

Statistical Optimization of CNN-LSTM Network Architectures: A Case Study in Autonomous Vehicle Control

by

Cameron Bentley

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Mechanical and Mechatronics Engineering

Waterloo, Ontario, Canada, 2024

© Cameron Bentley 2024

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This thesis introduces a novel framework for optimizing combined Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) architectures for kinematic control problems, with a specific focus on autonomous vehicle control, chosen for its combination of dynamics and scene recognition which bare similarities to other complex controls problems faced in mechatronics engineering.

A combined dataset and high-fidelity simulation environment is implemented using an an off-the-shelf game engine, a novel approach in the literature which is traditionally limited by the quality of openly available simulation environments, and enabling a hybrid approach of training neural network models via both Imitation Learning and Reinforcement Learning.

A comprehensive exploration of network structures and hyperparameters is undertaken using the Tree-structured Parzen Estimator (TPE) to systematically improve model performance, enabling more informed approaches to neural network structure and design. The research demonstrates the impact of varying temporal and spatial information through varying the emphasis on the CNN and LSTM layers of the network respectively, as well as the amount of context provided to the network.

The findings and methodology are adaptable to other problems in the kinematic optimization control space, and the particular similarities of other problems in the area are discussed.

Acknowledgements

I would like to thank Professor H. J. Kwon for his guidance and support throughout my studies, and for providing me with the opportunity to explore an area of research of personal interest. Additionally, I extend my gratitude to my office mate Dr. Moslem Sadeghi Goughari for his insights into properly choosing a project direction and ensuring I leave behind something that is extensible by future researchers.

I would also like to thank my supports outside of school, including my parents and partner Helen who have made the completion of this work possible. Your encouragement made me push on through a tumultuous two years and see this work to its conclusion and I could not have done it without you.

Table of Contents

Author's Declaration	ii
Abstract	iii
Acknowledgements	iv
List of Figures	viii
List of Tables	ix
List of Abbreviations	x
1 Introduction	1
1.1 Project Motivation	1
1.2 Neural Networks as a Controls Solution	2
1.3 Project Objectives	3
1.4 Contributions to the Literature	4
1.5 Broader Project Context	4
1.6 Thesis Structure	5
2 Literature Review	6
2.1 Background on Kinematic System Problems	6

2.1.1	Redundancy and Multiple Solutions	6
2.1.2	State Space	7
2.1.3	Nonlinear Systems	8
2.1.4	Real-Time Constraints & Neural Network Applications	9
2.2	Kinematic Network Structures	9
2.2.1	Imitation Learning and Reinforcement Learning	9
2.2.2	Separation of Control and Perception	12
2.3	Optimization Studies Across Machine Learning	14
2.3.1	Kinematic Systems	14
2.3.2	Convolutional Neural Networks	15
2.4	Datasets and Environments	17
2.5	Summary	19
3	Dataset & Environment Development	20
3.1	Case Study Selection	20
3.1.1	Trajectory Planning for Robotic Arms	21
3.1.2	Path Planning for Autonomous Vehicles	21
3.1.3	Gait Optimization for Legged Robots	24
3.1.4	Comparison of Case Studies	25
3.2	Integrating with Commercial Game Engines	26
3.3	Selection of Simulator	27
3.3.1	Accuracy of Simulation	28
3.3.2	Programmatic Interaction	28
3.3.3	Scalability	29
3.3.4	Visual Similarity	29
3.3.5	Comparison	30
3.4	Recording Simulation State	31
3.4.1	Interpreting Simulation Memory	31

3.4.2	Recording Visual State	32
3.4.3	Control Inputs	32
3.5	Reinforcement Learning API	33
3.5.1	Reward Structure	33
3.6	Scalability	34
3.7	Dataset Generation	35
3.8	Summary	35
4	Training Process	38
4.1	Base Network Structure	38
4.2	Key Parameters for Optimization	40
4.3	Efficient Selection of Network Structures	41
4.3.1	Tree-structured Parzen Estimator (TPE)	42
4.3.2	Covariance Matrix Adaptation Evolution Strategy (CMA-ES)	43
4.3.3	Comparison	43
4.4	Library Selection	44
4.4.1	Training Library	44
4.4.2	Search Library	44
4.5	Network Restrictions & Resources	45
4.6	Training Process	46
4.6.1	Pre-trained EfficientNetB0+LSTM Architecture	48
5	Summary and Conclusions	49
5.1	Applicability to Other Kinematic Systems	49
5.2	Limitations and Future Work	50
5.3	Conclusion	51
	References	53
	Glossary	58

List of Figures

2.1	Denavit-Hartenberg Parameters for a Manipulator Link. Image from [8] . .	9
3.1	Search Process for Speed Memory Address	36
3.2	Client and Server Interfaces	37
4.1	CNN+LSTM network structure	39
4.2	Loss function of the network over time	46
4.3	Hyperparameter importance	47
4.4	Loss function of pretrained CNN network over time	48

List of Tables

3.1	Decision Matrix for Case Study Selection	26
3.2	Decision Matrix for Simulator Selection	30
4.1	Parameters for Convolutional Neural Network (CNN) optimization	40
4.2	Parameters for Long Short-Term Memory (LSTM) optimization	40
4.3	Parameters for data preprocessing optimization	41

List of Abbreviations

AI Artificial Intelligence [2](#), [27](#), [30](#), [35](#)

ANNs Artificial Neural Networks [2](#), [3](#), [10](#), [17](#), [19](#), [20](#), [22](#), [24](#), [25](#), [35](#), [51](#)

API Application Programming Interface [18](#), [33](#)

CMA-ES Covariance Matrix Adaptation Evolution Strategy [43](#)

CNN Convolutional Neural Network [ix](#), [12](#), [13](#), [15](#), [16](#), [39](#), [40](#), [42](#), [45](#), [50](#), [51](#)

DoF Degrees of Freedom [6–8](#), [18](#), [21](#), [22](#), [25](#)

EI Expected Improvement [43](#)

GPU Graphics Processing Unit [45](#)

GRU Gated Recurrent Unit [39](#)

IL Imitation Learning [4](#), [9–12](#), [14](#), [19–21](#), [27](#), [30](#), [32–35](#), [51](#)

ILO Imitation Learning from Observation [11](#)

IRL Inverse Reinforcement Learning [11](#)

KDE Kernel Density Estimation [42](#)

LIFO Last In, First Out [34](#)

LSTM Long Short-Term Memory [ix](#), [15](#), [39](#), [40](#), [45](#), [50](#), [51](#)

MDP Markov Decision Process [7](#)

PD Proportional-Derivative [24](#), [26](#)

PES Predictive Entropy Search [43](#)

PID Proportional-Integral-Derivative [1](#), [39](#)

RL Reinforcement Learning [4](#), [9–12](#), [14](#), [16–20](#), [24](#), [26–28](#), [33–35](#), [44](#), [51](#)

RNN Recurrent Neural Network [16](#)

SVMs Support Vector Machines [3](#)

TAS Tool-Assisted Speedrun [33](#)

TPE Tree-structured Parzen Estimator [16](#), [42–44](#)

UE4 Unreal Engine 4 [18](#)

Chapter 1

Introduction

1.1 Project Motivation

Control theory as a discipline has been around since ancient times, with the first recorded use of feedback control systems in the form of water clocks in ancient Egypt and Greece [1]. This early form of control theory is defined by the concept of using feedback to adjust the system's behaviour based on the difference between the desired and actual output.

As engineering and technology have advanced, so too has the field of control theory. Following James Watt's invention of the centrifugal governor (for regulation of steam engine speed), the need to mathematically model and control systems became more apparent. This led to the creation of the formal field of control theory, which is now a fundamental part of engineering and technology.

The approaches towards classical control theory initially started with a focus on linear systems, where the system's behaviour could be described by a set of linear differential equations. This allowed for the development of control systems that could be mathematically modelled and solved, leading to the development of techniques such as [Proportional-Integral-Derivative \(PID\)](#) control.

However, as the types of problems that control theory was applied to became more complex, the limitation of needing to model the system as a set of linear equations became apparent. Much of the work in modern control theory has focused on the development of techniques to handle uncertainty and disturbances in systems, as well as techniques to handle non-linear systems.

At this point, many of the next generation of control problems are not well-suited to these deterministic models. Engineering continues to expand into more systems-based problems, such as autonomous vehicles, robotics, and other complex systems. Although traditional approaches to control theory are still applicable in the subcomponents of these systems, the systems as a whole are often too complex to be modelled with such approaches.

Hence, the motivation of this thesis is to explore the use of [Artificial Neural Networks \(ANNs\)](#) in kinematic system contexts (especially the overall system, rather than the subcomponents), aiming to bridge the gap between the traditional approach, generally a domain of mechanical engineering, and the neural network approach, generally a domain of computer science. The goal is to develop a framework of best practices for the efficient training of neural networks in kinematic optimization contexts.

1.2 Neural Networks as a Controls Solution

In 1943, McCulloch and Pitts proposed a mathematical model of a neuron [2], laying the foundation for the field of [ANNs](#). The model was based on the idea that a neuron is a binary threshold unit that receives inputs from other neurons and produces an output; the output is produced by applying a step function to the weighted sum of the inputs. The model was later extended by Rosenblatt in 1958, who introduced the perceptron [3], a single-layer neural network that could learn to classify patterns following a [Supervised Learning](#) algorithm.

Initial work originated from the field of neuroscience. Inspired by the structure and function of the brain, early researchers were interested in understanding the brain’s computational capabilities. However, their simplicity gave them limited ability to solve problems; after a decade of research, it was shown that a single-layer perceptron could not learn to classify patterns that were not linearly separable [4]. Soon after, the field of [ANNs](#) fell out of favour, with greater focus on rule-based systems.

After a few decades, the field of [ANNs](#) was revived by the introduction of the [Back-propagation](#) algorithm [5]. This algorithm allowed for the training of multi-layer neural networks, which could learn to approximate any function given enough data and computational resources, enabling the field to solve non-linearly separable problems. However, despite the algorithm’s success, the field of [ANNs](#) was still limited by the computational resources available at the time; published in 1986, the backpropagation algorithm was computationally expensive and required large amounts of data to train models effectively, which was not feasible at a time where both compute time and data storage were expensive. Hence, the field of [Artificial Intelligence \(AI\)](#) moved in the direction of more simple

statistical models, such as [Support Vector Machines \(SVMs\)](#), which were state-of-the-art for many years following their 1995 introduction [6].

This ability to model nonlinear functions is particularly important in kinematic system contexts, and is why the application of [ANNs](#) to kinematic system problems is now of interest. Since the 1990s, compute power and data storage have become exponentially cheaper, leading to an extensive growth in [ANNs](#); these models have become the state-of-the-art in many fields, with additional developments furthering their capabilities in tasks such as image recognition and natural language processing.

