RESEARCH REPORT

An application-based study of the evolution from model-based to model-free control

Author:
Gideon ILUNG (2241186)

Supervisor:
Krupa PRAG

UNIVERSITY OF THE
WITWATERSRAND,
JOHANNESBURG

submitted to
the Faculty of Science, in fulfilment of the requirements for the degree of
BSc with Honours in Mathematical Science

in the

School of Computer Science and Applied Mathematics

November 21, 2022

Declaration of Authorship

I, Gideon ILUNG (2241186), declare that this Research Report titled, "An application-based study of the evolution from model-based to model-free control" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this Research Report has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this Research Report is entirely my own work.
- I have acknowledged all main sources of help.
- Where the Research Report is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

Abstract

Faculty of Science
School of Computer Science and Applied Mathematics

BSc with Honours in Mathematical Science

An application-based study of the evolution from model-based to model-free control

by Gideon ILUNG (2241186)

Control theory is a broad field that consists of designing methods or algorithms to control engineered dynamical systems and machines. the goal of these methods is to influence the system to reach a setpoint [1] whilst minimising the effects of overshooting [2],steady-state error [3] and stability while also ensuring the system reaches its setpoint optimally.

The challenge with classical analytical methods is that they do not take into account disturbances to the system and have difficulty in handling actuator[4] constraints.

In this research, numerical techniques which can handle disturbances to the system and actuator constraints whilst determining optimal policies in real-time will be explored. Comparisons between these numerical techniques will be made based on solution quality and computational efficiency. These will be measured by running numerous simulations and recording the average error and the average time taken to receive a control response.

Across relevant literature Model Predictive Control, which is a model based control has been used in industry for many years due to the methods accuracy and theoretical guarantees. Due to the difficulties of obtaining and solving mathematical models, model free controllers such as Reinforcement Learning(RL) and Neuroevolution(NE) have been introduced in control literature. RL assumes the system to be Markov, which is generally not true in the control setting hence RL algorithms have been omitted from this research.

Throughout this research, comparisons will be made to establish which control mechanism would be better suited in.

[1] Setpoint: desired state or outcome for the system to reach

[2] Overshooting: Output exceeds past the desired setpoint

[3] Steady State Error: a measure of accuracy a control system has at tracking its input and output reaching its predicted state

[4] Actuator: a component of a system that is responsible for the movement and controlling of a system

Acknowledgements

I would like to thank my supervisor Ms Krupa Prag for her support and guidance. In particular the motivation and ensuring that I produce work to the best of my ability.

Contents

List of Figures

List of Tables

List of Abbreviations

PID	**P**roportional **I**ntegral **D**erivative
MPC	**M**odel **P**redictive **C**ontrol
RL	**R**einforcement **L**earning
LTI	**L**inear **T**ime **I**nvariant
NE	**N**euro- **E**volution
NEAT	**N**euro- **E**volution of **A**ugmenting **T**opologies
GA	**G**enetic **A**lgorithm
PSO	**P**article **S**warm **O**ptimisation
DP	**D**ynamic **P**rogramming
ANN	**A**rtificial **N**eural **N**etwork

List of Symbols

x	displacement	m
θ	angle	deg
K_P	PID Proportional parameter	
K_I	PID Integral parameter	
K_D	PID Derivative parameter	
k	Horizon length	
N	Population size	
ρ	elite sample size	
m	number of generations	
μ_m	mutation rate	
μ_l	link mutation rate	
μ_e	edge mutation rate	
μ_n	node mutation rate	
$c1$	weight parameter	
$c2$	weight parameter	
$c3$	weight parameter	

Chapter 1

Introduction

Every entity, both natural and man-made is dynamic. From the aging of humans to the motion of machines as time changes so does the behavior of these systems. Influencing the behavior of these systems has been a concern to society from the dawn of time. Be it to slow down the aging process or to maximize the productivity of a batch plant some external factor is required to influence the system in some sort of manner. Control theory is a field that focuses on designing control laws or policies that may guarantee optimal behaviors from a dynamic system. Models in various fields such as engineering, biology, and economics consist of dynamic systems, and understanding how to influence these systems for desired outcomes is beneficial. The goal of control theory is to determine the optimal way to influence a system such that an objective is achieved.

Control theory has a plethora of applications in various industries such as engineering, finance, and economics. Despite the vast knowledge available within each respective industry, these industries suffer from inefficiency due to human error. The fact that the systems within these industries are dynamic implies that improvements to influence the behavior of these systems can be made to reduce operational costs.

1.1 Aims, Objectives, and Research Questions

The importance of designing efficient controllers has long been recognized by the industry. However, many systems within industry are plagued by inefficiency and increasingly high costs which can usually be traced back to human operation. Systems within modern society are becoming more complex, making determining efficient strategies to control these systems challenging. Traditionally, Proportional Integral Derivative (PID) controllers have been widely used in industry to control the behavior of these systems, but as the degree of freedom within these systems increases, the controller is rendered useless which emphasizes the need for better control mechanisms. The following research questions that will be addressed through this research include:

- Which control mechanisms are currently used in practice?
- How do these mechanisms perform relative to accuracy and computational efficiency
- What are the recent research advancement within controller design and how applicable are they to industry?
- Are the characteristics of the proposed control schemes comparable?

With the above questions taken into consideration, the aim of this research report is to be able to:

- Make comparisons of the solution quality of the proposed control schemes.
- Determine the computational efficiency of the various control schemes.

Solution quality will be evaluated by running numerous simulation runs and evaluating the robustness of the controller against the presence of noise within the system. Assuming data is sampled at short, fixed time durations, it is important that a control response is delivered to the system before the next sampling state. After a controller has been designed, the computational efficiency will be measured by the time taken for the controller to deliver a control response once state information is given to the controller.

1.2 Method Overview

The control design schemes introduced throughout this research can be classified as either model-based or model-free controllers. Model-based controllers make use of a mathematical model which described the dynamics of the system as described in [1]. this model is then used to determine an appropriate control policy to influence the behavior of the system to reach the desired outcome. Model Predictive Control (MPC) could be considered a model-based controller. Within the MPC framework, the model is used to make future state predictions and optimises the control response over short time intervals. Although these control schemes have been shown to be very effective in industry, the effectiveness of the controller is highly dependent on the accuracy of the mathematical model. Model-free controllers as the name suggests, do not make use of a mathematical model. Proposed model-free controllers are the Proportional Integral Derivative Controller (PID), Neuro-Evolution (NE), Reinforcement Learning (RL) and Neuro-Evolution of Augmenting Topologies (NEAT).A detailed discussion of each method can be found in Chapter 2.

1.3 Overview of report

This report contains 5 chapters. In chapter 2, a brief background of relevant topics in control theory is introduced and discussed whilst also introducing possible control schemes to influence the behavior of dynamic systems. Chapter 3 gives a detailed report on how the various control schemes were implemented and in chapter 4, results were obtained with some discussions made based on the results. Chapter 5 concludes the report and gives possible improvements that could be made in future works.

Chapter 2

Background

Control theory studies the influence of control responses on a system. It has been a heavily focused area due to its applications in engineering. The aim of this chapter is to give an overview of the field control theory and possible control methods. Control theory focuses on designing controllers that regulate the behavior of dynamic systems. This regularisation is achieved by providing a response to the system. The control response is dependent on the current state of the system, constraints on the state of the system and the control response. As seen in figure 2.1 this creates a feedback loop between the controller and system. Combinations of these control responses over time are referred to as a control policy and the goal of control theory is to determine the optimal control policy.

Proposed control frameworks have been introduced throughout this paper, which can be categorised as either model-based or model-free control mechanisms. In regards to the model free control systems it is generally assumed that the system is controllable.

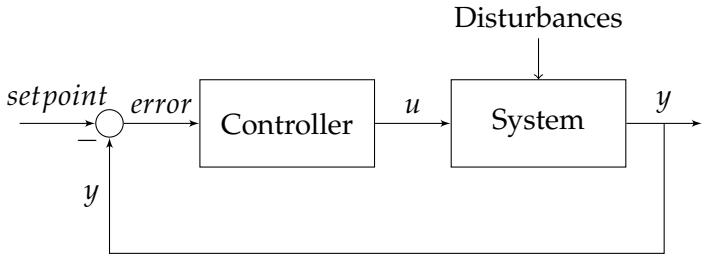

FIGURE 2.1: General Control overview

2.1 Terminology

In the following section, terminology and ideas commonly used in the field of control theory is discussed to give an overview of what control theory consists of. Additional information such as what a state space is, difference between time-invariant and time varient models are discussed in detail in Appendix B

2.1.1 SISO, and MIMO control

An important characteristic of a system is the number of inputs and outputs. SISO control systems are described as systems with a single input and return a single control response whilst MIMO control has multiple inputs are returned multiple control responses.

2.1.2 Open-Loop and Closed-Loop Systems

A control system is said to be a closed-loop system (example seen in figure 2.3) if there is some sort of feedback to the system, usually, this is the case when the next state of the system is dependent on the current state of the system.

An open-loop control system (example seen in figure 2.2) differs from a closed-loop in the sense that there is no feedback given to the controller or if the next state does not depend on the current state. Although open-loop controllers are relatively easy to construct, they are unreliable due to the fact that there is no feedback, which makes the handling of disturbances poor.

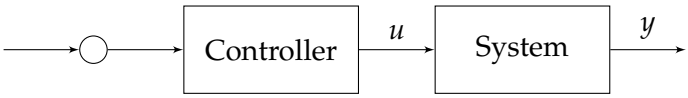

FIGURE 2.2: Example of Open-Loop System

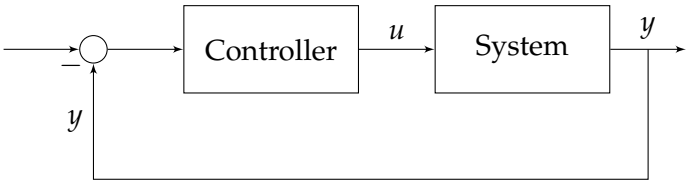

FIGURE 2.3: Example of Closed Loop

2.1.3 Controllability

Controllability determines whether there exists a path from one state to another, regardless of the starting and ending state. Given two states $x_1, x_2 \in X$, a system is said to be controllable if $\exists t \mapsto u(t)$ s.t the system can be steered from x_1 to $x_2, \forall x_1, x_2 \in X$. Depending on the systems linearity and/or time variance determining controllability changes.

Given that the system is a linear time invariant system, the system is said to be controllable if $Rank(R) = n$ where $R = [B, AB, A^2B, \ldots, A^nB]$ and n is the number of state variables.

If the system is nonlinear and starting at x_0, by converting the system into the control-affine form:

$$\dot{x} = f(x) + \sum_{i=1}^{m} g_i(x)u_i$$

The system is said to be controllable from x_0 if the $dim\langle ad_f|R(f)\rangle(x) = n$. if this condition holds $\forall x_0 \in X$ the system is said to be controllable.

2.1.4 Observability

Given the systems, initial state x_0 and is currently in state x_t, a system is said to be observable if it is possible to trace the path taken by the system in order to reach x_t.

For LTI systems, observability is determined by $Rank(R) = n$ where $R = [C, A^2C, \ldots, A^nC]^T$.

For nonlinear systems, utilizing the control-affine form observability is determined using the fact that the system is said to be observable if $dim[d\Theta(x_0)] = n$. Where $\Theta(x_0)$ is the space in which all Lie derivatives lie in.

2.1.5 Disturbance

Disturbance (also known as noise) are external factors that influence the behavior of the system. As described in [2], depending on the system the number of factors that induce noise is generally unknown and/or too large. Assuming $\bar{y}$ is the output of the system influenced by noise one could model the disturbance to the system using Table 2.1.

Examples of noise		
1	$\dot{x} = f(x, u + \epsilon)$	actuator error
2	$\dot{x} = f(x, u) + \epsilon$	state error

(2.1)

Where ϵ is the external noise. Noise handling analytically is generally difficult, which leads to certain assumptions being made when handling noise rejection. In application, a lot of systems tend to have disturbance, which makes applying analytical solutions difficult. Numerical techniques do not assume anything about disturbances to the system, hence for application processes, numerical schemes are favored over analytical.

2.2 Model Definition

Models in the field of Control Theory were defined as below

$$\max \quad \phi[x(t_1)] + \int_{t_0}^{t_1} L(\dot{\boldsymbol{x}}, \boldsymbol{x}, \boldsymbol{u}, t)dt, \tag{2.2}$$

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}), \tag{2.3}$$

$$\boldsymbol{x}(t_0) = \boldsymbol{x}_0 \qquad \boldsymbol{x}(t_1) = \boldsymbol{x}_T,$$

subject to

$$\begin{aligned} g_i(\boldsymbol{x}, \boldsymbol{u}) &= 0 \qquad \text{for } i = 1, ..., n \\ h_j(\boldsymbol{x}, \boldsymbol{u}) &\geq 0 \qquad \text{for } j = 1, ..., m \end{aligned} \tag{2.4}$$

where equation 2.2 is the cost of the state which is to be minimized from t_0 to t_1 and

equation 2.3 is the model describing the system dynamics behaviour equations 2.4 are constraints that the system must abide by.

2.3 Bellman's Principle of Optimality

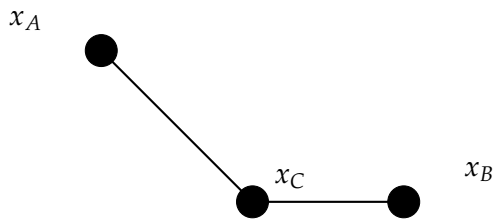

FIGURE 2.4: Illustration of bellman optimality condition, $\min(J(x_A,x_C)) + \min(J(x_C,x_B)) = \min(J(x_A,x_B))$

Bellmans principle of optimality states that whatever the initial state and initial control are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision. This is a useful principle as it allows for the development of numerical schemes. Taking into account Figure 2.4, the condition states that the optimal state trajectory from x_A to x_B is equivalent to the addition of the optimal state trajectories x_A to x_C and x_C to x_B. The proof of this principle can be found in Appendix A.1.

