} = A_2 \left(\frac{\partial (\underline{w}_k^1)^+}{\partial \underline{w}_k^1} \cdot \frac{\partial \underline{w}_k^1}{\partial \theta_{b_1}} \right) \tag{5.55}$$

where the AF derivative scaling can be simplified to:

$$\left(\frac{\partial (\underline{w}_k^1)_{HL}^+}{\partial \underline{w}_k^1} \cdot \frac{\partial \underline{w}_k^1}{\partial \theta_{b_1}} \right) = \begin{bmatrix} \frac{\partial (\underline{w}_{0,k}^1)_{HL}^+}{\partial \underline{w}_{0,k}^1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{\partial (\underline{w}_{h,k}^1)_{HL}^+}{\partial \underline{w}_{h,k}^1} \end{bmatrix} \tag{5.56}$$

The hidden layer output is another matrix product with the second matrix weight, so the derivative is taken as

$$\frac{\partial \underline{w}_k^2}{\partial \theta_{A_2}} = \begin{bmatrix} \left((\underline{w}_k^1)_{HL}^+\right)^T & \cdots & 0^{1\times h} \\ \vdots & \ddots & \vdots \\ 0^{1\times h} & \cdots & \left((\underline{w}_k^1)_{HL}^+\right)^T \end{bmatrix} \tag{5.57}$$

Lastly, the derivative with respect to the second bias is the identity

$$\frac{\partial \underline{w}_k^2}{\partial \theta_{b_2}} = I^{m\times m} \tag{5.58}$$

All of these separate components can be compiled at every time step into Eq. (5.50) and subsequently, Eq. (5.48).

5.5.3 Convolutional Neural Network

Convolution neural networks have been shown to be extremely powerful at pattern recognition. A CNN is comprised of one or more convolutional layers that feed into one or more MLP layers. These convolutional layers work by scanning a kernel across the incoming data, which can have any number of dimensions, and convolving the kernel with the data. This process allows a CNN to recognize translationally invariant patterns in space as is commonly found in image recognition. Similarly, as previously discussed, CNNs work well at playing board games, since they essentially share the same pattern recognition problem. Another use, more recently explored, is the ability for CNNs to estimate the states of system given some time history of data. Wilson and Prazenica used a 1D CNN to predict the inflow states of a rotor system based on measurements of the rotor blade states. Similarly, Kang et al. used a 2D CNN to approximate rotorcraft dynamics. Approximating system states is, again, essentially a pattern recognition problem.

The dimension of the CNN refers to the dimension of the the incoming data. In image recognition, 3D CNNs may be used where two of those dimensions represent the actual image with the third dimension containing color data such as RGB values. In 2D image recognition,

the images are set to a grayscale, where each pixel takes on a value between 0 and 255, and those pixels make up the image. In terms of dynamics estimation and approximation, 2D or 1D CNNs are generally used. 1D CNNs are considered in this work, as they ease the computational burden and fit the dynamics estimation problem better. Kang et al. used a 2D kernel on a time history of data that convolves not only the same state over time but also other states over that time interval. Depending on the system, this may or may not be beneficial because the arbitrary ordering of the states in the state vector then plays a role in the training. With a 1D CNN, each state is handled individually, which removes any arbitrariness. A similar effect could be found by using a 2D kernel that is the same width as the state vector dimension. In this case, all of the states would be convolved together over time and the state vector order would not matter.

The CNN based policy maps a time-history of states to a control input:

$$\mu_{CNN} : \mathcal{X}^{N_d} \times \theta \mapsto \mathcal{U} \tag{5.59}$$

where $\mathbb{N}_d \in \mathbb{N}_{>1}$ is the number of data input to the CNN. The time-history input, also known as a feature map, is assembled as

$$X^{CNN} = \begin{bmatrix} \underline{x}_{k-N_d+1}{}^T \\ \vdots \\ \underline{x}_k{}^T \end{bmatrix} \tag{5.60}$$

and the input layer scaling given in Eq. (5.32) is applied to time step individually. The forward pass through the convolutional layers of a 1D CNN is done using

$$\underline{w}^j_{k,i} = b^j_{k,i} + \sum_{l=1}^{h_{j-1}} conv(\psi^{j-1}_{li}, (\underline{w}^{l-1}_{k,i})^+) \tag{5.61}$$

where i is the neuron at the current layer, j is the layer, b is the bias, h_{j-1} is the number

of neurons in the previous layer, and ψ is the filter acting on the lth neuron of the previous layer going to the ith neuron of the current layer [46]. The AFs used in MLP networks are used here as well, along with an additional step. Prior to passing the current neuron information $\underline{w}_k^j$ to the next layer another operation, termed pooling, is performed where a separate kernel is scanned along the data, and instead of performing convolutions, it is common for this kernel to either select the maximum value or average all of the values. Once the data are propagated through all of the convolutional and pooling layers, they are passed to MLP layers. Figure 5.6 shows an example CNN equipped with ReLU AFs and a softmax output layer, which is used for classification problems.

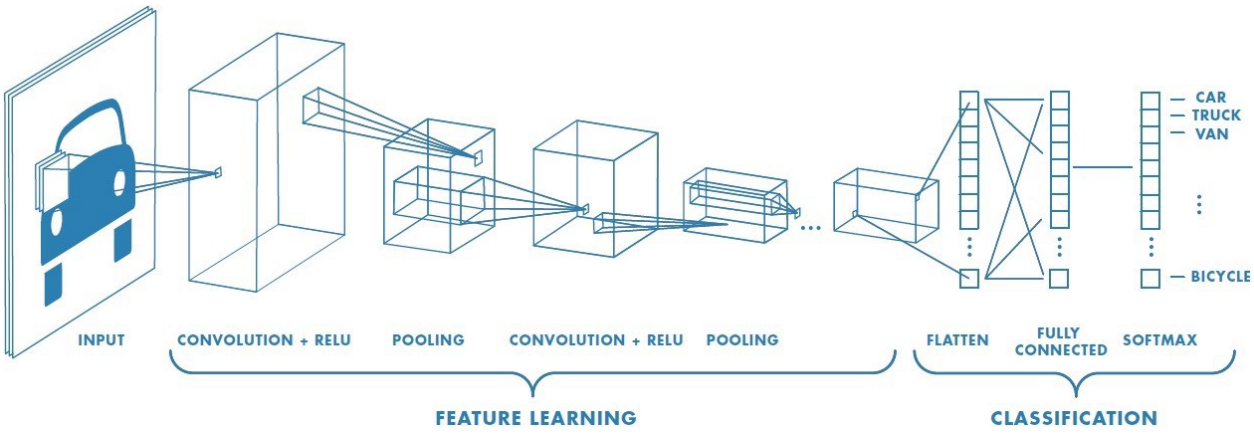

Figure 5.6 Example convolutional neural network used for image recognition [2]

6 Regulation Stability

Closed-loop stability of MPC systems has been thoroughly studied with regards to conventional OCS problems and SFCLs. In [47] and [26], the basic assumptions for exploring the regulation stability of MPC systems are explained:

Assumption 8. *$f(0^n, 0^m) = 0^n$ and under zero control, given $\underline{x}_0 \neq 0$, the state either becomes unbounded or converges to zero (i.e. there are no limit cycles)*

Assumption 9. *$\exists\, U, \underline{u}_k \in \mathcal{U}\ \forall\ k \in \mathbb{N}_{[0,N_h-1]}$ such that $\underline{x}_{N_h} = 0^n$*

Assumption 10. *$V^{\mu^*}(\underline{x}_k) = 0 \iff \underline{x}_k = 0^n\ \forall\ k \in \mathbb{N}_{[0,N_h]}$*

Assumption 11. *$V^{\mu^*}(\underline{x}_k) \Rightarrow \infty$ when $||\underline{x}_k|| \Rightarrow \infty\ \forall\ k \in \mathbb{N}_{[0,N_h]}$*

Assumption 12. *$\exists\ \frac{\partial V^{\mu^*}(\underline{x}_k)}{\partial \underline{x}_k}\ \forall\ \underline{x}_k \in \mathcal{X}, k \in \mathbb{N}_{[0,N_h]}$*

These assumptions concern the system behavior, existence of a solution, and cost function design assumptions given in Section 3.2.1. First, if the system reaches the origin, it must remain there without any control effort. A constraint placed on the optimization $\underline{x}_{N_h} = 0^n$ is commonly found as this complements the assumption that the origin is an equilibrium point. Work has been done to show how this stabilizing constraint can be relaxed, however. While the stabilizing terminal constraint is not necessary, the existence of a control trajectory that would satisfy that constraint is required. Essentially, Assumption 9 regards the system's controllability. If the system were linear, controllability could be easily determined using the controllability Gramian; however, controllability is much harder to analyze for generic nonlinear systems. Building on the cost function assumptions given in Section 3.2.1, the value function must be able to properly detect the system, be radially unbounded, and have proper positive (semi)definiteness properties. Additionally, the value function derivative must exist over the entire admissible state space, which not only means the cost function but also the dynamics function must have a derivative. A value function that has these

properties can be used as a Lyapunov function and subsequently be used to show that the control inputs make the system dissipative.

Consider the OCS case where $\mu : \Theta \times \mathbb{N}_{[0,N_h-1]} \mapsto \mathcal{U}$. Along the optimal trajectory X^* of a finite horizon optimal control problem produced by the optimal OCS policy μ^*, the value function should be decreasing. Let $V_{N_h}^{\mu^*}(\underline{x}_0)$ be the optimal value for an N_h time step problem starting at $\underline{x}_0$. Then

$$V_{N_h}^{\mu^*}(\underline{x}_0) - V_{N_h-1}^{\mu^*}(\underline{x}_1) = L(\underline{x}_0, \underline{u}_0^*) \tag{6.1}$$

Now, subtract $V_{N_h}^{\mu^*}(\underline{x}_1)$ from both sides and rearrange to obtain

$$V_{N_h}^{\mu^*}(\underline{x}_0) - V_{N_h}^{\mu^*}(\underline{x}_1) = L(\underline{x}_0, \underline{u}_0^*) + V_{N_h-1}^{\mu^*}(\underline{x}_1) - V_{N_h}^{\mu^*}(\underline{x}_1) \tag{6.2}$$

If the right hand side of this equation is nonnegative, then the optimal value function is nonincreasing along the optimal trajectory. Since L is a positive semidefinite function, if

$$0 \leq V_{N_h-1}^{\mu^*}(\underline{x}_1) - V_{N_h}^{\mu^*}(\underline{x}_1) \tag{6.3}$$

is true, then the right hand side of Eq. (6.2) is nonnegative. In [47], the following lemma and proof are explained:

Lemma 1. *If* $0 < N_h < \hat{N}_h$, $\underline{x}_{N_h} = 0^n$, *and* $\underline{x}_{\hat{N}_h} = 0^n$, *then* $V_{N_h}^{\mu^*}(\underline{x}_0) \geq V_{\hat{N}_h}^{\hat{\mu}^*}(\underline{x}_0)\ \forall\ \underline{x}_0$

Proof. The OCS μ^* on $k \in \mathbb{N}_{[0,N_h)}$ gives the value $V_{N_h}^{\mu^*}(\underline{x}_0)$ and terminal condition $\underline{x}_{N_h} = 0^n$. A new control policy

$$\hat{\mu}_k^* = \begin{cases} \mu_k^* & \text{for } k \in \mathbb{N}_{[0,N_h)} \\ 0^m & \text{for } k \in \mathbb{N}_{[N_h,\hat{N}_h)} \end{cases} \tag{6.4}$$

gives the value $V_{N_h}^{\hat{\mu}^*}(\underline{x}_0)$, which is equal to $V_{N_h}^{\mu^*}(\underline{x}_0)$, and by definition, shows $V_{N_h}^{\mu^*}(\underline{x}_0) \geq V_{\hat{N}_h}^{\hat{\mu}^*}(\underline{x}_0)$. □

The proof applies to finite horizon optimal control problems (one stage of a receding horizon control problem). Additional work is required to show that a receding horizon OCS policy results in a dissipative system. In [47] and [48] a continuous time derivation of MPC regulation stability is given, while [49] and [50] cover discrete-time cases. Porting work from [47] to a discrete-time system gives Theorem 1.

Theorem 1. *Consider a system with Assumptions 8-12 holding and the constraint* $\underline{x}_{N_h} = 0^n$. *Then, the OCS MPC policy asymptotically stabilizes the system around the origin.*

Proof. Using Bellman's equation, two steps of the MPC problem can be written as

$$V^{\mu^*}_{N_h}(\underline{x}_0) = L(\underline{x}_0, \underline{u}^*_0) + V^{\mu^*}_{N_h-1}(\underline{x}_1) \tag{6.5}$$

$$V^{\mu^*}_{N_h}(\underline{x}_1) = L(\underline{x}_1, \underline{u}^*_1) + V^{\mu^*}_{N_h-1}(\underline{x}_2) \tag{6.6}$$

where two optimal control problems are being solved over $N_h \in \mathbb{N}$ length horizons with the second starting one time step after the first. If $V^{\mu^*}_{N_h}(\underline{x}_0) \geq V^{\mu^*}_{N_h}(\underline{x}_1)$, then the system is asymptotically stable. Subtracting the two equations yields

$$V^{\mu^*}_{N_h}(\underline{x}_0) - V^{\mu^*}_{N_h}(\underline{x}_1) \geq L(\underline{x}_0, \underline{u}^*_0) + V^{\mu^*}_{N_h-1}(\underline{x}_1) - L(\underline{x}_1, \underline{u}^*_1) - V^{\mu^*}_{N_h-1}(\underline{x}_2) \tag{6.7}$$

Expanding $V^{\mu^*}_{N_h-1}(\underline{x}_1)$ using Bellman's equation again and simplifying gives

$$V^{\mu^*}_{N_h}(\underline{x}_0) - V^{\mu^*}_{N_h}(\underline{x}_1) \geq L(\underline{x}_0, \underline{u}^*_0) + V^{\mu^*}_{N_h-2}(\underline{x}_2) - V^{\mu^*}_{N_h-1}(\underline{x}_2) \tag{6.8}$$

If the right hand side of this equation is positive semidefinite, then the optimal value over the $N_h \in \mathbb{N}$ horizon is nonincreasing. Using Lemma 1,

$$V^{\mu^*}_{N_h-2}(\underline{x}_2) \geq V^{\mu^*}_{N_h-1}(\underline{x}_2) \tag{6.9}$$

which proves the right hand side is positive semidefinite. Then,

$$V_{N_h}^{\mu^*}(\underline{x}_0) \geq V_{N_h}^{\mu^*}(\underline{x}_1) \tag{6.10}$$

and the regulating OCS MPC policy asymptotically drives the system to the origin. □

In [51], it is shown how the terminal constraint $\underline{x}_{N_h} = 0^n$ can be relaxed. Finding the exact OCS that satisfies $\underline{x}_{N_h} = 0^n$ can be quite difficult, so another piecewise control policy is introduced, which assumes the existence of some region around the origin $\mathcal{W} \subset \mathcal{X}$ where the system can be stabilized using a linear feedback control law. The dual-mode control policy is given by

$$\mu_k^*(\underline{x}_k) = \begin{cases} \underline{u}_k^* & \text{when } \underline{x}_k \in \mathcal{W}^c \\ -K\underline{x}_k & \text{when } \underline{x}_k \in \mathcal{W} \end{cases} \tag{6.11}$$

where $\underline{u}_k^* \in \mathcal{U}$ comes from the optimal OCS, the superscript "c" denotes the complement, and $K \in \mathbb{R}^{m \times n}$ is a stabilizing gain. In [26], this idea is leveraged to show stability of policies used to approximate the optimal OCS. A stabilizable region $\mathcal{W}_p$ is defined as

$$\mathcal{W}_p(\bar{N}_h, a, P) := \{\underline{x}_k \in \mathcal{X} : \bar{V}^{\mu}(\underline{x}_i) \leq \beta\} \tag{6.12}$$

where $a \in \mathbb{R}_{>0}$ is a scalar weight, $P \in \mathbb{R}_{>0}^{n \times n}$ is a state weighting matrix, $\bar{N}_h > N_h$ is some number of time steps greater than that required to satisfy $\underline{x}_{N_h} = 0^n$, $\bar{V}^{\mu}(\underline{x}_i, \theta) = \bar{J}(X^{\mu}, U^{\mu}) = \sum_{k=i}^{i+\bar{N}_h-1} \left(L(\underline{x}_k, \underline{u}_k) + a\underline{x}_k{}^T P \underline{x}_k\right)$ is the value function used to provide a bound, and $\beta \in \mathbb{R}_{>0}$ is the bound. The ellipsoid $\mathcal{W}_p$ is the region where a chosen control policy can stabilize the system, as compared to $\mathcal{W}$, which is specific to a linear feedback law. From [26], Theorem 2 explains stability of NN optimal control approximations, though it is not necessarily relegated to only NN based policies.

Theorem 2. *If Assumptions 1-7 are satisfied and the control policy is continuous with respect to the system state, there exists a scalar $\bar{a} \in \mathbb{R}_{>0}$ and matrix $P \in \mathbb{R}_{>0}^{n \times n}$ such that, for any*

$\bar{N}_h \geq N_h$ *and for any* $a \geq \bar{a}$*, the following properties hold:*

1. *There exist suitable scalars* $\delta_k \in \mathbb{R}_{>0}$ *such that, if* $||\underline{u}_k^* - \mu(\underline{x}_k, \underline{\theta}, k)|| \leq \delta_k$*,* $k \in \mathbb{N}_{[i,i+\bar{N}_h]}$*, then* $\underline{x}_k \in \mathcal{W}_p(\bar{N}_h, a, P)\ \forall\ k \in \mathbb{N}_{[i,i+\bar{N}_h]}, \underline{x}_i \in \mathcal{W}_p(\bar{N}_h, a, P)$

2. *For any compact set* $\mathcal{W}_d \subset \mathcal{X}$*, there exists a finite integer* $T \geq i$ *and suitable scalars* $\delta \in \mathbb{R}_{>0}$ *such that, if* $||\underline{u}_k^* - \mu(\underline{x}_k, \underline{\theta}, k)|| \leq \delta_k$*,* $k \in \mathbb{N}_{[i,i+\bar{N}_h]}$*, then* $\underline{x}_k \in \mathcal{W}_d\ \forall\ k \in \mathbb{N}_{>T}, \underline{x}_i \in \mathcal{W}_p(\bar{N}_h, a, P)$

The proof for Theorem 2 is given in [24], and values for the suitable scalars δ_k are determined in [26]. This theorem essentially states that, in finite time, an approximation of the optimal OCS given by a control policy will drive the system to a stabilizable region of the origin where a policy such as Eq. (6.11) can be used. If δ_k can be determined, then a control policy can be selected accordingly. In [26], the width of a single layer NN is varied to achieve suitable accuracy. A theorem and proof of similar effect is given in [28]. In [52], a stability result is provided for discrete OCS policies acting on continuous systems with bounded disturbances and measurement error that can be applicable to state feedback policies.

These proofs work around the case where $N_o = 1$. A single control input $\underline{u}_k$ is implemented, propagating the system from $\underline{x}_k$ to $\underline{x}_{k+1}$. The trajectory between $\underline{x}_k$ and $\underline{x}_{k+1}$ is not considered, just the two end points. Similarly, the actual shape of $\underline{u}_k$ is not important, just that it moves the system from $\underline{x}_k$ to $\underline{x}_{k+1}$. The MPC approach used here involves cases where $N_o \geq 1$, which means the system may propagate at a higher frequency than the controller's operation. If the state at every N_o time steps is taken to be $\underline{x}_k$ and that subset of the higher frequency state trajectory has a nonincreasing cost, then the low frequency MPC system will be stable according to Theorem 1 or 2. Assuming the control $\underline{u}_k$ is the optimal OCS along the trajectory from $\underline{x}_k$ to $\underline{x}_{k+1}$, the cost between k and $k+1$ will also be nonincreasing via the Bellman equation and assumptions on $L(\underline{x}_k, \underline{u}_k)$, which means the high frequency system will also be stable. If the control $\underline{u}_k$ is not optimal but has a bounded approximation error,

then Theorem 2 and work in [28] prove that the system will remain within the stabilizable region $\mathcal{W}$.

6.1 Linear Feedback Control

The control policy $\underline{u}_k = K\underline{x}_k$, where $K = vec^{-1}_{m\times n}(\underline{\theta})$ is a constant feedback gain, is considered here. This control policy is attractive for its simplicity and robustness but it lacks in approximation power. Clearly, a linear feedback control law will only be able to stabilize the system if $\underline{x}_0 \in \mathcal{W}$. Let $\Theta_{\mathcal{W}} \subseteq \Theta$ be the set of policy parameters that produce stabilizing feedback gains. Clearly, if a linear feedback policy is used, $\underline{x}_0 \in \mathcal{W}$ and $\theta \in \Theta_{\mathcal{W}}$; then, $\underline{x}_k \in \mathcal{W}\ \forall\ k \in \mathbb{N}_{[0,\infty)}$.

Assumption 13. *Given a constant linear feedback policy, there exists a unique solution to Problem 1:* $\underline{\theta}^* = argmin_{\underline{\theta}}\, V^{\mu}(\underline{x}_0)$

Empirically, it can be shown that optimal regulators, stationary horizon or receding horizon, produce stable solutions when the optimization horizon is sufficiently large. If the horizon were infinite, then the optimal solution would need to stabilize the system as the cost would be infinite otherwise. In very short horizon problems, destabilizing, or at least non-stabilizing, solutions can be produced as the control input component of the cost function can outweigh the state trajectory or the unstable solution may cause the state trajectory to quickly approach the origin, which can be optimal for regulation problems given the states do not reach or overshoot the origin within the short horizon.

For example, assume a linear system will be controlled via receding horizon optimizations of a quadratic cost with an OCS policy; then, the optimal solution at every N_o steps is given by the finite horizon linear quadratic regulator (FHLQR). Given a long enough horizon the FHLQR behaves essentially the same as the infinite horizon linear quadratic regulator (IHLQR), because the ARE dynamics occur at the end of the horizon where the states are likely already close to zero. Under very short horizons, however, the ARE does not reach a constant solution, and the gains associated with the dynamic portion of the ARE solution

are not necessarily stabilizing. So, suppose $N_o = 1$ and N_h is short enough such that K_0 does not stabilize the system. Since a single step approach is used, K_0 will be implemented repeatedly, which results in CG feedback, and the closed-loop system will not be stable. If N_h were long enough such that the ARE reached a constant solution, then $K_0 \approx K_\infty$, where K_∞ is the IHLQR gain, and the closed-loop system will almost exactly replicate the stable IHLQR solution.

In Lemma 2, it is shown that the linear feedback policy stabilizes nonlinear systems as the optimization horizon approaches infinity, given that the initial condition is within the stabilizable region $\mathcal{W}$.