Yet despite the success of [ANNs](#), the field is still limited by the lack of interpretability of these models - which are often referred to as “black boxes” - and the extreme computational resources required to train them. The latter limitation is particularly important in kinematic system contexts, where the models are often required to produce results in real-time or in resource-constrained environments (such as embedded systems).

These two main limitations - interpretability and computational resource requirements - are in fact interrelated, in the sense that the lack of ability to interpret the model makes it difficult to understand the best ways in which to optimize it. Classical approaches such as [SVMs](#) have the advantage of easily interpretable hyperparameters, which can be tuned to optimize the model’s performance. Yet, in the field of [ANNs](#), the hyperparameters are often chosen through trial and error, as the relationship between the hyperparameters and the model’s performance is not well understood.

This limitation exists both on the [Inference](#) and [Training](#) sides of [ANNs](#); although inference latency is incurred on every prediction the network makes, training resource requirements can be a significant limiting factor in the complexity of model that can be employed as well. This is where the field of [ANNs](#) has the potential to improve, and is hence a key area of focus in this thesis.

1.3 Project Objectives

The principal objective of this thesis is to develop a framework of best practices for the efficient training of neural networks in kinematic optimization contexts. This will be achieved through the following research objectives:

1. Selection of a problem domain that is representative of the challenges faced in kinematic optimization, with a focus on problems that are resource-constrained;

2. Development of a training and testing dataset that is representative of the problem domain;
3. Implementation of a training pipeline that allows for the efficient evaluation of hyperparameters, with a focus on statistical measures for hyperparameter optimization;
4. Evaluation of the performance of the trained models on the testing dataset, with a focus on the interpretability of the findings; and
5. Summarization of a framework of best practices for the efficient training of neural networks in kinematic optimization contexts.

1.4 Contributions to the Literature

The novel contributions of this thesis to the literature are twofold.

First, a novel approach is taken to the dataset and environment development, which achieves fidelity exceeding that of the existing literature with minimal development time through the integration with existing consumer-grade simulation software.

Second, an objective approach to selecting neural network structures in a representative kinematic system problem is described, which is the first of its kind in the problem domain; similar approaches have been taken in the computer vision and natural language processing domains, but have not been applied to kinematic optimization contexts.

1.5 Broader Project Context

It is important to note that the study described in this thesis is implemented in the context of a broader research project, which aims to bridge the gap between [Imitation Learning \(IL\)](#) and [Reinforcement Learning \(RL\)](#) in kinematic optimization contexts. Hence, the implementation details of the training pipeline are both designed to be generalizable to other problem domains, and also to be compatible with the broader research project's goals. The findings of this thesis will be used to inform the development of the broader research project, and will be used to evaluate the performance of the models developed in that project.

1.6 Thesis Structure

Chapter 1 provides a brief introduction to the field of artificial neural networks, with a focus on the history of the field and the motivation for this thesis.

Chapter 2 contains a literature review describing the existing approaches taken to optimize neural network performance in a variety of problem sets, along with the necessary background knowledge on the unique challenges faced in each of these domains. Additionally, the available open dataset resources are discussed.

Chapter 3 describes the implementation of the training and testing dataset, along with the development of the training pipeline. The dataset-specific hyperparameters are discussed, along with their hypothesized impact on the model’s performance.

Chapter 4 presents the implementation of the training pipeline, along with the approach taken towards automated hyperparameter optimization, and the results of this optimization.

Chapter 5 discusses the applicability of the results to other problem domains, the limitations of the study, and recommendations for future work.

Chapter 2

Literature Review

2.1 Background on Kinematic System Problems

Kinematic systems are systems that are governed by the laws of motion. The behaviour of these systems can hence be described by the kinematic equations that govern their motion, derived from the laws of physics. Such systems are a cornerstone of mechanical engineering, and include simple systems such as pendulums, to more complex systems such as robotic manipulators and vehicular systems.

Deterministic control refers to the ability to control a system with certainty, meaning that the system will behave in a predictable manner given a set of inputs. The simplicity of the system is a key factor in determining the ease of deterministic control. Simplicity can be defined in terms of the number of [Degrees of Freedom \(DoF\)](#) in the system, as well as the linearity of the system.

2.1.1 Redundancy and Multiple Solutions

Redundancy in kinematic systems refers to the existence of multiple solutions to a given problem. The [DoF](#) of a system is the number of independent parameters that define the configuration of the system. For example, a simple pendulum has one [DoF](#), as the position of the pendulum is defined by the angle of the pendulum. A six-[DoF](#) robotic manipulator, on the other hand, has six different parameters that define its configuration; if attempting to reach a specific point in space, there are multiple configurations of joint positions that will result in the end effector reaching that point.

2.1.2 State Space

The state space of a system is the set of all possible states that the system can be in. The *state* of a system is defined by a set of variables that describe the configuration of the system at a given point in time. For example, the state of a simple pendulum can be defined by the angle of the pendulum and the angular velocity of the pendulum, as this fully describes the configuration of the pendulum at a given point in time.

A system moves through its state space over time, transitioning from one state to another based on the laws of motion that govern the system, as well as the control *actions* that are applied to the system. When designing a control system for a kinematic system, it is important to consider the desired next state of the system (described with a *reward*), as well as the control actions that will move the system to that state.

To model the state space of a system, as well as the transitions between states and the rewards associated with each state, a [Markov Decision Process \(MDP\)](#) can be used. Consider a 6-DoF robotic manipulator that has the task of navigating to an object in space. The state space of the system can be defined by the set of all possible joint configurations of the manipulator:

$$\begin{aligned} S &= s_1, s_2, s_3, \dots \\ s_1 &= \theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6 \end{aligned}$$

Here, θ_i represents the angle of the i th joint of the manipulator.

The actions that the robot can take can be defined by the set of all possible joint movements that the robot can make:

$$\begin{aligned} A &= a_1, a_2, a_3, \dots \\ a_1 &= \Delta\theta_1, \Delta\theta_2, \Delta\theta_3, \Delta\theta_4, \Delta\theta_5, \Delta\theta_6 \end{aligned}$$

These actions describe the changes in joint angles that the robot can make.

The transitions between states, $T(s, a, s')$, can be defined by the probability of moving from state s to state s' given action a . This probabilistic framework allows for the modelling of uncertainty in the system, such as noise in the sensors or actuators; the robot may not move to the exact desired state due to these uncertainties, nor actually exist in the state that the sensors report. Such a framework is essential for designing robust control systems

that can handle these uncertainties. The rewards associated with reaching each state, $R(s)$, describe the value of reaching that state.

The optimal *policy* for the robot, $\pi(s)$, is defined as the strategy that specifies the action that should be taken in each state to maximize the expected cumulative reward over time. The goal of the robot is to learn this policy such that it can navigate to the desired object in space whilst optimizing for some objective, such as minimizing the time taken to reach the object, or minimizing the energy consumed in reaching the object.

2.1.3 Nonlinear Systems

In many kinematic systems, the equations that describe the relationship between the inputs and outputs of the system are nonlinear, resulting in complex dynamics that are difficult to model and control. Nonlinear systems can exhibit behaviours such as oscillations, bifurcations, and chaos.

Further considering the example of the 6-DoF robotic manipulator, the relationship between the joint angles and the position of the end effector is nonlinear, as the forward kinematic equations, which map joint angles to end effector position, involve trigonometric functions. The Denavit-Hartenberg parameter representation [7] of the manipulator describes the relationship between the joint angles and the position of the end effector, the parameters of which are visible in Figure 2.1:

$$[T] = [Z_0][X_0][Z_1][X_1][Z_2][X_2][Z_3][X_3][Z_4][X_4][Z_5][X_5][Z_6][X_6]$$

$$[Z_i] = \begin{bmatrix} \cos \theta_i & -\sin \theta_i & 0 & 0 \\ \sin \theta_i & \cos \theta_i & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[X_i] = \begin{bmatrix} 1 & 0 & 0 & r_{i,i+1} \\ 0 & \cos \alpha_{i,i+1} & -\sin \alpha_{i,i+1} & 0 \\ 0 & \sin \alpha_{i,i+1} & \cos \alpha_{i,i+1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Due to the nonlinear nature of each transformation, coupled with the multiplication of these transformations to obtain the end effector position, the relationship between the joint angles and the end effector position is highly nonlinear.

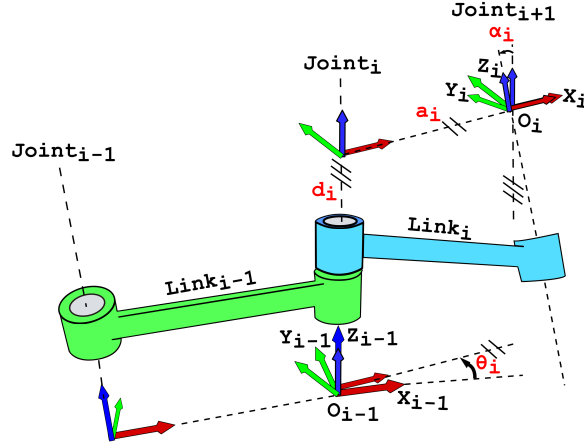


Figure 2.1: Denavit-Hartenberg Parameters for a Manipulator Link. Image from [8]

2.1.4 Real-Time Constraints & Neural Network Applications

As environmental factors such as gravity and friction can affect the behaviour of a kinematic system, real-time control is essential to ensure that the system can respond to these changes in a timely manner.

The ability of neural networks to learn complex, nonlinear relationships between inputs and outputs makes them well-suited for modelling and controlling kinematic systems. Neural networks can approximate any continuous function, given enough data and computational resources, making them a powerful tool for learning the complex dynamics of kinematic systems. However, the constraints of available computing technology necessitates the optimization of neural network structures to ensure that they can be used in real-time.

2.2 Kinematic Network Structures

2.2.1 Imitation Learning and Reinforcement Learning

The two main paradigms for training neural networks to control kinematic systems are [IL](#) and [RL](#). In [IL](#), the model learns to imitate the behaviour of an expert by training on a dataset of expert demonstrations. Mathematically, this can be modelled as:

$$\theta^* = \arg \min_{\theta} \sum_{i=1}^N L(f_{\theta}(s_i), a_i)$$

Where θ are the parameters of the neural network, L is the loss function, f_{θ} is the neural network, s_i is the state of the system at time i , and a_i is the action taken by the expert at time i . In effect, the goal of the model is to learn the mapping from states to actions that minimizes the difference between the actions taken by the expert and the actions predicted by the model. With a large enough dataset of expert demonstrations, the model can learn to imitate the expert's behaviour in a wide variety of scenarios.

In [RL](#), the model learns to make decisions by interacting with an environment and receiving feedback in the form of rewards or penalties, without any expert demonstrations. The model learns to maximize the cumulative reward over time by exploring the environment and learning which actions lead to the highest rewards. Mathematically, this can be modelled as:

$$\theta^* = \arg \max_{\theta} \sum_{i=1}^N \sum_{t=1}^T R(s_{i,t})$$

Where R is the reward function, $s_{i,t}$ is the state of the system at time t in episode i , and θ are the parameters of the neural network. The goal of the model is to learn the policy that maximizes the expected cumulative reward over time.