2.4 Bellman Equation

Bellmans equation is a necessary condition of optimality expressed as an equation in dynamic systems. The equation is mainly used in discrete settings. The continous version of this condition is the Hamiltonian-Jacobi-Bellman(HJB) equation which extends bellmans equation into the continous setting. Since numerical schemes are used throughout this project discussions of the HJB equation is omitted. Bellmans equation A.1 defines the value of a particular state which is dependant on the value of future states where $V(x)$ is an unknown functional. Within numerical schemes such as MPC and RL $V(x)$ is determined iteratively. The proof that bellmans equation statisfies optimality can be found in section A.2.

$$V(x_i) = \max_{a_i}\{F(x_i, a_i) + \beta V(x_{i+1})\} \tag{2.5}$$

2.5 Proportional Integral Derivative Control

the Proportional Integral Derivative (PID) controller is a fixed-feedback controller that approximates the optimal control based on the error between the current state of the system and the goal state. The control policy computed by the controller is governed equation 2.6. More Information could be found in [3].

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}. \tag{2.6}$$

The controller is said to be modular as it consists of 3 individual components. The PID controller had a major impact in the field of engineering but its performance many came into question when dealing with MIMO systems.

2.5.1 Proportional Error

The proportional Error Term $K_p e(t)$, computes some proportion of the error in the current state and makes use of that to compute the optimal control policy. It determines the control policy solely based on the magnitude of the error in the current state

2.5.2 Integral Error

The Integral Error term $K_i \int_0^t e(\tau)d\tau$,keeps track of all the past errors throughout the system. The Integral Term can be said to keep track of previous errors made by the controller and to prevent the controller from repeating the same errors. The integral term has known disadvantages such as integral windup. It occurs when the accumulated error over time gets too large resulting in poor control policies being calculated. In practice, it is common to reset the Integral Error or turn off the integral term when certain conditions indicate an integral windup may occur.

2.5.3 Derivative Error

The Derivative Error term $K_d \frac{de(t)}{dt}$, computes the rate of change of the error based on the current state of the system. It serves as a future predictor within the controller as it allows a response based on the magnitude of the rate of change. For example, if there is a sharp increase in the error the derivative term returns a value proportional to the magnitude of the increase, effectively allowing the controller to counter possible increases in the error function. The derivative term also has known issues, such as noise amplification and instability. Multiple techniques have been introduced to counteract these issues but in practice it is generally advised to not use the Derivative term when designing a controller.

2.5.4 Parameter Tuning

The parameters K_p, K_i, K_d have a huge influence on the effectiveness of the PID controller, which leads to an important aspect of finding the optimal parameters for the system in question. The optimal parameters vary depending on the system hence parameter optimisation algorithms have been introduced. The parameter optimisation process can be described using figure 2.5. Some of the highly regarded tuning algorithms are

- Ziegler–Nichols
- Tyreus Luyben
- Cohen–Coon
- Åström-Hägglund
- Global Optimisation Metaheuristics

more information on parameter tuning can be found in [4]

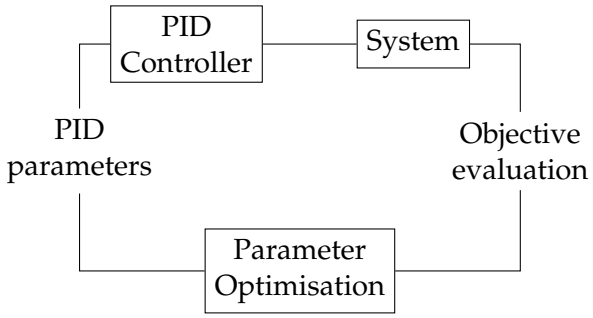

FIGURE 2.5: General overview of PID parameter tuning

2.6 Model Predictive Control

Model Predictive Control (MPC) is a feedback control scheme that converts the control theory problem to an optimization problem over small time intervals. Similar to figure 2.6, The controller makes use of a mathematical model to predict the future states of the system. The controller makes k future predictions and makes use of the control policies that minimize the function value over the prediction horizon. Once the sub-optimal control policies are determined, only one control response is sent to the system and the method is repeated again. The optimizer determines how effective the controller is hence 2 optimization algorithms were used. The algorithm was first introduced in the late 20^{th} century for process control in the petrochemical industry. MPC has been an attractive method for controlling dynamical systems due to its ability to easily handle MIMO systems and constraints. MPC has also gained recent attention in the academic community due to its accuracy. Further discussion on MPC could be found [1].

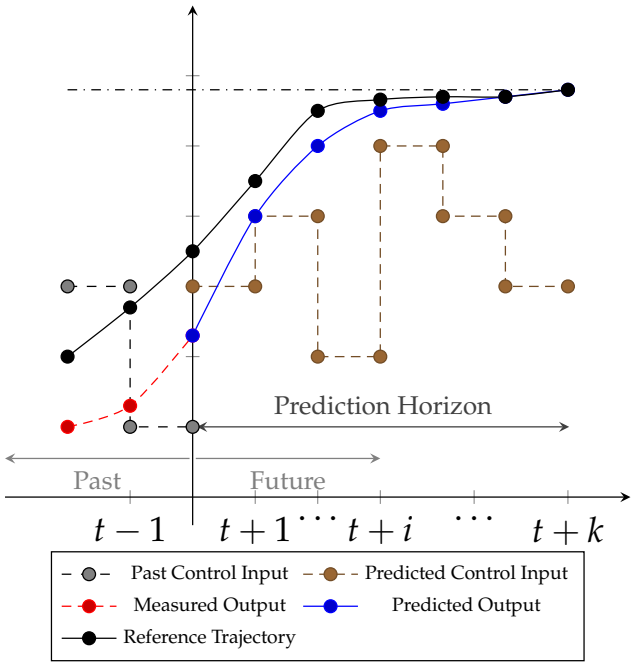

FIGURE 2.6: Graphical representation of the MPC algorithm

2.6.1 Dynamic Programming

Given the action space is a small finite set, dynamic programming is a brute force algorithm that tries all possible combinations of the possible control policy over the prediction horizon are returns the first control response of the optimal control policy. More information on Dynamic Programming in the field of Control Theory is discussed in [5]

2.6.2 Genetic Algorithm

As the prediction horizon k increases or the action space increases, making use of dynamic programming tends to be infeasible. An alternative optimizer introduced is Genetic Algorithm (GA). Genetic Algorithm is a metaheuristic that was inspired by Darwin's theory of evolution. The algorithm starts off with a population of possible control policies with an associated fitness value. At each generation (or iteration) possible solutions mate to create better solutions. The algorithm has been celebrated due to its applicability to a wide range of problems, both discrete and continuous. Further discussions on the use of GA could be found in [6]

2.7 Data-Driven Model Predictive Control

An extension of MPC which makes use of historical data obtained from the system. Although MPC is a powerful framework, its performance is heavily dependent on the mathematical model used to make future predictions. Due to the difficulty of determining a mathematical model for certain systems, assumptions and simplifications are used to determine a mathematical model which approximates the behavior of the system. By making use of historical data and the approximate model, Data-Driven Model Predictive Control (DDMPC) determines the appropriate control response. DDMPC makes use of the historical data to determine the range in which future states should lie in, in conjunction with the approximate mathematical model to determine the next state. The application of DDMPC is mainly used in controlling stochastic systems, due to the difficulty of obtaining mathematical models that encapsulate the entire system as mentioned in [7]. Further discussion on DDMPC can be found in [8] [1].

2.8 Reinforcement Learning

Reinforcement learning (RL) is a subset of Machine learning (ML) which focuses on decision-making processes with the goal being to be able to construct a model that can determine the optimal decision process based on the current state of the system. The basic idea behind RL is learning from previous iterations to improve its decision-making process. In order to achieve this multiple components are used to determine the optimal decision process.

2.8.1 Environment

In RL literature [9] [10], an environment is said to be a system with which the agent has interaction with. In the context of Control theory, this would be the system that the agent would have iterations with.

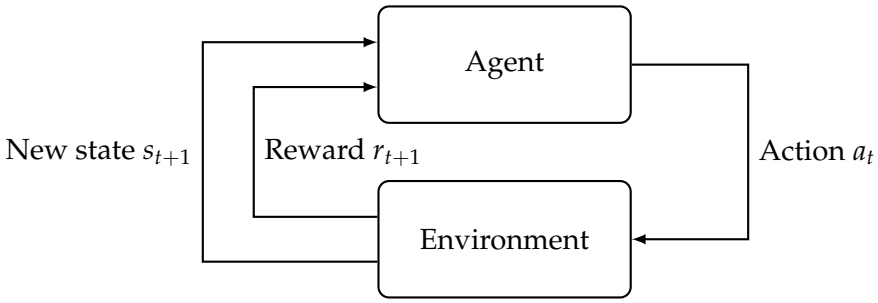

FIGURE 2.7: General Reinforcement learning flowchart

2.8.2 Reward

A reward (a reward at time step t is usually denoted as R_t) is the feedback that is used to evaluate the effectiveness of the decision taken by the agent. if the agent has returned a control response which steers the system closer to the desired state then usually a high reward is returned to the agent to have a positive effect on the learning process. if the control response influences the system to deviate away from the desired state usually a negative reward is returned to the agent to prevent the agent from returning the bad control response again.

2.8.3 Markov Assumption

The Markov assumption is a fundamental property of RL as it is used in the formulation of many RL algorithms. The property states that decisions and values are assumed to be a function only of the current state. This is generally not true in the control setting, especially when there are inequalities involving integrals over time. Due to this assumption RL has been omitted from this project. An alternative model free control scheme is introduced in the next section, which does not make this assumption.

2.9 Neuro-Evolution

Neuro-evolution as discussed in [11], is a subset of computational intelligence that makes use of an Artificial Neural Network (ANN) to learn the dynamics of a particular system. This is extremely useful in the control theory setting as it removes the need of a mathematical model which describes the dynamic behavior of the system. Although it allows for the creation of controllers without a mathematical model, this framework fails to give information on controllability and observability. In application, it is generally assumed that the system in question is controllable. Neuro-Evolution follows the same principles mentioned in Darwinism (a discussion on Darwinism found here [12]) which uses the ideas of populations mating and interacting which facilitate the transformations of species over generations that are better suited for the environment. These ideas are translated to be used in computational models, in the case of control theory to construct an efficient controller. The method of determining an efficient ANN controller varies depending on the optimization metahueristic used but the general overview is the same. A population of randomly generated networks is initialized. The effectiveness of the network is usually determined by some fitness function. depending on the fitness value, a subset of the networks in the population will be used to generate a new set of ANNs that will

replace low-ranking ANNs in the next iteration (generally referred to as generation in the relevant literature). Commonly used metahueristics are:

- Genetic Algorithm
- Differential Evolution
- Particle Swarm Optimisation
- Simulated Annealing

These methods differ from standard neural-network-based RL techniques as for each control response an associated reward is required to update the model and the topology of the network is generally fixed (example by figure 2.8). With neuro-evolution, the above 2 properties do not hold, as the topology of the network may change and evaluations are usually determined after a fixed window of exploration [13]. An example of a neural network with changing topology can be seen by figure 2.9. The structure of heuristics allow distribution of computation across multiple machines which makes scaling these algorithms computationally efficient. During the formulation and training of the network, the Markov assumption is not assumed.

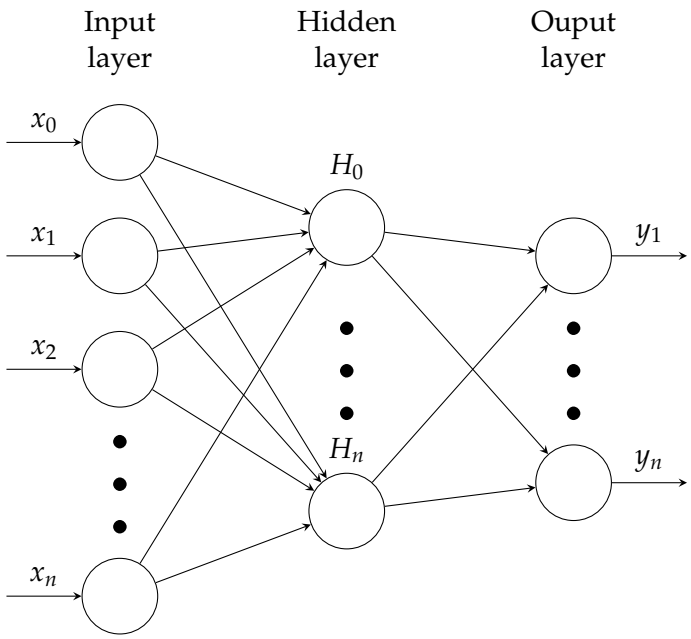

FIGURE 2.8: Artificial Neural Network with Fixed Topology

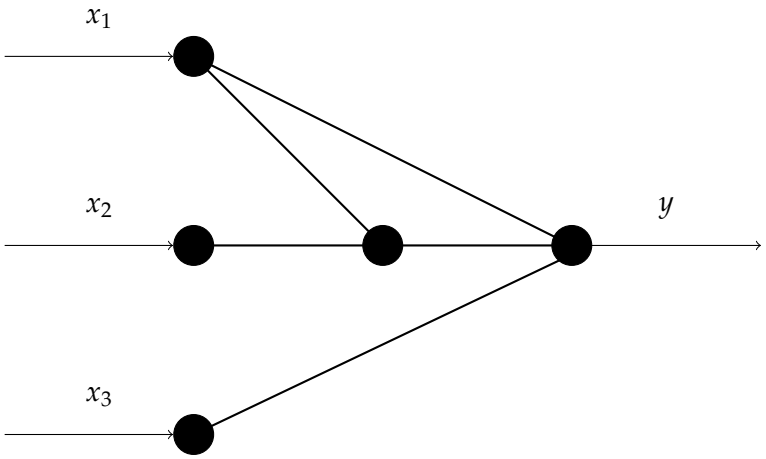

FIGURE 2.9: Artificial Neural Network without fixed topology

2.10 Benchmarks

The cart pole problem has been used as a benchmark for comparison of control frameworks, particularly model free methods. based on previous experiments done the following results where obtained.