Lemma 2. *Given $\underline{x}_0 \in \mathcal{W}$ and a constant linear feedback policy, there exists $\underline{\theta}^*_\infty \in \Theta_\mathcal{W}$ such that, as $N_h \to \infty$ the solution to Problem 1 $\underline{\theta}^*$ approaches $\underline{\theta}^*_\infty$ and produces an asymptotically stable trajectory around the origin.*

Proof. Since $\underline{x}_0 \in \mathcal{W}$, there exists some gain $\bar{K} = \text{vec}^{-1}_{m\times n}(\bar{\underline{\theta}}), \bar{\underline{\theta}} \in \Theta_\mathcal{W}$ that produces an asymptotically stable trajectory. Given $||\underline{x}_k|| \to 0$ as $k \to \infty$, then $||\underline{u}_k|| \to 0$ and according to Assumptions 3 and 4, $L(\underline{x}_k, \underline{u}_k) \to 0$, which shows $V^\mu_\infty(\underline{x}_0, \bar{\underline{\theta}}) < \infty$. There is no requirement that $\bar{K}$ be the solution to Problem 1, so given Assumption 13, $V^\mu_\infty(\underline{x}_0, \underline{\theta}^*) \leq V^\mu_\infty(\underline{x}_0, \bar{\underline{\theta}})$, which shows the optimal solution to the infinite horizon problem is stabilizing. Now, consider the value function derivative given in Eq. (4.25). Given the assumptions on $L(\underline{x}_k, \underline{u}_k)$ and the convergence of both $\underline{x}_k$ and $\underline{u}_k$, $\frac{\partial V^\mu(\underline{x}_0)}{\partial \underline{x}_k} \to 0^n$ and $\frac{\partial V^\mu(\underline{x}_0)}{\partial \underline{u}_k} \to 0^m$ as $k \to \infty$, which means $\frac{\partial V^\mu(\underline{x}_0)}{\partial X}$ and $\frac{\partial V^\mu(\underline{x}_0)}{\partial U}$ become constant as $N_h \to \infty$. The control policy gives $\frac{\partial U}{\partial X} = blkdiag\left(\text{vec}^{-1}_{m\times n}(\underline{\theta})\right)$ and $\frac{\partial U}{\partial \underline{\theta}} = X_{[0,N_h-1]}$, which are both constant as $N_h \to \infty$. Lastly, the dynamics derivative can be written as:

$$\frac{\partial \underline{x}_{k+1}}{\partial \underline{\theta}} = \left(\frac{\partial f(\underline{x}_k, \underline{u}_k)}{\partial \underline{x}_k} + \frac{\partial f(\underline{x}_k, \underline{u}_k)}{\partial \underline{u}_k}\text{vec}^{-1}_{m\times n}(\underline{\theta})\right)\frac{\partial \underline{x}_k}{\partial \underline{\theta}} + \frac{\partial f(\underline{x}_k, \underline{u}_k)}{\partial \underline{u}_k}{\underline{x}_k}^T \tag{6.13}$$

Using Assumption 12, $\left|\left|\frac{\partial f(\underline{x}_k, \mu(\underline{x}_k))}{\partial \underline{x}_k}\right|\right|, \left|\left|\frac{\partial f(\underline{x}_k, \mu(\underline{x}_k))}{\partial \underline{u}_k}\right|\right| < \infty \; \forall \; \underline{x}_k \in \mathcal{X}$, which shows $\left|\left|\frac{\partial \underline{x}_{k+1}}{\partial \underline{\theta}}\right|\right| < \infty$ as $N_h \to \infty$. Since $\frac{\partial \underline{x}_{k+1}}{\partial \underline{\theta}}$ and $\frac{\partial U}{\partial X}$ are bounded and all other components approach zero

as $k \to \infty$, both terms of the value function derivative become constant as $N_h \to \infty$ and $argmin_{\underline{\theta}} \left(V^{\mu}_{N_h}(\underline{x}_0)\right) \to \underline{\theta}^*_{\infty}$. □

Solving infinite horizon problems numerically is intractable, so now it is shown that there exists some finite optimization horizon that produces a stabilizing feedback gain. While the exact length of this horizon is unknown, this enables an empirical search to find a horizon that is suitable. The optimal linear feedback gain, for any horizon length, depends on the initial condition, excluding linear systems over infinite horizons, which suggests that the optimization horizon that produces a stabilizing gain also varies with respect to the initial condition. This is important for receding horizon control where the initial condition for the repeated optimizations is varying throughout time.

Theorem 3. *There exists* $\bar{N}_h \in \mathbb{N}_{(0,\infty)}$ *such that,* $\bar{\underline{\theta}} \in \Theta_{\mathcal{W}}$ *where* $\bar{\underline{\theta}} = argmin_{\underline{\theta}} V^{\mu}_{\bar{N}_h}(\underline{x}_0)$

Proof. Let the optimal control inputs $\underline{u}^*_k$ come from the infinite horizon linear SFCL using $\underline{\theta}^*_{\infty}$. The optimal inputs can be approximated by a finite horizon linear SFCL using $\bar{\underline{\theta}} = argmin_{\underline{\theta}} V^{\mu}_{\bar{N}_h}(\underline{x}_0)$. In Lemma 2, it is shown that the finite horizon solution converges to the stable infinite horizon solution as $\bar{N}_h \to \infty$. So, there exists some finite horizon $\bar{N}_h \in \mathbb{N}_{(0,\infty)}$ such that, $||\underline{u}^*_k - \mu(\underline{x}_k, \bar{\underline{\theta}})|| < \delta_k, \delta_k > 0 \; \forall \; k \in \mathbb{N}_{[0,\bar{N}_h-1]}$ and using Theorem 2, $\underline{x}_k \in \mathcal{W}_p \; \forall \; k \in \mathbb{N}_{[0,\bar{N}_h-1]}$. Given the control policy is a linear SFCL, $\mathcal{W}_p = \mathcal{W}$ and $\bar{\underline{\theta}} \in \Theta_{\mathcal{W}}$. □

7 Linear Quadratic Regulator

Consider the discrete time LQR problem where the system's states propagate with the linear dynamics

$$\underline{x}_{k+1} = A\underline{x}_k + B\underline{u}_k \tag{7.1}$$

and the optimal control input minimizes the quadratic cost function

$$J(X,U) = \frac{1}{2}\underline{x}_{N_h}{}^T S_{N_h}\underline{x}_{N_h} + \frac{1}{2}\sum_{k=0}^{N_h-1} \underline{x}_k{}^T Q\underline{x}_k + \underline{u}_k{}^T R\underline{u}_k \tag{7.2}$$

where $S_{N_h} \in \mathbb{R}^{n\times n}_{\geq 0}$ is the terminal state weighting matrix, $Q \in \mathbb{R}^{n\times n}_{\geq 0}$ is the running state weighing matrix, and $R \in \mathbb{R}^{m\times m}_{>0}$ is the control weighting matrix. Two methods are considered here: open-loop control and linear feedback control.

7.1 Open-Loop Control

Consider the OCS policy $\mu : \Theta \times \mathbb{N}_{[0,N_h-1]} \mapsto \mathcal{U}$ given by:

$$\underline{u}_k = \theta_k \tag{7.3}$$

where $\Theta = \mathcal{U} = \mathbb{R}^{mN_h}$, i.e. there are no control bound constraints. Now, Eq. (4.25) will be explored using the presented dynamics, policy, and cost function. It is easy to recognize that the states do not directly affect the control inputs and it is given that the parameter is the OCS, which means:

$$\frac{\partial U}{\partial X} = 0^{mN_h\times n(N_h+1)} \tag{7.4}$$

and

$$\frac{\partial U}{\partial \theta} = \begin{bmatrix} I^{m\times m} & \cdots & 0^{m\times m} \\ \vdots & \ddots & \vdots \\ 0^{m\times m} & \cdots & I^{m\times m} \end{bmatrix} \tag{7.5}$$

Using this result and given $\theta = U$, Eq. (4.25) can be rewritten as:

$$\frac{\partial V^{\mu}(\underline{x}_0)}{\partial X}\frac{\partial X}{\partial U} + \frac{\partial V^{\mu}(\underline{x}_0)}{\partial U} = 0^{mN_h} \tag{7.6}$$

and explicitly defined. Taking derivatives of the cost function with respect to its inputs yields:

$$\frac{\partial V^{\mu}(\underline{x}_0)}{\partial \underline{x}_k} = \begin{cases} \underline{x}_k{}^T Q & \text{if } k < N_h \\ \underline{x}_k{}^T S & \text{if } k = N_h \end{cases} \tag{7.7}$$

over $k \in \mathbb{N}_{[0,N_h]}$, and

$$\frac{\partial V^{\mu}(\underline{x}_0)}{\partial \underline{u}_k} = \underline{u}_k{}^T R \ \ \forall \ \ k \in \mathbb{N}_{[0,N_h-1]} \tag{7.8}$$

These can be used to populate the terms $\frac{\partial V^{\mu}(\underline{x}_0)}{\partial X}$ and $\frac{\partial V^{\mu}(\underline{x}_0)}{\partial U}$ as:

$$\frac{\partial V^{\mu}(\underline{x}_0)}{\partial X} = \begin{bmatrix} \underline{x}_0{}^T Q & \underline{x}_1{}^T Q & \cdots & \underline{x}_{N_h-1}{}^T Q & \underline{x}_{N_h}{}^T S \end{bmatrix} \tag{7.9}$$

and

$$\frac{\partial V^{\mu}(\underline{x}_0)}{\partial U} = \begin{bmatrix} \underline{u}_0{}^T R & \underline{u}_1{}^T R & \cdots & \underline{u}_{N_h-2}{}^T R & \underline{u}_{N_h-1}{}^T R \end{bmatrix} \tag{7.10}$$

The term $\frac{\partial X}{\partial U}$ is a block matrix and is explained well by [53]:

$$\frac{\partial \underline{x}_k}{\partial \underline{u}_j} = \begin{cases} 0^{n\times m} & \text{if } k \leq j \\ B & \text{if } k = j+1 \\ A\frac{\partial \underline{x}_{k-1}}{\partial \underline{u}_j} & \text{if } k > j+1 \end{cases} \tag{7.11}$$

over $k \in \mathbb{N}_{[0,N_h]}$ and $j \in \mathbb{N}_{[0,N_h-1]}$, which gives:

$$\frac{\partial X}{\partial U} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 0 \\ B & 0 & 0 & \cdots & 0 & 0 \\ AB & B & 0 & \cdots & 0 & 0 \\ A^2B & AB & B & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ A^{N_h-2}B & A^{N_h-3}B & A^{N_h-4}B & \cdots & B & 0 \\ A^{N_h-1}B & A^{N_h-2}B & A^{N_h-3}B & \cdots & AB & B \end{bmatrix} \tag{7.12}$$

with $0 := 0^{m \times n}$ for brevity.

Evaluating the result of these matrices then gives:

$$\frac{dV^{\mu}(\underline{x}_0)}{d\theta} = \begin{bmatrix} \underline{u}_0{}^T R + \underline{x}_{N_h}{}^T S_{N_h} A^{N_h-1} B + \sum_{i=0}^{N_h-2} \underline{x}_{i+1}{}^T Q A^i B \\ \underline{u}_1{}^T R + \underline{x}_{N_h}{}^T S_{N_h} A^{N_h-2} B + \sum_{i=0}^{N_h-3} \underline{x}_{i+2}{}^T Q A^i B \\ \vdots \\ \underline{u}_{N_h-2}{}^T R + \underline{x}_{N_h}{}^T S_{N_h} AB + \underline{x}_{N_h-1}{}^T QB \\ \underline{u}_{N_h-1}{}^T R + \underline{x}_{N_h}{}^T S_{N_h} B \end{bmatrix} \tag{7.13}$$

Setting this equal to $0^{m \times N_h}$ yields the optimal control inputs as a function of the optimal trajectory:

$$U^* = \begin{bmatrix} -R^{-1}B^T \left((A^{N_h-1})^T S_{N_h} \underline{x}_{N_h} + \sum_{i=0}^{N_h-2} (A^i)^T Q \underline{x}_{i+1} \right) \\ -R^{-1}B^T \left((A^{N_h-2})^T S_{N_h} \underline{x}_{N_h} + \sum_{i=0}^{N_h-3} (A^i)^T Q \underline{x}_{i+2} \right) \\ \vdots \\ -R^{-1}B^T \left(A^T S_{N_h} \underline{x}_{N_h} + Q \underline{x}_{N_h-1} \right) \\ -R^{-1}B^T S_{N_h} \underline{x}_{N_h} \end{bmatrix} \tag{7.14}$$

Now, let $\underline{\lambda}_k \in \mathbb{R}^n$ be defined over $k \in \mathbb{N}_{[1,N_h]}$ as:

$$\underline{\lambda}_k = \begin{cases} (A^{N_h-k})^T S_{N_h} \underline{x}_{N_h} + \sum_{i=0}^{N_h-k-1} (A^i)^T Q \underline{x}_{i+k} & \text{if } 1 \leq k < N_h \\ S_{N_h} \underline{x}_{N_h} & \text{if } k = N_h \end{cases} \tag{7.15}$$

Then, the optimal control input at every time step $k \in \mathbb{N}_{[0,N_h-1]}$ can be written simply as:

$$\underline{u}_k = -R^{-1} B^T \underline{\lambda}_{k+1} \tag{7.16}$$

which can be substituted into Eq. (7.1):

$$\underline{x}_{k+1} = A\underline{x}_k - BR^{-1}B^T \underline{\lambda}_{k+1} \tag{7.17}$$

This is still not useful when trying to compute the actual inputs as the value $\underline{\lambda}_k$ depends on the trajectory. Looking back to Eq. (7.6), the term $\frac{\partial V^\mu(\underline{x}_0)}{\partial X}\frac{\partial X}{\partial U}$ can be analyzed further as the values of $\underline{\lambda}_k$ represent this part of the derivative. Taking the derivative of that term with respect to X yields:

$$\frac{\partial}{\partial X}\left(\frac{\partial V^\mu(\underline{x}_0)}{\partial X}\frac{\partial X}{\partial U}\right) = \frac{\partial^2 V^\mu(\underline{x}_0)}{\partial X^2}\frac{\partial X}{\partial U} + \frac{\partial V^\mu(\underline{x}_0)}{\partial X}\frac{\partial^2 X}{\partial X \partial U} \tag{7.18}$$

$$= \begin{bmatrix} Q & Q & \cdots & Q & S \end{bmatrix} \frac{\partial X}{\partial U} + 0^{n \times nN_h} \tag{7.19}$$

which is a constant and does not depend on X. This leads to the assumption that

$$\lambda_k = S_k \underline{x}_k \; \forall \; k \in \mathbb{N}_{[1,N_h]} \tag{7.20}$$

and Eq. (7.17) can be rewritten as:

$$\underline{x}_{k+1} = A\underline{x}_k - BR^{-1}B^T S_{k+1}\underline{x}_{k+1} \tag{7.21}$$

$$= (I + BR^{-1}B^T S_{k+1})^{-1} A\underline{x}_k \tag{7.22}$$

Setting Eq. (7.20) and Eq. (7.15) equal yields the system:

$$S_k\underline{x}_k = (A^{N_h-k})^T S_{N_h}\underline{x}_{N_h} + \sum_{i=0}^{N_h-k-1} (A^i)^T Q\underline{x}_{i+k} \tag{7.23}$$

$$S_{N_h}\underline{x}_{N_h} = S_{N_h}\underline{x}_{N_h} \tag{7.24}$$

The second equation is trivial as it is simply the boundary condition. Using the first equation and working backward from N_h, a couple terms can be explicitly written:

$$S_{N_h-1}\underline{x}_{N_h-1} = A^T S_{N_h}\underline{x}_{N_h} + Q\underline{x}_{N_h-1} \tag{7.25}$$

$$= A^T S_{N_h}(I + BR^{-1}B^T S_{N_h})^{-1} A\underline{x}_{N_h-2} + Q\underline{x}_{N_h-1} \tag{7.26}$$

$$= \left(A^T S_{N_h}(I + BR^{-1}B^T S_{N_h})^{-1} A + Q\right) \underline{x}_{N_h-1} \tag{7.27}$$

$$S_{N_h-2}\underline{x}_{N_h-2} = (A^2)^T S_{N_h}\underline{x}_{N_h} + A^T Q\underline{x}_{N_h-1} + Q\underline{x}_{N_h-2} \tag{7.28}$$

$$= A^T \left(A^T S_{N_h}(I + BR^{-1}B^T S_{N_h})^{-1} A + Q\right) \underline{x}_{N_h-1} + Q\underline{x}_{N_h-2} \tag{7.29}$$

$$= \left(A^T S_{N_h-1}(I + BR^{-1}B^T S_{N_h-1})^{-1} A + Q\right) \underline{x}_{N_h-2} \tag{7.30}$$

These results give the discrete matrix Riccati equation (MRE):

$$S_k = A^T S_{k+1}(I + BR^{-1}B^T S_{k+1})^{-1} A + Q \tag{7.31}$$

The full optimal solution is then written over $k \in \mathbb{N}_{[0,N_h-1]}$ as:

$$\theta_k = \underline{u}_k \tag{7.32}$$

$$\underline{u}_k = -R^{-1}B^T\underline{\lambda}_{k+1} \tag{7.33}$$

$$\underline{\lambda}_{k+1} = S_{k+1}\underline{x}_{x+1} \tag{7.34}$$

$$\underline{x}_{k+1} = (I + BR^{-1}B^TS_{k+1})^{-1}A\underline{x}_k \tag{7.35}$$

$$S_k = A^TS_{k+1}(I + BR^{-1}B^TS_{k+1})^{-1}A + Q \tag{7.36}$$

with the boundary conditions S_{N_h} and $\underline{x}_0$. The control policy can be written as $\underline{u}_k = K_k\underline{x}_k$ where $K_k = -R^{-1}B^TS_{k+1}(I + BR^{-1}B^TS_{k+1})^{-1}A$.

7.2 Linear Feedback

Now, the goal is to attain a constant optimal linear feedback gain while considering the linear system and quadratic cost function given by Eqs. (7.1) and (7.2), respectively. Consider the case where $\mathcal{U} = \mathbb{R}$, so $\Theta \subset \mathbb{R}^{1\times n}$. By constraining the control to a singular input at each time, the reshaping of K to θ is avoided and the feedback law is given by:

$$u_k = \theta\underline{x}_k \;\forall\; k \in \mathbb{N}_{[0,N_h-1]} \tag{7.37}$$

The different components needed to evaluate Eq. (4.25) are now explored. Let $S_{N_h} = Q$, then the value function derivatives can be formed as:

$$\frac{\partial V^\mu(\underline{x}_0)}{\partial X} = \begin{bmatrix}\underline{x}_0{}^TQ & \underline{x}_1{}^TQ & \cdots & \underline{x}_{N_h-1}{}^TQ & \underline{x}_{N_h}{}^TQ\end{bmatrix} \tag{7.38}$$

and

$$\frac{\partial V^\mu(\underline{x}_0)}{\partial U} = \begin{bmatrix}\underline{u}_0{}^TR & \underline{u}_1{}^TR & \cdots & \underline{u}_{N_h-2}{}^TR & \underline{u}_{N_h-1}{}^TR\end{bmatrix} \tag{7.39}$$

which is similar to the open-loop case. Next, the control policy derivatives are taken, over

all time steps $k \in \mathbb{N}_{[0,N_h-1]}$, to be

$$\frac{\partial \mu(\underline{x}_k)}{\partial \underline{x}_k} = \theta \tag{7.40}$$

$$\frac{\partial \mu(\underline{x}_k)}{\partial \theta} = \underline{x}_k{}^T \tag{7.41}$$

which are used to form

$$\frac{\partial U}{\partial X} = \begin{bmatrix} \theta & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & \theta & 0 \end{bmatrix} \tag{7.42}$$

and

$$\frac{\partial U}{\partial \theta} = X^T_{[0,N_h-1]} \tag{7.43}$$

Lastly, the state trajectory derivative is formed using the components

$$\frac{\partial f(\underline{x}_k, u_k)}{\partial \underline{x}_k} = A \tag{7.44}$$

$$\frac{\partial f(\underline{x}_k, u_k)}{\partial u_k} = B \tag{7.45}$$

which apply for all time steps $k \in \mathbb{N}_{[0,N_h-1]}$ and, along with the policy derivatives, give

$$\frac{\partial X}{\partial \theta} = \begin{bmatrix} 0^{n \times n} \\ B\underline{x}_0{}^T \\ \bar{A}B\underline{x}_0{}^T + B\underline{x}_1{}^T \\ \vdots \\ \sum_{i=0}^{N_h-1} \bar{A}^i B \underline{x}^T_{N_h-1-i} \end{bmatrix} \tag{7.46}$$

where $\bar{A} = A + B\theta$.

Using these components, the value function derivative is then found to be

$$\frac{dV^\mu(\underline{x}_0)}{d\theta} = \tilde{X}\left(G + PM\right) X^T_{[0,N_h-1]} \tag{7.47}$$

with the following definitions:

$$\tilde{X} := \begin{bmatrix} \underline{x}_0{}^T & \cdots & \underline{x}_{N_h-1}{}^T \end{bmatrix} \tag{7.48}$$

$$G := \begin{bmatrix} \theta^T R + \bar{A}^T Q B & \cdots & 0^n \\ \vdots & \ddots & \vdots \\ 0^n & \cdots & \theta^T R + \bar{A}^T Q B \end{bmatrix} \tag{7.49}$$

$$P := \begin{bmatrix} \theta^T R \theta + \bar{A}^T Q \bar{A} & \cdots & 0^{n\times n} \\ \vdots & \ddots & \vdots \\ 0^{n\times n} & \cdots & \theta^T R \theta + \bar{A}^T Q \bar{A} \end{bmatrix} \tag{7.50}$$

$$M := \begin{bmatrix} 0^n & 0^n & \cdots & 0^n & 0^n \\ B & 0^n & \cdots & 0^n & 0^n \\ \bar{A}B & B & \cdots & 0^n & 0^n \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \bar{A}^{N_h-2}B & \bar{A}^{N_h-3}B & \cdots & B & 0^n \end{bmatrix} \tag{7.51}$$

A detailed derivation of this result is given in Appendix B.