Notably, the [IL](#) paradigm requires a dataset of expert demonstrations, which can be recorded and provided to the model without any interaction with the environment; only once a certain level of performance is reached (as measured by the loss function) need the model be deployed in an interactive environment. This has significant implications for the speed and nature of training. Consider the manner in which [ANNs](#) are trained:

1. The model is initialized with random weights (θ)
2. The input data for state s_i is fed into the model, and the model outputs an action $f_{\theta}(s_i)$
3. The loss function is calculated based on the difference between the predicted action $f_{\theta}(s_i)$ and the expert action a_i
4. The weights of the model are updated using stochastic gradient descent to minimize the loss function

5. Steps 2-4 are repeated for each state in the dataset.

Provided the network is sufficiently complex, and the rate of learning is not too high, the model will immediately begin to imitate the expert’s behaviour; the direction of the gradient will be towards the expert’s actions, and the model will begin to converge to a solution that imitates the expert. One key issue with the [IL](#) paradigm is the lack of exploration; the model is only exposed to the expert’s actions, and may not learn to explore the environment to find better solutions. Hence, the model can not only never exceed the performance of the expert, but may also fail to generalize to new scenarios not covered by the expert’s demonstrations.

Torabi et al. [9] describe the state of the research in [Imitation Learning from Observation \(ILO\)](#), a method by which the model is provided with the expert’s state information as input (s_i), but not the corresponding action (a_i). The model must learn to predict the actions necessary to reach the expert’s state, and the authors posit that this method is more robust than [IL](#) to novel scenarios, as it requires the model to learn the underlying dynamics of the system, rather than simply imitating the expert’s actions. However, this begins to increase the complexity of the problem, as the model must learn to predict the expert’s actions based on the expert’s state information, rather than solely imitating the expert’s actions.

In contrast, the [RL](#) paradigm requires the model to interact with the environment from the start, learning from the feedback it receives through (initially) randomized exploration. There is no guarantee that a particular line of exploration will lead to a high reward, and the model must explore a wide variety of approaches to determine an optimal policy. This has both high startup costs (as the model must explore the environment to learn the optimal policy) and high computational costs (as the model must explore a wide variety of approaches to determine the optimal policy).

This initial startup problem is a significant issue in the field of [RL](#), and a variety of approaches have been proposed to address it. In [10], the authors implement a Human-AI Copilot Optimization (HACO) approach through which the [RL](#) model explores the environment in a safe manner by leveraging human demonstrations to guide the exploration process, and more quickly address the underlying network structures leading to the need for human intervention. One noted issue with this particular hybrid approach to blending [IL](#) and [RL](#) is the reduction of the training speed of the model; simulations must run at a slower pace to allow for human intervention.

Additionally, some approaches to [ILO](#) involve the use of [RL](#) reward functions, further blurring the line between the two paradigms. Referred to as [Inverse Reinforcement Learning \(IRL\)](#), this approach uses the expert’s demonstrations to learn a reward function that

can be used to train an [RL](#) model. The model learns to maximize the reward function, rather than imitating the expert’s actions directly, allowing for more robust generalization to new scenarios. However, the model must still explore the environment to learn the optimal policy, incurring the same need for environmental interaction as other [RL](#) approaches. [\[11\]](#) describes the application of such an approach to autonomous driving, where the driving style of a human driver is modelled as a cost function, and the model learns to minimize this cost function to imitate the human driver’s behaviour.

Addressing the gulf between the [IL](#) and [RL](#) approaches is thus a key component to the development of efficient kinematic control systems. The [IL](#) approach is efficient in terms of training time, but lacks the ability to generalize to new scenarios, while the [RL](#) approach is able to generalize to new scenarios, but requires significant computational resources to explore the environment and learn the optimal policy. Additionally, the dataset requirements of each are significantly different; [IL](#) requires a dataset of expert demonstrations, while [RL](#) requires the model to interact with the environment from the start. A hybrid approach that combines the strengths of both paradigms could lead to more efficient and effective kinematic control systems, but necessitates a combined environment and expert demonstration system enabling the use of both paradigms.

2.2.2 Separation of Control and Perception

The problem of kinematic control is both a control problem and an environmental interpretation problem. The control problem is the task of determining the control actions that will move the system to the desired state, while the environmental interpretation problem is the task of interpreting the state of the environment and determining the optimal control actions to take.

In classical control theory, it is generally accepted that it is beneficial to separate the control and perception components of a system; this separation is known as the separation principle. As described by Kruglikov [\[12\]](#), the separation principle simplifies the control problem by treating the state estimation problem (perception), and the feedback control problem (control), as separate entities. This separation allows for the optimization of each component in isolation, simplifying the overall control problem.

This same theory can be applied to deep learning models. In [\[13\]](#), the authors proposed a hybrid architecture that utilizes a perception module to interpret the state of the environment (utilizing a variety of techniques including LiDAR mapping combined with Mask R-CNN instance segmentation), building a 3D representation which can be interpreted by humans. This representation is then fed into a control module, which uses a finite state

machine to determine the optimal control actions to take. The authors posit that this separation of control and perception both simplifies the control problem and allows for increased interpretability of the model, as the 3D representation of the environment can be visualized and understood by humans, and the control actions taken by the model can be easily interpreted.

Indeed, the problem of poor interpretability is a common issue in deep learning models, which the research has attempted to address through a variety of techniques - compartmentalization of functions being only one of them. In [14], the authors propose LIME, a method for explaining the predictions of complex models by approximating the decision boundary with a simpler, interpretable model; this decision boundary is generated by perturbing the input data to generate a synthetic input, and observing the change in the output of the model. In the case of image recognition tasks, this can mean removing certain pixels from the image to observe the change in the model’s prediction, hence approximating what features of the image are most important to the model’s decision; in the context of a task such as driving, this could mean frame-by-frame comparison of input data to determine the features encouraging a robot to initiate (or not initiate) a turn. In [15], the authors propose a method for visualizing the features learned by a CNN using saliency maps, helping inform the user of the features that the model is using to make its decisions.

Further, this issue of poor interpretability is not solved by the separation of control and perception. Indeed, inside both the control and perception modules, the models are still complex neural networks that are difficult to interpret. In perception, the question of the best manner in which to fuse data from different sensors is open; the *TransFuser* model [16] uses a transformer-based architecture to fuse data from image and LiDAR sensors to build a more accurate representation of the environment, and even in models such as [13], the Mask R-CNN implementation is a complex neural network that is difficult to interpret.

Notably, these models are trained for road driving tasks, which require the interpretation of certain environmental cues (such as lane markings, traffic lights, and other vehicles) to make decisions following a set of rules (traffic laws). The separation of control and perception in this context has significant benefits considering external regulatory constraints, as the control module can be verified to follow these rules, while the perception module can be more verifiably trained to interpret the environment.

However, in the context of other kinematic systems, there is a risk that imposing separation may limit the ability of the model to learn complex dynamics; the control module may not have access to the full state of the environment, as its inputs are first preprocessed by the perception module. In the context of robotic manipulation, summarized in the

review paper [17], the authors present approaches used by 15 different research groups focused on [RL](#) for robotic pick-and-place tasks.

In [18], the authors propose a novel approach that integrates perception and control by leveraging a pre-trained neural model augmented with online dynamics adaption to achieve [One-Shot Learning](#) of manipulation skills. The framework allows the robot to rapidly adapt to new tasks from a single demonstration, and the authors posit that this integration of perception and control is essential for the model to reach this level of adaptability; the model of the environment [Configuration Space](#) is not fitted to the task, but rather a collection of manipulation strategies that can be augmented to fit the task.

Fundamentally, the concept of separation begins to break down when the output layers of the perception and control modules are connected and the model is trained end-to-end. Even if the initial components were trained individually, the model will learn to optimize the entire system, and the separation of control and perception will be lost.

2.3 Optimization Studies Across Machine Learning

Of particular note throughout the literature is the use of optimization techniques to objectively define the best network structures for a given task. However, in the area of kinematic systems, systematic application of optimization techniques to define the best network structures is uncommon, representing a gap in the literature.

2.3.1 Kinematic Systems

Robotic Manipulators

Across the 15 robot manipulator studies reviewed in [17], no authors focused on a systematic approach to optimizing the network structure of their models.

In [18], the authors used a single network structure, with the only investigation regarding a difference in network structure being the addition of a second input field accepting the previous state-action pair as temporal context. They found the temporal context was necessary for determining if a collision had occurred, but did no further investigation into the impact of temporal context on the model’s performance.

The strongest result in this field is presented in [19], where the authors focus on learning from demonstration (or [IL](#)) to train a model to perform object manipulation. The authors

implement 3 CNN-only models and 3 CNN+LSTM models, finding an improvement in training and validation losses on the CNN+LSTM models, despite lower parameter counts in the CNN+LSTM models (indicating less complexity), suggesting that the temporal context provided by the LSTM layers is beneficial for efficient task completion.

Autonomous Vehicles

In autonomous vehicle studies, the same trend of a lack of systematic optimization of network structures is present.

One of the first papers attempting to directly analyze the impact of certain network parameters (namely, the trade-off between spatial information, captured by a CNN, and temporal information, captured by LSTM layers) was [20]. The paper presents a comparison of the performance of a CNN-only architecture with a CNN+LSTM architecture, finding that the CNN+LSTM architecture outperformed the CNN-only architecture in terms of the mean squared error of the predicted steering angle by 35%. However, the authors did not investigate the impact of varying the number of LSTM layers, the number of units in each LSTM layer, or the number of filters in the CNN layer; the only difference between the two architectures was the addition of the LSTM layer, showing the importance of temporal context, but not the extent to which varying its parameters would impact the model’s performance.

A simplified system is explored in [21], where the authors implement a CNN-only architecture with three convolutional feature maps followed by pooling layers and a fully-connected layer output. The authors find that their network is able to predict the steering angle with a mean error across the 3000-frame validation dataset of approximately 4%, showing that simple architectures can deduce some of the relationships involved in vehicle control. However, the simplified system involves the use of a visually simple driving simulator (the CARSIM simulator), and is held at a constant speed of 60km/h. The constant speed simplification is a significant limitation; it reduces the complexity of the problem by removing the need for the network to learn to predict the steering angle based on the speed of the car, which has significant temporal dependencies.

2.3.2 Convolutional Neural Networks

Further focusing on the CNN layers is relevant to kinematic systems, as the ability of CNN layers to capture spatial information is essential for interpreting the state of the

environment. In this problem domain, there exists a significant amount of research into the optimization of neural structures to capture spatial information.