Comparison of algorithms	
Algorithm	number of iterations till optimal policy found
SARSA [14]	420
Q-Learning [14]	420
PPO [15]	200

The following results will be used as benchmarks for the control schemes mentioned above where possible.

2.11 Conclusion

Frameworks for determining the control function both analytically and numerically were introduced, with more focus on numerical techniques as no assumptions are made on external disturbances to the system. Within the numerical framework, there are two classes of approaches looked into. The first is model-based which makes use of a mathematical model to determine the optimal control policy. The effectiveness of these techniques is dependent on the accuracy of the model. The second type explored is model-free techniques, which do not rely on a mathematical model but do not prove information such as controllability and observability. Since these models make use of ANNs to approximate control policies they tend to be computationally cheaper in comparison to MPC.

Chapter 3

Methodology

The goal of this project is to compare different control mechanisms based on their solution quality and their computational efficiency. The solution quality produced by the controller will be evaluated by running 30 trials on a cart pole simulator within the Open AI Gym simulator [16], and the robustness of the controller will be evaluated by adding noise to the sampled measurements from the simulator and observing how effective the control mechanism handles noise. The computational efficiency of the controller will be determined by measuring the time taken for the controller to return a control response to the system. This response time will be measured throughout a single simulation run to determine the mean and standard deviation of the control response. For the learning-based controllers, additional measurements will be considered. The convergence rate of the learning-based controllers will be measured to determine how quickly these control mechanisms learn the optimal control response, in addition to that, the number of iterations and time taken to learn the optimal control law will also be taken into consideration.

The computational experiments described in the methodology are obtained using Python 3.8.10 running on Intel® Core™ i5-8400 CPU @ 2.80GHz × 6, 16GB, 64-bit Ubuntu 20.04 LTS. All code used throughout this research could be found here.

Throughout this chapter, the implementation of PID, MPC, NE, and NEAT is discussed in detail. Before the aforementioned control mechanisms are discussed, an overview of GA and PSO are given as they are used in each of the control mechanisms to approximate the control policy and/or optimize the parameters of the control mechanisms. Before the control schemes are discussed a brief overview of cart pole is provided.

3.1 Cartpole Overview

Cart-pole is a well-known classical control theory problem. Over the years the cart pole problem has been used as a benchmark for comparison and testing of possible control schemes. Derivation of the differential equations of the cart-pole problem can be found in Section 3.1.1, which is modeled over the forces as seen in Figure 3.1. Given an inverted pendulum that moves, the goal of the controller in this setting would be to be able to determine which direction should the cart be pushed in such that the pendulum remains upright for as long as possible

3.1.1 Derivation of Cart pole Model

the following variables were used in the derivation of the model of this problem:

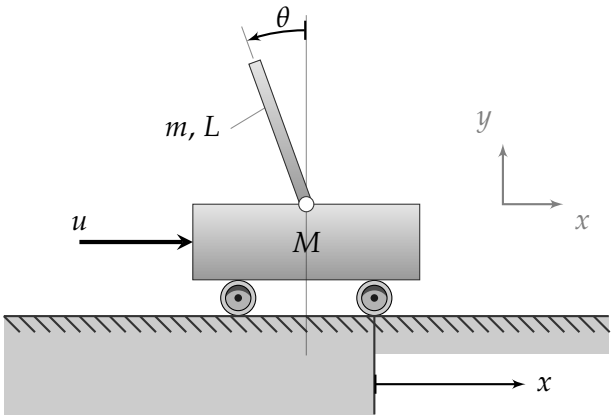

FIGURE 3.1: Cartpole diagram

$x(t)$: horizontal displacement of cart
$x_p(t)$: horizontal displacement of pole
$y_p(t)$: vertical displacement of pole
$\theta(t)$: rotational displacement of pole
L : Length of pole
M : Mass of cart
m : Mass of pole
u : applied force

assuming the pole is uniform and $\ell = \frac{L}{2}$, the position of the center of mass of the pole could be defined as:

$$\begin{aligned} x_p(t) &= x(t) + \ell \sin(\theta) \\ y_p(t) &= \ell \cos(\theta) \end{aligned} \tag{3.1}$$

with the velocity of the pole being:

$$\begin{aligned} \dot{x}_p(t) &= \dot{x}(t) + \ell\dot{\theta} \cos(\theta) \\ \dot{y}_p(t) &= -\ell\dot{\theta} \sin(\theta) \end{aligned} \tag{3.2}$$

since the cart is on the ground, it can be assumed that it has no potential energy. However, the potential energy of the pole can be determined using the following equation:

$$\begin{aligned} U &= mgh \\ &= mgy_p \\ &= mg\ell \cos(\theta) \end{aligned} \tag{3.3}$$

the kinetic energy of the system is determined by the summation of the kinetic energy of the cart and pole:

$$\begin{aligned} T &= T_{\text{cart}} + T_{\text{pole}} \\ &= \frac{1}{2}M\dot{x}^2 + \frac{1}{2}m(\dot{x}_p^2 + \dot{y}_p^2) \\ &= \frac{1}{2}(M+m)\dot{x}^2 + m\ell\dot{x}\dot{\theta}\cos(\theta) + \frac{1}{2}m\ell^2\dot{\theta}^2 \end{aligned}$$

if we let $D = M + m$ we obtain:

$$T = \frac{1}{2}D\dot{x}^2 + m\ell\dot{x}\dot{\theta}\cos(\theta) + \frac{1}{2}m\ell^2\dot{\theta}^2 \tag{3.4}$$

the Lagrangian of the system can be determined by subtracting 3.3 from 3.4

$$L = \frac{1}{2}D\dot{x}^2 + m\ell\dot{x}\dot{\theta}\cos(\theta) + \frac{1}{2}m\ell^2\dot{\theta}^2 - mg\ell\cos(\theta) \tag{3.5}$$

using the Euler-Lagrange equation on the horizontal displacement we have:

$$\begin{aligned} \frac{d}{dt}\left[\frac{\partial L}{\partial \dot{x}}\right] - \frac{\partial L}{\partial x} &= u \\ D\ddot{x} + m\ell\left[\ddot{\theta}\cos(\theta) - \dot{\theta}^2\sin(\theta)\right] &= u \end{aligned} \tag{3.6}$$

using the Euler-Lagrange equation on the rotational displacement we have:

$$\begin{aligned} \frac{d}{dt}\left[\frac{\partial L}{\partial \dot{\theta}}\right] - \frac{\partial L}{\partial \theta} &= 0 \\ \ddot{x}\cos(\theta) + \ell\ddot{\theta} - g\sin\theta &= 0 \end{aligned} \tag{3.7}$$

in this case, the expression is set to zero as there is no external moment of inertia applied to the system.

Let :

$$\begin{aligned} X_1 &= x \\ X_2 &= \dot{x} \\ X_3 &= \theta \\ X_4 &= \dot{\theta} \end{aligned}$$

since the system returns the state information $\vec{X}$ it is required to have a model in the form:

$$\dot{\vec{X}} = F(\vec{X}, u)$$

solving for $\ddot{x}$ and $\ddot{\theta}$ in 3.5,3.6 we obtain the following state equation:

$$\begin{aligned}
\dot{X}_1 &= X_2 \\
\dot{X}_2 &= A\left[u + m\ell \sin(X_3) X_2^2 - mg\cos(X_3)\sin(X_3)\right] \\
\dot{X}_3 &= X_4 \\
\dot{X}_4 &= g\ell^{-1}\sin(X_3) - \ell^{-1}\cos(X_3) A\left[u + m\ell \sin(X_3) X_2^2 - mg\cos(X_3)\sin(X_3)\right]
\end{aligned} \tag{3.8}$$

where $A = \frac{1}{D - \cos^2(X_3) m}$

3.1.2 Mathematical model

Since the goal of the cart-pole problem is to minimise the rotational displacement of the pole, It is possible to mathematically formulate the problem as

$$\min \int_0^\infty X_3^2(t) dt, \tag{3.9}$$

$$\begin{aligned}
\dot{X}_1 &= X_2, \\
\dot{X}_2 &= A\left[u + m\ell \sin(X_3) X_2^2 - mg\cos(X_3)\sin(X_3)\right], \\
\dot{X}_3 &= X_4, \\
\dot{X}_4 &= g\ell^{-1}\sin(X_3) - \ell^{-1}\cos(X_3) A\left[u + m\ell \sin(X_3) X_2^2 - mg\cos(X_3)\sin(X_3)\right],
\end{aligned}$$

subject to:

$$-4.8 < X_1 < 4.8 \tag{3.10}$$

$$-0.418 < X_3 < 0.418 \tag{3.11}$$

where $A = \frac{1}{D - \cos^2(X_3) m}$,
t is time and u is the control variable

since the input of the system should be either zero or one, the control policy u will be passed through the sigmoid function before the policy is sent to the system.

3.2 Genetic Algorithm

Genetic Algorithm (GA), as discussed in [17], is a common meta-heuristic that closely mimics the natural selection process to determine the global optima. The general process of the algorithm can be found in Figure 3.2. Starting off with a population size of N candidate solutions are randomly initialised. This population of candidate solutions will then evolve over the generations, favoring better solutions. During each generation, a portion of the population of size ρ, which contains the fittest candidates, will be carried over to the next generation, whilst other candidate solutions for the next generation will be created through Crossover. Crossover is a genetic operator that generates a new candidate solution based on the genetic information of parent solutions (current individuals in the population). Nearing the end of each generation a solution may stand a chance of being mutated, which may result in

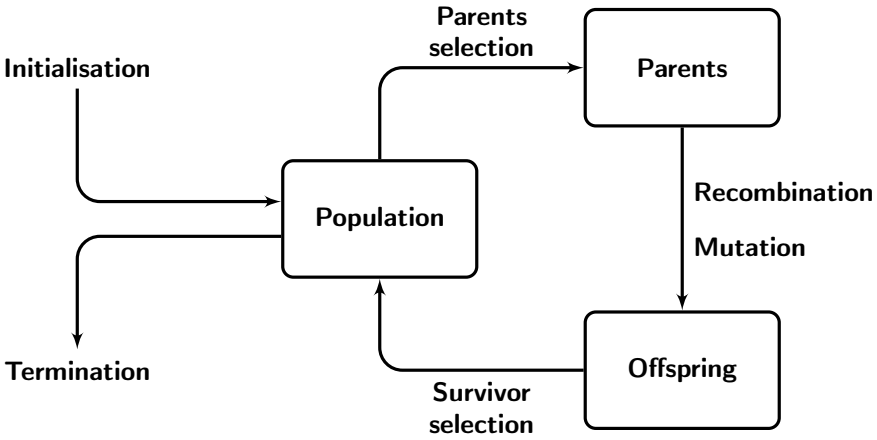

FIGURE 3.2: Genetic Algorithm Flowchart

better solutions being found. By introducing this random change we introduce a probability of a mutated candidate solution returning a better solution.

3.2.1 Selection

An integral part of GA is selecting good candidate solutions within the population in order to generate new solutions. Commonly used selection operators are Roulette and Tournament Selection [17]. Roulette Selection samples a random solution from the population based on the population's fitness values. Although it has been commonly used for parent selection, it is limited due to the fact that in order to construct a reliable probability distribution all fitness values associated with the candidate solutions within the population must be non-negative. Tournament selection addresses the issues mentioned with roulette selection. Two unique solutions are sampled from the population and the solution with the better fitness function is kept to be used for crossover. This process is repeated again to find the second parent which will be used in generating new solutions.

3.2.2 Crossover

Crossover is an operator that generates new candidate solutions by combining subsets of the parent solutions. A commonly used crossover operator is single-point crossover. As seen in Figure 3.3, an index i is selected and from this index to the index of the last gene, the values of the genes are swapped, which results in two new candidate solutions formed.

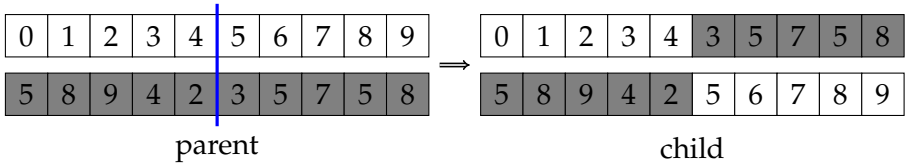

FIGURE 3.3: Example of crossover in GA

3.2.3 Mutation

The mutation operator is responsible for perturbing candidate solutions in order to diversify the population's search space. An example of this mutation is bit flipping in binary encoding problems. An example of applying this operator to a candidate solution can be seen in Figure 3.4. The rate at which mutations occurs, μ_m, is usually set to a small value. Similar to natural biological organisms only having a small chance of their genetic information being mutated, resulting in either a better or worse solution.

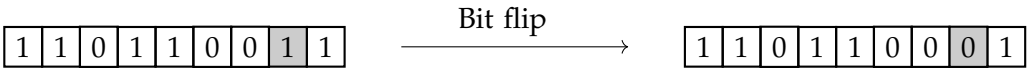

FIGURE 3.4: Visualization of the bit-flip.

3.2.4 GA Algorithm

In this research, GA is used to optimise the different control mechanisms. A general algorithmic process of the algorithm is given Algoritm 1.

Algorithm 1 Pseudo-code for Genetic Algorithm

```
procedure GA
    for each agent i = 1, ..., N do
        Initialise agent_i's gene x_i within search space
        Assign a fitness value to agent_i
        Add agent to population set
    while termination criterion not met do
        Select candidates within population for crossover
        Generate new candidate solutions using selected candidates
        Update the population of candidate solutions
        Mutate subset of population
        Update the fitness scores of the candidate solutions
    return Fittest member within the population
```

3.3 Particle Swarm Optimisation

Particle Swarm Optimisation (PSO), as discussed in [17], is a swarm based meta-heuristic that makes use of social behavioural characteristics to search for the optimal solution. Each particle's movement is influenced by its local best solution and the global best solution found up until that point in time. The position update is determined using equation (3.12).

$$x_{t+1}^i = x_t^i + c_1 r_1 (p_t^i - x_t^i) + c_2 r_2 (g_t - x_t^i). \tag{3.12}$$

The iteration count is represented by t, whilst i represents the agent number and the variables p_t^i, g_t represent the agents personal best solution and global best solution respectively. the coefficients r_1 and r_2 are uniformly sampled values between $[0, 1)$. The parameters c_1, c_2 are weight parameters. A visual representation of how PSO searches and updates the agents position can be found in Figure 3.5. Throughout

the experiments conducted, the parameters c_1, c_2 where both set to a fixed value of two.