7.3 Infinite Horizon Approximation Linear Feedback

Working with the problem from the last section, a simplified solution, in comparison to the previous result, can be found for the infinite horizon LQR if a simplifying assumption is made. The finite horizon LQR is derived through optimizing the OCS and the time varying linear feedback control comes out as a result. Additionally, if the horizon goes to infinity, the solution to the MRE stays constant for all time steps and a constant linear feedback gain is found, the infinite horizon LQR. Here, the assumption is made that each control input is independent from the others; that is to say:

$$dU \approx \frac{\partial U}{\partial \theta} d\theta \tag{7.52}$$

Then, using the chain rule, Eq. (4.12) can be written as

$$\left(\frac{\partial V^{\mu}(\underline{x}_0)}{\partial X}\frac{\partial X}{\partial U}+\frac{\partial V^{\mu}(\underline{x}_0)}{\partial U}\right)X_{[0,N_h-1]}^{T}=0^{1\times n} \tag{7.53}$$

For most systems, if $\underline{x}_0 = 0^n$ then $X = 0^{n\times N_h+1}$ and Eq. (7.53) would be true for any value of θ. Assuming $\underline{x}_0 \neq 0^n$ then $X \neq 0^{n\times N_h+1}$, and the optimal solution relies on

$$\frac{\partial V^{\mu}(\underline{x}_0)}{\partial X}\frac{\partial X}{\partial U}+\frac{\partial V^{\mu}(\underline{x}_0)}{\partial U}=0^{1\times N_h} \tag{7.54}$$

which is equivalent to Eq. (7.6), from the open-loop portion. However, given the linear feedback constraint, $\Theta \neq \mathcal{U}$. The problem considers searching within the parameter space $\Theta = \mathbb{R}^{1\times n}$ while the dynamics are constrained to

$$\underline{x}_{k+1} = (A + B\theta)\,\underline{x}_k \tag{7.55}$$

$$= \bar{A}\underline{x}_k \tag{7.56}$$

Using Eq. (7.39), Eq. (7.54) can be written as

$$G + UR = 0 \tag{7.57}$$

where $0 := 0^{1\times N_h}$ and

$$G := \frac{\partial V^{\mu}(\underline{x}_0)}{\partial X}\frac{\partial X}{\partial U} \tag{7.58}$$

Note, since the control inputs are scalar values, $U^T = U$. Then, substituting the control law gives

$$G + X_{[0,N_h-1]}^{T}\theta^T R = 0 \tag{7.59}$$

The optimal value of θ is then found as a least squares regression:

$$\theta = -R^{-1}G^T X_{[0,N_h-1]}^{T}\left(X_{[0,N_h-1]}X_{[0,N_h-1]}^{T}\right)^{-1} \tag{7.60}$$

This equation is not trivial to solve as both G and $X_{[0,N_h-1]}$ depend on θ, but it does help explain the general workings of this algorithm. The true optimal solution is a time varying linear feedback gain. Given that this solution is constraining the problem to be a constant gain, it would make sense that there is an error minimization problem within the overarching optimal control problem. Some more work can be done to show the gain's independence of $\underline{x}_0$ and develop a dynamic equation for determining G. Developing the value of G yields

$$G^T = B^T \begin{bmatrix} \sum_{i=0}^{N_h-1} (A^i)^T Q \bar{A}^{i+1} & \sum_{i=0}^{N_h-2} (A^i)^T Q \bar{A}^{i+2} & \cdots & \sum_{i=0}^{1} (A^i)^T Q \bar{A}^{i+N_h-1} & Q\bar{A}^{N_h} \end{bmatrix} \underline{x}_0 \quad (7.61)$$

Let S be defined as

$$S := \begin{bmatrix} \sum_{i=0}^{N_h-1} (A^i)^T Q \bar{A}^{i+1} & \sum_{i=0}^{N_h-2} (A^i)^T Q \bar{A}^{i+2} & \cdots & \sum_{i=0}^{1} (A^i)^T Q \bar{A}^{i+N_h-1} & Q\bar{A}^{N_h} \end{bmatrix} \quad (7.62)$$

Then, the last three terms of S can be written fully as:

$$S_{N_h} = Q\bar{A}^{N_h} \quad (7.63)$$

$$S_{N_h-1} = Q\bar{A}^{N_h-1} + A^T Q \bar{A}^{N_h} \quad (7.64)$$

$$S_{N_h-2} = Q\bar{A}^{N_h-2} + A^T Q \bar{A}^{N_h-1} + (A^2)^T Q \bar{A}^{N_h} \quad (7.65)$$

A single equation can be formed to express these terms. Given the boundary condition $S_{N_h} = Q\bar{A}^{N_h}$, the rest of S can be found with:

$$S_k = Q\bar{A}^k + A^T S_{k+1} \quad (7.66)$$

The state trajectory is completely determined by $\underline{x}_0$ and θ as

$$X_{[0,N_h-1]} = \begin{bmatrix} I & \bar{A} & \bar{A}^2 & \cdots & \bar{A}^{N_h-2} & \bar{A}^{N_h-1} \end{bmatrix} \underline{x}_0 \quad (7.67)$$

Then, to simplify this result, let $\tilde{A}$ be defined as

$$\tilde{A} := \begin{bmatrix} I & \bar{A} & \bar{A}^2 & \dots & \bar{A}^{N_h-2} & \bar{A}^{N_h-1} \end{bmatrix} \tag{7.68}$$

Now, returning to Eq. (7.59), the values for G and $X_{[0,N_h-1]}$ can be substituted to obtain

$$\underline{x}_0{}^T S^T B + \underline{x}_0{}^T \tilde{A}^T \theta^T R = 0 \tag{7.69}$$

Then, the optimal value of θ can be found as

$$\theta = -R^{-1} B^T S \tilde{A}^T (\tilde{A}\tilde{A}^T)^{-1} \tag{7.70}$$

Still, S and $\tilde{A}$ depend on θ so the solution to this equation is not trivial. However, the total solution is given, over $k \in \mathbb{N}_{[0,N_h-1]}$, as:

$$u_k = \theta \underline{x}_k \tag{7.71}$$

$$\tilde{A}_k = (A + B\theta)^k \tag{7.72}$$

$$\underline{x}_{k+1} = (A + B\theta)\underline{x}_k \tag{7.73}$$

$$S_k = Q\tilde{A}_k + A^T S_{k+1} \tag{7.74}$$

$$\theta = -R^{-1} B^T S \tilde{A}^T (\tilde{A}\tilde{A}^T)^{-1} \tag{7.75}$$

given the boundary conditions $\underline{x}_0 = x_0$ and $S_{N_h} = Q\bar{A}^{N_h}$.

A fixed point iteration technique is used to solve for θ following the scheme:

$$\theta^{i+1} = \theta^i - \alpha \left(\theta^i + R^{-1} B^T S^i (\tilde{A}^i)^T (\tilde{A}^i (\tilde{A}^i)^T)^{-1} \right) \tag{7.76}$$

where $i \in \mathbb{N}_{[0,\infty)}$ is the iteration and $\alpha \in \mathbb{R}_{(0,1)}$ is a learning rate.

7.4 Linear Quadratic Gaussian

The linear quadratic Gaussian (LQG) is a combination of the LQR and Kalman filter. The linear system in Eq. (7.1) is amended with the process noise $\underline{\omega} \sim \mathcal{N}(0, \sigma_{\underline{\omega}}) \in \mathbb{R}^n$ and a measurement model is added:

$$\underline{x}_{k+1} = A\underline{x}_k + B\underline{u}_k + \underline{\omega}_k \tag{7.77}$$

$$\underline{y}_k = C\underline{x}_k + \underline{\nu}_k \tag{7.78}$$

where $\underline{y}_k \in \mathbb{R}^r$ is the measurement, $C \in \mathbb{R}^{r \times n}$ is the measurement matrix, and $\underline{\nu}_k \sim \mathcal{N}(0, \sigma_{\underline{\nu}}) \in \mathbb{R}^r$ is the measurement noise. The process noise and measurement noise are assumed to be uncorrelated, i.e. each instance of noise is an independent event. Given a stochastic dynamic system, state trajectories come from a probability distribution within the state space. Exact values of the state cannot be easily used within the cost function, so the cost function is modified to include the expectation operator

$$\tilde{J}(X, U) = J(\mathbb{E}[X], U) \tag{7.79}$$

The expectation operator is linear and, given deterministic values, simply returns the same value. The weighting matrices are design variables, and the control inputs are known. So, the expected cost can be simplified to

$$\tilde{J}(X, U) = \frac{1}{2}\hat{\underline{x}}_{N_h}{}^T S_{N_h} \hat{\underline{x}}_{N_h} + \frac{1}{2}\sum_{k=0}^{N_h-1} \hat{\underline{x}}_k{}^T Q \hat{\underline{x}}_k + \underline{u}_k{}^T R \underline{u}_k \tag{7.80}$$

where $\hat{\underline{x}}_k = \mathbb{E}[\underline{x}_k]$. Taking the expectation of Eq. (7.77)

$$\hat{\underline{x}}_{k+1} = A\hat{\underline{x}}_k + B\underline{u}_k \tag{7.81}$$

gives the expected state trajectory, and following the same steps as the deterministic finite

horizon LQR problem, the full optimal solution of θ can be written over $k \in \mathbb{N}_{[0,N_h-1]}$ as:

$$\theta_k = \underline{u}_k \tag{7.82}$$

$$\underline{u}_k = -R^{-1}B^T\underline{\lambda}_{k+1} \tag{7.83}$$

$$\underline{\lambda}_{k+1} = S_{k+1}\hat{\underline{x}}_{x+1} \tag{7.84}$$

$$\hat{\underline{x}}_{k+1} = (I + BR^{-1}B^T S_{k+1})^{-1}A\hat{\underline{x}}_k \tag{7.85}$$

$$S_k = A^T S_{k+1}(I + BR^{-1}B^T S_{k+1})^{-1}A + Q \tag{7.86}$$

The main result is that the optimal control inputs come from a linear control law

$$\underline{u}_k = K_k\hat{\underline{x}}_k \tag{7.87}$$

where $K_k = -R^{-1}B^T S_{k+1}\left(I + BR^{-1}B^T S_{k+1}\right)^{-1}A$, similar to the finite horizon LQR. Compared to the deterministic case, however, the expected value $\hat{\underline{x}}_k$ is used for feedback and must be determined through other means.

For this problem, the Kalman Filter (KF) is known to be the optimal solution. [54] thoroughly explain the KF and other means of estimation. Here, portions of the KF derivation are presented, and a detailed derivation is given in Appendix D. The state estimate is propagated and corrected, respectively, using

$$\hat{\underline{x}}^-_{k+1} = A\hat{\underline{x}}_k + B\underline{u}_k \tag{7.88}$$

$$\hat{\underline{x}}_k = \hat{\underline{x}}^-_k + L_k\left[\underline{y}_k - C\hat{\underline{x}}^-_k\right] \tag{7.89}$$

where the superscript "$-$" denotes an uncorrected estimate, the absence of the superscript "$-$" denotes the corrected estimate, and $L_k \in \mathbb{R}^{n\times r}$ is a feedback gain to be optimized. The KF minimizes the covariance $P_k \in \mathbb{R}^{n\times n}$ of the state estimates by directly optimizing the

time-varying feedback gain L_k

$$min_{L_k}\, tr(P_k) \;\; \forall \;\; k \in \mathbb{N}_{[0,N_h]} \tag{7.90}$$

Using Eq. (7.88), Eq. (7.89), and the definition of covariance

$$P_k = \mathbb{E}\left[\underline{e}_k \underline{e}_k{}^T\right] \tag{7.91}$$

where $\underline{e}_k = \underline{\hat{x}}_k - \underline{x}_k$ is the estimation error, the dynamics of the covariance can be found. Similar to the state estimate, the covariance is propagated and corrected at each time step using

$$P_{k+1}^- = AP_kA^T + \Omega \tag{7.92}$$

$$P_k = (I - L_kC)\, P_k^- \tag{7.93}$$

Optimizing the feedback gain according to Eq. (7.90) using the covariance dynamics gives the Kalman gain

$$L_k = P_k^- C^T \left(CP_k^- C^T + \Lambda\right)^{-1} \tag{7.94}$$

8 Simulation Environments

The PPMPC algorithm is tested within two simulation environments: cart-pole and dot intercept. The cart-pole environment is implemented as a deterministic system that tests the nonlinear predictive capabilities and unmodelled dynamics. The dot intercept environment shows decision making capabilities within a stochastic environment.

8.1 Cart-Pole

The cart-pole system, which consists of a pendulum attached to a movable cart, is quite common in both control theory and machine learning. Force inputs are applied to the cart and can be used to control the angle of the pendulum. While the system is quite simple, its dynamics are nonlinear and provide some unique tasks. The pendulum swing up task

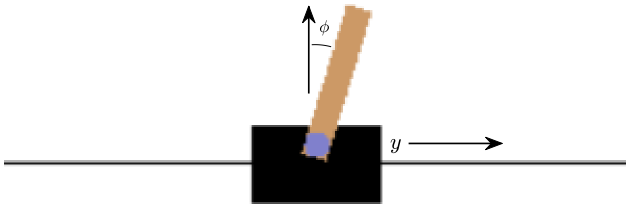

Figure 8.1 Cart-pole environment

provides a challenge for predictive controllers and reinforcement learning algorithms while balancing the upright pendulum gives a nice platform for linear control techniques. Some research has also focused on controlling variants with multiple pendulums attached to each other. Glück et al. present a brief history of work involving the triple cart-pole and derive a nonlinear controller for the swing up maneuver. The regular cart-pole system, used to test the control algorithm presented here, is described by the following two nonlinear equations:

$$\ddot{y} = \frac{f - m\sin\phi(L\dot{\phi}^2 + g\cos\phi)}{M + m(1 - \cos^2\phi)} \tag{8.1}$$

$$\ddot{\phi} = \frac{g\sin\phi + \ddot{y}\cos\phi}{L} \tag{8.2}$$

where y is the position of the cart, ϕ is the angle between the pendulum and straight up, f is the force applied to the cart, m is the plumb bob mass, M is the cart mass, and L is the pendulum length. A discrete state space representation can be derived with the state vector $\underline{x} = \begin{bmatrix} y & \dot{y} & \phi & \dot{\phi} \end{bmatrix}^T$ using Euler integration:

$$f(\underline{x}_k, \underline{u}_k) = \underline{x}_k + \Delta t f_c(\underline{x}_k, \underline{u}_k) \tag{8.3}$$

where $\Delta t \in \mathbb{R}_{>0}$ is the discrete time step and $f_c : \mathcal{X} \times \mathcal{U} \mapsto \mathbb{R}^n$ is the continuous dynamics function corresponding to the state vector $\underline{x}$. The pole angle is constrained to the range $(-\pi, \pi]$ where zero corresponds to the pole being straight up. This constraint can be enforced with the appropriate wrapping function on the pole angle.

8.1.1 Cart-Pole Cost Function

The cart-pole will be subject to a quadratic cost function, aimed at regulating the system. The cart-pole dynamics are presented such that he origin corresponds to the pole pointing straight up at the unstable equilibrium point. The cost function is given by

$$J(X, U) = \frac{1}{2} \underline{x}_{N_h}{}^T S \underline{x}_{N_h} + \frac{1}{2} \sum_{k=0}^{N_h - 1} \underline{x}_k{}^T Q \underline{x}_k + R u_k{}^2 \tag{8.4}$$

where

$$S = Q = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{8.5}$$

and $R = 0.001$. The horizon length N_h is varied over the presented simulations. The partial derivatives of the cost function are computed as

$$\frac{\partial J(X,U)}{\partial X} = \begin{bmatrix} \underline{x}_0{}^T Q & \cdots & \underline{x}_{N_h-1}{}^T Q & \underline{x}_{N_h}{}^T S \end{bmatrix} \tag{8.6}$$

$$\frac{\partial J(X,U)}{\partial U} = \begin{bmatrix} Ru_0 & \cdots & Ru_{N_h-1} \end{bmatrix} \tag{8.7}$$

8.2 Dot Intercept

In the dot intercept environment, a single interceptor is tasked with reducing the total incurred cost from several incoming dots. A goal is placed at a random position $y_{target} \in \mathbb{R}_{[y_{min},y_{max}]}$ at the bottom of the play area. The vertical position is taken to be $z \in \mathbb{R}_{[z_{min},z_{max}]}$ with positive being up, and the horizontal position $y \in \mathbb{R}_{[y_{min},y_{max}]}$ is positive to the right. Each dot and the interceptor abide by the stochastic differential equation:

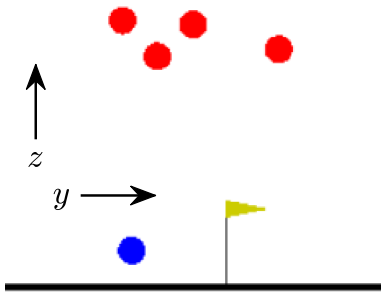

Figure 8.2 Dot intercept environment

$$d\underline{x}^i = f_c(\underline{x}^i, \underline{u}^i)dt + \sigma^i(\underline{x}^i, \underline{u}^i)dW \tag{8.8}$$

where $i \in \mathbb{N}_{[0,N_d]}$ in an index over the agents, $i = 0$ corresponds to the interceptor, $N_d \in \mathbb{N}$ is the number of dots, $\underline{x}^i = [y^i \;\; \dot{y}^i \;\; z^i \;\; \dot{z}^i]^T$, $\underline{u}^i = \begin{bmatrix} f_y^i & f_z^i \end{bmatrix}^T$ is the control input vector consisting of y and z force inputs, $f_c : \mathcal{X}^{play} \times \mathcal{U} \mapsto \mathbb{R}^4$ is the infinitesimal mean, $\sigma^i : \mathcal{X}^{play} \times \mathcal{U} \mapsto \mathbb{R}^{4\times 4}$ is the infinitesimal standard deviation for the i-th agent, $\mathcal{X}^{play} \subseteq \mathbb{R}^4$ is the play area for a

single agent, and W is a Wiener process. The dynamics given by the mean are

$$f_c(\underline{x}^i, \underline{u}^i) = A\underline{x}^i + B\underline{u}^i + \underline{g} \tag{8.9}$$

where

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad B = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \tag{8.10}$$

and $\underline{g} = [0 \;\; 0 \;\; 0 \;\; -g]^T$ with g being the acceleration due to gravity. Using Euler integration, the system can be written in a discrete time form as:

$$\underline{x}^i_{k+1} = \underline{x}^i_k + f(\underline{x}^i, \underline{u}^i) + \sigma^i(\underline{x}^i_k, \underline{u}^i_k)\sqrt{\Delta t}\underline{w}^i_k \tag{8.11}$$

where $\underline{w}^i_k \in \mathbb{R}^4$ such that $\forall\, j \in \mathbb{N}_{[1,4]}\ \underline{w}^i_{k,j} \sim \mathcal{N}(0,1)$ with $\underline{w}^i_{k,j}$ being the j-th element of $\underline{w}^i_k$ and the mean dynamics are propagated with

$$f(\underline{x}^i, \underline{u}^i) = \begin{cases} \Delta t f_c(\underline{x}^i_k, \underline{u}^i_k) & \text{if } \underline{x}^i_3 > z_{min} \\ \begin{bmatrix} \underline{x}^i_1 & 0 & z_{min} & 0 \end{bmatrix}^T & \text{otherwise} \end{cases} \tag{8.12}$$

which includes collision detection with the bottom of the play area. In its most basic form, the standard deviation will be used as

$$\sigma^i(\underline{x}^i_k) = \begin{bmatrix} \sqrt{v^i_1(\underline{x}^i_k)} & 0 & 0 & 0 \\ 0 & \sqrt{v^i_2(\underline{x}^i_k)} & 0 & 0 \\ 0 & 0 & \sqrt{v^i_3(\underline{x}^i_k)} & 0 \\ 0 & 0 & 0 & \sqrt{v^i_4(\underline{x}^i_k)} \end{bmatrix} \tag{8.13}$$

where $v_j(\underline{x}_k^i) \geq 0 \; \forall \; \underline{x}^i \in \mathcal{X}^{play}$, $j \in \mathbb{N}_{[1,4]}$ are variances of the process noise in each state. Ground collision is considered within the variance as well, so each variance is either constant or zero:

$$v_j(\underline{x}_k^i) = \begin{cases} v_j & \text{if } \underline{x}_3^i > z_{min} \\ 0 & \text{otherwise} \end{cases} \tag{8.14}$$

Note, if $v_j = 0 \; \forall \; j \in \mathbb{N}_{[1,4]}$, then the dynamics are deterministic.

A free final time is used within the dot intercept environment. If the interceptor hits a dot, touches the bottom of the play area, or leaves any other side of the play area, the simulation is stopped. Additionally, if all of the dots come into contact with the ground, the simulation is stopped. Once the simulation ends, the cost at the terminal state is evaluated.

8.2.1 Dot Intercept Cost Function

The cost function used within the dot intercept environment does not satisfy all of the assumptions presented in Section 3.2.1, which are needed when proving regulation stability. However, given the nature of the problem presented in the dot intercept environment, these assumptions are not applicable, although special consideration is given to make the cost function continuous. Subsequently, gradient-based optimization methods can be used without the concern of undefined derivatives. Additionally, the cost function partially satisfies the radially unboundedness assumptions, which helps ensure the interceptor does not leave the play area.

The objective is to create a cost function that when used to optimize the parameters of an adequately powerful policy (e.g. a NN), discrete decisions can be made. If the policy being used is not capable of approximating abstract functions, decision making is likely not attainable. Traditionally, decision making problems are solved using RL algorithms where some differentiable approximating function is used to predict the value of using a control input in a specific state. The policy parameters are then optimized based on this approximating function, which essentially turns some abstract discrete decision making problem into a straightforward optimization problem. Instead of simply defining the desired discrete

outputs, say by returning a positive reward if the correct dot is intercepted, a continuous cost function is designed to return the lowest cost when the correct decision is made. Collision between the interceptor and dots is evaluated by comparing the distances between the two centers and the radii of the bodies:

$$\sqrt{(\underline{x}_1^0 + \underline{x}_1^i)^2} + \sqrt{(\underline{x}_3^0 + \underline{x}_3^i)^2} \leq r^{int} + r^i \tag{8.15}$$

where $r^{int}, r^i \in \mathbb{R}_{>0}$ are the radii of the interceptor and i-th agent, respectively.

The cost function is comprised of three main components and follows the same general structure given for the soft terminally constrained control minimization problem given by Eq. (3.11). The cost function is given as

$$J(X, U) = J_{dot}(\underline{x}_{N_h}) + J_{bounds}(\underline{x}_{N_h}) + J_{inputs}(U) \tag{8.16}$$

where the first two components are the terminal cost

$$\phi(\underline{x}_{N_h}) := J_{dot}(\underline{x}_{N_h}) + J_{bounds}(\underline{x}_{N_h}) \tag{8.17}$$

and the last accounts for the control inputs

$$J_{inputs}(U) = \sum_{k=0}^{N_h-1} \underline{u}_k{}^T R \underline{u}_k \tag{8.18}$$

The objective is to create a cost function that will return a minimal value when the dot that will land closest to the target is intercepted. This is done by centering one 2D Gaussian curve on each dot and varying the depth of each curve based on the predicted landing location of that particular dot. Additionally, a component is added to the depth that entices the interceptor to actively engage the dots instead of waiting for them to fall and making an interception close to the target. Figure 8.3 shows a representation of the terminal cost

component. In this case, it is assumed $N_d = 2$, $y \in \mathbb{R}_{[-1,1]}$, and $z \in \mathbb{R}_{[0,2]}$. The optimal decision would be to intercept the dot centered on $y = 0$, since that curve is deeper.

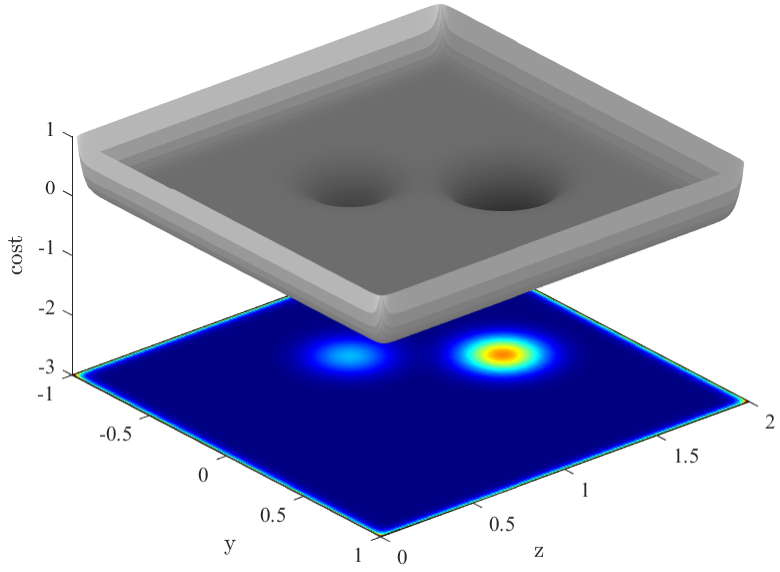

Figure 8.3 Dot intercept cost function example representation.

The N_d Gaussian curves $\eta : \mathcal{X}^{play} \times \mathcal{X}^{play} \mapsto \mathbb{R}_{<0}$ are summed to create a plane with multiple local minima:

$$J_{dot}(\underline{x}_{N_h}) = \sum_{i=1}^{N_d} \eta(\underline{x}^i, \underline{x}^0) \tag{8.19}$$

where $\underline{x}^0 \in \mathcal{X}^{play}$ is the interceptor state and $\underline{x}^i \in \mathcal{X}^{play}$ is the state of the i-th dot. The 2D Gaussian curves are evaluated as

$$\eta(\underline{x}^i, \underline{x}^0) = -\delta(\underline{x}^i)\, exp\left(\frac{(\underline{x}^0 - \underline{x}^i)^T C_{pos} (\underline{x}^0 - \underline{x}^i)}{2{\sigma^i_{dot}}^2}\right) \tag{8.20}$$

where $\delta : \mathcal{X}^{play} \mapsto \mathbb{R}^{>0}$ evaluates the predicted landing location of the i-th dot and returns the depth for that particular curve, $\sigma^i_{dot} \in \mathbb{R}_{>0}$ is the standard deviation of the Gaussian

curve, and the matrix $C_{pos} \in \mathbb{R}^{4\times4}$ is used to measure the y and z positions:

$$C_{pos} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{8.21}$$

The depth of each curve is evaluated based on a 1D Gaussian curve centered at the target, as shown in Figure 8.4, and a z-axis component:

$$\delta(\underline{x}^i) = c_z(x_3^i - z_{min}) + c_d\, exp\left(\frac{(x_{1,land}^i - y_{target})^2}{2{\sigma_{target}}^2}\right) \tag{8.22}$$

where $c_z, c_d \in \mathbb{R}_{>0}$ are tuning coefficients, $\underline{x}_{1,land}^i$ is the predicted y landing position of the i-th dot, and $\sigma_{target} \in \mathbb{R}^{>0}$ is the standard deviation. The z-axis component is a simple linear function that decreases in value as the dots fall towards the ground, which means it is optimal for the interceptor to actively engage the dots. Note, both the 2D and 1D Gaussian curves used here do not represent probability distributions for any random variables. The Gaussian curve just supplies the desired shape needed for this task. So, the standard deviations σ_{dot}^i and σ_{target} are simply used to adjust the width of the curves and do not really function as standard deviations. The predicted landing location for each dot can be found with kinematic equations, assuming the dots have zero control inputs. Given an estimate of a dot's current state, or the exact state in the deterministic case, the landing position can be found with:

$$\Delta t_{land}^i = \frac{-x_4^i - \sqrt{{x_4^i}^2 - 2gx_3^i}}{g} \tag{8.23}$$

$$x_{1,land}^i = x_2^i \Delta t_{land}^i + x_1^i \tag{8.24}$$

A Kalman filter can be used to optimally estimate each dot's states. The final component comes from the position constraints. It is desirable for the interceptor to not leave the play

area or come into contact with the bottom of the play area, which can be viewed as the ground. Exponential functions are used to achieve this with

$$J_{bounds}(\underline{x}_{N_h}) = e^{-c_b(\underline{x}_1^0 - y_{min})} + e^{c_b(\underline{x}_1^0 - y_{max})} + e^{-c_b(\underline{x}_3^0 - z_{min})} + e^{c_b(\underline{x}_3^0 - z_{max})} \tag{8.25}$$

where $c_b \in \mathbb{R}_{>0}$ is used to adjust how quickly the soft boundary constraints activate. In Figure 8.3 the boundary cost function can be seen as the cost increases around the edges of the play area. The larger c_b becomes, the closer each exponential function approaches a discontinuous step from 0 to ∞, which is displayed in Figure 8.4.