A variety of approaches to the optimization of CNN architectures are described in the survey paper [22], which describes the evolution of the area from Bayesian approaches into RL approaches in the very recent past. The entrance of RL into neural architecture search primarily became mainstream in 2017 [23], after Zoph and Le trained models on the CIFAR-10 32x32 image classification dataset, with the structure of their networks being informed by RL. The controller Recurrent Neural Network (RNN) guides the selection of network hyperparameters, informed by a reward signal linked to the model’s performance.

However, despite the interest in RL approaches, there is minimal evidence that such an approach exceeds classical statistical methods. Real et al. discovered in 2019 that an evolutionary approach to network optimization matched the performance of RL approaches in terms of the quality of the network structures found [24]. The authors posit that the evolutionary approach is more efficient in terms of computational resources, as the RL approach requires the training of a separate controller model to guide the search process, which can be computationally expensive.

A key gap in the literature as described by [22] is the lack of application of Bayesian optimization techniques to the optimization of network structures, despite preliminary results being promising; the review authors reference [25], the NAS-Bench-101 search dataset, which found some evidence that Bayesian optimization techniques could outperform the evolutionary algorithms with a minimal increase in computational cost (due to the need to generate a distribution modelling the performance of the network structures).

Bayesian optimizers are of particular interest in highly dimensional search spaces where both neural architectures and their algorithms are varied simultaneously, with Tree-structured Parzen Estimator (TPE) being explored in [26], and both TPE and random forests being explored in [27], implemented with the SMAC library [28].

There is also some extension of objective parameter selection to sequence models, where a sequence of data is used as input to the model. In [29], the authors implement a genetic algorithm for optimizing hyperparameters of a CNN network aimed at text classification tasks and sentiment analysis; effectively, the sentiment of words and groupings of words is the output of the model. Although a significantly different problem domain, the paper highlights the utility of informed searching techniques to guide the selection of hyperparameters for sequential models.

In conclusion, there exists some robust research into the optimization of network structures for CNN models, but this research is primarily focused on problem domains with

significant differences to kinematic systems. There is a significant gap in the literature regarding the optimization of network structures for kinematic systems.

2.4 Datasets and Environments

A variety of datasets and simulation environments have been developed for training ANNs on kinematic systems. Much like the research into the compartmentalization of control and perception, many of these datasets have been developed to address only a specific subset of the kinematic control problem. Perhaps the most well-known collection of datasets in the self-driving car domain is the KITTI suite [30], which provides no ground truth for the actions taken by the human driver, but does provide a large dataset of images and LiDAR data for training perception models.

Observation-only datasets are common in gait optimization datasets, where the focus is often on collecting human gait data for the purpose of recreating these gaits in robots. One such example is [31], but this dataset provides no information on the actions taken by the human operator to achieve the observed gaits, which is generally a limitation of the methodology, since directly extracting the muscle activations from the human operator is not feasible currently.

These limited datasets allow for the training of models that can interpret the state of the environment, but lack significant information that aids in the training of models that can control the environment. ApolloScape [32] is one dataset that provides both scene parsing training data (labelled by human annotators) and trajectory training data (matched pairs of input images and output actions taken by the human operator) for self-driving applications. However, this dataset provides no method by which to test the generalization of the model to new scenarios, as there is no corresponding simulation environment in which to test the model’s performance. The lack of simulation also restricts the ability to implement RL, as there is no safe manner in which the agent can explore the environment to learn the optimal policy without a simulated representation.

The focus of significant research in recent years has hence been on the development of simulation environments that can be used to train models for kinematic control tasks. The CARLA simulator [33] is one such example, providing a realistic 3D environment in which to train self-driving models. The simulator provides a variety of sensors, such as cameras and LiDAR, that can be used to interpret the state of the environment, as well as a physics engine that can be used to simulate the dynamics of the environment.

Indeed, much of the benchmark work in navigational tasks has been done in the CARLA

environment. A key limitation of the CARLA environment is the lack of a robust physics engine; the physics engine is based on the standard PhysXVehicles model provided for [Unreal Engine 4 \(UE4\)](#) [33], which uses a simplified sprung mass collection model for the vehicle dynamics. This is necessary for a generic implementation that can be leveraged by a wide variety of applications, which is the goal of the [UE4](#) platform, but the generic nature of the platform results in a simulation that cannot accurately model dynamics such as wind resistance, tire deformation, or suspension dynamics.

Another challenge is the use of generic assets; there is a high cost involved in the generation of realistic 3D assets, and the CARLA environment uses a variety of freely available assets that are of lesser quality than that which is used in commercial applications. This can lead to issues with the generalization of models trained in such environments to real-world scenarios.

Robotic manipulator tasks are one area in which simulation environments exist at a reasonable level of fidelity for the problem type, and there is good crossover between the simulation and dataset approaches. This is because traditional 6-[DoF](#) robotic manipulators are often employed in reasonably controlled environments, where a simulation of sufficiently representative visual fidelity is easier to achieve. Traditional programming approaches to such systems also often involve the use of simulation software; one example includes CoppeliaSim, formerly V-REP [34]. However, the commercial viability of such software causes a significant barrier to entry for researchers.

In the open source space, examples are generally built on Gymnasium [35], an open-source [Application Programming Interface \(API\)](#) for integrating simulation environments with [RL](#) models. Gymnasium provides a variety of environments for training [RL](#) models, including the classic CartPole environment, which is a simple 2D environment in which the agent must balance a pole on a cart by moving the cart left or right. In the 6-[DoF](#) robotic manipulator domain, the panda-gym environment [36] is a popular choice, designed for training [RL](#) models to control a Franka Emika Panda robotic manipulator. Although the environment is simplified, there is the ability to augment the environment with additional assets to attempt to increase the fidelity of the simulation.

It is likely that the state of the art in simulation environments primarily exists in the commercial self-driving car domain, where there is significant investment in the development of realistic simulation environments. Such companies with large deployed fleets of self-driving and sensor-equipped cars, such as Waymo and Tesla, have the ability to augment their simulations with real-world data, allowing for the development of generalizable models. However, the lack of access to high fidelity combined simulations and datasets is a significant barrier to entry for researchers in the field.

2.5 Summary

The literature review has highlighted a variety of key issues in the field of kinematic control. First, a background on kinematic systems outlines the key limitations of deterministic control, and the complexity of nonlinear systems. The state representation is introduced as a method for parameterizing the system, and the approaches to learning the optimal policy for a system are discussed, including [IL](#) and [RL](#). There are clear limitations to both approaches, with [IL](#) lacking some ability to generalize to new scenarios, and [RL](#) requiring significant computational resources to explore the environment and learn the optimal policy. The external constraints on kinematic systems such as real-time operation are also described, indicating the need for efficient control systems.

The separation of control and perception is discussed, with the benefits of this separation being the simplification of the control problem and the increased interpretability of the model. However, the issue of poor interpretability is not solved by the separation of control and perception, given the complexity of the [ANNs](#) used in each of these modules, and there are certain limitations to this separation in presupposing the key details that the control module requires to make decisions, which are not provided as outputs from the perception model.

Studies in the methods for optimizing network structures are discussed, with a variety of approaches being proposed, including Bayesian optimization, [RL](#), and evolutionary algorithms. However, there is a significant gap in the literature regarding the optimization of network structures for kinematic systems, with most research being focused on other problem domains.

Finally, the datasets and environments available for training [ANNs](#) on kinematic systems are discussed, with a variety of datasets and simulation environments being developed for this purpose. The necessity of combined dataset and environment systems is highlighted to enable the hybrid combinations of [IL](#) and [RL](#) approaches, and the limitations of current simulation environments are discussed, with a significant gap in the literature regarding the development of high-fidelity simulation environments for kinematic systems.

With this identified state of the research, the following chapters focus on the development of a high-fidelity simulation and dataset combination for training [ANNs](#) on kinematic systems, with the goal of enabling the optimization of network structures for these systems, and the implementation of the objective parameter selection techniques to this novel problem domain.

Chapter 3

Dataset & Environment Development

Following the identified gap in the existing literature, and to enable the comparison of [RL](#) and [IL](#) algorithms in the context of kinematic systems, the development of a combined dataset and environment is necessary. This is a novel contribution of the research, as the prior work in the field has focused on either the development of a dataset or the development of an environment, but not both. The dataset is a collection of expert demonstrations that will be used to train with an [IL](#) approach, while the environment is a simulation of the kinematic system that will be used to train the [RL](#) algorithms. The development of the dataset and environment will be discussed in the following sections.

3.1 Case Study Selection

With a broad field of problems under the umbrella of kinematic systems, the selection of a representative case study is essential to improve the generalizability of the results. The case study should:

- Be representative of a broad class of kinematic systems, with impacts from both the agent and the environment on the system dynamics;
- Be complex enough to require the use of [ANNs](#) for effective control;
- Be reasonably documented in the literature to allow for the comparison of results with prior work; and
- Be simple enough to allow for the development of a dataset and environment within the scope of the research.

3.1.1 Trajectory Planning for Robotic Arms

The trajectory planning problem for robotic arms is a well-documented problem in the literature, with a variety of control strategies and optimization techniques available.

The existing approaches to robotic programming (namely, the use of teach pendants or manual programming) are time-consuming, presenting a significant opportunity for the development of an automated solution. Additionally, this approach lends itself well to the development of a dataset of expert demonstrations, aiding in the bootstrapping of the [IL](#) approach.

However, the physics complexity of the problem can largely be abstracted away, as inverse kinematics solvers can solve for the torque requirements necessary to overcome the forces of gravity and friction. This abstraction may limit the generalizability of the results to other kinematic systems.

The detailed kinematics of 6-[DoF](#) manipulator systems are mentioned in the earlier literature review. The problem complexity can be described with the concept of [Configuration Space](#). Such manipulators have a 6-[DoF](#) configuration space. The [Configuration Space](#) obstacles are generally static and operation is largely not in dynamic environments, and path planning is generally not time constrained, reducing the incentive for improving efficiency on this area of the problem. Depending on the problem formulation, the [Configuration Space](#) can be considered holonomic, as the manipulator is capable of moving in any direction in the [Configuration Space](#). Although the dynamics are complex and nonlinear, the controls are relatively well-understood, and enjoy a high level of repeatability, with existing commercial solutions available.

3.1.2 Path Planning for Autonomous Vehicles

The problem of automating driving tasks can generally be divided into two categories: perception and control. The perception problem involves the identification of objects in the environment, while the control problem involves the planning of a path through the environment to reach a desired goal.

The perception problem in traffic environments is complex due to the high degree of uncertainty and variability in the environment. However, the control problem is well-suited to the development of a dataset of expert demonstrations, as the desired path can be determined by a human operator. By selecting an environment in which the perception problem is abstracted away, the focus can be placed on the control problem; an ideal

environment for such a study would be a closed-course track with no other vehicles or pedestrians.