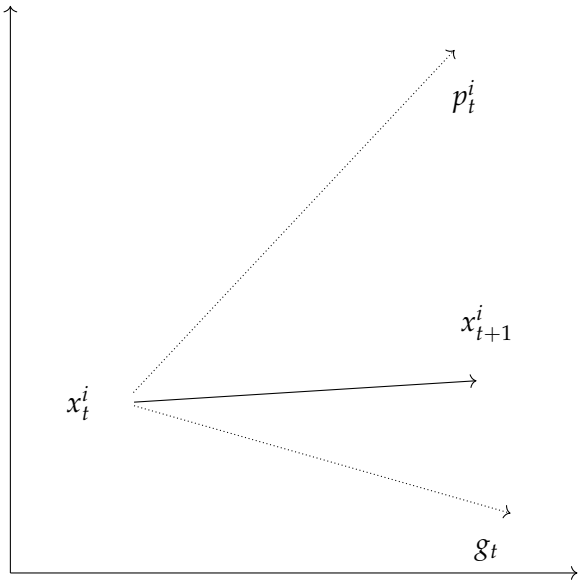

FIGURE 3.5: PSO Update

3.3.1 PSO Algorithm

PSO is used in varying frameworks in order to determine the optimal control policy, Hence a general overview of the implementation of the algorithm is given in Algorithm 2.

Algorithm 2 Pseudo-code for Particle Swarm Optimisation

```
procedure PSO
    for each agent i = 1, ..., N do
        Initialise agents position x_i within search space
        Set agents personal best position p_i to current position
        if fitness(p_i) < fitness(g) then
            g ← p_i
    while termination criterion not met do
        for each agent i = 1, ..., M do
            Update agents position using equation 3.12
            if fitness(x_i) < fitness(p_i) then
                p_i ← x_i
            if fitness(p_i) < fitness(g) then
                g ← p_i
    return g
```

3.4 Proportional Integral Derivative Controller

The Proportional Integral Derivative Controller is a fixed-based controller that determines the sub-optimal control response using the following equation.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}.$$

Where K_p, K_i, K_d are the proportional, integral and derivative parameters respectively. the error of the system at time t is denoted as $e(t)$. Since state information is sampled at fixed-time differences the following equation was used in implementation,

$$u_k = K_p e_k + K_i \sum_{j=0}^{k} e_j \Delta t + K_d \frac{e_k - e_{k-1}}{\Delta t}, \tag{3.13}$$

where k is the current iteration within the simulation. The first term makes use of some proportion of the to determine the control response, the second term makes use of past measurements and the third makes use of the rate of change of the error to make future predictions to determine the appropriate control response.

3.4.1 Parameter Optimisation

The solution quality of the control response returned by the PID controller is dependant on the K_p, K_i, K_d parameters. For this experiment, each parameter will be bounded between $[-10, 10]$. GA and PSO will be used in order to determine the optimal parameters for the PID controller.

Fitness function

The fitness value for each set of parameters will be measured by the total error across 500 iterations. S is the set-point of the system.

Algorithm 3 PID parameter fitness

```
1:  procedure FITNESS(K_p, K_i, K_d, S)
2:      controller ← PID(K_p, K_i, K_d)        ▷ Creating controller with input parameters
3:      state ← System.getState()              ▷ Obtaining the initial state from the system
4:      error ← 0
5:      k ← 0
6:      while termination criterion not met do
7:          e ← (state − S)^2
8:          error ← error + e
9:          u ← controller.response(e, k)      ▷ Calculates control response using 3.13
10:         state ← System.Step(u)             ▷ Applies control, gets next state
11:         k ← k + 1
12:     return error
```

3.5 Model Predictive Control

Model Predictive Control (MPC) is a fixed-based controller that determines the optimal control policy by converting the control problem to an optimisation problem

over small time intervals and by making use of the mathematical model to make future predictions. The variable k denotes the horizon length, which essentially is a measure of how many iterations into the future will the algorithm look into to determine the control response. The function L_t is the cost of the current state the system is in. For example with regards to the cart-pole problem $L_i = X_3^2$. Given that the system is at time t with a horizon length k, the goal would be to determine the control response u that optimises equation 3.14.

$$\sum_{i=t}^{t+k} L_i(X_i, u_i)\Delta t. \tag{3.14}$$

3.5.1 Algorithm

Given a mathematical model $\dot{X} = f(X, u)$, that describes the dynamics of the system The general algorithmic process of the MPC framework that is presented in Algorithm 4 can be used.

Algorithm 4 Model Predictive Control

```
1: procedure MPC(Ẋ,k,stop)
2:     X ← System.getState()          ▷ Obtaining the initial state from the system
3:     t ← 0
4:     while termination criterion not met do
5:         u ← optimise(Ẋ, X, k)      ▷ Determine the control optimising 3.14
6:         X ← System.Step(u[0])      ▷ Applies first control, gets next state
7:         t ← t + 1
```

Utilising the proof in section A.2 it is possible to conclude that for $k \geq 2$, MPC can be written as an optimisation problem in the form of equation A.1 and that MPC will yield locally optimal results for $k \geq 2$.

3.6 Neuro-Evolution

Neuro-Evolution (NE) is an adaptive learning control mechanism which approximates the optimal control function $u(t)$. By using the universal approximation theorem, it was shown in [18] and Appendix A.3 that neural networks can be used to approximate and control dynamical systems.The main advantage NE has over Reinforcement learning is that the Markov Assumption is not a necessary requirement for the learning process of the control scheme. Using the information stated above, the goal of using neuro evolutionary techniques is to approximate the optimal control function $u(t)$ without any knowledge of differentiating between efficient or inefficient control responses. This is achieved by restructuring the weight matrices of the network into a single vector and optimising the weights using GA or PSO . The fitness of each network is determined using an equation similar to the functional 3.14.

3.6.1 Fitness Function

The fitness function of the network is determined over a fixed duration of time. Let $L_i(X_i, u_i)$ denote the cost of the state at iteration i and x be the vector representation

Algorithm 5 Neuro Evolution Fitness

```
1:  procedure FITNESS(x, v)
2:      model ← reconstruct(x, v)        ▷ Reconstruct network in traditional format
3:      X ← System.getState()            ▷ Obtaining the initial state from the system
4:      t ← 0
5:      error ← 0
6:      while termination criterion not met do
7:          u ← model.feedForward(X)     ▷ u determined using feed-forward
8:          X ← System.Step(u)           ▷ Applies control, gets next state
9:          error ← error + L(X, u)
10:         t ← t + 1
11:     return error
```

of the network we can determine the fitness of the network using the Algorithm 5 and determine the optimal network using the Algorithms 1 and 2.

Due to truncation issues encountered when dealing with the cart pole simulator, the fitness function which was a minimisation problem was converted to a maximisation problem with a penalty parameter. with L_i defined using equation 3.17

$$L_i(X_i, u_i) = R_i - X_i[3]^2 \tag{3.15}$$

Where $R_i = 1$ when X_i satisfies equations 3.10,3.11 and $R_i = -1$ otherwise

3.7 Neuro-Evolution of Augmenting Topologies

Neuro-Evolution of Augmenting Topologies (NEAT) which was introduced by Stanley [13] is an adaptive learning control mechanism which builds on the disadvantages of NE. The universal function approximation theorem states that a network with a large enough hidden layer can be used to approximate any function, in the context of control theory, any control function $u(t)$. The issue with predefined network dimensions is it may be using more nodes than required, which in turn increases the computational expense of determining control responses. NEAT addresses these issues by starting off with a network of input and output nodes only, then adding complexity to the network only when required. The fitness value of candidate networks is determined using algorithm 5.

3.7.1 Representation of Genomes

As mentioned above there is no specified structure within the possible network architecture which requires a new way of representing the network. An example of this representation can be seen in figure 3.7 and figure 3.6. Each genome representation will consist of node genes, which give a description of the node type of the specified node and edge genes which detail information on edges between nodes. The following information is derived from the edge genes

- In: denotes where the signal will be travelling from along that edge
- Out: denotes where the signal will be travelling to
- Weight: denotes the weight parameter along that edge
- Enabled/Disabled: denotes whether the edge is active or not

- Innov: is the innovation number which is a unique number assigned to each edge which will be used during crossover

Node Genes	Node 1 Type: Input	Node 2 Type: Input	Node 3 Type: Input	Node 4 Type: Output	Node 5 Type: Hidden
Edge Genes	In: 1 Out: 4 Weight: 0.7 Enabled Innov 1	In: 2 Out: 4 Weight: -0.5 Disabled Innov 2	In: 3 Out: 4 Weight: 0.5 Enabled Innov 3	In: 2 Out: 5 Weight: 0.2 Enabled Innov 4	In: 5 Out: 4 Weight: 0.4 Enabled Innov 5	In: 1 Out: 5 Weight: 0.6 Enabled Innov 6

FIGURE 3.6: Genome representation of the network in 3.7

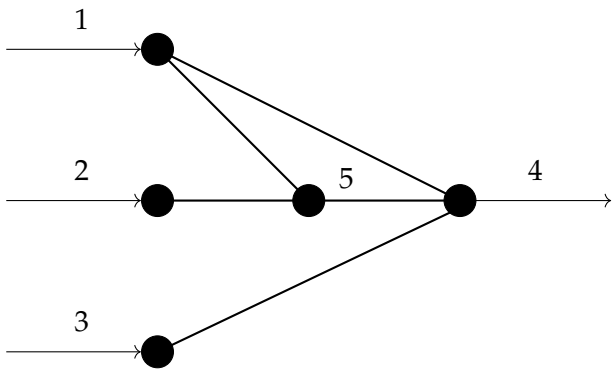

FIGURE 3.7: Network representation of genome in 3.6

3.7.2 Mutations

Within the network architecture, there exist 2 types of mutations, edge and node mutations. Edge mutations occur by adding new edges between nodes, mutating the weight value along the edge, disabling existing edges and enabling disabled edges. The type of edge mutation is determined by randomly a random value and setting up a probability distribution of each type of edge mutation that can occur. Node mutations occur by adding or removing hidden nodes within the network. Similarly to the edge mutation, The type of node mutation is determined by sampling a random value and setting up a probability distribution of each type of node mutation that can occur.

3.7.3 Crossover

As mentioned above, the candidate networks in the population have different topological structures, which makes using traditional crossover functions relatively difficult. By using the innovation number we can generate new candidate networks by matching the edge genes with the same innovation numbers and applying crossover operators. Genes that do not match can be classified as either disjoint or excess genes. Referencing Figure 3.8, The genes with innovation numbers 4 and 5 can be classified as disjoint edges and the gene with innovation number 7 can be classified as an excess gene. Given there are matching genes, the gene that would be selected would be determined randomly. For excess and disjoint genes, the candidate network with the better fitness value will determine whether the genes will be accepted. For example, assuming the second network has a higher fitness, the genes with innovations 4 and 5 will be selected whilst the gene with innovation number 7 will be rejected.

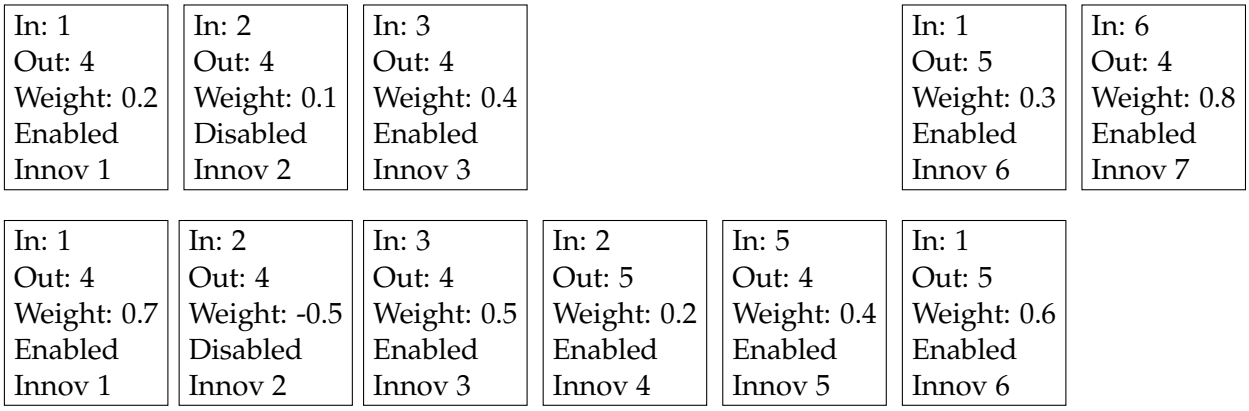

FIGURE 3.8: Crossover between 2 networks

3.7.4 Speciation

The main drawback with NEAT is its ability to differentiate between networks of different topologies. Since larger networks take longer to optimise and there is no unique network representation of approximating the control policy function, networks within the population are grouped based on their topological similarities. Given two networks, we can denote E as the number of excess edges, D the number of disjoint edges between the 2 networks and $\overline{W}$ the average weight difference between matching genes. The equation used to determine the similarity between

networks is,

$$\delta = \frac{c1E}{N} + \frac{c2D}{N} + c3\overline{W}. \tag{3.16}$$

The constants $c1, c2, c3$ are weight parameters which allow for adjusting the significance of each term. The variable N is the maximum number of genes between the 2 candidate networks. If the similarity value computed using equation (3.16) is less than some threshold value δ_{th}, it is possible to conclude that the 2 networks are similar. Once the networks have been grouped together, the candidate network within each species group is sorted and the bottom half of the networks within each group is eliminated. The remainder of networks is then used to generate new candidate networks using the crossover operator.