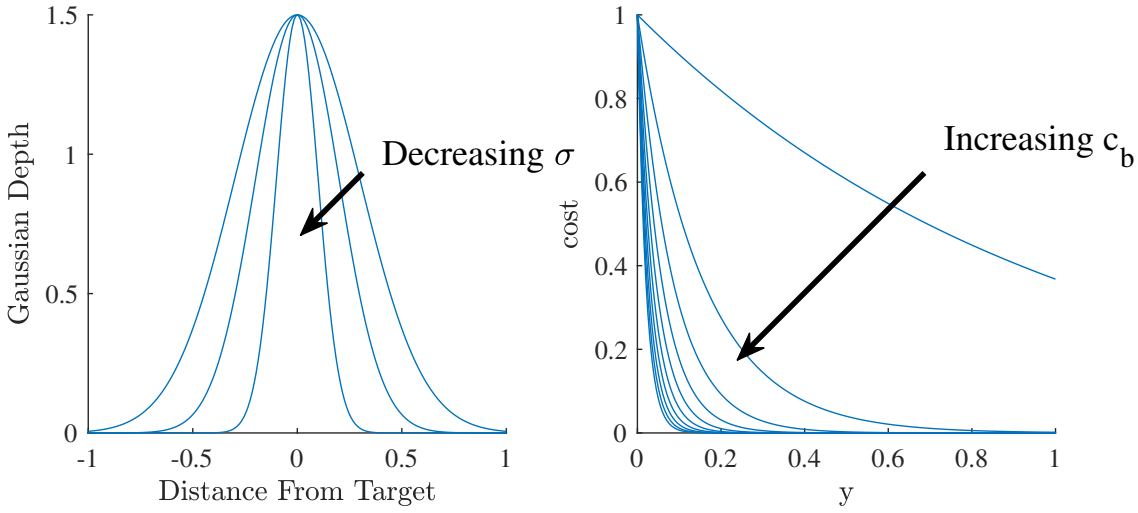

Figure 8.4 Effects of varying the depth standard deviation and the boundary constraint constant.

The standard deviation of the 2D Gaussian curves must be adjusted such that a sufficiently low value is returned when a dot is intercepted while also retaining a distinction between the dots. Given that the simulation is stopped when the distance between the interceptor and a dot is less than the sum of their radii and the position of the interceptor is used within the cost function evaluation, the cost function will never return the global minimum value. Since the 2D Gaussian curves are centered on the dots, if the standard deviation is too

small, the interceptor may not be able to enter the area where a lower cost will be observed. For example, if the radii of the interceptor and dot were both 1.5 and the standard deviation of the 2D Gaussian curve were set to 1, then upon collision the cost corresponding to a Gaussian curve evaluated at three times its standard deviation is returned, which is close to zero. If the standard is set too large, the interceptor will not be able to decipher between two dots that are close together, as their separate Gaussian curves will merge together into a single curve.

The standard deviation of the 1D Gaussian curve used in part to evaluate the depth of the 2D curves can be adjusted to define the area around the target in which it is undesirable for a dot to land. Given the standard deviation is sufficiently large, the policy should learn to intercept the dot that will land closest to the target anywhere within the admissible y-range. The standard deviation of the curve can also be reduced so that the interceptor should learn to only intercept dots that will land within a subset of the y-range. Within Figure 8.4 it can be seen that for the small standard deviation case, a dot landing farther than 0.5 units away from the target will produce no extra depth in its 2D curve. In the largest standard deviation case, a dot would need to land in excess of 1 unit away from the target to produce little depth in its 2D curve.

Consider Eq. (8.16), two of the terms are solely functions of the system state and one only depends on the control inputs. Hence, the cost derivative with respect to the state and control trajectories can be written as

$$\frac{\partial J(X,U)}{\partial X} = \frac{\partial J_{dot}(\underline{x}_{N_h})}{\partial X} + \frac{\partial J_{bounds}(\underline{x}_{N_h})}{\partial X} \tag{8.26}$$

$$\frac{\partial J(X,U)}{\partial U} = \frac{\partial J_{inputs}(U)}{\partial U} \tag{8.27}$$

The latter equation is straight forward to evaluate as the cost is quadratic with respect to the control inputs:

$$\frac{\partial J_{inputs}(U)}{\partial U} = \begin{bmatrix} \underline{u}_0{}^T R & \cdots & \underline{u}_{N_h-1}{}^T R \end{bmatrix} \tag{8.28}$$

Now, the two state dependent components will be evaluated individually. First, only the terminal state is used for both components, so the partial derivative of the terminal state with respect to the state trajectory can be written as

$$\frac{\partial \underline{x}_{N_h}}{\partial X} = \begin{bmatrix} 0^{n\times n} & \cdots & 0^{n\times n} & I^n \end{bmatrix} \tag{8.29}$$

where $\frac{\partial \underline{x}_{N_h}}{\partial X} \in \mathbb{N}^{n\times nN_h}$. The 2D Gaussian curve component derivative can be written as a summation of individual derivatives

$$\frac{\partial J_{dot}(\underline{x}_{N_h})}{\partial \underline{x}_{N_h}} = \sum_{i=1}^{N_d} \frac{\partial \eta(\underline{x}^i, \underline{x}^0)}{\partial \underline{x}_{N_h}} \tag{8.30}$$

where each curve's derivative can be expanded as

$$\frac{\partial \eta(\underline{x}^i, \underline{x}^0)}{\partial \underline{x}_{N_h}} = \begin{bmatrix} \frac{\partial \eta(\underline{x}^i, \underline{x}^0)}{\partial \underline{x}^0} & 0^{4\times 4} & \cdots & 0^{4\times 4} \\ 0^{4\times 4} & \frac{\partial \eta(\underline{x}^i, \underline{x}^0)}{\partial \underline{x}^1} & \cdots & 0^{n\times n} \\ \vdots & \vdots & \ddots & \vdots \\ 0^{4\times 4} & 0^{4\times 4} & \cdots & \frac{\partial \eta(\underline{x}^i, \underline{x}^0)}{\partial \underline{x}^{N_d}} \end{bmatrix} \tag{8.31}$$

The derivative will be evaluated in terms of the individual agents' states and reassembled to form the entire state vector. First, define the curve

$$\mathcal{N}^i_{dot}(\underline{x}_{N_h}, \sigma^i_{dot}) := exp\left(\frac{(\underline{x}^0 - \underline{x}^i)^T C_{pos} (\underline{x}^0 - \underline{x}^i)}{2{\sigma^i_{dot}}^2}\right) \tag{8.32}$$

Then the interceptor and dot components can be computed with

$$\frac{\partial \eta(\underline{x}^i, \underline{x}^0)}{\partial \underline{x}^0} = -\delta(\underline{x}^i)\, \mathcal{N}^i_{dot}(\underline{x}_{N_h}, \sigma^i_{dot}) \frac{(\underline{x}^0 - \underline{x}^i)^T C_{pos}}{{\sigma^i_{dot}}^2} \tag{8.33}$$

and

$$\frac{\partial \eta(\underline{x}^i, \underline{x}^0)}{\partial \underline{x}^i} = \mathcal{N}^i_{dot}(\underline{x}_{N_h}, \sigma^i_{dot}) \left(\delta(\underline{x}^i) \frac{(\underline{x}^0 - \underline{x}^i)^T C_{pos}}{{\sigma^i_{dot}}^2} - \frac{\partial \delta(\underline{x}^i)}{\partial \underline{x}^i}\right) \tag{8.34}$$

The depth is a function of the dots' estimated landing positions. The assumption is made that perfect state information exists for each dot, and, regardless of the system's true stochastic process, a deterministic parabolic trajectory is followed to the ground from the current state. The derivative of the depth can then be written as

$$\frac{\partial \delta(\underline{x}^i)}{\partial \underline{x}^i} = \begin{bmatrix} 0 & 0 & c_z & 0 \end{bmatrix} + c_d exp\left(\frac{(x^i_{1,land} - y_{target})^2}{2{\sigma_{target}}^2}\right)\frac{(x^i_{1,land} - y_{target})}{{\sigma_{target}}^2}\frac{\partial x^i_{1,land}}{\partial \underline{x}_{N_h}} \tag{8.35}$$

where

$$\frac{\partial x^i_{1,land}}{\partial x_{N_h}} = \begin{bmatrix} \frac{\partial x^i_{1,land}}{\partial x^i_1} & \frac{\partial x^i_{1,land}}{\partial x^i_2} & \frac{\partial x^i_{1,land}}{\partial x^i_3} & \frac{\partial x^i_{1,land}}{\partial x^i_4} \end{bmatrix} \tag{8.36}$$

and each component can be evaluated. First, the initial y-position acts as a simple offset, so

$$\frac{\partial x^i_{1,land}}{\partial x^i_1} = 1 \tag{8.37}$$

The y-velocity is assumed to be constant, which gives

$$\frac{\partial x^i_{1,land}}{\partial x^i_2} = \Delta t^i_{land} \tag{8.38}$$

The quadratic equation, which is used to calculate the time to land Δt^i_{land}, depends on the two z-axis states. Using the chain rule, the two z-axis derivatives can be written as

$$\frac{\partial x^i_{1,land}}{\partial x^i_3} = x^i_2\frac{\partial \Delta t^i_{land}}{\partial x^i_3} \tag{8.39}$$

$$\frac{\partial x^i_{1,land}}{\partial x^i_3} = x^i_2\frac{\partial \Delta t^i_{land}}{\partial x^i_4} \tag{8.40}$$

where

$$\frac{\partial \Delta t^i_{land}}{\partial x^i_3} = \frac{1}{\sqrt{{x^i_4}^2 - 2gx^i_3}} \tag{8.41}$$

and

$$\frac{\partial \Delta t^i_{land}}{\partial x^i_4} = -\frac{1}{g}\left(1 + \frac{x^i_4}{\sqrt{{x^i_4}^2 - 2g x^i_3}}\right) \tag{8.42}$$

Lastly, the soft boundary constraints are solely a function of the interceptor position, which is contained within the interceptor state. The interceptor state can be extracted from the full state vector with

$$\frac{\partial \underline{x}^0}{\partial \underline{x}_{N_h}} = \begin{bmatrix} I^4 & 0^{4\times 4} & \cdots & 0^{4\times 4} \end{bmatrix} \tag{8.43}$$

and the boundary constraint derivative is written as

$$\frac{\partial J_{bounds}(\underline{x}_{N_h})}{\partial \underline{x}^0} = \begin{bmatrix} \frac{\partial J_{bounds}(\underline{x}_{N_h})}{\partial x^0_1} & 0 & \frac{\partial J_{bounds}(\underline{x}_{N_h})}{\partial x^0_3} & 0 \end{bmatrix} \tag{8.44}$$

where

$$\frac{\partial J_{bounds}(\underline{x}_{N_h})}{\partial x^0_1} = c_b\left(e^{c_b(x^0_1 - y_{max})} - e^{c_b(x^0_1 - y_{min})}\right) \tag{8.45}$$

and

$$\frac{\partial J_{bounds}(\underline{x}_{N_h})}{\partial x^0_3} = c_b\left(e^{c_b(x^0_3 - z_{max})} - e^{c_b(x^0_3 - z_{min})}\right) \tag{8.46}$$

The soft boundary constraints are designed such that the cost function will return large values when the interceptor has left the play area. Increasing the value of c_b increases the constraining effect that these bounds have, as the returned value will grow much faster. When considering numerical optimization, the derivative of these bounds it is then important to consider. If the magnitude of the derivative increases dramatically with respect to the interceptor position, extremely large changes in θ can be made, which will inevitably make the optimization fail. If the value of c_b is too low, then the bounds will not work as a sharp constraint and may affect the overall interception performance.

9 Simulation Results

9.1 Nonlinear Oscillator

In Chapter 1, the advantage of using the PPMPC method within a simple nonlinear system is explored for a single initial condition. A side effect of the nonlinear dynamics and finite horizon is that the optimal control gain depends on the initial condition as seen in Figure 9.1. The MPC problem is not well posed at $\underline{x}(0) = 0^2$, as this is a fixed point,

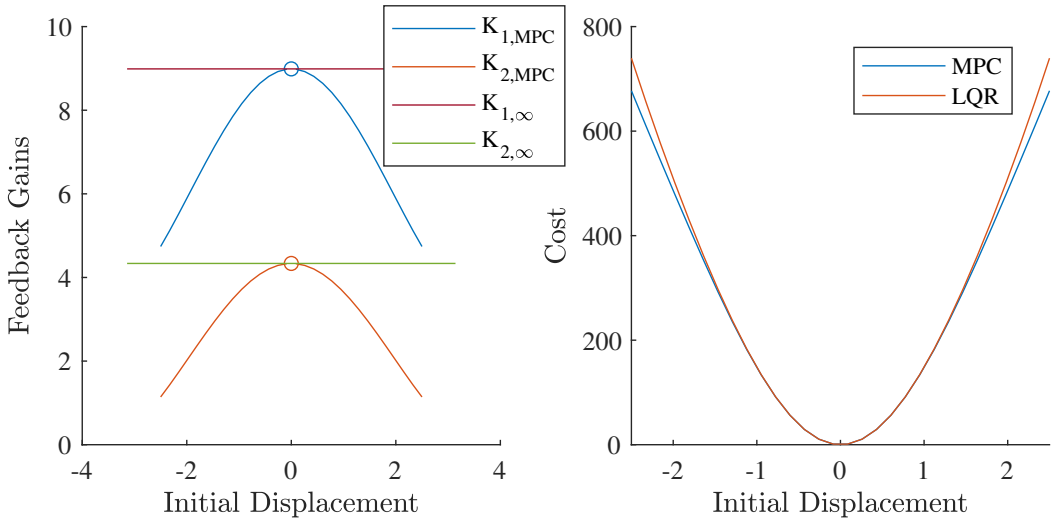

Figure 9.1 Nonlinear oscillator regulation with the linear quadratic regulator and model predictive control derived linear feedback gains. Left shows the value of gains derived and right shows the resulting cost given a varying initial displacement with zero initial speed.

so any value of K will result in the same trajectory of $\underline{x}(t) = 0^2 \ \forall \ t \in [0, \infty)$. However, within Figure 9.1, it is clear that the MPC gain converges to the LQR gain when the initial condition approaches 0^2. As the initial displacement increases, the linearization error increases, which causes the difference in the LQR and MPC costs to increase. While the MPC performance is only slightly better than the LQR, even at larger initial displacements, this shows that calculating optimal feedback gains using MPC can give lower cost solutions than the linearized counterpart. Given a more regnant nonlinear component, the benefit of MPC will be more prevalent as seen in later sections of this chapter.

9.1.1 Constant Gain Regulation over Finite Horizons

It has been shown that time-varying linear state feedback control produces optimal solutions for regulation of linear systems over finite horizons with quadratic cost functions. Constant gain policies are attractive in their ease of implementation and the straightforward assessment of closed-loop stability. In Section 6.1, it is shown that the optimal solution linear feedback gain converges to a constant solution as the optimization horizon approaches infinity and there exists some finite horizon in which the optimal CG policy stabilizes the system around the origin. Figure 9.2 shows not only how the solution to Problem 3.17 converges to the IHLQR solution as $N_h \to \infty$ but also how the solution to Problem 4, the infinite horizon assumption (IHA) presented in Section 7.3, and FHLQR converge.

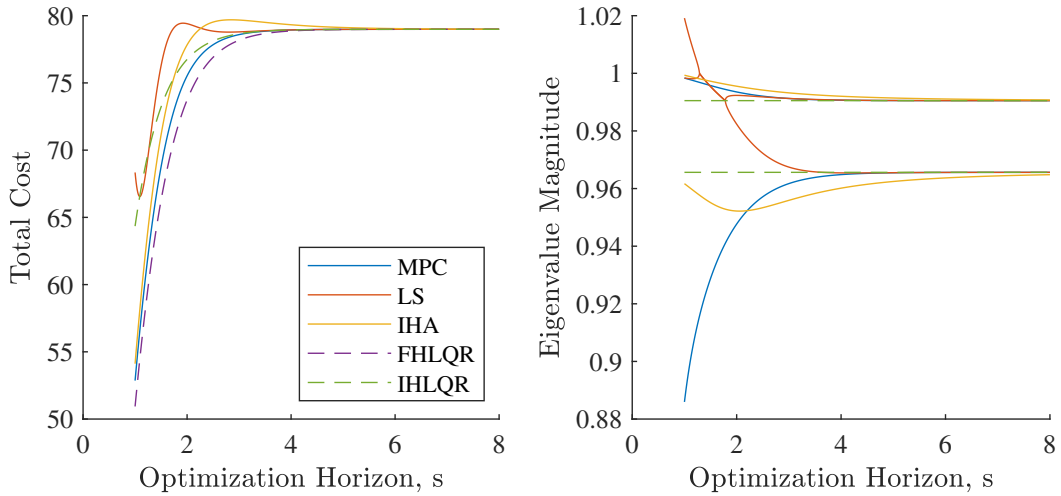

Figure 9.2 Linear oscillator regulation with varying constant gain policy techniques and the time-varying finite horizon linear quadratic regulator solution. Left shows the the total cost observed by each policy, and the right shows the magnitude of the closed-loop eigenvalues.

Figure 9.2 presents the total cost observed by each control policy over the increasing optimization horizon and the magnitude of the closed-loop eigenvalues for the linear oscillator created when $b = 0$. The FHLQR is known to be the true optimal solution and, as expected, observes the lowest cost across the range of optimization horizons. The MPC solution is bounded by the FHLQR and IHLQR, which is also expected. Since the MPC solution is optimized with respect to the the finite horizon, a better performing solution is produced in

comparison to the infinite horizon solution. The two approximation methods LS and IHA produce solutions that vary from being slightly better to worse than the IHLQR. Both the LS and IHA policies are optimized without direct regard for the system dynamics, which significantly hinders performance. However, it can be seen that both the LS and IHA solutions converge to the IHLQR solution as the horizon approaches infinity.

9.2 Stationary Horizon Cart-Pole Regulation

Optimal policies can be determined offline and implemented online similar to explicit MPC, which significantly reduces online computational requirements. Regulation of the cart-pole using a variety of control policy architectures is considered in this section for varying initial conditions, and the performance of the control policies is compared in terms of both minimal cost function output and modelling error handling. The cart-pole offers unique dynamics that allow the polices to be tested in not only nearly linear but also fully nonlinear contexts.

9.2.1 Pole Stabilization

The cart-pole system is regulated from an initial condition of $\underline{x}_0 = [1 \;\; 0 \;\; 4\pi/18 \;\; 0]^T$ over a 1 second using both linear control theory derived policies and various PPMPC derived control policies. The FHLQR provides viable solutions at an optimization horizon of $N_h = 101$ with the nonlinear dynamics model linearized around the origin. The IHLQR is not included as it does not stabilize the system given the large initial pole angle. Similarly, as the optimization horizon is increased, the FHLQR performance degrades quickly as it converges to the IHLQR solution. The FHLQR is derived using the classical methods, but a saturation function is used to constrain the outputs to the admissible region, which also degrades the performance. In comparison, the PPMPC derived policies are optimized with the saturation functions active, which improves overall performance as not only are the nonlinear dynamics better handled but also the nonlinear input saturation functions are taken into account.

The PPMPC approach is used to find an optimal OCS, CG, TG, and NN policy. Additionally, the OCS state-control trajectory is used to perform a nonlinear least squares

Table 9.1 Optimal regulation cost of the cart-pole system over 1 second using various control policies ($N_h = N_s = N_o = 101$, $\Delta t = 0.01s$) with $\underline{x}_0 = [1 \ 0 \ 4\pi/18 \ 0]^T$ and the weights $S = Q = diag(\{1 \ 0 \ 2 \ 0\})$, $R = 0.001$.

Controller	$FHLQR$	OCS	TG	NN	CG	LS
Cost	30.7618	27.8194	27.8255	28.3434	30.4748	38.1267

regression and find another optimal CG policy, which within the context of this example is referred to as the LS policy. The OCS, CG, and TG policies use the discontinuous min-max output layer; however, the derivatives used within training are handled differently. In the OCS case, the derivative at the boundary is assumed to be the identity, and if any control input moves outside of the admissible control range during a training step, that control input is artificially returned to the boundary. In the CG case, no special actions are generally required unless the entire control trajectory were to lie on a boundary. Since the policy parameters directly affect all the control inputs, training information can be passed throughout the horizon. The TG policy is trained using the leaky ReLU method, where the derivatives at and beyond the boundaries are assumed to be some small positive constant. The NN policy is implemented as a SLP and uses a sigmoid output layer to help with derivative scaling.

Table 9.1 shows the total cost incurred by each control policy. All policies perform similarly except for the LS policy, which is a CG architecture trained using a nonlinear least-squares regression with the OCS state-control trajectory. In direct comparison to the the CG policy, found by solving Problem 1, the LS policy performs poorly, clearly showing the advantage of direct optimization of the policy parameters as opposed to the least squares regression. As expected, the OCS policy returns the lowest cost and the TG policy returns a similar result, up to some numerical precision. If $m \leq n$, i.e. there are at most the same number of control inputs as states, the TG policy has universal approximation power and can perfectly replicate the optimal OCS. In this example, $m = 1$ and $n = 4$, and it is clear that, at a particular time step, there exists some gain K_k that maps $\underline{x}_k^*$ to $\underline{u}_k^*$. Hence, the TG policy returns a value comparable to the OCS policy. Numerical optimization results

in some discrepancy. Similarly, the a SLP NN with 64 hidden nodes is implemented and returns an equally comparable cost. Given $N_h = 101$, a SLP network with 101 hidden nodes could theoretically act as a TG policy and return the exact optimal solution. At 64 hidden nodes, some approximation is made but the overall solution is almost identical.