Vehicle handling dynamics are well-understood and can be accurately modeled, allowing for the development of a simulation environment that has a high degree of fidelity to the real-world system. Additionally, unlike problems such as robotic arm control, the variety of vehicle states and actions is much larger, necessitating a less deterministic approach to control, making ANNs an ideal choice; this complexity may improve the generalizability of the results to other kinematic systems which also have a large state and action space. Between commercial self-driving vehicle development, academic research, and the availability of simplified open-source datasets, there is a significant amount of literature in the field to compare the results of the research to.

The Configuration Space of the problem is dynamic; although only a 3-DoF problem (2D position and steering angle), the rate of motion through the Configuration Space is much higher than the robotic arm problem, and the knowledge of the environment is limited to the immediate surroundings of the vehicle, making the known Configuration Space much smaller. The problem is also time-constrained, due to the requirement to process sensor data and make control decisions in real-time. The Configuration Space is nonholonomic; the vehicle must follow a trajectory through the environment to reach the goal, introducing velocity planning as a control factor, rather than simply moving to a desired configuration.

Kinematic control of vehicles can be modelled with a bicycle model, which is a simplified model of the vehicle dynamics that captures key aspects of the system. Consider the following state equations:

$$\begin{aligned}\dot{x} &= v \cos(\theta) \\ \dot{y} &= v \sin(\theta) \\ \dot{\theta} &= \frac{v}{L} \tan(\delta)\end{aligned}$$

Where x and y are the position of the vehicle, θ is the heading of the vehicle, v is the velocity of the vehicle, L is the wheelbase of the vehicle, and δ is the steering angle of the vehicle. The control problem is to determine the steering angle δ and the velocity v to reach a desired goal.

However, there are limitations to this simplification. First, the value of velocity v cannot be directly controlled; the control input is the acceleration of the vehicle, which affects the velocity. Second, the model does not account for the forces acting on the vehicle, such as friction and gravity, which can affect the motion of the vehicle - restricting both cornering

and acceleration. The model can be expanded to include dynamic effects - the forces on the vehicle. The dynamic bicycle model includes the following models of the forces:

$$\begin{aligned} F_x &= m(\dot{v}_x + v_y\dot{\theta}) \\ F_y &= m(\dot{v}_y - v_x\dot{\theta}) \end{aligned}$$

The change in velocity is hence modelled by the forces acting on the vehicle:

$$\begin{aligned} \dot{v}_x &= \frac{F_x}{m} - v_y\dot{\theta} \\ \dot{v}_y &= \frac{F_y}{m} + v_x\dot{\theta} \end{aligned}$$

The cornering angle of the vehicle is dependent on the forces and inertial properties of the vehicle:

$$\dot{\theta} = \frac{1}{I_z}(F_{yf}l_f - F_{yr}l_r)$$

Where I_z is the moment of inertia of the vehicle, l_f and l_r are the distances from the center of mass to the front and rear axles, and F_{yf} and F_{yr} are the lateral forces on the front and rear tires. The lateral forces are dependent on the tire slip angle and the tire characteristics:

$$\begin{aligned} F_{yf} &= C_{\alpha f}\alpha_f \\ F_{yr} &= C_{\alpha r}\alpha_r \end{aligned}$$

Where $C_{\alpha f}$ and $C_{\alpha r}$ are the cornering stiffness of the front and rear tires, and α_f and α_r are the slip angles of the front and rear tires. This cornering stiffness depends on the tire characteristics and the loading on the tires, and is hence a function of m , l_f , and l_r (as the weight distribution of the car affects the loading on the tires), as well as thermal state, tire pressure, and tire compound, which are not easily modelled.

Even with this simplified model - which ignores the effects of roll and pitch on the interior and exterior tires, as well as the effects of aerodynamics and tire deformation - the problem is nonlinear from the very first equations, due to the presence of the $\cos(\theta)$ and $\sin(\theta)$ terms. Still, this simplified representation is useful for evaluating the similarity of the control problem to other kinematic systems.

3.1.3 Gait Optimization for Legged Robots

Gait optimization is a complex problem that necessitates the coordination of multiple limbs to achieve a desired goal. Although the physics of the problem are well-understood, there are significant trade-offs between speed, stability, and energy efficiency that must be considered, which makes a deterministic approach to control challenging. Thus, ANNs are well-suited to the problem, as they can learn the complex relationships between the state of the system and the desired actions.

Recent advances in the application of RL to gait optimization have shown promising results; a 2020 paper by Lee et al. demonstrated state-of-the-art performance in environments with deformable terrain and dynamic footholds using a RL approach trained in a simulated environment [37], areas of control typically challenging for state machine or optimization approaches.

However, a key difficulty of the problem is the development of a dataset of expert demonstrations. One approach to generating expert demonstrations is to match the gait of test subjects to the robot, which is complex and limited by the sensor hardware mounted to subjects. Existing datasets include the CMU MoCap dataset, which contains motion capture data of human subjects performing a variety of activities [31]; the dataset focuses on solely optical motion capture, lacking force and environmental data which could easily be collected in a simulation environment. Other motion capture datasets, such as the PhysioNet dataset, contain force and accelerometer data, but focus on gait measurements from patients with various medical conditions [38], which may be a poor representation of an expert gait.

Creating such a simulated environment to accurately model the dynamics of the system is itself difficult. As the key areas of interest in the field are over rough terrain or in dynamic environments, the simulation environment must be able to accurately model these conditions to ensure the generalizability of the results to the real-world system. Development of such a simulation environment may be beyond the scope of the research. It is worth noting, however, that there is some debate in the research regarding the fidelity of model necessary for models to achieve a good understanding of the problem kinematics; the GLiDE model, trained by Xie et al., utilized a reduced-order model of the system and achieved results competitive with a Proportional-Derivative (PD) controller trained on a deterministic inverted pendulum model control scheme [39].

The kinematics of gait recognition, using the example of a bipedal robot, are complex. Given n as the number of joints in the system, the Configuration Space is n -dimensional, as the n joints represent n degrees of freedom. However, considering the objective of

navigating the robot to certain locations, the pose of the robot in the environment can be considered as a 6-DoF problem, as the robot centroid can be placed in positions of numerous $(x, y, z, \theta_x, \theta_y, \theta_z)$ values. Hence, depending on the exact problem formulation, the Configuration Space that the network is expected to navigate may vary in dimensionality.

Let θ_1 , θ_2 , and θ_3 represent the angles of the three joints of a legged robot - hip, knee, and ankle joint. The position of the foot relative to the hip can be given by:

$$\begin{aligned} x &= l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3) \\ y &= l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) + l_3 \sin(\theta_1 + \theta_2 + \theta_3) \end{aligned}$$

Where l_1 , l_2 , and l_3 are the lengths of the segments of the leg. In this simplified case, the inverse kinematics can be computed deterministically, allowing for the selection of a desired foot position with the necessary joint angles as equation outputs. However, this does not describe the stability of the robot, nor the energy efficiency of the gait, which are critical to the control problem.

The position of the feet is also a function of friction, similar to the vehicle control problem. However, the weight distribution varies significantly more with the gait problem, as the robot is not constrained to a fixed wheelbase, and the center of mass of the robot can move significantly with each step. This frictional modelling is more complex for other reasons; the robot is not constrained to a flat surface, and the frictional forces are dependent on the terrain.

Finally, real-time operation is significantly more critical in the gait problem, as slow predictions can lead to the robot falling over, a significantly less recoverable state than a vehicle beginning to reach the limits of its handling.

3.1.4 Comparison of Case Studies

To compare the case studies, a decision matrix was developed to evaluate the case studies based on the criteria outlined above. The decision matrix is shown in Table 3.1.

The vehicle planning case study was selected as the most suitable case study for the research, as it scored the highest in the decision matrix.

The case study is complex enough to require the use of ANNs for effective control, unlike arm path planning. Although arm control is a complex problem from a control output perspective, the path planning subset is less complex due to the simplified environmental

Problem	Generalizability	Complexity	Feasibility	Result
Arm Planning	1	1	3	5
Vehicle Planning	2	2	3	7
Robotic Gait	2	3	1	6

Table 3.1: Decision Matrix for Case Study Selection

interpretation, and existing control strategies (such as [PD](#) controllers) have largely solved the problem of the inverse kinematics of the system to an acceptable degree.

The physics of the problem are well-understood and simulatable enough to allow for the development of a dataset and environment within the scope of the research, unlike robotic gait optimization, which requires a high-fidelity simulation environment and expensive motion capture hardware to generate expert demonstrations.

Finally, the case study is likely most representative of a broad class of kinematic systems, as the balance between path planning and control provided by the vehicle planning problem is likely to cover a wider range of kinematic systems. Navigation problems are ubiquitous in robotics; similar network architectures and training strategies might have applications in aerospace, marine, and other terrestrial vehicle control problems. By comparison, the robotic gait problem is more specialized, and the arm control problem does not have the same level of complexity in environmental interpretation.

3.2 Integrating with Commercial Game Engines

As discussed in the literature review, the open source simulators available for kinematic systems are limited in their visual fidelity, as they are primarily designed for research purposes. To improve the generalizability of the results to real-world systems, the simulation environment should have a high degree of visual similarity to the real-world system.

Given the cost and complexity of developing a high-fidelity simulation environment from scratch, integrating the simulation environment with a commercial game engine is a cost-effective solution. This approach has been used in prior research implementing [RL](#) algorithms. One such example is provided in [\[40\]](#), where the authors used the Gran Turismo Sport game engine to simulate the dynamics of a vehicle in a racing environment to train a [RL](#) agent. Key to their success was the ability to interact with the game engine programmatically; in particular, their implementation extracted key parameters from the game engine to inform the [RL](#) agent’s decisions, including the velocity of the

vehicle (in component form). The reward structure of the [RL](#) agent was also based on metrics extracted from the game engine, such as the current lap time.

This integration has the advantage of providing a high degree of visual similarity to the real-world system, as visual fidelity is a key selling point of commercial game engines. The physics model of the game engine also benefits from these financial resources, as the game engine developers have the resources to accurately model even complex effects, such as tire deformation and aerodynamics, which are critical to the control of a vehicle.

In open-source simulators, implementing concepts such as aerodynamic maps is difficult, due to the computational cost of simulating the fluid dynamics of the system. However, in the case of the *Gran Turismo* game engine, the developers have specifically developed aerodynamics maps and tire loading models for the vehicles in the game through wind tunnel and road testing [\[41\]](#), simplifying lift, drag, and cornering calculations at runtime; this level of detail is not available in open-source simulators.

An additional advantage is the ease of generating expert demonstrations in a game engine. In the case of the *Gran Turismo* game engine, the authors were able to use both the game’s built-in [AI](#) to generate demonstrations as well as data from online players; this served as a comparison against which their [RL](#) agent could be evaluated, but could also be used as a dataset for [IL](#) approaches, which are typically time-consuming to generate.

However, integrations such as the one in [\[40\]](#) have typically required the cooperation of the game engine developers, as the game engine is not designed to be interacted with programmatically. Hence, a novel contribution of this research is the development of a method to interact with a commercial game engine programmatically, enabling the recording of game state and the provision of control inputs to the game engine, building both a dataset for [IL](#) and an environment for [RL](#).