3.7.5 Algorithm

A general algorithmic process of NEAT is given in Algorithm 6. Due to truncation issues encountered when dealing with the cart pole simulator, the fitness function which was a minimisation problem was converted to a maximisation problem with a penalty parameter. with L_i defined using equation 3.17

$$L_i(X_i, u_i) = R_i - X_i[3]^2 \tag{3.17}$$

Where $R_i = 1$ when X_i satisfies equations 3.10,3.11 and $R_i = -1$ otherwise

Algorithm 6 Pseudo-code for NEAT

```
1:  procedure NEAT
2:      for each agent i = 1, ..., N do
3:          Initialise agent_i's genome with input and output nodes only
4:          Randomly create connections between the input and output nodes
5:          Assign a fitness value to agent_i
6:          Add agent to population set
7:      while termination criterion not met do
8:          Divide agents into subspecies using speciation
9:          Sort the agents within each subspecies based on their fitness
10:         Eliminate bottom half of agents within each subspecies from population
11:         Select agents within population for crossover
12:         Generate new agents using selected candidates
13:         Update the population of agents
14:         Mutate subset of population
15:         Update the fitness scores of the agents
16:     return Fittest member within the population
```

Chapter 4

Results

In the following chapter results obtained using the various control mechanisms is recorded. The information recorded consists of the performance of the controller. The metrics used to measure the performance were the average error, average response time and where applicable, the average learning rate.The response time of a control scheme was measured by determining the time taken for the control scheme to return a control response after state information is given to the control scheme.These results are tabulated and portrayed through figures for comparison. Following the results, a detailed discussion is provided.

4.1 PID Results

In the following section comparisons of the PID controller with different parameters are made.Information regarding implementation of the PID controller can be found in section 3.4.

4.1.1 Results

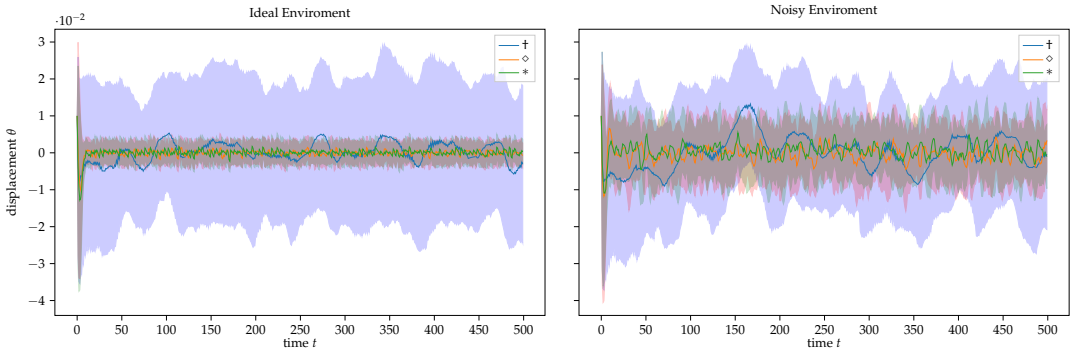

FIGURE 4.1: Comparison of PID parameters in Table 4.1 with mean displacement & error margin. This graph was achieved by evaluating the behaviour of the controller after parameter optimisation

PID parameter Comparison									
Parameters				**Ideal**			**Noisy**		
	K_P	K_I	K_D	mean error	mean std	mean response time (ms)	mean error	mean std	mean response time (ms)
†	0.5	1	1	0.0021	0.0024	0.0118	0.0038	0.0026	0.0107
◇	6.9387	-1.5199	0.3818	0.0006	0.0021	0.0111	0.0015	0.0022	0.0105
∗	10.4830	-1.9495	0.5857	0.0006	0.0020	0.0109	0.0015	0.0020	0.0110

TABLE 4.1: PID Parameter comparison. The above results where obtained after parameter optimisation. The PID control scheme was repeatedly run and the average performance was recorded.highlighted row is the best result

4.1.2 Discussion

Throughout the experiment the parameter configuration † was manually selected whilst $\diamond, *$ were obtained through parameter optimisation. Detailed discussion on how the parameters where optimised can be found in Chapter 3. When comparing the mean error in Table 4.1 and the plots in Figure 4.1 it is observed that the accuracy of the control response is highly dependent on the parameter selection, meaning parameter optimisation is a vital step in setting up an effective and efficient controller. The main downside of the PID controller is its poor noise handling. As seen in Table 4.1 and the plots in Figure 4.1 it is apparent that performing parameter optimisation reduces the capability of the PID controller to handle noise. On average the mean error between the ideal and noisy environment was 150%. Considering the mean response time in Table 4.1, it is reasonable to conclude that the parameter selection has minimal effect on the control response of the controller.

4.2 MPC Results

In the following section, comparisons of the MPC controller with different prediction horizon k values. Dynamic programming(DP) & Genetic Algorithm(GA) was used to determine the control response. For Genetic Algorithm population size $N = 10$, number of generations $m = 20$ and elite sample size $\rho = 2$ is used to determine the optimal control response. The above parameters were selected in comparison to others allowing for lower control response time and lower mean displacement to other parameters tested.

4.2.1 Results

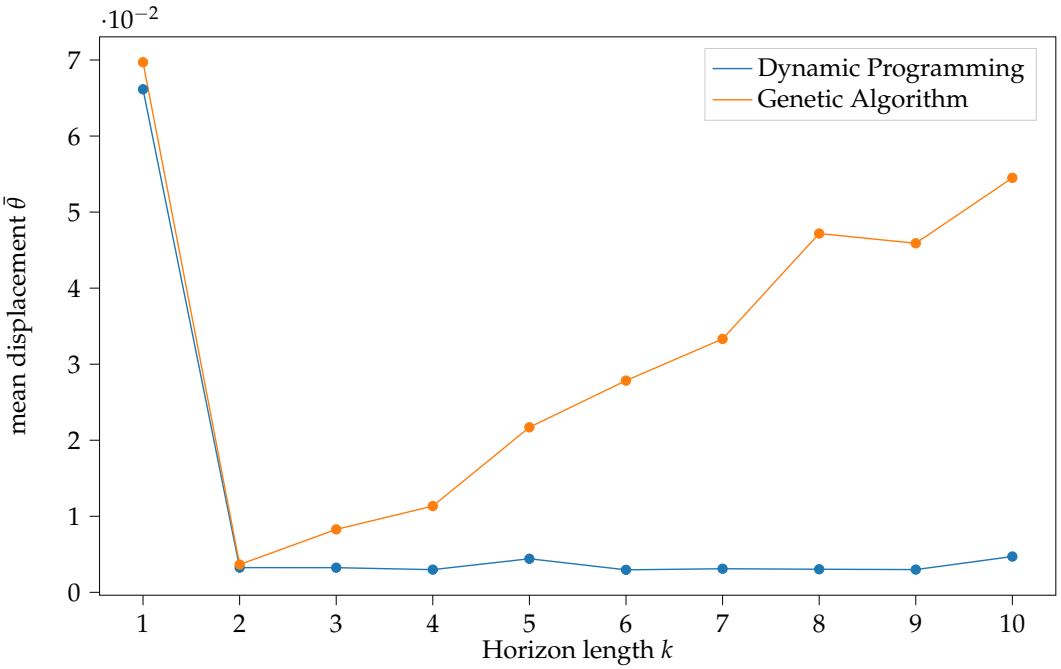

FIGURE 4.2: Mean error of MPC controller with different k values.

MPC parameter Comparison (Dynamic Programming)							
Parameters		**Ideal**			**Noisy**		
	k	mean error	mean std	mean response time (ms)	mean error	mean std	mean response time (ms)
†	1	4.5628	20.4859	0.0592	3.3033	20.0617	0.0625
◇	5	0.0010	0.0028	1.2646	0.0011	0.0028	1.2890
∗	10	0.0010	0.0031	42.9945	0.0012	0.0021	42.1758

TABLE 4.2: Comparison of the different parameters within the MPC control scheme, which is optimised using DP. The highlighted row is the best result.

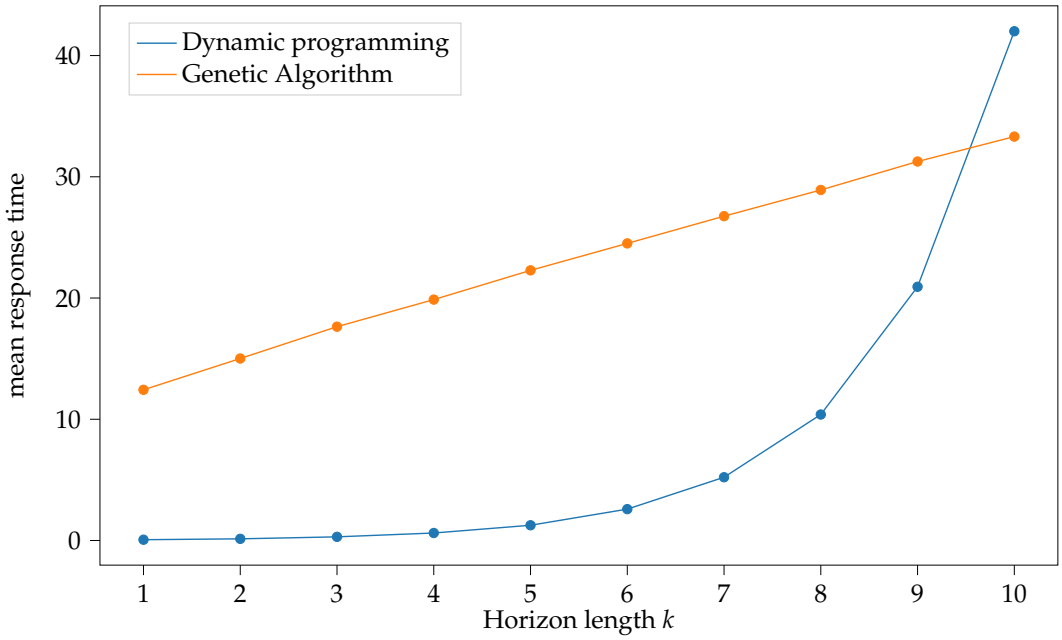

FIGURE 4.3: mean response time of MPC controller with different k values

MPC parameter Comparison (Genetic Algorithm)							
Parameters		**Ideal**			**Noisy**		
	k	mean error	mean std	mean response time (ms)	mean error	mean std	mean response time (ms)
†	1	1.1550	17.9542	12.8167	2.5016	17.8009	12.4244
◇	5	0.0054	0.0052	22.9147	0.0051	0.0040	22.0166
∗	10	0.0093	0.0115	33.9767	0.0136	0.0093	32.6021

TABLE 4.3: Comparison of the different parameters within the MPC control scheme, which is optimised using GA. The highlighted row is the best result.

4.2.2 Discussion

MPC has a limited amount of hyper-parameters in comparison to the other controllers discussed within this research, making it ideal for quick deployment of controller design due to the fact that not much preparation is required to construct an optimal controller. As seen in Figure 4.2, increasing the prediction horizon k results in improved behavior of the system. The trade-off of increasing k can be seen in Figure 4.3 which illustrates that increasing the prediction horizon k results in higher response times. For longer prediction horizons it would be better to use stochastic global optimisation in comparison to deterministic algorithms as seen in Figure 4.3, the response time when using DP increases exponentially whilst the response time whilst using stochastic algorithms such as GA increases linearly making it a better optimiser for determining the control response in comparison to DP over longer horizons. When taking the results in Table 4.2 and Table 4.3 the difference between the mean error under ideal and noisy environments is relatively small indicating that

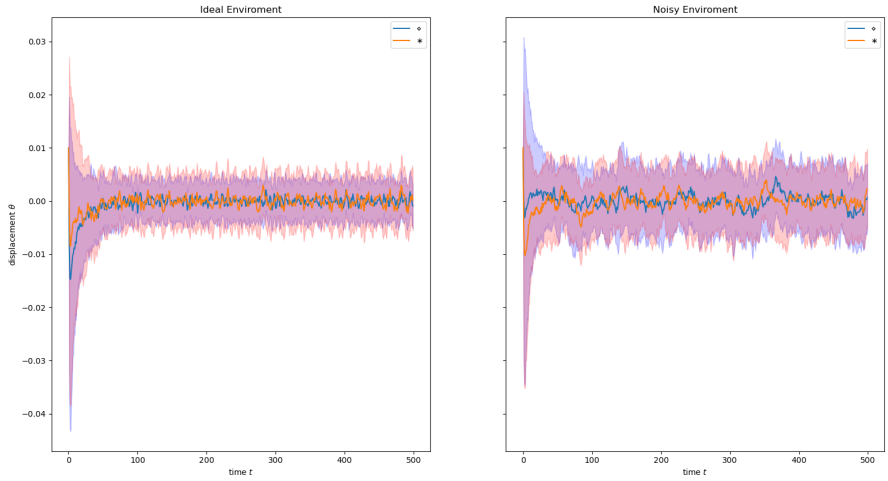

FIGURE 4.4: Comparison of MPC parameters in Table 4.2 with mean displacement. The results for $k = 1$ where omitted from the graph due to the the large standard deviation and error making the other 2 schemes not visible

FIGURE 4.5: Comparison of MPC parameters in Table 4.3 with mean displacement.The results for $k = 1$ where omitted from the graph due to the the large standard deviation and error making the other 2 schemes not visible

the MPC controller is effective at noise handling, which is important, particularly in engineering systems where a control response may result in devastating effects.

4.3 NE Results

In this section an ANN is with dimensions $[4, 4, 1]$ is optimised using evolutionary algorithms. The algorithms used to optimise the network are GA & PSO. For comparison purposes. The maximum number of generations allowed is 200 generations.