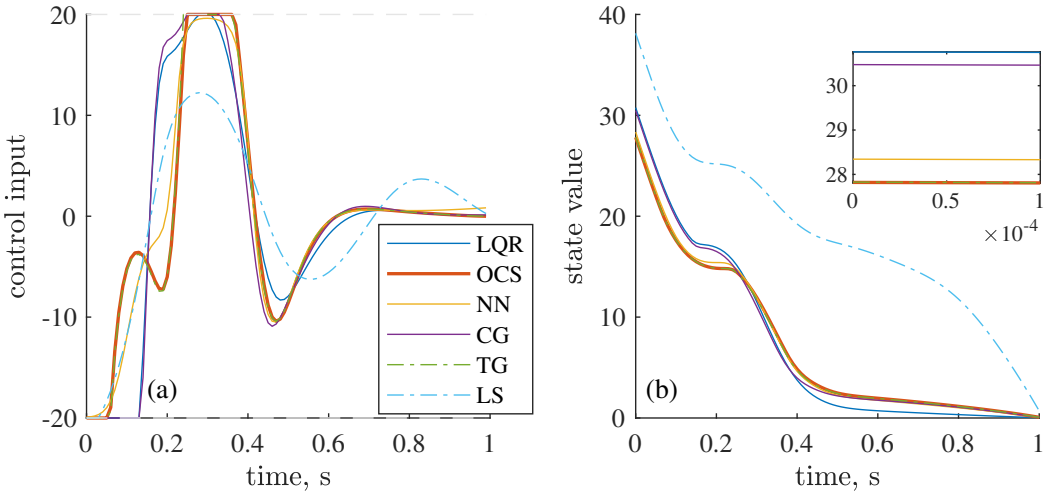

Figure 9.3 Optimal regulation of the cart-pole system over 1 second using various control policies ($N_h = N_s = N_o = 101$, $\Delta t = 0.01s$) with $\underline{x}_0 = [1\ \ 0\ \ 4\pi/18\ \ 0]^T$ and the weights $S = Q = diag(\{1\ \ 0\ \ 2\ \ 0\})$, $R = 0.001$.

Figure 9.3 shows the control inputs and instantaneous state values throughout the simulation. The optimal solution reaches both the maximum and minimum of the admissible control input range. As noted before, the control inputs calculated by the FHLQR are saturated to remain within the admissible range, which hinders its performance. Additionally, this saturation could lead to the IHLQR failing to control the system. The CG policy presented performs better than the FHLQR, even though it is not a time-varying policy, because the control input constraints are considered during the optimization. This problem shows that a simple feedback policy can be trained to perform almost as well as the OCS policy, given that the initial state is within the stabilizable region $\mathcal{W}$. Additionally, Fig. 9.3 shows the poor performance of the LS policy as it does not track any of the other results.

Table 9.2 Average regulation cost of the cart-pole system over 1 second using various control policies ($N_h = N_s = N_o = 101$, $\Delta t = 0.01s$) with $\underline{x}_0 = [0 \ 0 \ \phi_0 \ 0]^T$ where $\phi_0 \sim U(-\pi/18, \pi/18)$ and the weights $S = Q = diag(\{1 \ 0 \ 2 \ 0\})$, $R = 0.001$ over 5000 trials.

Controller	$FHLQR$	$IHLQR$	TG	NN	CG	LS
Cost	0.8288	0.8821	0.8268	0.8722	0.8634	1.686

9.2.2 Stabilization with Stochastic Initial Condition

Another useful application of offline policy determination is finding optimal policies over a range of initial conditions. A TG, NN, and CG policy are optimized by solving Problem 2 assuming that $\underline{x}_0 = [0 \ 0 \ \phi_0 \ 0]^T$, where $\phi_0 \sim U(-\pi/18, \pi/18)$ is the initial pole angle ($\pi/18 = 10°$) and $U(a, b)$ is a uniform distribution over $[a, b]$. Table 9.2 shows a comparison of the policies' performance including the FHLQR, IHLQR, and a CG policy found via LS regression of the FHLQR state-control trajectory. In this case, the initial condition $\underline{x}_0 = [0 \ 0 \ 0.1392 \ 0]^T$ is in close proximity to the origin ($0.1392rad \approx 8°$) and the linear controllers work well. The FHLQR state-control trajectory is nearly optimal as seen by its relative performance in Table 9.2 and Fig. 9.4, so for the sake of computation time, the LS policy is not trained using OCS state-control trajectories. The TG, NN, and CG policies were trained using the ADAM gradient descent method where the gradient was approximated using Eq. (4.28). The number of samples N_b was varied from 1000 to 3000, depending on the size of the policy parameter. The available memory limits N_b as the components of Eq. 4.25 can become quite large, even for a single trajectory, but computing N_b derivatives in parallel increases the memory usage more. Computing the expected derivative via serial batches of samples that can be combined to produce N_b in total will relieve memory issues but increase computation time.

A TG policy is trained offline with the uniformly distributed random initial condition $\underline{x}_0 = [y_0 \ 0 \ \phi_0 \ 0]^T$ where $y_0 \sim U(-1, 1)$ and $\phi_0 \sim U(-4\pi/18, 4\pi/18)$. For this larger region of the state space, a CG policy no longer effectively controls the system. Given the deterministic initial condition $\underline{x}_0 = [1 \ 0 \ 4\pi/18 \ 0]^T$, it is shown in Sec. 9.2.1 that the CG policy controls

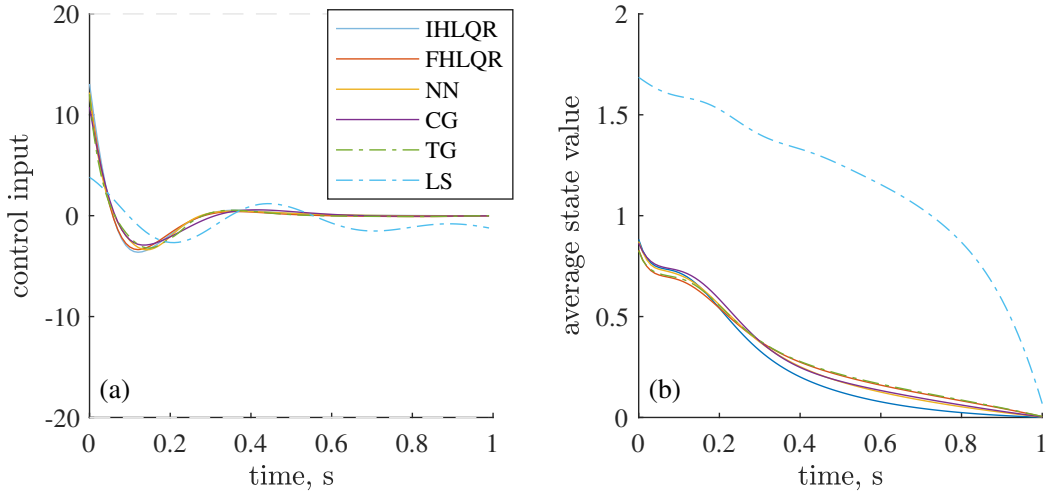

Figure 9.4 Sample regulation of the cart-pole system over 1 second using various control policies ($N_h = N_s = N_o = 101$, $\Delta t = 0.01s$) with $\underline{x}_0 = [0\ \ 0\ \ 0.1392\ \ 0]^T$ in (a), and the average state values of the control policies over 5000 trials in (b) with the weights $S = Q = diag(\{1\ \ 0\ \ 2\ \ 0\})$, $R = 0.001$.

the system in a nearly optimal fashion. However, controlling the system over a range of the state space requires more approximating power, which the CG policy does not have. In comparison, the TG policy has significantly more approximating power as there are N_h feedback gains being determined. Figure 9.5 shows a comparison of the trained TG policy and the FHLQR. A sample control trajectory is given where $\underline{x}_0 = [0.8532\ \ 0\ \ 0.5957\ \ 0]^T$. The TG policy, while providing a more optimal solution in terms of the cost function ($V^{TG}(\underline{x}_0) = 16.39$ and $V^{FHLQR}(\underline{x}_0) = 17.74$), produces an erratic control trajectory that is likely not optimal. However, over 5000 trials with sampled uniformly distributed $\underline{x}_0$, the TG policy return an average cost of 6.84 while the FHLQR returns 7.54. Even without saturating the FHLQR inputs, i.e. allowing the FHLQR to break the control constraints, the average cost returned is 7.05, which is still higher than the trained TG policy.

9.2.3 Constant Gain State Feedback Convergence

In Section 6.1 it is shown that, given the initial condition $\underline{x}_0$ is in the stabilizable region $\mathcal{W}$ then the optimal constant gain feedback policy over an infinite horizon stabilizes the system around the origin. Additionally, it is shown that there exists some finite optimization horizon

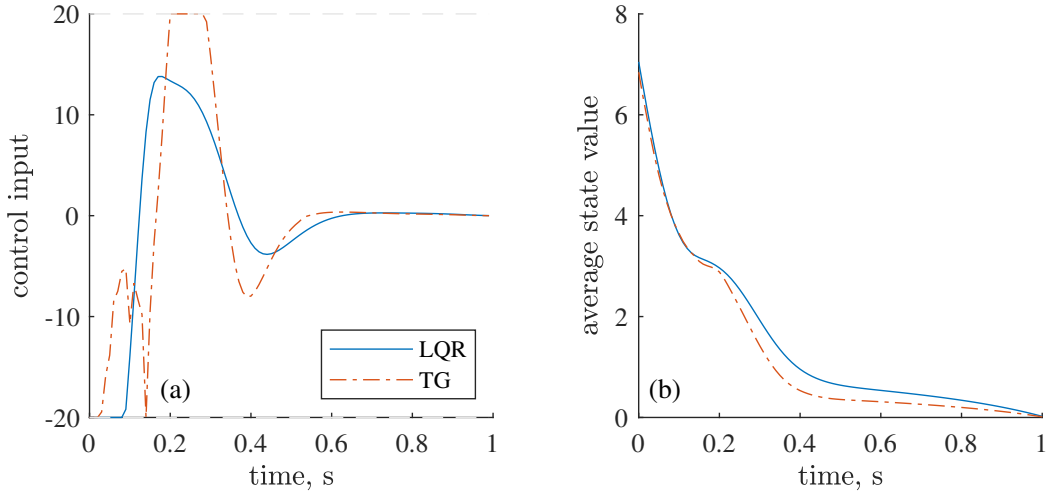

Figure 9.5 Sample regulation of the cart-pole system over 1 second using a time-varying linear feedback policy and the finite horizon linear quadratic regulator ($N_h = N_s = N_o = 101$, $\Delta t = 0.01s$) with $\underline{x}_0 = [0.8532\ \ 0\ \ 0.5957\ \ 0]^T$ in (a), and the average state value of the two policies over 5000 trials in (b) with the weights $S = Q = diag(\{1\ \ 0\ \ 2\ \ 0\})$, $R = 0.001$.

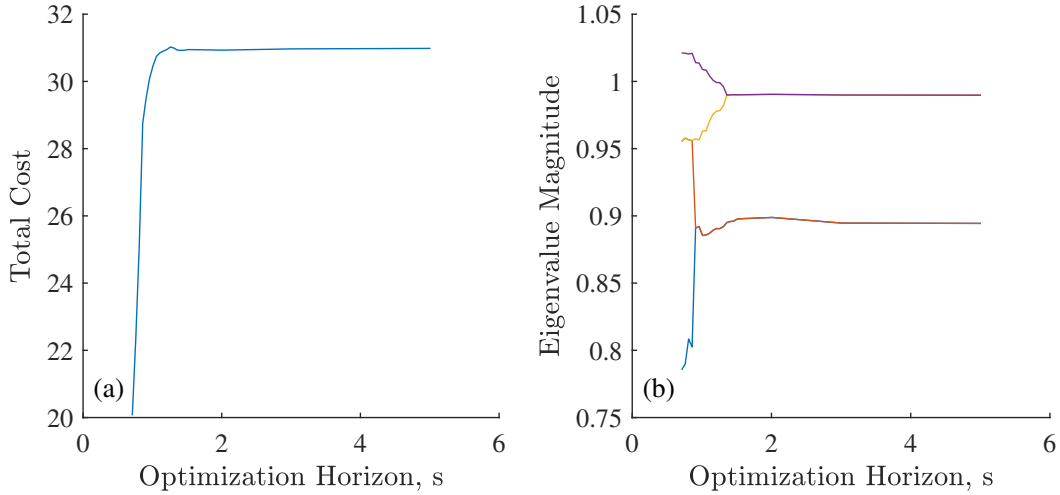

Figure 9.6 Total cost and closed-loop eigenvalues convergence of the cart-pole system controlled by a constant linear state feedback policy over an increasing optimization horizon with the initial condition $\underline{x}_0 = [1\ \ 0\ \ 4\pi/18\ \ 0]^T$.

such that a stabilizing policy is produced. Figure 9.6 shows the total cost and magnitude of the closed-loop eigenvalues over increasing horizon length. Note, the policy is optimized for the nonlinear system but the eigenvalues are computed using a linearized system. As expected, the incurred cost initially increases as there are simply more time steps to sum over,

and eventually the cost converges to a constant value, which occurs when the system begins to reach the origin within the horizon. The convergence of the cost shows that the system is stabilized as the cost would be increasing otherwise. The magnitude of the closed-loop eigenvalues gives similar information as short horizons do not produce stabilizing policies, but given minor increases in the optimization horizon, the eigenvalues converge to constant stable values.

9.2.4 Swing-Up Maneuver

Depending on the initial pole angle, the cart-pole offers both simple regulation and complex nonlinear regulation. When the pole passes horizontal, control inputs have an opposite effect on its angular acceleration. The swing-up maneuver entails the initial pole angle $\phi_0 = \pi$ starts with the pole pointing down and the controller regulating the system. Three policies are trained to regulate the system: an OCS, a NN, and a TG. The genetic algorithm was used to optimize the OCS and NN policy parameters. Since the TG policy has universal approximation power, a LS regression is performed to make the TG controller reproduce the OCS controller's solution. For this example, the NN policy is implemented as a SLP with 512 hidden nodes. Figure 9.7 shows the pole angle and cart position along with the instantaneous cost.

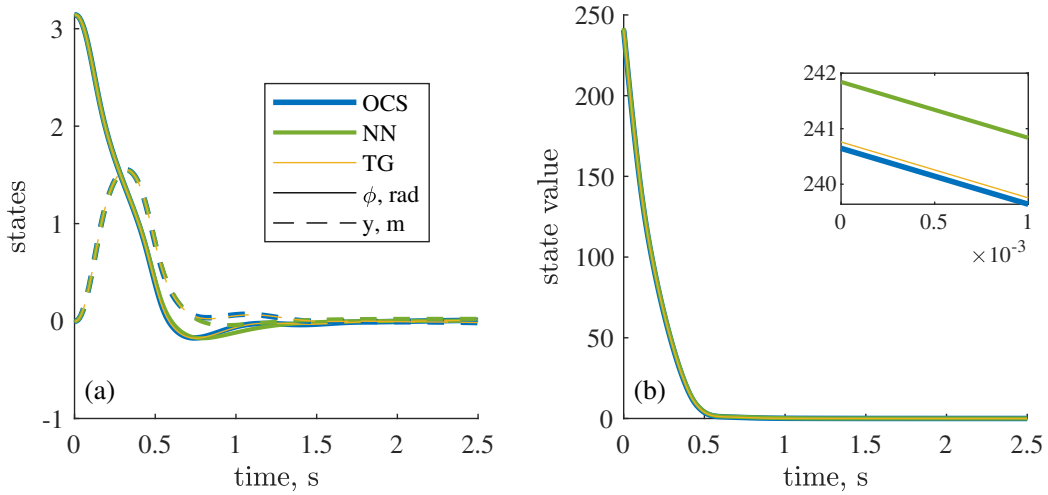

Figure 9.7 Cart-pole swing up maneuver using nonlinear and time-varying control policies.

There are two main issues that need to be addressed when performing this maneuver. First, as previously mentioned, the controller must be nonlinear or time-varying as the sign of the control power changes when the pole passes $\pi/2$. The second issue involves the pi-to-pi wrapping assumed to be used within the model. If the initial condition is $\underline{x}_0 = [0\ 0\ \pi\ 0]^T$ there are two optimal trajectories that return the same cost. The controller can initiate the motion, swing-up the pole in either direction, and incur the same cost as the pi-to-pi wrapping induces some symmetry. Without the pi-to-pi wrapping, every initial condition has its own unique optimal state-control trajectory, but depending on the physical goal of the controller, the pi-to-pi wrapping makes the system operate in a more straightforward manner. Both $\phi = 0$ and $\phi = 2\pi$ correspond to the pole pointing up. With the wrapping function included, $\phi = 2\pi$ becomes $\phi = 0$ and the controller understands that the pole is pointed straight up. Without the wrapping, if the initial angle is started outside of $-\pi$ to π range, extra motion and maneuvers will be used to regulate the system. For instance, if $\phi_0 = 6\pi$, then the controller would need to perform three full rotations in the negative direction to regulate the system.

9.3 Receding Horizon Cart-Pole Regulation with Modelling Error

Taking the trained policies from Section 9.2.1, Fig. 9.8 shows their relative performance given varying error in the pole length modelling. Clearly, the NN policy is not robust to unmodelled dynamics as it performs poorly even with slight variation in pole length. The LS policy is the most robust as its solution varies only slightly over the range of the modelling error. Interestingly, the OCS is mildly robust in comparison to the linear feedback policies. Given that the OCS has no feedback component, it might be assumed that it would perform poorly given modelling error. The CG policy performs nominally if the pole becomes shorter but begins to fail when the length error exceeds $0.4m$, approximately. As the pole length decreases, the system reacts to inputs faster and requires less aggressive forces to be controlled, since the moment arm of the plumb bob mass decreases. Therefore, the CG policy stabilizes the system, although not optimally, when the pole is shorter than what is

modelled within the controller. Oppositely, as the moment arm of the plumb bob increases, the system reacts to inputs slower and requires larger forces to control, which means at some point the CG policy will begin to fail as the system will not respond appropriately to its inputs.

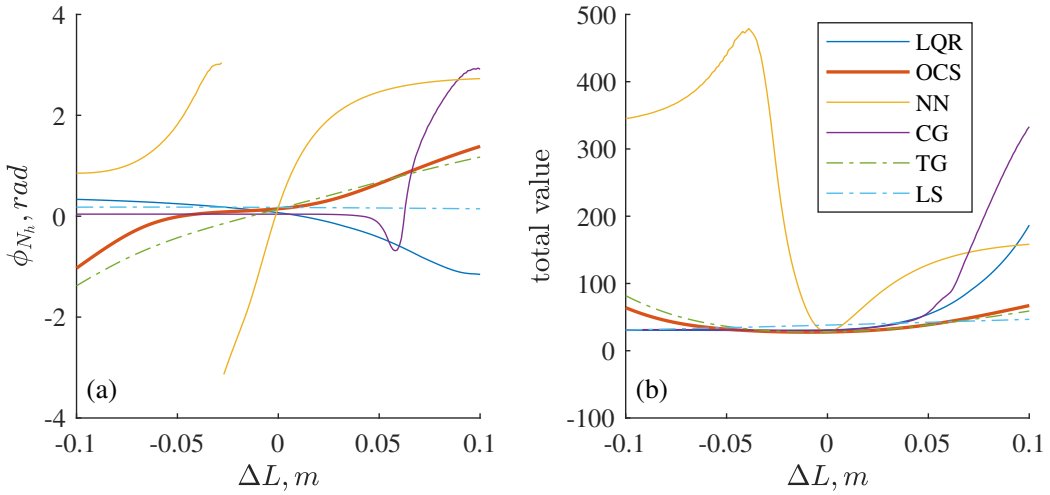

Figure 9.8 Regulation of the cart-pole system over 1 second using various control policies ($N_h = N_s = N_o = 101$, $\Delta t = 0.01s$) with varying amounts of pole length modelling error, the initial condition $\underline{x}_0 = [1\ \ 0\ \ 4\pi/18\ \ 0]^T$, and the weights $S = Q = diag(\{1\ \ 0\ \ 2\ \ 0\})$, $R = 0.001$.

Figure 9.9 shows the cart-pole regulated using two receding horizon controllers: one with an OCS policy and one with a CG policy. The effects of pole length modelling error on the controllers' comparative performance is explored with modelling errors of $0.07m$, $0m$, $-0.07m$, and $-0.15m$ in (a), (b), (c), and (d) of Fig. 9.9, respectively. The initial condition $\underline{x}_0 = [1\ \ 0\ \ 4\pi/18\ \ 0]^T$ and optimization horizon $N_h = 101$ parallel that of the example shown in Sec. 9.2.1. As seen in Fig. 9.8, both the OCS and CG policies regulate the system poorly when the modelling error is $0.07m$. Using receding horizon control introduces an implicit state feedback that increases the robustness of the controller, which is prevalent in (a) of Fig. 9.9 where both controllers effectively regulate the system. Both controllers induce some moderate oscillations at the beginning of the simulation, but the OCS policy reduces the amplitude of the oscillations faster than the CG policy. In (b), the nominal case, there

is some discrepancy between the two trajectories, but generally, they are the same, which is expected given the results in Sec. 9.2.1 where the CG policy performed similarly to the OCS policy. As the modelling error becomes negative, it is expected that the CG policy will perform better than the OCS policy, and this is the case, although to a lesser degree at $-0.07m$. With moderate negative error, the OCS controller exhibits some small scale oscillations not seen with the CG controller, while at large negative error, $-0.15m$, the OCS policy fails to stabilize the system. The CG controller successfully stabilizes the system in both cases with little transient oscillation.

Since each control input of the OCS policy is not constrained by a state-feedback function, it can deliver very aggressive corrections to systems that are not following the predicted trajectory. If a single step receding horizon strategy is used, i.e. $N_o = 1$, the OCS policy will most likely produce the most robust solutions. However, in this example, optimizations are performed every 10 simulation steps, which allows a build up of error prior to the controller optimizing the policy again. The state feedback component of the CG policy adds some inherent robustness as the control trajectory is not fixed. Clearly, the CG policy will not work in all cases, as seen in Fig. 9.8, but given some knowledge about potential issues that could arise within a system and the relative frequency between the dynamics and controller optimization, a suitable policy can be chosen to handle those scenarios.