3.3 Selection of Simulator

Following the review of the literature, it was determined that the selection of a simulator for the kinematic system was a critical decision in the development of the dataset and environment. Key requirements for the simulator include:

- The ability to simulate the kinematic system accurately, including the dynamics of the system and the constraints on the system;
- The ability to interact with the simulator programmatically;

- The ability to scale the training runs across multiple machines in resource-constrained environments; and
- A high degree of visual similarity to the real-world system to aid the transfer of learned policies to a real-world implementation.

3.3.1 Accuracy of Simulation

As detailed in the literature review, many simulators have been developed for autonomous vehicle control, but all are focused on the perception problem, with dynamics simulations that are too simple to be useful for the threshold control problem of driving at high speeds. Accurate simulation generally restricts the use of most video games and open-source simulators, as they are not designed to accurately model the dynamics of the system. However, a variety of simulation games are available that model the dynamics of vehicles accurately, such as *Gran Turismo* and *Forza Motorsport*. This category shall broadly be referred to as *consumer racing simulators*.

Greater accuracy is available with some products that have a crossover between the gaming audience and the professional racing audience. *BeamNG Drive*, for example, is a physics simulation game that models the dynamics of vehicles accurately, including the deformation of the vehicle body and the interaction of the vehicle with the environment; the developers also make an equivalent commercial product for manufacturers to test vehicle dynamics in a controlled environment. This category shall broadly be referred to as *professional racing simulators*.

3.3.2 Programmatic Interaction

The ability to interact with the simulator programmatically is a key requirement for the development of the dataset and environment. The simulator must provide an interface to extract key parameters of the simulation state, such as the position and velocity of the agent, and the lap time, as well as the ability to provide control inputs to the simulator.

This is less common in consumer racing simulators, as the primary audience for these games is the gaming community, rather than the research community. However, some consumer racing simulators, such as *Gran Turismo*, have been used in prior research to develop [RL](#) agents, indicating that it is possible to interact with these simulators programmatically.

Professional racing simulators are more likely to provide an interface for programmatic interaction, as they are designed for professional use and are more likely to be used in

research and development. For example, *BeamNG Drive* provides an interface for programmatic interaction, allowing users to extract key parameters of the simulation state and provide control inputs to the simulator. However, this is locked behind the commercial version of the simulator, which is significantly more expensive than the consumer version.

3.3.3 Scalability

The ability to scale the training runs across multiple machines is essential for the development of the dataset and environment. The simulator must provide an interface to distribute the training runs across multiple machines, allowing for the parallelization of the training process.

As consumer racing simulators are primarily designed for use on resource-constrained consumer systems, they often have lower system requirements, due to the aforementioned ability to model the dynamics of cars on real-world data [41].

For professional racing simulators, the system requirements are often higher, as they are designed for professional use and are more likely to be used in research and development.

3.3.4 Visual Similarity

A high degree of visual similarity to the real-world system is essential for the transfer of learned policies to a real-world implementation. The simulator must accurately model the visual appearance of the real-world system.

This is an area of particular strength for consumer racing simulators, as visual fidelity is a key selling point of these games. The developers of these games have the financial resources to accurately model the visual appearance of the real-world system, making them ideal for the transfer of learned policies to a real-world implementation.

Professional racing simulators also have a high degree of visual similarity to the real-world system. However, the standard assets of these simulators are often less detailed than those of consumer racing simulators, as it is expected that professional users will have access to more detailed assets for their specific use case.

3.3.5 Comparison

Considering the author’s possession of *Gran Turismo 3* and *BeamNG Drive*, the comparison of the two simulator types, with a particular focus on the specifics of the two, is shown in 3.2.

Simulator	Accuracy	Programmability	Scalability	Graphics
<i>Gran Turismo 3</i>	2	2	1	2
<i>BeamNG Drive</i>	1	1	2	1

Table 3.2: Decision Matrix for Simulator Selection

Despite the higher overall score of *BeamNG Drive*, extra emphasis is placed on the importance of scalability. *Gran Turismo 3* was developed for the PlayStation 2 console, which has significantly lower system requirements than a modern PC. Indeed, the game is able to run without a dedicated GPU for graphics acceleration, enabling the use of the GPU for inference and training tasks. Given the restricted hardware available to the author, this is a key consideration.

Gran Turismo 3 is also generally regarded as a less accurate and programmable simulator than *BeamNG Drive*, however, the relative lack of graphical fidelity, a result of the game’s age, can be mitigated by virtue of the emulation process. The PCSX2 emulator, which is used to run the game on a modern PC, allows for the upscaling of the game’s graphics to a higher resolution, improving the visual fidelity of the game.

An additional advantage not captured in the decision matrix, and a key consideration in the selection of *Gran Turismo 3*, is the ability to use the game’s built-in AI to generate expert demonstrations. The game’s AI is capable of driving the vehicle around the track at a high level of performance, providing a dataset of expert demonstrations that can be used to train the IL approach. Given the modular nature of the *BeamNG* simulator, preplanned AI paths are not available, increasing the cost of generating expert demonstrations.

Finally, the programmatic interaction with *BeamNG Drive* would necessitate the use of a commercial upgrade to the simulator, and the use of memory extraction techniques to interact with game engines appears to be a novel contribution to the field, as the majority of research in this area has focused on the use of game engines that provide a programmatic interface. Hence, the decision was made to use *Gran Turismo 3* as the simulator for the research.

3.4 Recording Simulation State

With the selection of the simulator complete, an interface is developed to record the simulation state. Recording the simulation state is a problem of two parts:

- Recording key parameters of the simulation state, such as the position and velocity of the agent, and the lap time (to be used as a reward signal); and
- Recording the visual state of the simulation, to be used as an input to neural network.

3.4.1 Interpreting Simulation Memory

With the lack of a programmatic interface to the vehicle state, the memory of the game engine must be interpreted to extract the key parameters of the simulation state.

Direct connection to the game engine’s memory is implemented with the ctypes library for Python; however, although this provides an interface to the game engine’s memory, the memory structure is not documented, making interpretation of the memory difficult.

The undocumented nature of the game engine’s memory structure necessitates the use of reverse engineering techniques. The use of a memory scanner, such as Cheat Engine (employed in this research) [42], is a common technique in the reverse engineering community to identify the memory addresses of key parameters in a game’s memory.

The memory scanner works by scanning the memory of a process for a known value, such as the current speed (displayed on screen). However, due to the high volume of memory addresses in a process, filtering of the results is necessary to identify the memory address of the desired parameter. This is done by changing the value of the parameter in the game and scanning for the new value, repeating this process until a single memory address is identified.

To implement this, the game engine is run in a window, with a Lua script running in parallel. The user plays the game, searches the memory for the observed speed value at one interval, and then changes the speed, scanning a second time, repeating until the memory address containing the speed value is identified, as shown in Figure 3.1.

The same process is repeated to determine the memory addresses of the RPM and gear values, as well as the lap time. There is a consistent pattern in the relative memory addresses of these values, as they are stored in a contiguous block of memory, likely a result of an object-oriented design pattern in the game engine. However, as the location of the

memory block changes with each run of the game engine, the memory addresses must be determined at runtime.

The current position of the vehicle necessitates a slightly different approach. It was not possible to find a memory address from which to directly extract the vehicle’s pose. Instead, the vehicle’s position is extracted by exploiting the minimap feature of the game engine, which displays the vehicle’s position in the environment (visible in the top left of Figure 3.1). Instead of filtering based on the value of the parameter, the memory scanner is used to filter across two memory addresses: the minimap’s X and Y coordinates. With the oval test track used to initially map the memory model of the game, the memory dump of the game engine is filtered to instead look for a relative *increase* in two variables (presumably the X and Y coordinates of the dot on the minimap) to begin, followed by a relative *decrease* in the X coordinate as it continues around the first corner. This process is repeated until the memory addresses of the X and Y coordinates of the minimap are identified, which are then used as a proxy for the vehicle’s position; this process generally requires a few laps of the approx. 41sec track to filter down the list of memory addresses matching the expected rate of variable change, due to the lack of a specific value in the search process.

With the key parameters of the simulation state identified, the next step is to record the visual state of the simulation.

3.4.2 Recording Visual State

Screenshots of the current emulator display must be generated at high frequency and with low processing latency. The Python library DXcam is implemented [43], which leverages the built-in Windows Desktop Duplication API to capture the screen at high frame rates. The PAL copy of the game engine is used (as this is the version of the game engine that the researcher owns a copy of), which runs at 50Hz; the screen capture is synchronized with the vertical blanking interval of the game engine to ensure that the screen capture is not corrupted by tearing artifacts.

3.4.3 Control Inputs

For generation of an expert demonstration dataset for IL, the control inputs to the game engine must be recorded. The PyGame library is used to intercept controller inputs to the game engine (throttle, brake, and analog steering inputs), which are then recorded as lines

in a `.csv` file. Each image in the dataset is paired with the corresponding control inputs, and the extracted memory values, forming the dataset for the [IL](#) approach.

3.5 Reinforcement Learning API

To enable the use of [RL](#) algorithms in the simulation environment, a reward interface must be made available for the agent’s actions to be evaluated.

An [API](#) standard for [RL](#) environments is Gymnasium [\[35\]](#), which forked from OpenAI’s Gym once internal development ceased. This [API](#) is implemented for the simulation environment, utilizing a forked version of PCSX2 designed for [Tool-Assisted Speedrun \(TAS\)](#) applications, available from [\[44\]](#). This forked version implements the ability to pause the game engine with hotkeys whilst maintaining the game engine’s internal state, and step forward through individual frames of the game engine. Using the same input emulation technique as is used to provide control inputs, the following Gymnasium [API](#) bindings are implemented:

- `gym.make()`: Creates an instance of the simulation environment (not entirely automated due to initial memory address setup);
- `env.reset()`: Emulates press of F3 key to load a save state of the game engine recorded at the start of a lap by the researcher;
- `env.step(action)`: Takes an action in the environment and returns the next game engine frame (screenshot) by emulating Spacebar (to skip frame) and the control inputs for this frame;
- `env.close()`: Closes the environment and releases any resources used by the environment.

3.5.1 Reward Structure

Of particular interest in the field of [RL](#) is the reward structure of the environment. The reward structure is a critical component of the [RL](#) algorithm, as it guides the agent’s exploration of the state space and the selection of actions.

In the case of the memory extraction interface developed for the game engine, the reward structure is based on the lap time of the vehicle; the model is incentivized to

minimize the lap time, which implies an optimized trajectory through the environment. However, a key issue with this reward structure is the lack of intermediate rewards; the agent receives no feedback on the quality of its actions until the end of the lap, which may lead to slow learning rates and suboptimal policies.