4.3.1 Results

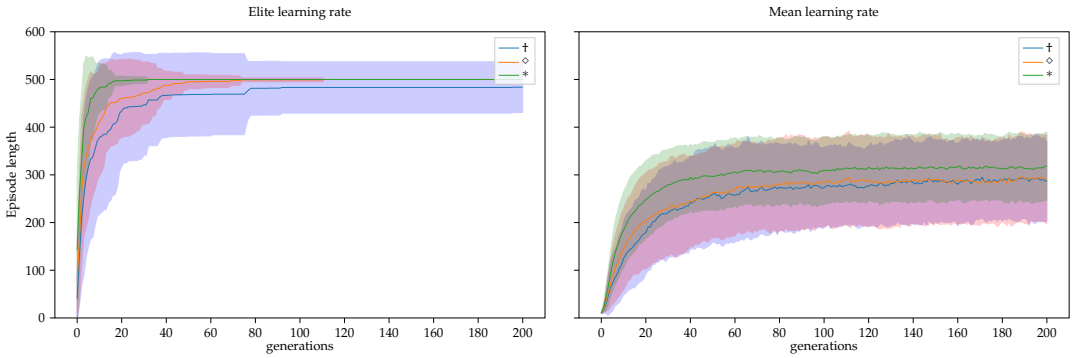

FIGURE 4.6: Comparison of the Learning rate when optimising network using PSO using parameters in Table 4.4. The following graph was obtained by running the training process several times and plotting the average learning rate with the standard deviation. The elite learning is rate represents the performance of the best ANN at each generation and the mean learning rate is the average performance of the population

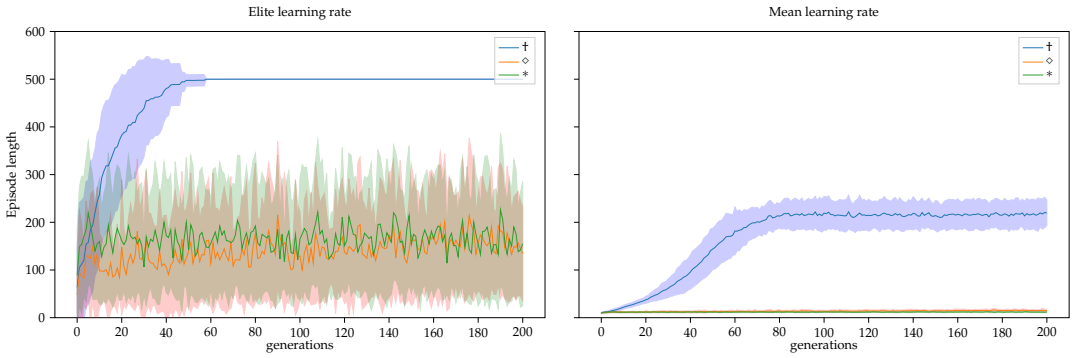

FIGURE 4.7: Comparison of the Learning rate when optimising network using GA using parameters in Table 4.5.The following graph was obtained by running the training process several times and plotting the avera learning rate with the standard deviation. The elite learning is rate represents the performance of the best ANN at each generation and the mean learning rate is the average performance of the population

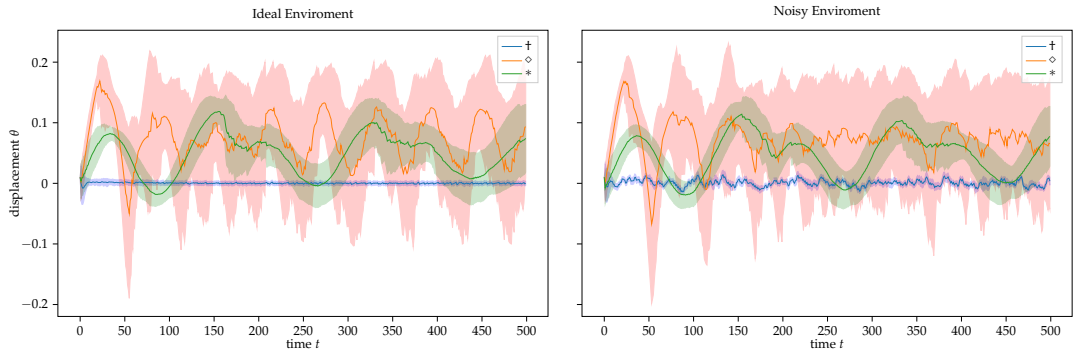

FIGURE 4.8: Comparison of NE parameters in Table 4.5. The following results where obtained by evaluating the ANN obtained after training/parameter optimisation several times and averaging the behaviour of the system using the NE control scheme optimised using GA

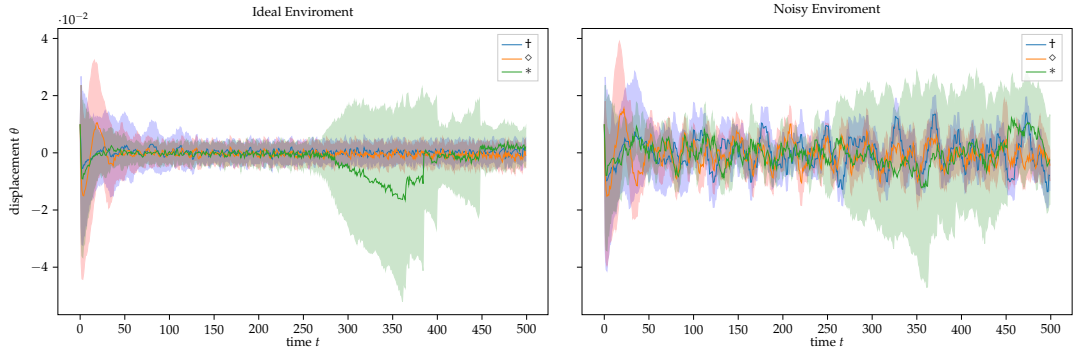

FIGURE 4.9: Comparison of NE parameters in Table 4.4. The following results where obtained by evaluating the ANN obtained after training/parameter optimisation several times and averaging the behaviour of the system using the NE control scheme optimised using PSO

NE parameter Comparison (Particle Swarm)										
Parameters					**Ideal**			**Noisy**		
	N	m	c_1	c_2	mean error	mean std	mean response time (ms)	mean error	mean std	mean response time (ms)
†	50	200	2	2	0.0008	0.0035	0.0349	0.0044	0.0041	0.0365
◇	100	200	2	2	0.0013	0.0046	0.0348	0.0035	0.0038	0.0361
∗	200	200	2	2	0.0028	0.0087	0.0344	0.0034	0.0081	0.035

TABLE 4.4: NE Parameter comparison using PSO.The above results where obtained by evaluating the performance of the model 30 times after training/parameter optimisation. highlighted row is the best result

4.3.2 Discussion

As seen in the results two stochastic optimisation algorithms where used to optimise the neural network. GA has the same amount of hyper-parameters as PSO, although

NE parameter Comparison (Genetic Algorithm)										
Parameters					**Ideal**			**Noisy**		
	N	m	ρ	μ_m	mean error	mean std	mean response time (ms)	mean error	mean std	mean response time (ms)
†	100	200	30	0	0.0008	0.0023	0.0373	0.0041	0.0024	0.0387
◇	100	200	30	0.3	0.0744	0.0260	0.0345	0.0745	0.0256	0.0372
∗	200	200	20	0.3	0.0489	0.0111	0.0353	0.0483	0.0087	0.0358

TABLE 4.5: NE Parameter comparison using GA. The above results where obtained by evaluating the performance of the model several times after training/parameter optimisation. The highlighted row is the best result

2 of the hyper-parameters c_1, c_2 are not problem specific. In literature it is usually advisable to to set $c_1 = c_2 = 2$. This simplifies hyper-parameter optimisation in comparison to GA. Comparing Figures 4.6 and 4.7 we can conclude that the elite agent within the PSO population learnt a sub-optimal solution at a faster rate than the elite agent in the GA population. This may be due to the fact that PSO generally explores the parameter space more than GA does. This claim is clear when looking at the mean learning rate of the population, GA performs worse in comparison to PSO. After a couple of iterations both algorithms converge to an episode length under 500. As defined in the objective of the project this is clear that the mean learning rate does not find a sub-optimal solution. This could be due to the fact that majority of the population converges to a local optima, which stochastic optimisation algorithms are not immune to. A possible resolution of this issue is to penalise candidate agents for being too close to other agents within parameter space, this may allow for the parameter space to be explored properly. When comparing the mean error results in Table 4.4, 4.5 it is observed that in both ideal and noisy environments optimising the ANN using PSO yields better results. Under noisy environments the mean error of the ANN optimised using PSO increased on average by 450%, whilst with GA the mean error had increased by 412%. This may be due to PSO over-fitting. The difference between the response time between the two methods are negligible. Both methods where used to optimise the same network structure so this is expected.

4.4 NEAT Results

NEAT is an extension of NE, with the main focus being the optimisation of the network topology. Thus no specified network structure was given to the algorithm. The only information given to the network was the number of input and output nodes, which are 4 and 1 respectively. The maximum number of generations allowed for these experiments where 30 generations as increasing this value resulted in overfitting.

4.4.1 Results

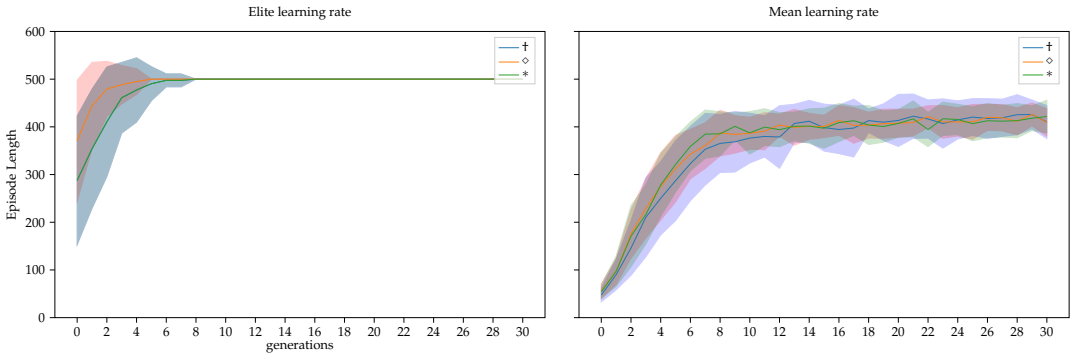

FIGURE 4.10: Learning rate of NEAT. The following graph was obtained by running the training process 30 times and plotting the mean learning rate with the standard deviation. The elite learning is rate represents the performance of the best ANN at each generation and the mean learning rate is the average performance of the population.

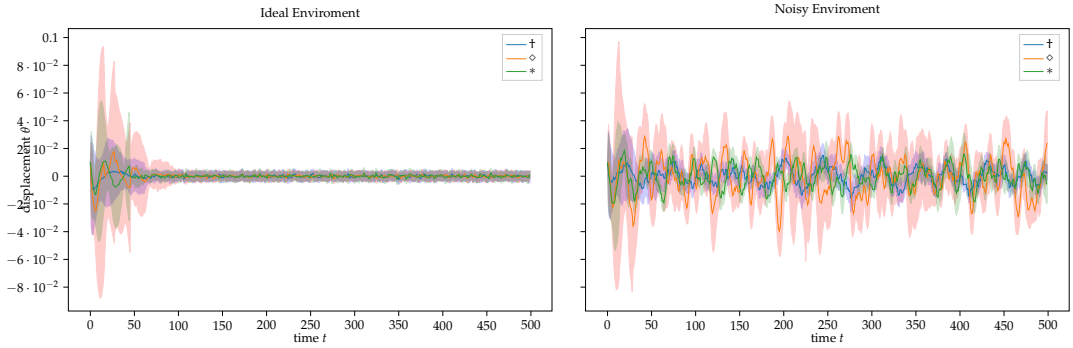

FIGURE 4.11: Comparison of NEAT parameters in Table 4.6.The following results where obtained by evaluating the ANN obtained after training/parameter optimisation 30 times and averaging the behaviour of the system using the NEAT control scheme.

4.4.2 Discussion

Observing the results it was determined that NEAT was able to learn a good approximation within 5-10 iterations. When taking into account Figure 4.10, NEAT was able to determine a sub-optimal solution within on average 5 generations. This shows that by both optimising the weights and structure the neural networks in the population are able to make good approximations of the control function. The downside

NEAT parameter comparison														
	Parameters							**Ideal**			**Noisy**			
	N	m	c_1	c_2	c_3	μ_l	μ_e	μ_n	mean error	mean std	mean response time (ms)	mean error	mean std	mean response time (ms)
+	20	30	1	1	0.4	0.4	0.4	0.2	0.0008	0.0048	0.0123	0.0049	0.0040	0.0172
◇	40	30	1	1	0.4	0.4	0.4	0.2	0.0014	0.0141	0.0118	0.0107	0.0105	0.0166
*	30	30	1	1	0.4	0.4	0.4	0.2	0.0011	0.0072	0.0119	0.0065	0.0047	0.0171

TABLE 4.6: NEAT Parameter comparison.The above results where obtained by evaluating the performance of the model 30 times after training/parameter optimisation. highlighted row is the best result

to NEAT is the number of hyper parameters required to be tuned. Looking at the behaviour of the controller under noisy conditions, the error of the controllers on average increased by 512 %. From that result it can concluded that NEAT may not perform well at error handling. The accuracy of the controller is highly dependant on the accuracy of the measurements sampled

4.5 Comparison of methods

In the following section, the best result of each of the proposed control schemes are used to make comparisons.