9.4 Dot Interception with Stochastic Initial Conditions

Two control policies are trained within the dot intercept environment to intercept one of two dots: a 1D CNN and 2-layer MLP network. The CNN policy is implemented with a single convolutional layer equipped with convolutional kernels of size 2, ReLU activation functions, average pooling with 1D kernel size 2, and 8 channels. The convolutional layer takes in 5 time steps of data and feeds into a SLP network with 64 nodes and ReLU activation functions. The MLP network is 2 layers of 128 nodes with ReLU activation functions. The horizontal play area is set to $[-1, 1]$ with the vertical range $[0, 3]$. The target remains at the fixed location $(0, 0)$, and the target standard deviation σ_{target}, used to determine which dot

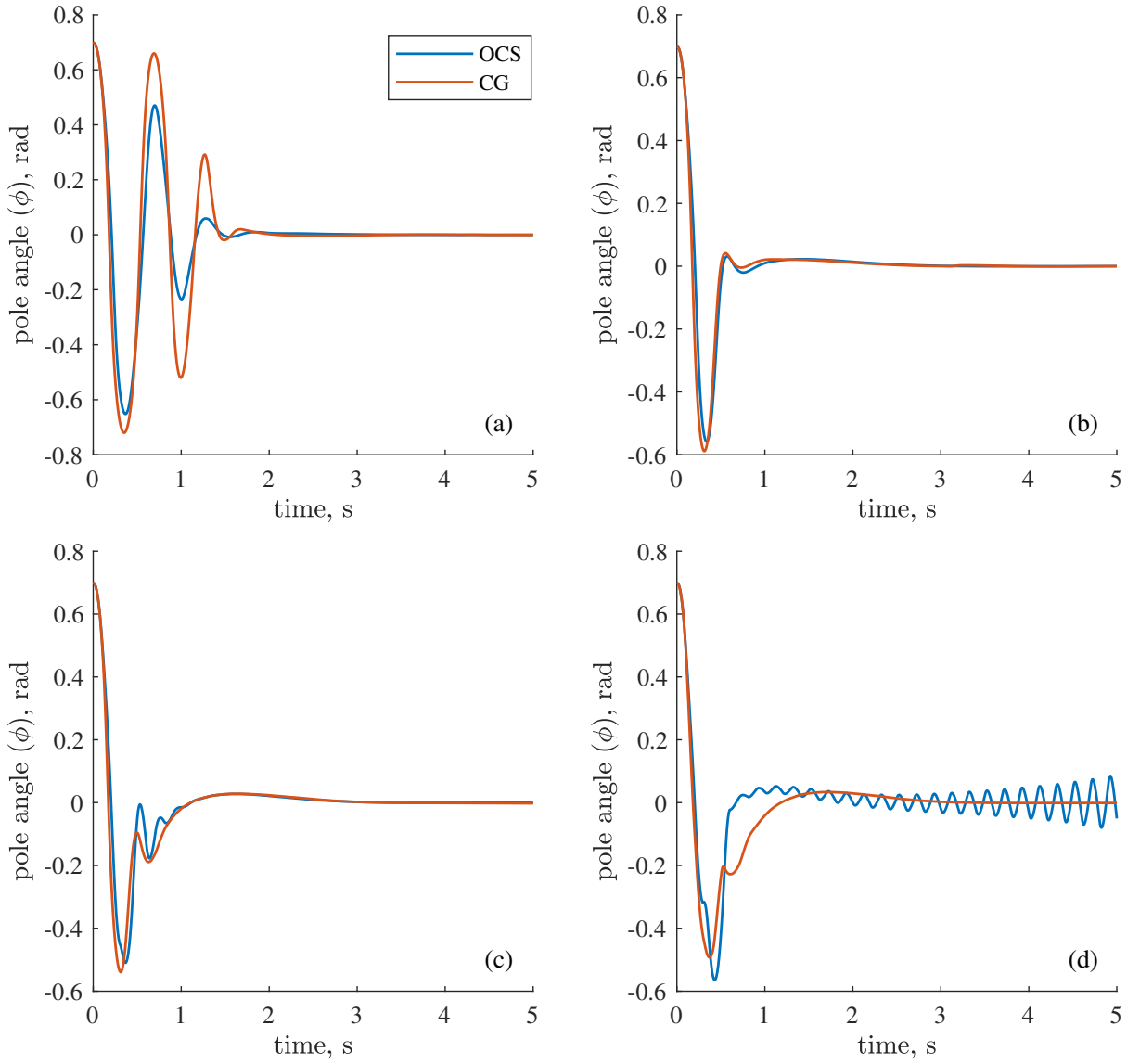

Figure 9.9 Receding horizon control of the cart-pole system using an OCS and CG policy with $N_h = 101$, $N_o = 10$, and varying error in pole length modelling error. In (a), (b), (c), and (d), the pole modelling error is $0.07m$, $0m$, $-0.07m$, and $-0.15m$, respectively.

should be intercepted, is set to 0.1. During training, the dots are initialized within the top and right 20% of the play area with radius $r^1, r^2 = 0.02m$. Their initial vertical velocity is sampled from a uniformly distributed random curve over the range $[0, 1]m/s$. Each dot randomly receives an initial horizontal velocity sampled from a normally distributed curve with standard deviation $0.1m/s$ and either a mean of $1m/s$ or $2m/s$. At the start of each trial, one dot is randomly selected to receive a horizontal velocity that results in it landing

within the region of $3\sigma_{target}$ ($[-0.3, 0.3]m$), which corresponds to an initial horizontal velocity with mean $1m/s$, and one dot receives a velocity with $2m/s$ that will cause it to overshoot the target. Each policy should then learn to recognize which dot will land within the 3σ range and intercept it. The interceptor is initialized with no velocity, a centered horizontal position, and a vertical position two times greater than its radius $r^{int} = 0.03m$.

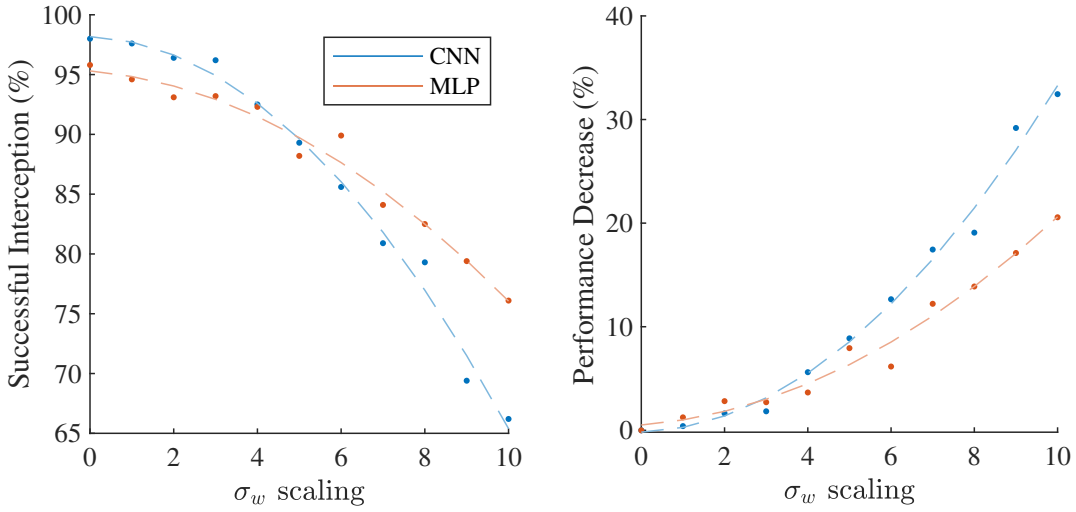

Figure 9.10 Dot interception using a 1D CNN and 2-layer MLP network with unmodelled process noise and the standard deviation $\sigma_w = diag\left(\{0.001, 0.01, 0.001, 0.01\}\right)$ scaled from 0 to 10.

Figure 9.10 shows the interception performance of the two policies given an increasing amount of unmodelled process noise. Each policy is trained with stochastic initial conditions and deterministic dynamics, and when tested in these conditions over 1000 trials, the CNN policy intercepted the correct dot 98% of time while the MLP policy intercepted the correct dot 95.8% of the time. This small discrepancy, and the occasional failure of both policies to intercept the dot, are likely due to training. However, both learn to intercept the correct dot almost all of the time. Adding process noise to the dots significantly changes these results. The process noise is added by scaling the base values $\sigma_{\underline{w}} = diag(\{0.001, 0.01, 0.001, 0.01\})$ from 1 to 10 (i.e. when scaled by 5 the standard deviation is $\sigma_{\underline{w}} = diag(\{0.005, 0.05, 0.005, 0.05\})$). Unexpectedly, the MLP policy proves more

robust to the process noise than the CNN policy as seen on the right side of Fig. 9.10. When the standard deviation is scaled by 10 and given as $\sigma_{\underline{w}} = diag(\{0.01, 0.1, 0.01, 0.1\})$, the MLP policy performs 20.6% worse while the CNN policy performs 32.4% worse. Note, the trend lines presented are quadratic fits given for comparison purposes. When the process noise is scaled by 10, the $3\sigma_w$ bounds on the dot and interceptor position process noise are within the same range as the radii of the dots themselves. This implies that, the process noise alone could cause the interceptor to miss regardless of the control inputs.

Within the cost function described in Sec. 8.2.1, a term is added with the purpose of enticing the interceptor to actively engage the dots by moving upward. As seen in Fig. 9.12, the CNN policy fails to move upward any considerable amount and intercepts the dot near its starting position. The trial depicted shows the interceptor move only slightly to intercept the dot, which would be optimal without the addition of the vertical position based component of the cost function. This moderate amount of motion can be observed over all trials with the CNN policy. It essentially learned to move the interceptor as little as possible. In comparison, the MLP network seen in Fig. 9.13 waits some time for the dot to fall and then actively engages it above the target. The more aggressive nature of the MLP policy could lend itself to performing better when the dots are subject to process noise as it may more actively track the dot to be intercepted.

9.5 Dot Interception with Process Noise

Another CNN policy of similar design to Sec. 9.4 is trained with not only random initial conditions but also stochastic dynamics. The CNN is extended to take in 11 time steps of data instead of the previous 5 and trained with the dots subjected to normally distributed zero mean process noise of standard deviation $\sigma_{\underline{w}} = diag(\{0.01, 0.1, 0.01, 0.1\})$. The environment characteristics remain identical to the previous section; most notably, the radii of the interceptor $r^{int} = 0.03m$ and dots $r^1, r^2 = 0.02m$ are of comparable size to the position process noise that the bodies are subject to. For the interceptor to intercept a dot, it must position itself within $0.05m$ of the dot being intercepted. Given the process noise standard

deviation, if both the vertical and horizontal position noises sampled when the interceptor is about to contact the dot lie on the $3\sigma_w$ bound with appropriate directions, then the interceptor and dot could move $2\sqrt{0.03^2 + 0.03^2} \approx 0.085m$ away from each other and cause the interceptor to miss.

Figure 9.11 shows percentages of interception and performance decrease of the CNN trained with fixed process noise over varying standard deviations. In this case, the CNN performs much better than the policies presented in Sec. 9.4. Taking in more time steps not only allows the CNN to analyze the mean trajectory of the dots better but also increases the overall size of the CNN, which increases its approximating capabilities. Given the fixed kernel size of 2, a CNN taking in 5 times steps of data performs 4 convolutions, while the CNN taking in 11 times steps performs 10. This increase in computations obviously affects the computation time and only minimally affects the performance with zero process noise. However, given the standard deviation $\sigma_w = diag(\{0.01, 0.1, 0.01, 0.1\})$, the larger CNN only performs 6.7% worse than the deterministic case as compared to the previous 32.4% deterioration.

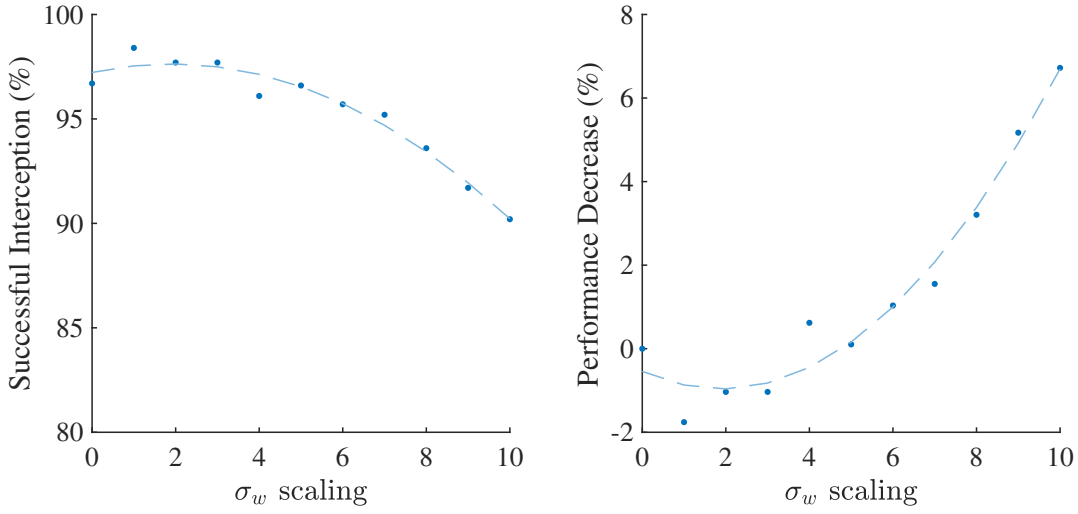

Figure 9.11 Dot interception using a 1D CNN trained with fixed process noise over trials with varying standard deviation $\sigma_w = diag\left(\{0.001, 0.01, 0.001, 0.01\}\right)$ scaled from 0 to 10.

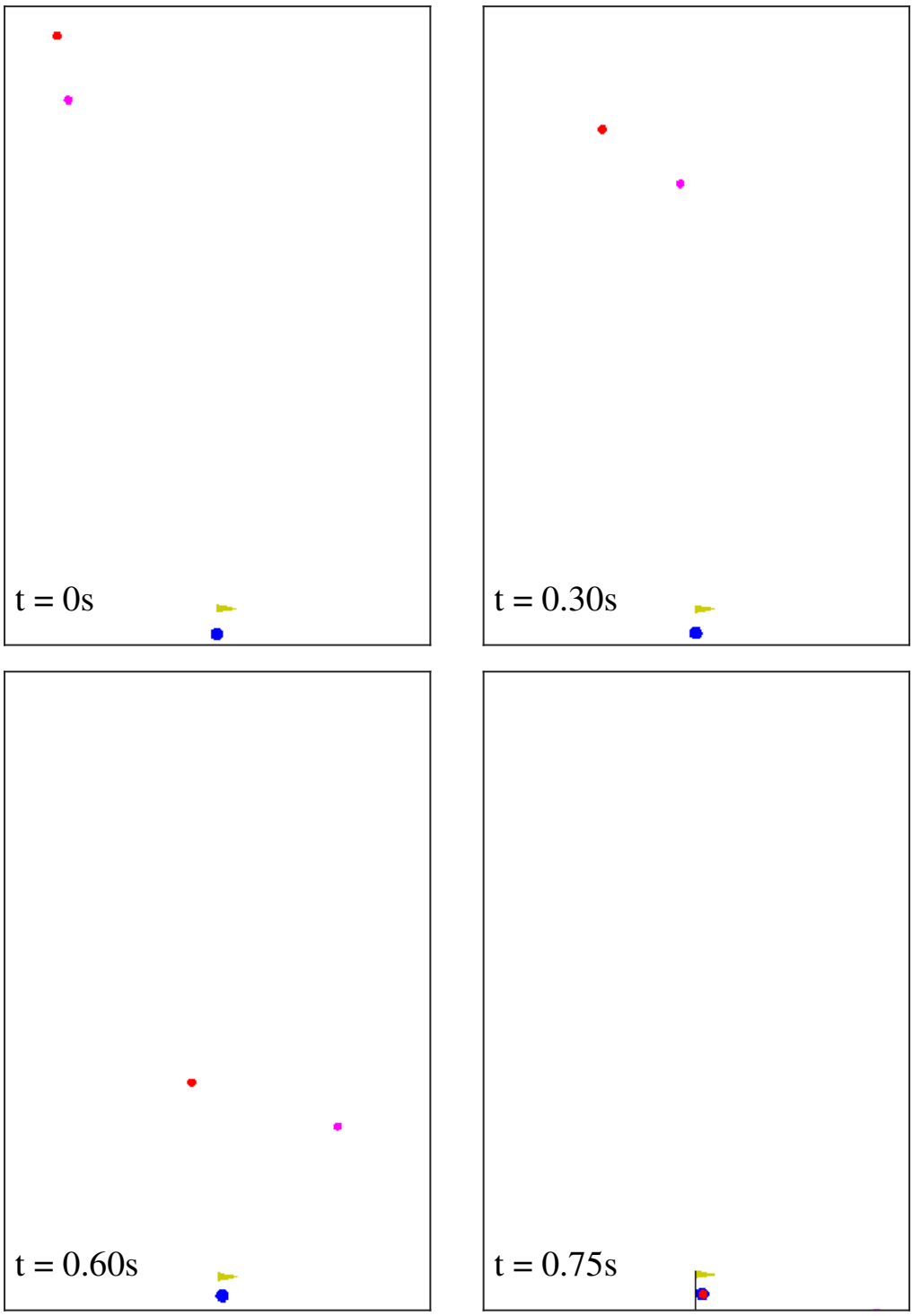

Figure 9.12 Sample trial of dot interception with no process noise using the 1D CNN policy that takes in 5 time steps of data.

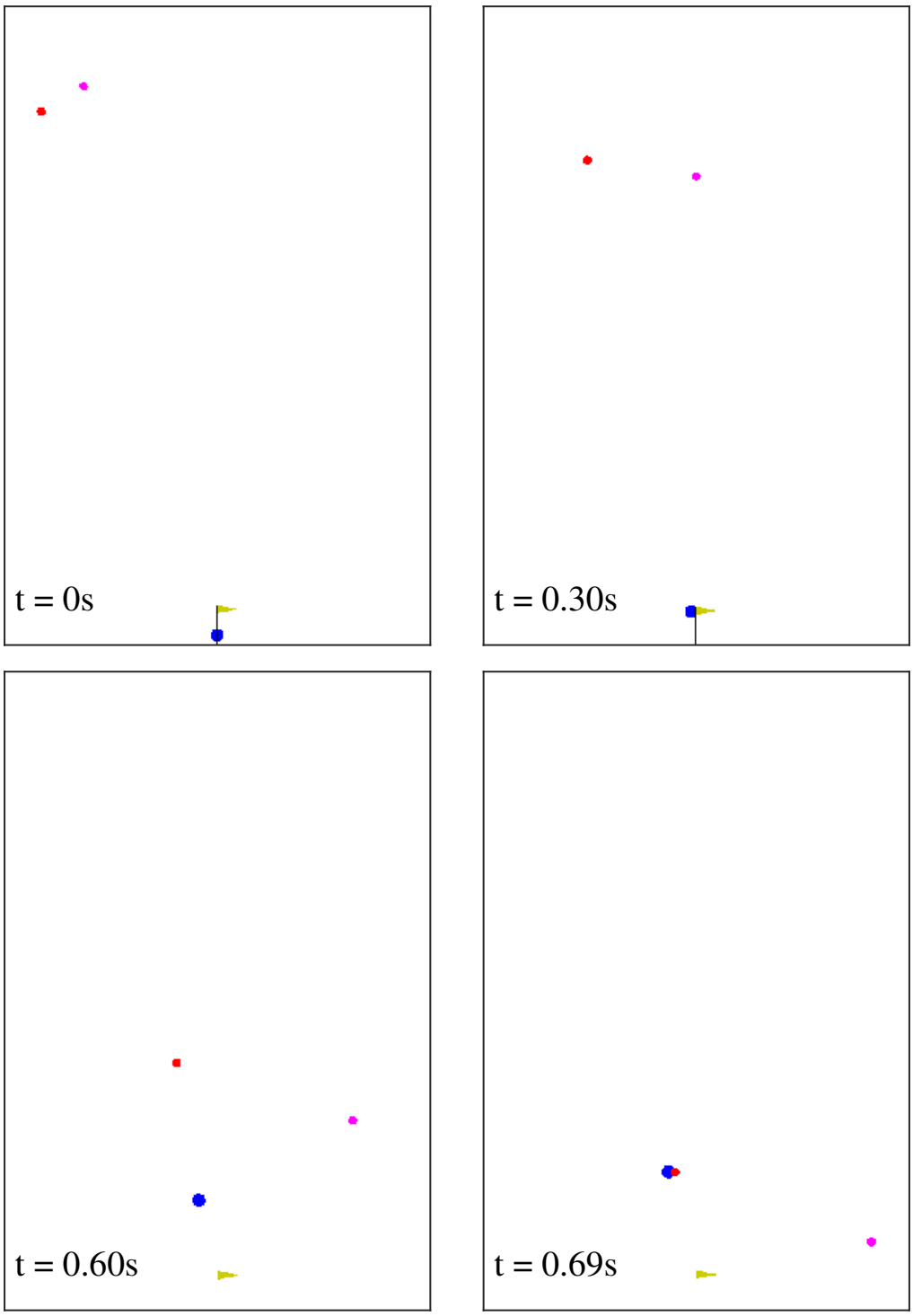

Figure 9.13 Sample trial of dot interception with no process noise using the 2-layer MLP policy.

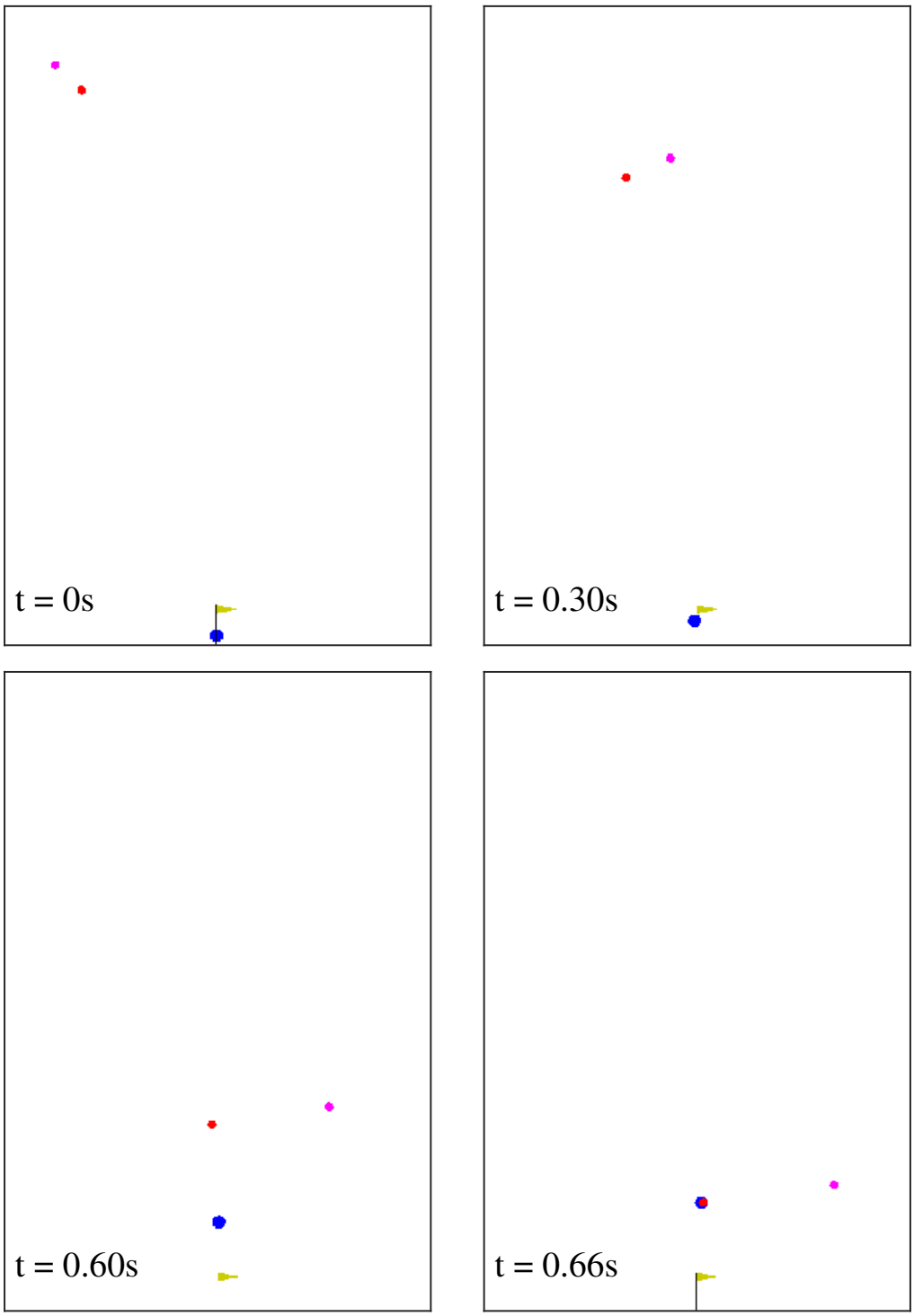

Figure 9.14 Sample trial of dot interception with process noise of standard deviation $\sigma_w = diag\left(\{0.01, 0.1, 0.01, 0.1\}\right)$ using the 1D CNN policy that takes in 11 time steps of data.

10 Conclusion

A model predictive control method is presented that encompasses not only receding horizon but also stationary horizon optimal control problems. The method, termed parameterized policy model predictive control (PPMPC), focuses on optimizing the parameters of a fixed structure policy, which includes conventional open-loop control sequences (OCSs) and state-feedback control laws. The control policy parameters are optimized with respect to a value function, which acts as a constrained cost function. Given the system dynamics, control policy, policy parameters, and an initial condition, the state and control trajectories can be computed over the optimization horizon. Evaluating a chosen cost function with the state and control trajectory produces the value of following the control policy from that initial condition. The policy parameters are then optimized to provide a minimal value. Note, not all policies can produce the true optimal solution given by an OCS policy, but solutions are considered optimal when the chosen policy is performing optimally. Since the dynamics and control policy are included in the value function, no functional constraints need to be placed on the problem unless there are some other desired outcomes like state bounds or closed-loop stability assurance. The control policies can then be determined offline, similar to explicit model predictive control, or online with new optimizations occurring every set number of simulation steps.