An alternative approach is the extraction of split times from the game engine’s memory, which would provide the agent with intermediate rewards at each split point on the track. This would allow the agent to receive feedback on the quality of its actions at each split point, which may lead to faster learning rates and more optimal policies. However, with the focus on [IL](#) for the preliminary model training, this enhanced reward signal is not implemented due to the complexity of finding the memory addresses of the split times in the game engine’s memory, which appear to change for each track and lap.

3.6 Scalability

To enable the scaling of training beyond the capabilities of a single machine in [RL](#) approaches, the ability to distribute the training runs across multiple machines is essential. Hence, scalability of implementation is a key consideration in the development of the environment.

The project uses a client-server architecture. The client runs on the machine with the PCSX2 emulator, and can run in data collection mode (where it sends all data), or inference mode (where it sends only screenshots and receives controller inputs as a response). The server runs on another machine and is where dataset storage, neural network training, and runtime inference occurs. The client currently requires Windows due to the particular methods of memory extraction and screenshot generation used, whereas the server side is OS-agnostic and only requires PyTorch support on the platform. A basic management UI is implemented using PySimpleGUI, and shown in [3.2](#); on the server side, this displays the most recent frame received, and the associated controller state and gear/RPM/speed.

PNG compression occurs on the client side to reduce network bandwidth consumption. The PNG is pickled with the other message data and transmitted using ZeroMQ, a networking library that handles certain transport characteristics (error handling, etc) automatically for the implementation. The server receives ZMQ messages into a queue and processes data in a [Last In, First Out \(LIFO\)](#) order to ensure inference occurs on the most recent frame.

The emulator is packaged into a Docker container for ease of deployment, enabling the scaling of training runs and data collection across multiple machines.

3.7 Dataset Generation

Following the simulator integration, an expert demonstration dataset is generated for the [IL](#) approach, to complement the simulation environment available for the [RL](#) approach. To ensure sufficient quantity and variety of expert demonstrations, an approximately 120GB dataset of driving demonstrations is generated, recording a variety of both the author’s interactions, and the game’s built-in [AI](#) interactions, which are recorded by using the demo playback feature of the game engine following a custom race loaded with the game’s [AI](#) opponents.

3.8 Summary

In this chapter, the selection of the case study and simulator for the research was discussed. The choice of the vehicle planning case study was made due to its feasibility and generalizability to a broad range of kinematic systems. The selection of the *Gran Turismo 3* game engine as the simulator is discussed, and its development is detailed, including the novel approach to using memory extraction techniques to develop a programmatic interface to the game engine, which may have implications for future research in developing simulation environments. Finally, a discussion of the scalability of the implementation and the generation of the expert demonstration dataset is provided. The next chapter will detail the development of the objective approach to optimizing [ANNs](#) for the kinematic system.

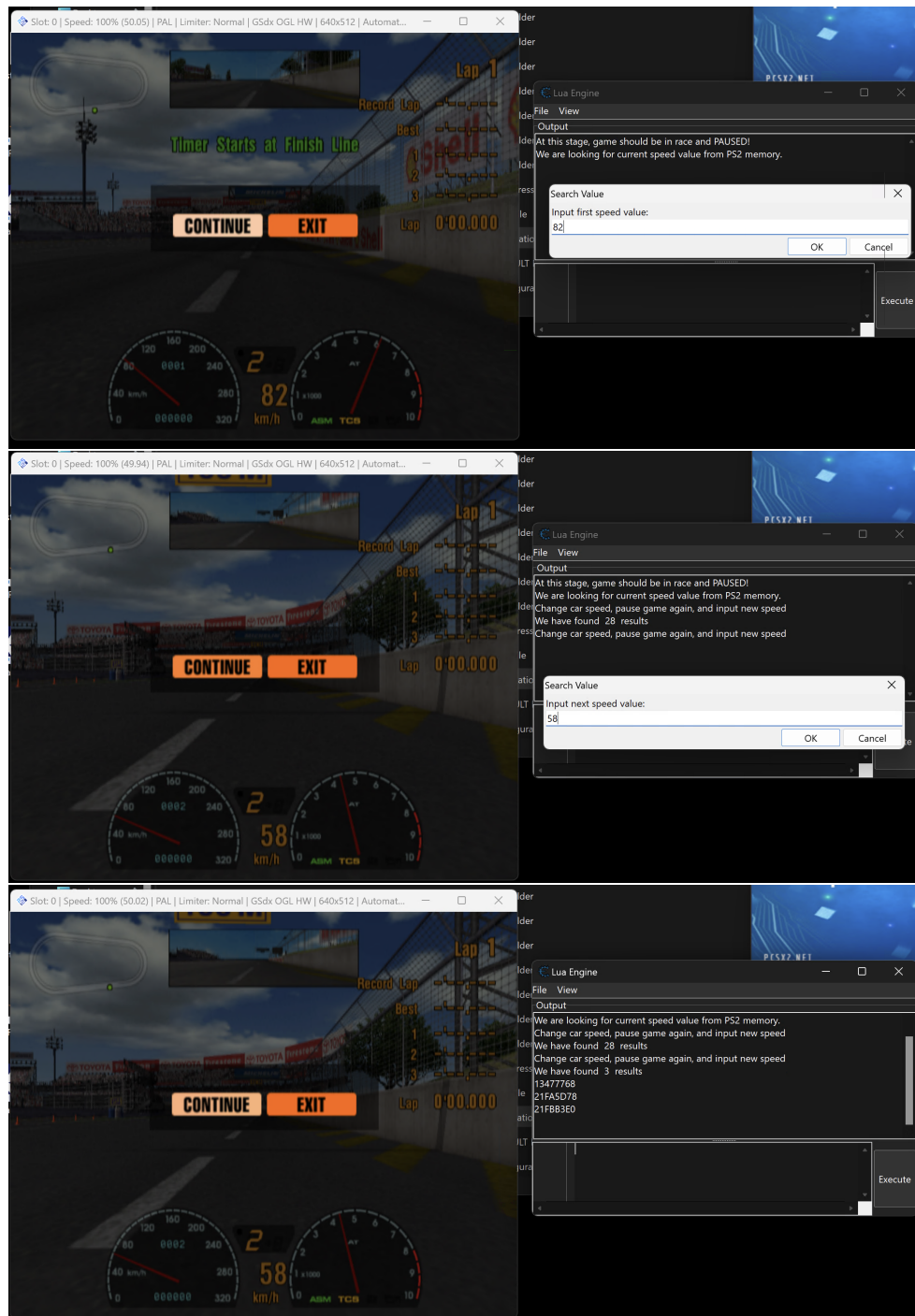
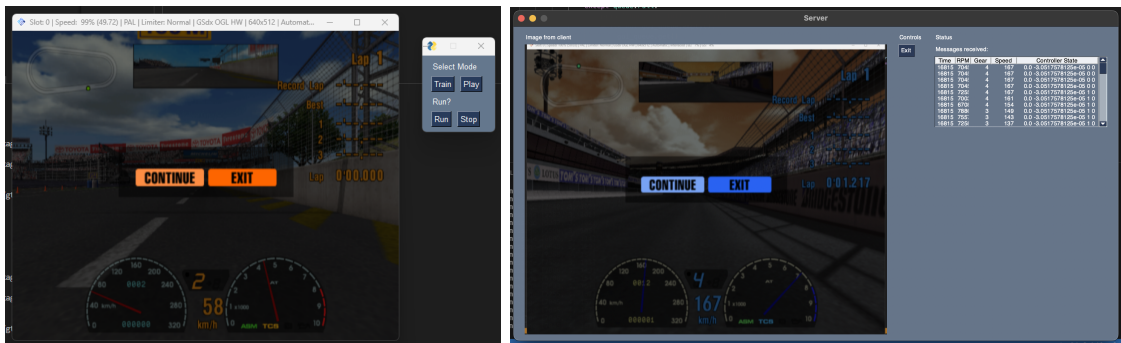


Figure 3.1: Search Process for Speed Memory Address



Chapter 4

Training Process

As identified during the literature review, there is a distinct lack of research on the optimal network structures for kinematic manipulation tasks. Using the dataset and environment developed in the previous chapter, this chapter will explore the methodology of the network structure optimization process.

4.1 Base Network Structure

The search space for neural networks can be infinite, so a base network structure must be defined. The base network structure will be used as the starting point for the optimization process, with parameters varied inside this framework. Recalling the earlier discussion on the separation of control and perception, the problem can similarly be separated into two components:

1. **Spatial Component:** The extraction of spatial features from the input images.
2. **Time Component:** The extraction of temporal features from the spatial features.

Consider the problem of controlling a vehicle through a corner, represented by a series of images. The spatial features extracted from the images include information such as the shape of the corner and the road surface. However, the vehicle's relationship with the corner is not static; the vehicle's position and orientation change over time, and the surface of the road may also change. These represent the temporal features of the problem; the relationship between the spatial features extracted from the images over time.

Following the literature review, the state-of-the-art approaches to image feature extraction employ CNNs [22]. Hence, the first layers in the network will be a CNN of varying parameters to extract spatial features from the input image.

For maintaining the temporal relationship between the spatial features, it is important to consider the types of temporal dependencies that may exist. For example, during cornering, the information informing the control decisions the agent should make are likely to be highly correlated with the vehicle's position and orientation in the previous time step. This could suggest that a simpler memory approach, such as a Gated Recurrent Unit (GRU), may be sufficient to capture the temporal dependencies; this replicates the context available to classical control methods such as PID controllers. However, there is also the possibility of long-term dependencies; the vehicle's position and orientation at the start of the corner might affect the strategy that the agent should take at the end of the corner. This suggests that a more complex memory approach, such as an LSTM, may be required. Given the results of prior work in the literature [19] [20] focusing on the use of LSTMs, to aid in the similarity of the approach, the network will use an LSTM to capture the temporal dependencies.

The structure of the network is visualized in the diagram shown in Figure 4.1.

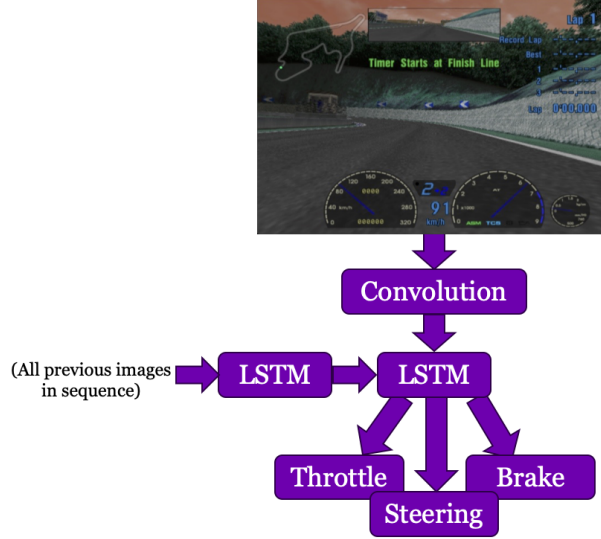


Figure 4.1: CNN+LSTM network structure

4.2 Key Parameters for Optimization

For the [CNN](#) component of the network, focused on the extraction of spatial features from the input images, the parameters listed in Table 4.1 will be optimized.