4.5.1 Results

Comparison of different control schemes						
Method	**Ideal**			**Noisy**		
	mean error	mean std	mean response time (ms)	mean error	mean std	mean response time (ms)
PID	0.0006	0.0020	0.0109	0.0015	0.0020	0.0110
MPC (DP)	0.0010	0.0028	1.2646	0.0011	0.0028	1.2890
MPC (GA)	0.0054	0.0052	22.9147	0.0051	0.0040	22.0166
NE (PSO)	0.0008	0.0035	0.0349	0.0044	0.0041	0.0365
NE (GA)	0.0008	0.0023	0.0373	0.0041	0.0024	0.0387
NEAT	0.0008	0.0048	0.0123	0.0049	0.0040	0.0172

TABLE 4.7: Comparison of best results obtained by the different control schemes

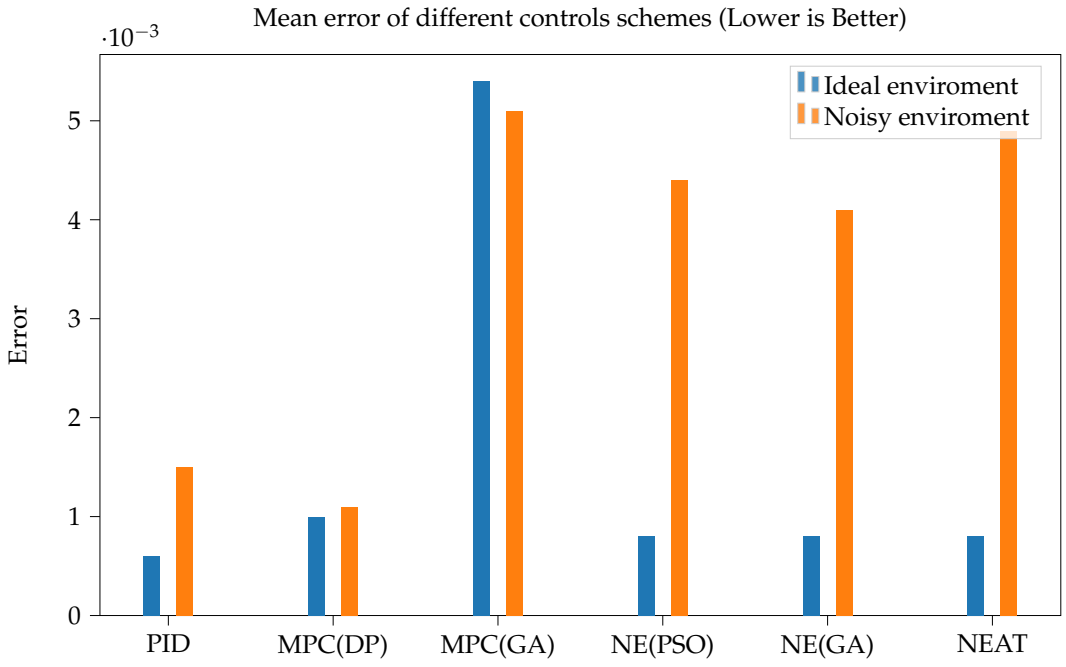

FIGURE 4.12: Mean error of proposed methods in Table 4.7

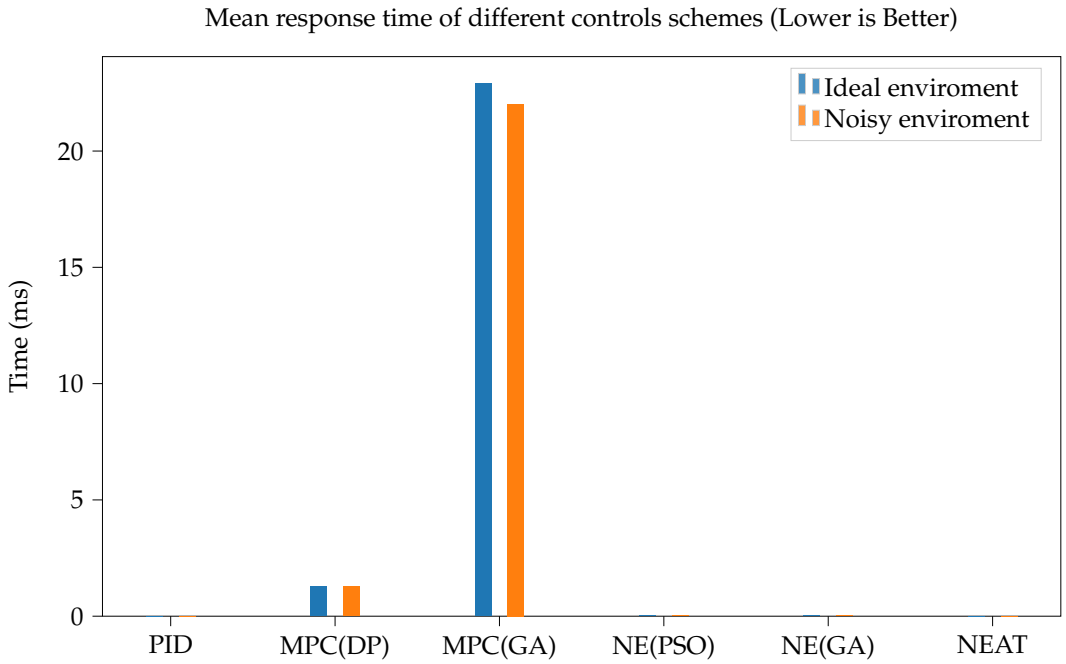

FIGURE 4.13: Mean response time of proposed methods in Table 4.7

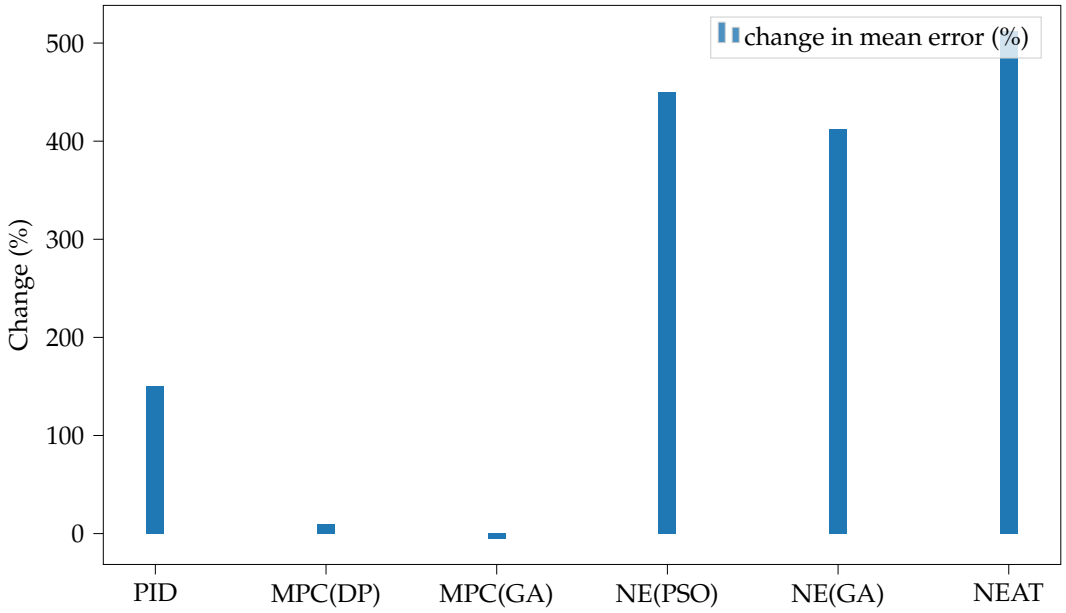

FIGURE 4.14: Change in mean error between noisy and ideal conditions of proposed methods in Table 4.7

4.5.2 Discussion

Under ideal conditions PID performed the best, with an mean error of 0.006 as seen in Table 4.7. This may be due to the simplicity of the controller. The drawback of the PID controller is that it does not scale well in terms of system complexity, Hence for more complex systems it is generally advised not to use the PID controller. Although the PID controller performed well under noisy conditions MPC proved to be more effective. When taking into consideration the percentage change of the mean error between noisy and ideal conditions in Figure 4.14, MPC had the lowest change.

from this result we may conclude that MPC is effective at noise handling. The disadvantage of the MPC control scheme as seen in Figure 4.13 is the response time. MPCs dependence of a mathematical model makes it effective at noise handling, but is computationally expensive. This is a major drawback as depending on the system, the state of the system may change drastically during the time MPC takes to compute the control response. This may lead to devastating impacts. MPC with GA as an optimiser improved under noisy conditions. this may be due to the stochasticity introduced by GA which allows the control scheme to improve under noisy conditions. NE and NEAT performed similarly under both ideal and noisy conditions. But when comparing the 2 algorithms learning rates it quite clear that NEAT was able to obtain a good approximation in 5 generations (Figure 4.10) whilst with NE it takes about 40 generations (Figure 4.6,4.6). From this we can conclude that the topology of the network plays an impact on how quickly the network learns to make an approximation of the control function. NE and NEAT are good alternatives to standard reinforcement learning algorithms as when comparing the following the results obtained by in [19], NE and NEAT outperformed standard RL algorithms with networks a fewer amount of neurons. This may be due to the fact that with NE you are directly optimising a loss function over a time horizon whilst with RL techniques learning is incremental.

Chapter 5

Conclusion

Recent research in control theory has provided a multitude of control schemes which traditionally could be classified as either model-based or model-free. In practise PID & MPC are generally used but are slowly being replaced with data-driven control schemes such as Reinforcement Learning & Neuroevolution. This sudden rise in use cases may be attributed to the recent interest in deep learning. One major benefit of model-free techniques is not much information is required to design an efficient controller, with the trade-off of this being a lot of simulation data is required in order to design these control schemes. Although Model-based controllers require human expertise's, They are quick to implement and have certain guarantees such as local optimally. Local optimally is extremely important as within certain systems it is possible to guarantee safety. Similarities between the control schemes, particularly between model-based and model-free control schemes are limited, Which makes comparisons between the control schemes generally difficult. Possible future work may be looking into designing hybrid models, which make use of data-driven & model based techniques in order to design efficient control schemes. This may provide the ease of implementation and practicality associated with data-driven control schemes and the theoretical guarantees associated with model-based control schemes. In the sections that follow possible improvements to the control schemes provided in this report are given which will then be followed by alternative control schemes.

5.1 Possible Improvements

5.1.1 PID

One major disadvantage as discussed earlier is that the integral term in the PID controller tends to grow exponentially as time moves forward, resulting in the integral term having a larger weighting in computing the control response. As a result two possible remedies to the issue above are discussed below.

Clamping

Clamping involves removing the integral term from the computation of the control response if the term exceeds a certain threshold. In practise this has shown to be very effective at reducing error.

Dampening

The integral term assumes that all previous errors have an equal weighting which is problematic as realistically decisions made at 100 previous time steps may have little

or no effect on the current state of the system. In order to rectify this a weight function as seen in equation 5.1 $w(\tau)$ is introduced such that $\int_0^t w(\tau)d\tau = 1$. Utilising this weight function it is possible to favour more recent errors.

$$u(t) = K_p e(t) + K_i \int_0^t w(\tau)e(\tau)d\tau + K_d \frac{de(t)}{dt}. \tag{5.1}$$

5.1.2 MPC

MPCs major disadvantage, as discussed above is it is computationally expensive determining the control response, especially when the governing equations are nonlinear. In order to remedy this a more computationally efficient model can be used within the MPC framework in order to make future state predictions. Two possible replacements are provided below.

Reduced Order Models

Reduced order models are models that reduce the computational complexity of obtaining numerical solutions, with the drawback being a reduction in the accuracy of the model. These models are generally favoured in practise as it drastically reduces the time taken to obtain results. It is commonly used in simulations but can be used in the context of model-based control theory as it allows model based frameworks to be used whilst reducing the computation expense. Commonly used techniques are

- Proper generalized decomposition
- Matrix interpolation
- Transfer function interpolation
- Piecewise tangential interpolation
- Loewner framework
- (Empirical) cross Gramian
- Krylov subspace methods

Physics Informed Neural Networks

Physics Informed Neural Networks (PINNS) are a class of ANNs that attempts to interpolate across data whilst satisfying a governing differential equation. This is usually done by setting up a loss function such that it minimises the error within the data set and reduces the error when substituting the solution into the differential equation. For example if we denote $f = u_t + L(u) + N(u) = 0$ to be some partial differential equation, where $L(u), N(u)$ are operators on u with $L(u)$ being a linear operator and $N(u)$ being a nonlinear operator. Utilising this expression the loss function of the network can be defined as:

$$Loss = Loss_{MSE} + f^2 \tag{5.2}$$

where $Loss_{MSE}$ is the Mean Squared Error of the network on the data set. f^2 is usually computed by using automatic differentiation on the output of the neural network.

5.1.3 Neuroevolution

Across literature there have been multiple improvements that have been introduced to increase learning rate and reducing computational expense. In this section 2 major improvements are discussed.

Novelty Search

Novelty search is a technique that ensures agents are not close to each other in parameter space. This is done by adding an additional term to the fitness function. This term which is commonly known as the novelty score is the distance of the network in parameter space to its nearest neighbour. By maximising this augmented fitness functional the algorithm ensures the fitness functional is optimised and that solutions do not tend to the same point in parameter space.

Encoding schemes

Encoding schemes attempt to transform the parameter space of the network into a latent space with the goal being a reduction in dimensionality. The population of networks is then optimised within this parameter space. This is extremely useful as it allows for the construction of deep neural networks at the computational cost of shallow networks.

5.2 Spiking Neural Networks

A more computationally feasible control scheme in introduced in the following section. Spiking Neural Networks (SNNs) are a subset of Artificial Neural Networks(ANNs) that mimic the brain more closely compared ANNs. instead of continuous values being propagated through the network, a series of binary values a passed through the network. in the literature [20], the binary information represents whether the voltage of a neuron exceeds a threshold. Due to the fact that binary information is propagated through the network makes SNNs less computationally expensive in comparison to the standard ANNs. A neuron is a biological cell that communicates with other cells using Dendrites (inputs) and Axons (outputs). An activation function (modeled as a differential equation) is used to determine the voltage of the neuron at a given time based on the input signals received. Many mathematical models have been proposed but the commonly used model in SNNs is the Leaky-Integrate-and Fire (LIF) Model [21]. SNNs are a promising mathematical model that can be used for controller design due to the activation functions used within this network. The differential equations used to describe the behavior of the neurons are usually with respect to the time, because of this we can assume that the processing of information within this network is temporal. Since the goal of control theory is to optimize systems that vary over time this model is assumed to be a good match. Although SNNs seem promising, they are rarely used in application due to the training difficulty. Weights including neuron parameters such as membrane capacitance and potential difference threshold are needed to be determined.