The PPMPC method resembles the deterministic policy gradient algorithm where a fixed policy, most commonly a deep neural network (NN), is used to optimize the output of another function that is being trained to reproduce the state-action values of a system. In this case, the policy is used to optimize a known value function with an analytical derivative, which enables closed-loop stability assurance for receding horizon problems given that the chosen control policy can replicate the optimal OCS trajectory to within some accuracy [26]. A common approach to optimizing state-feedback polices is to solve the OCS problem for a single initial condition or range of initial conditions, then perform a least-squares (LS) regression on the state and control trajectories with the desired state-feedback policy. As discussed

in [28], the LS approximation error has a large effect on the closed-loop performance of the policy. Assuming the chosen policy has nearly-universal or actual universal approximation capability, like an adequately sized NN or time-varying linear feedback gain (TG), the LS approach will produce an acceptable result. However, when considering policies with low approximation power, e.g. a constant linear feedback gain (CG), optimizing the policy parameters with respect to the value function directly, such as within the deterministic policy gradient algorithm, produces much better results in terms of value function optimality and subjective system performance.

Leveraging work from [26], it is shown that the infinite horizon optimal CG policy regulates nonlinear systems assuming that the system is stabilizable and the initial condition lies within a stabilizable region. Furthermore, it is shown that there exists some finite optimization horizon that produces a stabilizing feedback gain. It is well known that increasing the optimization horizon of an OCS policy produces "more stable" results. Assuming regulation is the objective, if more time steps are being considered via increasing the horizon length and the states are not approaching the origin, the cost will continue to increase. However, through implementation of the optimal OCS policy, it can be observed that the total cost converges to a constant finite value as the optimization horizon increases, assuming that there exists some stabilizing solution. The optimal CG policy acts in the same way; as the optimization horizon is increased, the feedback gain converges to a constant value and produces a constant cost.

Commonly, physical systems have control input limits, which are normally accounted for in conventional OCS optimal control problems by placing a functional constraint on the optimization. Linear optimal control methods, which inherently do not include nonlinear control bound constraints, can either be tuned in such a way that the control inputs produced do not violate the input limits or a saturation function can be placed on the linear controller, which will degrade its performance. By including the control boundaries within the policy and subsequently the value function, the policy parameters are optimized with respect to

those control bounds, along with the system dynamics and policy structure, at the expense of rendering linear state-feedback functions nonlinear in nature. Solving the optimal control problem with the control constraints intrinsically satisfied by the control policy negates the need for the use of Lagrange multipliers within the optimization; however, since even linear control policy architectures become nonlinear given a saturation function, special consideration must be given during optimization. The complexity of solving the optimal control problem with nonlinear functional constraints is not completely negated, but basic gradient-based or gradient-free optimizations can be used, given that the nonlinearities within the control policy are handled accordingly.

The value function derivative is presented for a generic cost function, dynamic system, and control policy. Given the structure of the derivative, comparing implementations of several control policies is straightforward. The derivative of each component, i.e. the cost function, system dynamics, and control policy, are separated such that altering one does not affect the others. Hence, optimizing a control policy simply requires inputting the related derivatives for that control policy and using them to evaluate the state and control trajectories. Within gradient-free optimization methods, the implementation is even more straightforward as all that is required is for the control policy to be implemented within the system model. Assuming a specific cost function, dynamic system, and control policy have been chosen, simplifications can be performed to decrease the computational burden. The dynamic system derivative $\frac{\partial X}{\partial U}$, in particular, can become quite large and computationally burdensome as both X and U grow with respect to the optimization horizon and the derivative is calculated using its own dynamic system given by Eq. 4, meaning the elements of the block matrix cannot be completely calculated in parallel like the control policy derivative $\frac{\partial U}{\partial X}$. Note, if the control policy contained some time dependent adapting component described by a dynamic system, for example, then the control policy derivative $\frac{\partial U}{\partial X}$ would also not be fully parallelizable.

Given a linear system and quadratic cost, the classical finite horizon linear quadratic regulator (FHLQR), a simplified derivative for optimal CG policies over finite horizons, and an

infinite horizon approximation with a CG policy are presented. Assuming the control policy is an OCS without any boundary constraints, i.e. $\mathcal{U} = \mathbb{R}^m$, the FHLQR is derived using the presented value function derivative for verification of the method. Instead of first defining Lagrange multipliers as is done with the Euler-Lagrange approach, a backwards look at the optimal control inputs reveals the matrix Riccati equation when using the value function derivative, which works similarly to the dynamic programming approach. An assumption is made about the solution to the Riccati equation, which then introduces the typical definition of the Lagrange multipliers; however, only the first Karush-Kuhn-Tucker condition is used to find this definition. A closed form solution for optimal CG policies over finite horizons was not determined, although a simplified value function derivative is given. Within the simplified derivative, it is clear that even within a linear-quadratic context, the optimal CG policy for a finite horizon relies on the initial condition. Even making an infinite horizon approximation, where the state and control trajectory are assumed to no longer be directly coupled, an iterative root finding algorithm must be used to find the optimal feedback gain.

The PPMPC method is used to regulate the cart-pole with various initial conditions and control policy architectures. Over a stationary horizon of 1 second with the continuous cart-pole dynamics model discretized at $\Delta t = 0.01s$, i.e. the horizon is comprised of 101 time steps, several control policies were implemented, and most produced similar results. The control policies used were an OCS, a CG, a TG, and a CG LS approximation of the optimal OCS. The control input weighting was low in comparison to the state weighting, which resulted in the optimal solution being aggressive and quickly regulating the system. The CG policy was able to almost replicate the true optimal OCS solution. Considering the control boundary saturation functions were active, i.e. the control limits were being reached, the power of directly optimizing the policy parameters is clearly shown as one solution, the CG, containing a total of 4 values to be computed, performed nearly as well as the OCS solution where each of the 100 control inputs was determined independently. This is even more evident in the receding horizon simulations where the CG solution not only performed

nearly identical to the OCS solution in the nominal case but also completely mitigated some modelling error in an off-nominal case.

The poor performance of the LS solution is clearly shown within the cart-pole simulations. A CG policy is trained using a LS regression and the optimal OCS state-control trajectory. Given the low approximation power of the CG policy, there is significant error within the LS regression. Since the CG policy is not able to accurately reproduce the OCS control trajectory, it cannot reproduce the OCS state trajectory. Additionally, the policy uses the states to calculate the control inputs, meaning when the OCS state trajectory is not reproduced, significantly more error propagates through the system. This phenomenon can be seen not only in the deterministic case but also in the stochastic case where several optimal state-control trajectories are used. In all of these cases, the direct CG solution performed significantly better.

Stationary time varying horizons are used to find optimal NN based policies within the dot intercept environment. Performance of a CNN policy and a MLP policy are compared given unmodelled process noise. It was assumed that the CNN layer would better handle noise as it brings in a time history of data that could be used to better deduce the mean trajectory. However, it was shown, for a particular case, that the MLP policy, which only observes the current state, was more robust to the unmodelled process noise. The MLP solution was much more proactive in intercepting the dots as it actively moved the interceptor upward. The CNN solution seemed to input the least amount of control power possible as it barely moved the interceptor. This discrepancy could be the reason that the MLP solution was able to track the dots subject to process noise as it was not only more aggressive in maneuvering but also shortened the distance that the dot had to drift away from its nominal trajectory. An additional CNN that brought in over twice the number of time steps of data, in comparison to the first CNN, was trained with the process noise present. This larger CNN maneuvered the dot in a similar manner to the deterministic MLP solution and performed significantly better when subjected to process noise. The increase in performance is likely from a combination

of training the policy in a stochastic environment and the increased approximation power of the larger convolutional layer.

The main purpose within the dot intercept environment is not necessarily to show that a NN can be trained to intercept a body in a parabolic trajectory, e.g. an enemy missile. The purpose is to show how discrete decisions can be made by designing a continuous differentiable cost function. Typically, decision problems are solved using RL algorithms where a discrete reward is given when the correct choice is made and the critic function approximates the reward signal in a differentiable way, which allows the policy to be updated via a gradient. However, this process does not lead to any stability proofs being available as the approximation of the reward function has unknown characteristics. It is assumed within optimal control proofs that the cost function is radially unbounded and non-decreasing, conditions that the cost function used within the dot intercept environment does not meet. However, some approximations could be made to argue that within some local region the cost function is radially unbounded and non-decreasing, and local stability proofs could potentially be reasoned. Assuring that control policies will make correct discrete decisions is vitally important to the mechanization of the world.

The PPMPC method was assessed by regulating the cart-pole system. Analyzing reference trajectory tracking capabilities of the method and training complex nonlinear policies to track complex trajectories would extend the work provided here. The compounding error effect seen within LS based policies was explored empirically but no definite proofs were given, or have been found, that show the LS solution performs only as well as the direct solution. The NN based policies within the dot intercept environment were trained under a narrow scope of initial conditions. Ideally, the interceptor would be able to make decisions given that the dots start anywhere in the play area and the target initializes anywhere along the bottom. Training the policies given those conditions produced limited results. Additionally, training a policy that could control multiple interceptors working cooperatively could show significant increase the performance of the algorithm.

REFERENCES

[1] Mohamed, H., Negm, A., Zaharan, M., and Saavedra, O., "Assessment of Artificial Neural Network for Bathymetry Estimation Using High Resolution Satellite Imagery in Shallow Lakes: Case Study El Burullus Lake," *International Water Technology Journal*, Vol. 5, No. 4, 2015, pp. 248–259.

[2] Saha, S., "A Comprehensive Guide to Convolutional Neural Networks - the ELI5 way," , 2018. URL `https://towardsdatascience.com/a-comprehensive-guide-to-...convolutional-neural-networks-the-eli5-way-3bd2b1164a53`.

[3] Richalet, J., Rault, A., Testud, L., and Papon, J., "Model Predictive Heurisitc Control: Applications to Industrial Processes," *Automatica*, Vol. 14, 1978, pp. 413–428. https://doi.org/10.1016/0005-1098(78)90001-8.

[4] Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, L., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D., "Mastering the Game of Go Without Human Knowledge," *Nature*, Vol. 550, 2017, pp. 354–359. https://doi.org/10.1038/nature24270.

[5] Lewis, F., Vrabie, D., and Syrmos, V., *Optimal Control*, 3rd ed., John Wiley & Sons, Inc., 2012.

[6] Burghart, J., "A Technique for Suboptimal Feedback Control of Nonlinear Systems," *Transactions on Automatic Control*, Vol. 14, No. 5, 1969, pp. 530–533. https://doi.org/10.1109/TAC.1969.1099251.

[7] Heydari, A., and Balakrishnan, S., "Closed-form Solution to Finite-horizon Suboptimal Control of Nonlinear Systems," *International Journal of Robust and Nonlinear Control*, Vol. 25, 2020, pp. 2687–2704. https://doi.org/10.1002/rnc.3222.

[8] Nanavati, R. V., Kumar, S. R., and Maity, A., "Suboptimal Closed-form Feedback Control of Input-affine Non-linear Systems," *Control Theory & Applications*, Vol. 14, No. 15, 2020, pp. 2064–2075. https://doi.org/10.1049/iet-cta.2019.1249.

[9] Misra, G., and Bai, X., "Output-Feedback Stochastic Model Predictive Control for Glideslope Tracking During Aircraft Carrier Landing," *Journal of Guidance, Control, and Dynamics*, Vol. 42, No. 9, 2019. https://doi.org/10.2514/1.G004160.

[10] Wilson, E., and Prazenica, R., "Autonomous Autorotation of a Multirotor Vehicle Using Output Feedback Model Predictive Control with Online Learning," *Advances in Feedback Control Architectures for Autonomous Systems I*, AIAA, 2021. https://doi.org/10.2514/6.2021-0379.

[11] Alessio, A., and Bemporad, A., "A Survey on Explicit Model Predictive Control," *Nonlinear Model Predictive Control. Lecture Notes in Control and Information Sciences*, Vol. 384, edited by L. Magni, D. Martino, and F. Allgöwer, Springer, Berlin, 2009, pp. 345–369. https://doi.org/10.1007/978-3-642-01094-1_29.

[12] Rawlings, J. B., and Mayne, D. Q., *Model Predictive Control: Theory and Design*, Nob Hill Publishing, 2009.

[13] Hovland, S., Gravdahl, T., and Willcox, K. E., "Explicit Model Predictive Control for Large-Scale Systems via Model Reduction," *Journal of Guidance, Control, and Dynamics*, Vol. 31, No. 4, 2008. https://doi.org/10.2514/1.33079.

[14] Chen, J., Chen, Y., Tong, L., Peng, L., and Kang, Y., "A Backpropagation Neural Network-Based Explicit Model Predictive Control for DC-DC Converters With High Switching Frequency," *Journal of Emerging and Selected Topics in Power Electronics*, Vol. 8, No. 3, 2020, pp. 2124–2142.

[15] Suykens, J., Vandewalle, J., and Moor, B. D., "Opitmal Control by Least Squares

Support Vector Machines," *Neural Networks*, Vol. 14, 2001, pp. 23–35. https://doi.org/10.1016/S0893-6080(00)00077-0.

[16] Sánchez-Sánchez, C., and Izzo, D., "Real-Time Optimal Control via Deep Neural Networks: Study on Landing Problems," *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 5, 2018. https://doi.org/10.2514/1.G002357.

[17] Chen, S., Saulnier, K., Atanasov, N., Lee, D., Kumar, V., Pappas, G., and Morari, M., "Approximating Explicit Model Predictive Control Using Constrained Neural Networks," *AACC, American Control Conference*, 2018.

[18] Sutton, R., and Barto, A., *Reinforcement Learning: An Introduction*, 2nd ed., MIT Press, 2018.

[19] Eisenberg, B. R., and Sage, A. P., "Closed Loop Optimization of Fixed Configuration Systems," *International Journal of Control*, Vol. 3, 1966, pp. 183–194. https://doi.org/10.1080/00207176608921377.

[20] Kleinman, D. L., and Athans, M., "The Design of Suboptimal Linear Time-Varying Systems," *IEEE Transactions on Automatic Control*, Vol. AC-13, 1968, pp. 150–159. https://doi.org/10.1109/TAC.1968.1098852.

[21] Colombo, R., Gennari, F., Annem, V., Rajendran, P., Thakar, S., Bascetta, L., and Gupta, S., "Parameterized Model Predictive Control of a Nonholonomic Mobile Manipulator: A Terminal Constraint-Free Approach," *IEEE 15th International Conference on Automation Science and Engineering*, 2019. https://doi.org/10.1109/COASE.2019.8843088.

[22] Parisini, T., and Zoppoli, R., "Multi-Layer Neural Networks for the Optimal Control of Nonlinear Dynamic Systems," *IFAC Design Methods of Control Systems*, 1991, pp. 393–398.

[23] Parisini, T., and Zoppoli, R., "Neural Networks for Feedback Feedforward Nonlinear Control Systems," *IEEE Transactions of Automatic Control*, Vol. 5, No. 3, 1994, pp. 436–449. https://doi.org/10.1109/72.286914.

[24] Parisini, T., and Zoppoli, R., "A Receding-horizon Regulator for Nonlinear Systems and a Neural Approximation," *Automatica*, Vol. 31, No. 10, 1995, pp. 1443–1451. https://doi.org/10.1016/0005-1098(95)00044-W.

[25] Zoppoli, R., and Parisini, T., "Neural Approximations for Finite- and Infinite-Horizon Optimal Control," *Neural Systems for Control*, edited by O. Omidvar and D. L. Elliot, Academic Press, 1997, Chap. 12, pp. 317–351.

[26] Parisini, T., and Zoppoli, R., "Nonlinear Stabilization by Receding-Horizon Neural Regulators," *International Journal of Control*, Vol. 70, 1998, pp. 341–362. https://doi.org/10.1080/002071798222271.

[27] Pin, G., Filippo, M., Pellegrino, F., Fenu, G., and Parisini, T., "Approximate Model Predictive Control Laws for Constrained Nonlinear Discrete-Time Systems: Analysis and Offline Design," *International Journal of Control*, Vol. 86, No. 5, 2013, pp. 804–820. https://doi.org/10.1080/00207179.2012.762121.

[28] Zoppoli, R., Sanguineti, M., Gnecco, G., and Parisini, T., *Neural Approximations for Optimal Control and Decision*, 1st ed., Springer, 2020.

[29] Bellman, R., "Dynamic Programming," *Science*, Vol. 153, No. 3731, 1966, pp. 34–37. https://doi.org/10.1126/science.153.3731.34.

[30] Zhou, Y., van Kampen, E.-J., and Chu, Q., "Incremental Approximate Dynamic Programming for Nonlinear Adaptive Tracking Control with Partial Observability," *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 12, 2018. https://doi.org/10.2514/1.G003472.

[31] Bertsekas, D., *Dynamics Programming and Optimal Control*, Vol. 1, Athena Scientific, 2017.

[32] Bertsekas, D., *Dynamics Programming and Opitmal Control*, Vol. 2, Athena Scientific, 2012.

[33] Watkins, C., "Learning From Delayed Rewards," Ph.D. thesis, King's College, 1989.

[34] Balakrishnan, S., and Biega, V., "Adaptive-Critic-Based Neural Networks for Aircraft Optimal Control," *Journal of Guidance, Control, and Dynamics*, Vol. 19, No. 4, 1996. https://doi.org/10.2514/3.21715.

[35] Farshidian, F., Hoeller, D., and Hutter, M., "Deep Value Model Predictive Control," , 2019.

[36] Saerens, M., and Soquet, A., "Neural Controller Based on Back-Propagation Algorithm," *IEE Proceedings-F, Radar and Signal Processing*, Vol. 138, 1991, pp. 55–62.

[37] Kingma, D. P., and Ba, J., "ADAM: A Method for Stochastic Optimization," *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[38] Holland, J., *Adaptation in Natural and Artificial Systems*, 1st ed., University of Michigan Press, 1975.

[39] Affenzeller, M., Winkler, S., Wagner, S., and Beham, A., *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications*, 1st ed., CRC Press/Taylor & Francis Group, 2009.

[40] Roodschild, M., nas, J. G. S., and Will, A., "A New Approach For the Vanishing Gradient Problem on Sigmoid Activation," *Progress in Artificial Intelligence*, Vol. 9, 2020, pp. 351–360. https://doi.org/10.1007/s13748-020-00218-y.

[41] Park, J., and Sandberg, I., "Universal Approximation Using Radial-Basis-Function Networks," *Neural Computation*, Vol. 3, No. 2, 1991, pp. 246–257. https://doi.org/10.1162/neco.1991.3.2.246.

[42] McCulloch, W., and Pitts, W., "A Logical Calculus of the Ideas Immanent in Nervous Activity," *Bulletin of Mathematical Biophysics*, Vol. 5, 1943, pp. 115–133. https://doi.org/10.1007/BF02478259.

[43] Kim, D., "Normalization Methods for Input and Output Vector in Backpropagation Neural Networks," *International Journal of Computer Mathematics*, Vol. 71, No. 2, 1999, pp. 161–171. https://doi.org/10.1080/00207169908804800.

[44] Ramachandran, P., Zoph, B., and Le, Q. V., "Searching for Activation Functions," , 2019. ArXiv:1710.05941v2.

[45] Kang, Y., Chen, S., Weng, X., and Cao, Y., "Deep Convolutional Identifier for Dynamic Modeling and Adaptive Control of Unmanned Helicopter," *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 30, No. 2, 2019. https://doi.org/10.1109/TNNLS.2018.2844173.

[46] Ince, T., Kiranyaz, S., Eren, L., Askar, M., and Gabbouj, M., "Real-Time Motor Fault Detection by 1-D Convolutional Neural Networks," *IEEE Transactions on Industrial Electronics*, Vol. 63, 2016, pp. 7067–7075. https://doi.org/10.1109/TIE.2016.2582729.

[47] Chen, C., and Shaw, L., "On Receding Horizon Feedback Control," *Automatica*, Vol. 18, No. 3, 1982, pp. 349–352. https://doi.org/10.1016/0005-1098(82)90096-6.

[48] Mayne, D., and Michalska, H., "Receding Horizon Control of Nonlinear Systems," *IEEE Transactions of Automatic Control*, Vol. 35, No. 7, 1990, pp. 814–824. https://doi.org/10.1016/0005-1098(82)90096-6.

[49] Keerthi, S., and Gilbert, E., "Optimal Infinite-Horizon Feedback Laws for a General Class of Constrained Discrete-Time Systems: Stability and Moving-Horizon Approximations," *Journal of Optimization Theory and Applications*, Vol. 57, No. 2, 1988, p. 265–293. https://doi.org/10.1007/BF00938540.

[50] Grüne, L., and Pannek, J., *Nonlinear Model Predictive Control: Theory and Algorithms*, 2nd ed., Springer, 2017. https://doi.org/10.1007/978-3-319-46024-6.

[51] Michalska, H., and Mayne, D., "Robust Receding Horizon Control of Constrained Nonlinear Systems," *IEEE Transactions of Automatic Control*, Vol. 38, No. 11, 1993, pp. 1623–1633. https://doi.org/10.1109/9.262032.

[52] Dontchev, A., Kolmanovsky, I., Krastanov, M., Veliov, V., and Vuong, P., "Approximating Optimal Finite Horizon Feedback by Model Predictive Control," *Sytems & Control Letters*, Vol. 139, No. 104666, 2020. https://doi.org/10.1016/j.sysconle.2020.104666.

[53] Dalamagkidis, K., Valavanis, K., and Piegl, L., "Nonlinear Model Predictive Control with Neural Network Optimization for Autonomous Autorotation of Small Unmanned Helicopters," *IEEE Transactions on Control Systems Technology*, Vol. 19, No. 4, 2010, pp. 818–831. https://doi.org/10.1109/TCST.2010.2054092.

[54] Crassidis, J., and Junkins, J., *Opitmal Estimation of Dynamic Systems*, 2nd ed., CRC Press/ Taylor & Francis Group, 2012.

[55] Glück, T., Eder, A., and Kugi, A., "Swing-up Control of a Triple Pendulum on a Cart With Experimental Validation," *Automatica*, Vol. 49, No. 3, 2013, pp. 801–808. https://doi.org/10.1016/j.automatica.2012.12.006.

[56] Wilson, E., and Prazenica, R., "Autonomous Autorotation of Tilt Rotor Aircraft Using Nonlinear Model Predictive Control," *UAS Guidance, Navigation, and Control*, AIAA, 2020. https://doi.org/10.2514/6.2020-1488.

PUBLICATIONS

- Wilson, E and Prazenica, R. "*Autonomous Autorotation of Tilt Rotor Aircraft Using Nonlinear Model Predictive Control*". UAS Guidance, Navigation, and Control, SciTech 2020. DOI: 10.2514/6.2020-1488

- Wilson, E and Prazenica, R. "*Autonomous Autorotation of a Multirotor Vehicle Using Output Feedback Model Predictive Control with Online Learning*". Advances in Feedback Control Architectures for Autonomous Systems I, SciTech 2021. DOI: 10.2514/6.2021-0379

- Wang, J., Wilson, E., and Velasquez A. "*Consensus-Based Value Iteration for Multiagent Cooperative Control*" IEEE Conference on Decision and Control, 2021. DOI: 10.1109/CDC45484.2021.9682831

- Wilson, E and Prazenica, R. "*Fixed Structure Policy Based Model Predictive Control*". Guidance and Control Architectures for Autonomous Systems VI, SciTech 2022. DOI: 10.2514/6.2022-1378

- IN REVIEW: Wilson, E and Prazenica, R. "*Fixed Structure Policy Based Model Predictive Control*". AIAA Journal of Guidance, Control, and Dynamics.

APPENDIX A - System Derivative

Consider the discrete nonlinear dynamics function

$$\underline{x}_{k+1} = f(\underline{x}_k, \underline{u}_k) \tag{1}$$

where $\underline{x}_k \in \mathbb{R}^n$ and $\underline{u}_k \in \mathbb{R}^m$ are the system state and control input at time step $k \in \mathbb{N}$, respectively. A nonlinear feedback control law $\mu_\theta : \mathbb{R}^p \times \mathbb{R}^n \mapsto \mathbb{R}^m$ is parameterized by $\theta \in \mathbb{R}^p$ and maps the current state to a control input: $\underline{u}_k = \mu_\theta(\underline{x}_k)$. The derivative of the dynamics with respect to this control parameter comes from two components: a change in the state and a change in the control inputs. The derivative can be written as:

$$\frac{\partial \underline{x}_{k+1}}{\partial \theta} = \frac{\partial f(\underline{x}_k, \underline{u}_k)}{\partial \underline{x}_k} \frac{\partial \underline{x}_k}{\partial \theta} + \frac{\partial f(\underline{x}_k, \underline{u}_k)}{\partial \underline{u}_k} \frac{\partial \underline{u}_k}{\partial \theta} \tag{2}$$

Now, the derivative of the control input is explored further. Taking the derivative of the control policy gives:

$$\frac{\partial \underline{u}_k}{\partial \theta} = \frac{\partial \mu_\theta(\underline{x}_k)}{\partial \theta} + \frac{\partial \mu_\theta(\underline{x}_k)}{\partial \underline{x}_k} \frac{\partial \underline{x}_k}{\partial \theta} \tag{3}$$

Substituting this back into (2) and rearranging yields:

$$\frac{\partial \underline{x}_{k+1}}{\partial \theta} = \left(\frac{\partial f(\underline{x}_k, \underline{u}_k)}{\partial \underline{x}_k} + \frac{\partial f(\underline{x}_k, \underline{u}_k)}{\partial \underline{u}_k} \frac{\partial \mu_\theta(\underline{x}_k)}{\partial \underline{x}_k} \right) \frac{\partial \underline{x}_k}{\partial \theta} + \frac{\partial f(\underline{x}_k, \underline{u}_k)}{\partial \underline{u}_k} \frac{\partial \mu_\theta(\underline{x}_k)}{\partial \theta} \tag{4}$$

APPENDIX B - Constant Gain Value Function Derivative

Given the discrete linear system

$$\underline{x}_{k+1} = A\underline{x}_k + Bu_k \tag{5}$$

where $\underline{x}_k \in \mathbb{R}^n$ is the system state at time step k, $A \in \mathbb{R}^{n\times n}$ is the dynamics matrix, $B \in \mathbb{R}^{n\times 1}$ is the control matrix, and $u_k \in \mathbb{R}$ is the control input, the goal is to find a constant gain feedback control law that minimizes the cost function

$$J = \frac{1}{2}\underline{x}_{N_h}{}^T Q \underline{x}_{N_h} + \frac{1}{2}\sum_{k=0}^{N_h-1} \underline{x}_k{}^T Q \underline{x}_k + R u_k{}^2 \tag{6}$$

subject to the system dynamics and $U_k = \theta \underline{x}_k$, where $\theta \in \mathbb{R}^{1\times n}$ is a vector of unknown parameters, $Q \in \mathbb{R}^{n\times n}_{\geq 0}$ is the state weighting matrix, and $R \in \mathbb{R}_{>0}$ is the control weighting parameter. The condition of first order optimality requires

$$\frac{dV^\mu(\underline{x}_0)}{d\theta} = 0^{1\times n} \tag{7}$$

where the derivative is given by:

$$\frac{dV^\mu(\underline{x}_0)}{d\theta} = \frac{\partial V^\mu(\underline{x}_0)}{\partial X}\frac{\partial X}{\partial \theta} + \frac{\partial V^\mu(\underline{x}_0)}{\partial U}\left(\frac{\partial U}{\partial \theta} + \frac{\partial U}{\partial X}\frac{\partial X}{\partial \theta}\right) \tag{8}$$

The required components for computing this derivative are given in section 7.2.

Now, the derivative will be evaluated term by term. First, consider the control derivative

portion:

$$\frac{\partial U}{\partial \theta} + \frac{\partial U}{\partial X}\frac{\partial X}{\partial \theta} = \begin{bmatrix} \underline{x}_0{}^T \\ \underline{x}_1{}^T + \theta B \underline{x}_0{}^T \\ \underline{x}_2{}^T + \theta\left(\bar{A}B\underline{x}_0{}^T + B\underline{x}_1{}^T\right) \\ \vdots \\ \underline{x}_{N_h-1}^T + \theta\sum_{i=0}^{N_h-2}\bar{A}^i B\underline{x}_{N_h-2-i}^T \end{bmatrix} \tag{9}$$

Multiplying this term to the value-control derivative

$$\frac{\partial V^\mu(\underline{x}_0)}{\partial U}\frac{dU}{d\theta} = u_0 R\underline{x}_0{}^T + u_1 R\left(\underline{x}_1{}^T + \theta B\underline{x}_0{}^T\right) + \cdots + U_{N_h-1}R\left(\underline{x}_{N_h-1}^T + \theta\sum_{i=0}^{N_h-2}\bar{A}^i B\underline{x}_{N_h-2-i}^T\right) \tag{10}$$

The second term of the derivative can now be expanded as

$$\frac{\partial V^\mu(\underline{x}_0)}{\partial X}\frac{\partial X}{\partial \theta} = \underline{x}_1{}^T QB\underline{x}_0{}^T + \underline{x}_2{}^T Q\left(\bar{A}B\underline{x}_0{}^T + B\underline{x}_1{}^T\right) + \cdots + \underline{x}_{N_h}Q\sum_{i=0}^{N-1}\bar{A}^i B\underline{x}_{N_h-1-i}^T \tag{11}$$

These two terms can now be added to get the total derivative

$$\frac{dV^\mu(\underline{x}_0)}{d\theta} = \sum_{i=0}^{N_h-1}\underline{x}_i{}^T\left(\theta^T R + \bar{A}^T QB\right)\underline{x}_i{}^T + \sum_{i=0}^{N_h-2}\sum_{j=0}^{i-1}\underline{x}_i{}^T\left(\theta^T R\theta + \underline{A}^T Q\bar{A}\right)\bar{A}^{i-j-1}B\underline{x}_j{}^T \tag{12}$$

Lastly, this can be rewritten in terms of matrix products as:

$$\frac{dV^{\mu}(\underline{x}_0)}{d\theta} = \begin{bmatrix} \underline{x}_0{}^T & \cdots & \underline{x}_{N_h-1}^T \end{bmatrix} \left(\begin{bmatrix} \theta^T R + \bar{A}^T Q B & \cdots & 0^n \\ \vdots & \ddots & \vdots \\ 0^n & \cdots & \theta^T R + \bar{A}^T Q B \end{bmatrix} + \right.$$

$$\left. \begin{bmatrix} \theta^T R\theta + \bar{A}^T Q\bar{A} & \cdots & 0^{n\times n} \\ \vdots & \ddots & \vdots \\ 0^{n\times n} & \cdots & \theta^T R\theta + \bar{A}^T Q\bar{A} \end{bmatrix} \begin{bmatrix} 0^n & 0^n & \cdots & 0^n & 0^n \\ B & 0^n & \cdots & 0^n & 0^n \\ \bar{A}B & B & \cdots & 0^n & 0^n \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \bar{A}^{N_h-2}B & \bar{A}^{N_h-3}B & \cdots & B & 0^n \end{bmatrix} \right) \begin{bmatrix} \underline{x}_0{}^T \\ \vdots \\ \underline{x}_{N_h-1}^T \end{bmatrix} \quad (13)$$

APPENDIX C - Cart-pole Derivative

Here, the cart-pole dynamics derivatives are presented. First, consider the Euler discretization:

$$\frac{\partial f(\underline{x}_k, \underline{u}_k)}{\partial \underline{x}_k} = I_n + \Delta t \frac{\partial f_c(\underline{x}_k, \underline{u}_k)}{\partial \underline{x}_k} \tag{14}$$

$$\frac{\partial f(\underline{x}_k, \underline{u}_k)}{\partial \underline{u}_k} = \Delta t \frac{\partial f_c(\underline{x}_k, \underline{u}_k)}{\partial \underline{u}_k} \tag{15}$$

The continuous time state space model of the cart-pole dynamics is:

$$f_c(\underline{x}, \underline{u}) = \begin{bmatrix} x_2 & \ddot{y} & x_4 & \ddot{\phi} \end{bmatrix}^T \tag{16}$$

where $\ddot{y}$ and $\ddot{\phi}$ are given by Eqs. (8.1) and (8.2), respectively. The Jacobian of the continuous time model is given by:

$$\frac{\partial f_c(\underline{x}_k, \underline{u}_k)}{\partial \underline{x}_k} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{\partial \ddot{y}}{\partial x_3} & \frac{\partial \ddot{y}}{\partial x_4} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{\partial \ddot{\phi}}{\partial x_3} & \frac{\partial \ddot{\phi}}{\partial x_4} \end{bmatrix} \tag{17}$$

where the partial derivative entries are expanded as follows. Let $\ddot{y}(\underline{x}, u) = \frac{\alpha(\underline{x}, u)}{\beta(\underline{x})}$, then

$$\alpha(\underline{x}, u) = u - m \sin x_3 \left(L {x_4}^2 + g \cos x_3 \right) \tag{18}$$

$$\beta(\underline{x}) = M + m \left(1 + \cos^2 x_3 \right) \tag{19}$$

and

$$\frac{\partial \ddot{y}}{\partial x_3} = \frac{\frac{\partial \alpha(\underline{x}, u)}{\partial x_3} \beta(\underline{x}) - \alpha(\underline{x}, u) \frac{\partial \beta(\underline{x})}{\partial x_3}}{\beta(\underline{x})^2} \tag{20}$$

These two partial derivatives are given as:

$$\frac{\partial \alpha(\underline{x}, u)}{\partial x_3} = -m\left(L{x_4}^2 + g\cos x_3\right)\cos x_3 + mg\sin^2 x_3 \tag{21}$$

$$\frac{\partial \beta(\underline{x})}{\partial x_3} = -2m\cos x_3 \sin x_3 \tag{22}$$

The other derivatives needed for the Jacobian are then given by:

$$\frac{\partial \ddot{y}}{\partial x_4} = \frac{1}{\beta(\underline{x})}\left(2Lx_4\left(u - m\sin x_3\right)\right) \tag{23}$$

$$\frac{\partial \ddot{\phi}}{\partial x_3} = \frac{1}{L}\left(\frac{\partial \ddot{y}}{\partial x_3}\cos x_3 - \ddot{y}\sin x_3 + g\cos x_3\right) \tag{24}$$

$$\frac{\partial \ddot{\phi}}{\partial x_4} = \frac{1}{L}\frac{\partial \ddot{y}}{\partial x_4}\cos x_3 \tag{25}$$

Lastly, the control derivative is given by:

$$\frac{\partial f_c(\underline{x}, u)}{\partial u} = \begin{bmatrix} 0 & \dfrac{\partial \ddot{y}}{\partial u} & 0 & \dfrac{1}{L}\dfrac{\partial \ddot{y}}{\partial u}\cos x_3 \end{bmatrix}^T \tag{26}$$

with

$$\frac{\partial \ddot{y}}{\partial u} = \frac{1}{\beta(\underline{x})}(L{x_4}^2 + g\cos x_3) \tag{27}$$

APPENDIX D - Dot Intercept Derivative

The dot intercept environment is described by simple parabolic motion with ground collision detection and process noise. The derivative of the $k+1$ state with respect to the k state can be written as

$$\frac{\partial \underline{x}_{k+1}^i}{\partial \underline{x}_k^i} = I^4 + \frac{\partial f(\underline{x}_k^i, \underline{u}_k^i)}{\partial \underline{x}_k^i} + \frac{\partial \sigma^i(\underline{x}_k^i, \underline{u}_k^i)}{\partial \underline{x}_k^i}\sqrt{\Delta t}\underline{w}_k^i \tag{28}$$

Note, the component $\frac{\partial \sigma^i(\underline{x}_k^i, \underline{u}_k^i)}{\partial \underline{x}_k^i}$ is not a trivial computation as the standard deviation $\sigma^i(\underline{x}_k^i, \underline{u}_k^i)$ is matrix valued. However, considering the case where the standard deviation is either a constant value or zero, dependent on ground collision, the system state has no effect on the process noise, and the last term is simply evaluated to be zero:

$$\frac{\partial \sigma^i(\underline{x}_k^i, \underline{u}_k^i)}{\partial \underline{x}_k^i}\sqrt{\Delta t}\underline{w}_k^i = 0^{4\times 4} \tag{29}$$

The system's mean dynamic derivative can be written as

$$\frac{\partial f(\underline{x}^i, \underline{u}^i)}{\partial \underline{x}_k^i} = \begin{cases} \Delta t \frac{\partial f_c(\underline{x}_k^i, \underline{u}_k^i)}{\partial \underline{x}_k^i} & \text{if } \underline{x}_3^i > z_{min} \\ 0^{4\times 4} & \text{otherwise} \end{cases} \tag{30}$$

which includes a discontinuous step when agent i makes contact with the ground. Some continuous approximations of the ground interaction can be made using a sigmoid function to alleviate the issue of discontinuity. The sigmoid maps to a range of $(0, 1)$ and, given some tuning, can be used to approximate discrete changes in the system dynamics as used by [56]. Consider the sigmoid approximation (SA) function

$$(\underline{x}_3^i)_{SA}^+ = \frac{1}{1 + e^{-c_s(\underline{x}_3^i - y_{min})}} \tag{31}$$

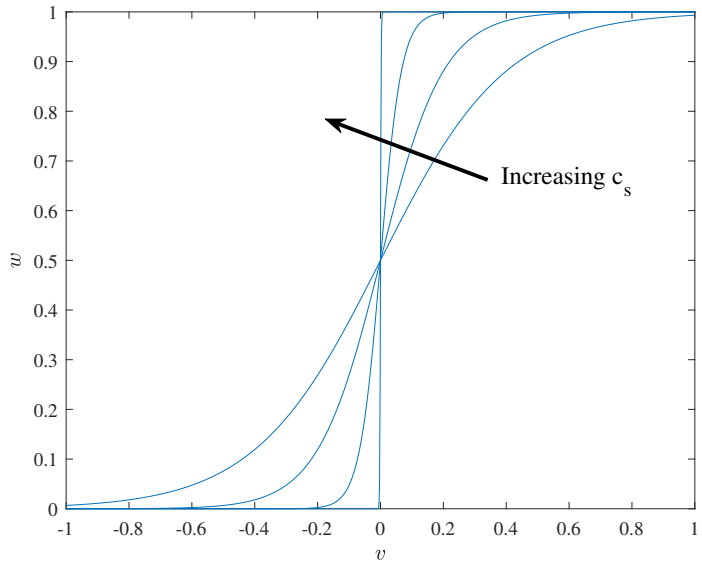

Figure 1 Sigmoid used for discrete step approximation assuming $y_{min} = 0$.

where $\underline{x}_3^i \in \mathbb{R}$ is the z-position of agent i and $c_s \in \mathbb{R}_{>0}$ is a tuning variable. As c_s increases the sigmoid will act more like a discrete step from 0 to 1, similar to how the boundary constraints are implemented. Figure 1 shows the effect of varying the value of c_s given $y_{min} = 0$. Again, similar to the boundary constraints, high values of c_s can result in extremely large derivatives, which may have a significant impact on numerical optimization. However, given the simple parabolic trajectory that the agent follows and the SA being based on the z-position, the derivative of the SA can be neglected.

Using the SA, the derivative of the system's mean dynamics can be written as

$$\frac{\partial f(\underline{x}^i, \underline{u}^i)}{\partial \underline{x}_k^i} = \Delta t \left[(\underline{x}_3^i)_{SA}^+ \frac{\partial f_c(\underline{x}_k^i, \underline{u}_k^i)}{\partial \underline{x}_k^i} + f_c(\underline{x}_k^i, \underline{u}_k^i) \frac{\partial (\underline{x}_3^i)_{SA}^+}{\partial \underline{x}_k^i} \right] \tag{32}$$

The second term can be expanded as

$$f_c(\underline{x}_k^i, \underline{u}_k^i) \frac{\partial (\underline{x}_3^i)_{SA}^+}{\partial \underline{x}_k^i} = \begin{bmatrix} 0^4 & 0^4 & f_c(\underline{x}_k^i, \underline{u}_k^i) \frac{\partial (\underline{x}_3^i)_{SA}^+}{\partial \underline{x}_{3,k}^i} & 0^4 \end{bmatrix} \tag{33}$$

which adds a dependence on the z-position that does not exist in the true system dynamics.

Ignoring this term yields

$$\frac{\partial f(\underline{x}^i, \underline{u}^i)}{\partial \underline{x}_k^i} \approx \Delta t(\underline{x}_3^i)_{SA}^+ \frac{\partial f_c(\underline{x}_k^i, \underline{u}_k^i)}{\partial \underline{x}_k^i} \tag{34}$$

and allows the SA function to just act as a switch that either passes derivative information or zeroes it out. The continuous dynamics derivative is taken to be

$$\frac{\partial f_c(\underline{x}_k^i, \underline{u}_k^i)}{\partial \underline{x}_k^i} = A \tag{35}$$

Considering the constant or zero standard deviation as previously discussed, the control input also has no impact on the process noise, so the derivative with respect to the control input is written as

$$\frac{\partial \underline{x}_{k+1}^i}{\partial \underline{u}_k^i} = \frac{\partial f(\underline{x}_k^i, \underline{u}_k^i)}{\partial \underline{u}_k^i} \tag{36}$$

where

$$\frac{\partial f(\underline{x}^i, \underline{u}^i)}{\partial \underline{u}_k^i} = \begin{cases} \Delta t \frac{\partial f_c(\underline{x}_k^i, \underline{u}_k^i)}{\partial \underline{u}_k^i} & \text{if } \underline{x}_3^i > z_{min} \\ 0^{4\times 2} & \text{otherwise} \end{cases} \tag{37}$$

The same SA function can be used to make a continuous approximation of this function:

$$\frac{\partial f(\underline{x}^i, \underline{u}^i)}{\partial \underline{u}_k^i} \approx \Delta t(\underline{x}_3^i)_{SA}^+ \frac{\partial f_c(\underline{x}_k^i, \underline{u}_k^i)}{\partial \underline{u}_k^i} \tag{38}$$

where

$$\frac{\partial f_c(\underline{x}_k^i, \underline{u}_k^i)}{\partial \underline{u}_k^i} = B \tag{39}$$

APPENDIX E - Kalman Filter

The Kalman filter derivation from [54] is given here. The state estimate $\hat{\underline{x}}_k \in \mathbb{R}^n$ is propagated and updated using

$$\hat{\underline{x}}_{k+1}^- = A\hat{\underline{x}}_k + B\underline{u}_k \tag{40}$$

$$\hat{\underline{x}}_k = \hat{\underline{x}}_k^- + L_k\left[\underline{y}_k - C\hat{\underline{x}}_k^-\right] \tag{41}$$

where $\hat{\underline{x}}_k^-$ denotes the state estimate prior to updating with the filter, $\hat{\underline{x}}_k$ is the corrected estimate of the state, $A \in \mathbb{R}^{n\times n}$ is the linear dynamics matrix, $B \in \mathbb{R}^{n\times m}$ is the linear control matrix, $\underline{u}_k \in \mathbb{R}^n$ is the known control input, $L_k \in \mathbb{R}^{n\times r}$ is the Kalman gain, $\underline{y}_k \in \mathbb{R}^r$ is the state measurement, and $C \in \mathbb{R}^{r\times n}$ is the measurement matrix of the model $\underline{y}_k = C\underline{x}_k + \underline{\nu}_k$ with $\underline{\nu}_k \sim \mathcal{N}(0^r, \sigma_{\underline{\nu}}) \in \mathbb{R}^r$ being the uncorrelated zero mean normally distributed measurement noise of standard deviation $\sigma_{\underline{\nu}}$. The goal is to find an optimal value of L_k such that the error $\underline{e}_k$ is regulated. Let the error terms be defined as

$$\underline{e}_k := \hat{\underline{x}}_k - \underline{x}_k \tag{42}$$

$$\underline{e}_k^- := \hat{\underline{x}}_k^- - \underline{x}_k \tag{43}$$

which can be substituted into the dynamics to obtain

$$\underline{e}_{k+1}^- = A\underline{e}_k - \underline{w}_k \tag{44}$$

where $\underline{w}_k \sim \mathcal{N}(0^n, \sigma_{\underline{w}}) \in \mathbb{R}^n$ is the uncorrelated zero mean normally distributed process noise with standard deviation $\sigma_{\underline{w}}$. The covariance P_k is expanded using Eq. (44) as

$$P_{k+1}^{-} = \mathbb{E}\left[\underline{e}_{k+1}^{-}\underline{e}_{k+1}^{-}{}^{T}\right] \tag{45}$$

$$= \mathbb{E}\left[\left(A\underline{e}_k - \underline{\omega}_k\right)\left(\underline{e}_k{}^{T}A^T - \underline{\omega}_k{}^{T}\right)\right] \tag{46}$$

$$= \mathbb{E}\left[A\underline{e}_k\underline{e}_k{}^{T}A^T\right] - \mathbb{E}\left[A\underline{e}_k\underline{\omega}_k{}^{T}\right] - \mathbb{E}\left[\underline{\omega}_k\underline{e}_k{}^{T}A^T\right] + \mathbb{E}\left[\underline{\omega}_k\underline{\omega}_k{}^{T}\right] \tag{47}$$

$$= AP_kA^T + \Omega \tag{48}$$

where $\Omega := \mathbb{E}\left[\underline{\omega}_k\underline{\omega}_k{}^{T}\right]$. Equation (48) is used to propagate the covariance matrix forward in time. This propagated covariance must be updated using the Kalman gain similar to the estimated state. Consider the estimate update equation and measurement model

$$\hat{\underline{x}}_k = \hat{\underline{x}}_k^{-} + L_k\left[C\underline{x}_k + \underline{\nu}_k - C\hat{\underline{x}}_k^{-}\right] \tag{49}$$

$$= \left(I - L_kC\right)\hat{\underline{x}}_k^{-} + L_kC\underline{x}_k + L_k\underline{\nu}_k \tag{50}$$

This can be written in terms of estimation error as

$$\underline{e}_k = (I - L_kC)\hat{\underline{x}}_k^{-} + +L_kC\underline{x}_k + L_k\underline{\nu}_k - \underline{x}_k \tag{51}$$

$$= (I - L_kC)\underline{e}_k^{-} + L_k\underline{\nu}_k \tag{52}$$

which can then be used with the covariance definition

$$P_k = \mathbb{E}\left[\underline{e}_k\underline{e}_k{}^{T}\right] \tag{53}$$

Substituting and gathering terms gives

$$P_k = \mathbb{E}\left[\left(I - L_kC\right)\underline{e}_k^{-}\underline{e}_k^{-T}\left(I - L_kC\right)^T\right] + \mathbb{E}\left[\left(I - L_kC\right)\underline{e}_k^{-}\underline{\nu}_k{}^{T}L_k{}^{T}\right] + \mathbb{E}\left[L_k\underline{\nu}_k\underline{e}_k^{-T}\left(I - L_kC\right)^T\right] + \mathbb{E}\left[L_k\underline{\nu}_k\underline{\nu}_k{}^{T}L_k{}^{T}\right] \tag{54}$$

which is simplified to

$$P_k = (I - L_k C)\, P_k^- \,(I - L_k C)^T + L_k \Lambda {L_k}^T \tag{55}$$

where $\Lambda := \mathbb{E}\left[\underline{\nu}_k {\underline{\nu}_k}^T\right]$. The diagonal elements of the covariance matrix are the squared errors of each state estimate. Hence, minimizing the trace of the covariance matrix will regulate the estimation error:

$$min_{L_k}\, tr(P_k) \;\; \forall \;\; k \in \mathbb{N}_{[0,N_h]} \tag{56}$$

Interestingly, this is another example of a fixed structure policy optimization. In this case, the update equation is assumed to take the form of Eq. (41), and the optimization of the time varying gain L_k is performed to produce the Kalman gain:

$$L_k = P_k^- C^T \left(C P_k^- C^T + \Lambda\right)^{-1} \tag{57}$$