5.2.1 Leaky-Integrate-and Fire Model

The LIF neuron model can be described by using the following differential equation

$$C_m \frac{dV_m}{dt} = G_L(E_L - V_m) + I_{ext}, \tag{5.3}$$

Where C_m is the Membrane Capacitance, which is essentially the amount of charge the neuron can hold. V_m is the potential difference across the membrane whilst E_L is the potential difference the membrane returns to after any temporary charge leaks away. G_L represents the magnitude of charge that could leak in and out of a neuron. I_{ext} represents external currents applied to the system. if V_m increases larger than some threshold (denoted as $V_{threshold}$), then the potential difference of the neuron will tend towards the resting potential (denoted as V_{rest}). A further discussion on the model could be found in [21].

Appendix A

Appendix

A.1 Proof of Bellman's Principle of Optimality

let $x_A, x_B, x_C \in S$.

where S is the state space

x_C is an intermediate state from x_A to x_B.

it is required to prove $\min J_{ab} = \min J_{ac} + \min J_{cb}$.

assuming $J_{ab} = J_{ac} + J_{cb}$ is the optimal state trajectory from x_A, x_B and

$\vec{J_{ac}} < J_{ac}$

$\vec{J_{cb}} < J_{cb}$

where $\vec{J_{ac}}, \vec{J_{cb}}$ are alternative paths from a to c,c to b with lower transition costs.

since $\vec{J_{ac}} < J_{ac}$

$$\begin{aligned} \vec{J_{ac}} &< J_{ac} \\ \vec{J_{ac}} + J_{cb} &< J_{ac} + J_{cb} \\ \vec{J_{cb}} + J_{cb} &< J_{ab} \end{aligned}$$

Which contradicts J_{ab} being the minimum path. Hence J_{ac} must be the minimum path from a to c.

since $\vec{J_{cb}} < J_{cb}$

$$\begin{aligned} \vec{J_{cb}} &< J_{cb} \\ J_{ac} + \vec{J_{cb}} &< J_{ac} + J_{cb} \\ J_{ac} + \vec{J_{cb}} &< J_{ab} \end{aligned}$$

Which contradicts J_{ab} being the minimum path. Hence J_{cb} must be the minimum path from c to b. Therefore by proof by contradiction $\min J_{ab} = \min J_{ac} + \min J_{cb}$.

A.2 Proof Bellman's Equation satisfies bellman optimality

A.2.1 Bellman operator

the bellman operator $T(V)(x)$ is defined as

$$T(V)(x) = \min_u (F(x,u) + \boldsymbol{E}V(f(x,u))) \tag{A.1}$$

where $x \in S$ and $V : S \to \mathbb{R}$, and $f(x,t)$ is the transition dynamics of the system the optimal value function is defined as V_t^*

A.2.2 Proof

suppose $v_t^* = T(v_{t+1}^*)$ and at terminal time T, $v_T^* = g_T$, where g_T is the value of the terminal state. for a given control policy u, $v_T^u = g_T$ and $v_t^u = F(x,u) + \boldsymbol{E}v_{t+1}^u(f(x,u))$ for all $t = T-1, \cdots, 0$. It is required to prove that if

$$v_t^* \leq v_t^u$$

then $J^* \leq J^u$

Given $v_t^* = T_t(v_{t+1}^*)$, $v_t^u \geq T(v_{t+1}^u)$ and $v_T^* = v_T^u = g_T$

$$\begin{aligned} v_t^u &\geq T_t(v_{t+1}^u) \\ &\geq T_t T_{t+1}(v_{t+2}^u) \\ &\vdots \\ &\geq T_t T_{t+1} \cdots T_{T-1}(v_T^u) \\ &\geq T_t T_{t+1} \cdots T_{T-1}(g_T) \\ &\geq v_t^* \end{aligned}$$

Therefore we may conclude that if $v_t^* \leq v_t^u$ then $J^* \leq J^u$.

A.3 Universal Function Approximator

Before illustrating the proof it is important to highlight some of the required definitions

A.3.1 Borel measure

a borel set is any set that can be formed from open sets through finite unions,intersections and relative complements

an outer measure μ is a borel regular measure if assuming B is a borel set

$$\forall A \subseteq \mathbb{R}^n, \exists B \subseteq \mathbb{R}^n$$

such that

$$A \subseteq B, \qquad \mu(A) = \mu(B)$$

Let $M(\Omega)$ be the borel regular measure on Ω

A.3.2 Hahn-Banach Theorem

every contionus linear functional defined on a subspace of a normed space X has a continuous extension to the whole of X

A.3.3 Reiz representation theorem

let H be a Hilbert Space whos inner product is linear in the first argument and anti-linear in its second argument
for all continous function $\psi \in H^*$ there exists a unique vector $f_\psi \in H$ such that

$$\psi(x) =< x, f_\psi >=< f_\psi | x >$$

A.3.4 discriminatory functions

A function σ is said to be discriminatory if for $\mu \in M(\Omega)$

$$\int_\Omega \sigma(w^T x + b) d\mu(x) = 0$$

for all $w \in \mathbb{R}^n$,$b \in \mathbb{R}$

A.3.5 sigmoidal functions

a function is σ is said to be sigmoidal if

$$\sigma(x) \to \begin{cases} 1 & x \to \infty \\ 0 & x \to -\infty \end{cases}$$

A.3.6 Proof

Let σ be a continuous discriminatory function, $C(\Omega)$ be the set of continuous functions on Ω.
Given a function $f \in C(\Omega)$, $\epsilon > 0$, there exists a sum $G(x)$ of the form

$$G(x) = \sum_{i=1}^{N} \alpha_i \sigma(w_i^T x + b_i)$$

for which

$$|G(x) - f(x)| < \epsilon$$

Let $S \subset C(\Omega)$ be the set of functions in the form of $G(x)$
it is required to prove the closure of S, denoted as R is $C(\Omega)$

assuming R is not $C(\Omega)$ this implies R is a closed subspace on $C(\Omega)$.
by the Hahn-Banach theorem, there exists a bounded linear functional on $C(\Omega)$, denoted as L such that $L(R) = L(S) = 0$. given $L \neq 0$.
By the Riesz representation theorem

$$L(h) = \int_{\Omega} h(x) d\mu(x)$$

where $\mu \in M(\Omega)$, $\forall h \in C(\Omega)$.

since $\sigma(w^T x + b) \in R$, $\forall w, b$

$$\int_{\Omega} \sigma(w^T x + b) d\mu(x) = 0$$

which is only true if $\mu = 0$ which is a contradiction since σ is discriminatory.
therefore we may conclude that S is dense in $C(\Omega)$ which then implies there exists $G(x)$ such that

$$|G(x) - f(x)| < \epsilon$$

Appendix B

Additional Background

B.0.1 State Space

The State Space of a system refers to the set of all possible configurations the system could be in. The State Space can be split into two categories which are described as discrete state spaces and continuous state spaces.

Discrete State Spaces are usually defined as the follows

$$M = \{X, E, S, G\}, \tag{B.1}$$

where X refers to the set of states. this usually represents the possible configurations the system may be in. E refers to the set of edges between the states. Elements of the set E usually represent the cost of transitioning from state a to state b, where $a, b \in X$. The set S, $S \subseteq X$ denotes the set of states the system may start in, and $G, G \subseteq X$ refers to the set of goal or termination states.

Continuous State Spaces (in the context of Control Theory) are usually defined as $\mathbb{R}^n$ Manifolds. This representation is commonly used over Vector Spaces as difficulties arise when characteristics of the system change over time. An example of this in chemical engineering, the concentration of a substance may decrease during some chemical process. A manifold can be described as a topological space where the space within a neighborhood of a state resembles a euclidean space.

B.0.2 Action Space

An action space can be described as the set of all possible admissible control responses, which is usually donated as U. The formulation of the discrete and continuous action spaces is similar to that of the state space. Control Theory consists of methods that try to determine the curve or surface $u(t)$ on the manifold U that steers the system from a starting state to the desired state.

B.0.3 Time-Invariant and Time variant Systems

A system is time-invariant when the system does not depend on time. Given the state x regardless of what the value of t is the next possible state will remain the same. Time invariant systems are defined as follows:

$$\dot{x}(t) = f(x(t), u(t))$$

On the other hand, time-variant systems depend on time. the next possible state of the system is not only dependent on the current state, but also on the value of time. Issues may arise when the differential equation contains singular points as the behaviour of the system near these points are generally difficult to predict. Systems of this nature are usually defined as follows:

$$\dot{x}(t) = f(x(t), u(t), t)$$

Further discussion on time-invariant and time-variant systems can be found in [22].

B.0.4 Dynamical System Models

Dynamical systems are usually modelled based on how the given system changes over time. Commonly used representations are

$$\dot{x}(t) = f(x, u), \tag{B.2}$$

$$x_{t+1} = f(x_t, u_t), \tag{B.3}$$

where x denotes the state of the system and u denotes the control policy. Equation B.2 is a differential equation, which is commonly used to describe system behavior over a continuous time frame. On the other hand, equation B.3 is used to describe system behavior over fixed time intervals. These definitions of models could be modified to represent time-variant systems. A detailed discussion on the mathematical definition of dynamical systems could be found in [23].

B.1 Pontryagin's maximum principle

One of the earliest studies conducted in the field assumed and addressed the following, given $x \in X$ and $u \in U$ which control policy u would optimally steer $x_0 \in X$ to $x_T \in X$ given some associated cost.

B.1.1 The Hamiltonian

The Hamiltonian in the control theory setting functions in the same manner as the Lagrangian in constrained Optimisation problems by converting the constrained problem into an unconstrained problem. In the case of control the total cost of the system, equation 2.2 is dependent on the system model equation 2.3. The Hamiltonian is defined by

$$H(\boldsymbol{x}(t), \boldsymbol{u}(t), \boldsymbol{\lambda}(t), t) = L(\boldsymbol{x}(t), \boldsymbol{u}(t), t) + \boldsymbol{\lambda}^T(t) f(\boldsymbol{x}(t), \boldsymbol{u}(t), t). \tag{B.4}$$

Given B.4 it is possible to determine the optimal control policy using the following

$$\begin{aligned} \frac{\partial H}{\partial u} &= 0, \\ \frac{\partial H}{\partial x} &= -\frac{d\lambda^T}{dt}, \\ \lambda^T(t_1) &= \frac{d\phi}{dx}\Big|_{x=x(t_1)} \end{aligned} \tag{B.5}$$

The research conducted in [24] a major contribution to the field, but its limitations were reached when trying to determine control policies when equations B.2,B.3 is nonlinear.

Bibliography

[1] J. Berberich, J. Köhler, M. A. Müller, and F. Allgöwer, "Data-driven model predictive control with stability and robustness guarantees," *IEEE Transactions on Automatic Control*, vol. 66, no. 4, pp. 1702–1717, 2020.

[2] B.-Z. Guo and Z.-L. Zhao, *Active disturbance rejection control for nonlinear systems: An introduction*. John Wiley & Sons, 2016.

[3] S. Bennett, "Development of the pid controller," *IEEE Control Systems Magazine*, vol. 13, no. 6, pp. 58–62, 1993.

[4] P. Gawthrop, "Self-tuning pid controllers: Algorithms and implementation," *IEEE Transactions on Automatic Control*, vol. 31, no. 3, pp. 201–209, 1986.

[5] D. P. Bertsekas *et al.*, "Dynamic programming and optimal control 3rd edition, volume ii," *Belmont, MA: Athena Scientific*, 2011.

[6] D. C. Dracopoulos, "Genetic algorithms and genetic programming for control," *Evolutionary algorithms in engineering applications*, pp. 329–343, 1997.

[7] R. Neck, "Stochastic control theory and operational research," *European Journal of Operational Research*, vol. 17, no. 3, pp. 283–301, 1984.

[8] U. Rosolia, X. Zhang, and F. Borrelli, "Data-driven predictive control for autonomous systems," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 259–286, 2018.

[9] P. Dayan and Y. Niv, "Reinforcement learning: The good, the bad and the ugly," *Current opinion in neurobiology*, vol. 18, no. 2, pp. 185–196, 2008.

[10] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[11] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: From architectures to learning," *Evolutionary intelligence*, vol. 1, no. 1, pp. 47–62, 2008.

[12] R. Dawkins *et al.*, "Universal darwinism," *Evolution from molecules to men*, pp. 403–425, 1983.

[13] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[14] S. Nagendra, N. Podila, R. Ugarakhod, and K. George, "Comparison of reinforcement learning algorithms applied to the cart-pole problem," in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, IEEE, 2017, pp. 26–32.

[15] J. Vendrow, G. Meyerowitz, and R. Malavalli, "Ece239as (rl) final report: Ppo implementation for openai environments,"

[16] O. G. AI, *Open gym ai classical control libary*, https://www.gymlibrary.ml/environments/classic_control/, 2022.

[17] A. P. Engelbrecht, *Computational intelligence: an introduction*. John Wiley & Sons, 2007.

[18] T. Chen and H. Chen, "Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems," *IEEE Transactions on Neural Networks*, vol. 6, no. 4, pp. 911–917, 1995.

[19] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," *arXiv preprint arXiv:1712.06567*, 2017.

[20] S. Ghosh-Dastidar and H. Adeli, "Spiking neural networks," *International journal of neural systems*, vol. 19, no. 04, pp. 295–308, 2009.

[21] Y.-H. Liu and X.-J. Wang, "Spike-frequency adaptation of a generalized leaky integrate-and-fire model neuron," *Journal of computational neuroscience*, vol. 10, no. 1, pp. 25–45, 2001.

[22] R. Melchers, "Simulation in time-invariant and time-variant reliability problems," in *Reliability and Optimization of Structural Systems' 91*, Springer, 1992, pp. 39–82.

[23] J. C. Willems, "Paradigms and puzzles in the theory of dynamical systems," *IEEE Transactions on automatic control*, vol. 36, no. 3, pp. 259–294, 1991.

[24] L. S. Pontryagin, *Mathematical theory of optimal processes*. CRC press, 1987.