Parameter	Description	Lower	Upper
num_layers	Number of layers	1	5
filters	Number of convolutional filters	16	128
kernel_sizes	Size of convolutional kernel for each layer	3	5
pool_sizes	Size of pooling kernel for each layer	2	4

Table 4.1: Parameters for [CNN](#) optimization

An additional area of interest is the use of pretrained starting points for the [CNN](#) component of the network. The use of pretrained models has been shown to improve the training speed of networks and the quality of the extracted features [9]. As a wide variety of pretrained models exist, EfficientNetB0 is chosen as a starting point as the model is capable of scaling to the arbitrary resolutions of input images [45], enabling minimal changes to the other variables in the network structure.

For the [LSTM](#) component of the network, focused on the extraction of temporal features from the spatial features, the parameters listed in Table 4.2 will be optimized.

Parameter	Description	Lower	Upper
lstm_layers	Number of layers	1	3
lstm_hidden_size	Number of hidden units	1	32

Table 4.2: Parameters for [LSTM](#) optimization

The selection of the ranges in Table 4.1 and Table 4.2 is based on the literature review and the results of prior work in the field [29], which have tested a wider search space in depth of convolutional layers (1 to 15 layers), but found that there is no significant improvement in performance beyond these ranges. Given increasing the number of layers has the highest impact on the computational cost, the search space is limited to a maximum of 5 convolutional layers.

Finally, some data preprocessing parameters are also optimized. The key parameters for the data preprocessing component of the network are listed in Table 4.3.

Parameter	Description	Lower	Upper
scale	Scaling factor for images	2	8
sequence_length	Number of input images	1	10
sampling_frequency	Frequency of input images	$\frac{1}{1}$	$\frac{1}{100}$

Table 4.3: Parameters for data preprocessing optimization

The scale of the image dramatically impacts the performance of the network; too small and the network may not be able to extract the necessary features, too large and the network may be too computationally expensive. The sequence length and sampling frequency are also important; the sequence length determines the number of images that the network will consider at each time step (the context it has in making its decision), and the sampling frequency determines the frequency at which the images are sampled.

At the maximum sampling rate, where the latest 1 of every 100 images is used, at the 50Hz dataset of the game, images will be spaced 2 seconds apart. Combined with a sequence length of 10, where the network considers 10 images at each time step, the network will have a maximum of 20 seconds of context in making its decision. However, it is expected that the network will have a higher performance with a lower sampling frequency, as 2 second intervals is likely to be insufficient for the network to accurately control cornering; these values are provided as an upper bound to illustrate the maximum context the network can have.

It is also important to note that irrespective of the sampling frequency, the network is constrained to a maximum single-image inference time of 200ms; hence, although the images in any particular sequence may be more than 200ms apart, the network will always make 5 control decisions per second.

4.3 Efficient Selection of Network Structures

The search space is extremely large; on the order of 10^{17} possible network structures. This is not possible to search exhaustively, so an informed search strategy is required. The utility of informed algorithms is also supported in the literature for other problem types [29].

4.3.1 Tree-structured Parzen Estimator (TPE)

The [TPE](#) algorithm is a Bayesian optimization algorithm. Let x represent a vector of hyperparameters, and let $f(x)$ be the objective function to be minimized (the loss function). The goal is to find the hyperparameters x that minimize $f(x)$.

The observed trials can be split into two parts: “good” outcomes, L , and “bad” outcomes, G ; the threshold objective function value along which the observed trials are split is y . Using [Kernel Density Estimation \(KDE\)](#), the algorithm models two probability distributions from the split observed trials:

$$l(x) = \frac{1}{n_l} \sum_{x_i \in L} K_h(x - x_i)$$
$$g(x) = \frac{1}{n_g} \sum_{x_i \in G} K_h(x - x_i)$$

where K_h is the kernel function with bandwidth h (set following the Silverman’s rule of thumb, $h = 1.06\sigma n^{-1/5}$), and n_l and n_g are the number of good and bad outcomes from the current trials (members of the sets L and G), respectively.

With this representation of the probability distributions, the algorithm can maximize the probability of a trial with parameters x_{new} having good performance by maximizing the ratio of the two distributions:

$$x_{new} = \arg \max \frac{l(x)}{g(x)}$$

Effectively, this identifies the regions of the hyperparameter space where the ratio of good to bad outcomes is highest, and selects the next trial from this region. This maximization is not directly possible, so sample candidates are drawn from the range for each hyperparameter and the ratio is calculated for each. The candidate with the highest ratio is selected as the next trial.

Handling of Conditional Parameters

A key advantage of the [TPE](#) algorithm is its ability to break down the optimization problem into conditional parameters. This is particularly useful for the [CNN](#) structure optimization

problem, where the number of layers in the network is a conditional parameter that affects whether these layers need an associated number of filters, kernel sizes, and pool sizes.

This is performed by expanding the tree structure beyond only the parameters x , which exist for all trials, to include conditional parameters x_c , which only exist for trials where the conditional parameter is true. This effectively defines a separate distribution for each branch of the conditional parameter tree, and allows for the comparison of performance across different branches of the tree by comparing the ratio of the distributions.

4.3.2 Covariance Matrix Adaptation Evolution Strategy (CMA-ES)

[Covariance Matrix Adaptation Evolution Strategy \(CMA-ES\)](#) instead attempts to estimate the covariance matrix of the hyperparameter search space, and uses this information to select the next set of hyperparameters to evaluate. A limitation of this approach is that in the case of conditional hyperparameters, the algorithm may struggle to estimate the covariance matrix accurately, due to changes in the validity of the estimated covariance matrix as the conditional hyperparameters change.

4.3.3 Comparison

Reviewing the literature, [TPE](#) is shown to match the performance of [CMA-ES](#) in [46]. This same paper also compared other algorithms, including [Predictive Entropy Search \(PES\)](#) and [Expected Improvement \(EI\)](#), and found that [TPE](#) and [CMA-ES](#) outperformed these algorithms in terms of validation error.

Parallelization of both algorithms is similar. In [TPE](#), the algorithm can be parallelized by sampling multiple candidates from the argmax step and evaluating them in parallel, returning the losses to the main process for inclusion in the good or bad sets. In [CMA-ES](#), the algorithm can be parallelized by sampling multiple candidates from the covariance matrix and evaluating them in parallel, returning the losses to the main process for inclusion in the covariance matrix estimation.

Given the conditional nature of the hyperparameters in the network structure optimization problem, [TPE](#) is selected as the search algorithm for the optimization process to avoid any potential issues with covariance estimation.

4.4 Library Selection

The selection of the library to implement both the training process and the search process is critical to the success of the project. The library must be capable of handling the complex network structures required for the problem, and must also be capable of handling the search process efficiently.

4.4.1 Training Library

Two primary options exist in deep learning research: TensorFlow and PyTorch. There is minimal difference in the performance of the two libraries, but since 2021, PyTorch has been the more popular choice as measured by Google Trends [47]. Hence, to maintain compatibility with the latest research, PyTorch will be used for the training process.

4.4.2 Search Library

With the selection of [TPE](#) to both extend the knowledgebase in the literature and fit the simultaneous optimization of hyperparameters and network parameters, the algorithm must be implemented. Although it is possible to implement this algorithm from scratch, hyperparameter optimization libraries exist that can handle the implementation and also provide useful interfaces for observing the optimization process. This is particularly useful given the long expected runtime of the optimization process.

The 2022 study by Shekhar et al. [48] provides a comprehensive comparison of the performance of SMAC, Optuna, and HyperOpt, three popular hyperparameter optimization libraries. Although SMAC is utilized by prior papers identified in the literature review [27], it does not include [TPE](#) by default [28], perhaps indicating why the authors of [48] found it to have the lowest runtime speed of the three libraries. On the other hand, the authors found Optuna has better performance and runtime scores in the NeurIPS black box optimization challenge than both HyperOpt and SMAC. Hence, Optuna will be used for controlling the search process.

Notably, all three options include infrastructure for parallelizing the search process, which is critical for integration with the [RL](#) training process.

4.5 Network Restrictions & Resources

To perform real-time control, it is necessary for the network to make control decisions within a restricted time frame.

Reaction time research differs dramatically in humans depending on the task, and is difficult to measure, with variance in subtasks within the same task; for example, it takes more time for a human operator driving forward to move their foot to a brake pedal than the closed-loop control involved in maintaining a steering wheel. Additionally, there is no guarantee that the reaction time of a human operator is an optimal target for the network. Hence, the restriction on inference time is instead based on the performance of similar implementations.

The most similar implementation to this thesis is given in [49]; the authors employ a combined CNN-LSTM trained on a 30-hour dataset of control decision & screenshot pairs recorded by a human operator driving in *Grand Theft Auto 5*, achieving class-leading performance when then evaluated on the *Udacity* dataset. Such performance is achieved with an approximate time consumption per frame of 200ms [49].

Taking this as a benchmark, the network must be capable of making control decisions within 200ms, resulting in approximately 5 control decisions per second. This is enforced by running 10 single-image test inferences through the network after the network has completed each epoch. If the average inference time is greater than 200ms, the network is terminated and the hyperparameters are recorded.

An additional concern is the efficient use of resources. Network optimization can either be accomplished on a fixed-epoch basis, where each network structure is exposed to the same number of data samples, or on a fixed-time basis, where each network structure is exposed to as many data samples as possible within a fixed time frame. The fixed-time basis is judged to be more representative of the real-world scenario, where compute time is a valuable resource, and the objective is to achieve the best performance within a fixed time frame.

The network must be capable of running on a single RTX 3060Ti Graphics Processing Unit (GPU) with 8GB of memory, but simpler network structures may be able to run more training cycles in a given timeframe with higher batch sizes (more samples fed through the network at once, taking advantage of the parallel processing capabilities of the GPU). Hence, before training begins, the batch size is dynamically adjusted to the maximum possible value that the hardware can handle without running out of memory. This is accomplished by slowly increasing the batch size with error handling wrapping the training function; if the network runs out of memory, the batch size is reduced by 10% to provide

some buffer, and the full training run begins with this batch size. To enable a reasonable number of network structures to be tested, the training time is limited to 24 hours for each network structure.

To maintain the efficiency of the training process, pruning is employed, where models that are not expected to achieve a better objective value than the current best model are terminated early, following evaluation at 15-minute intervals on the test set.

4.6 Training Process

The combined CNN+LSTM architecture is trained on the dataset with the hyperparameters of each training cycle selected by the TPE algorithm. Each network is trained for 24 hours, and the performance of the network is evaluated on the testing dataset. The loss function of the network over time is plotted, and the final loss of the network is recorded, with the best result being 1.927.

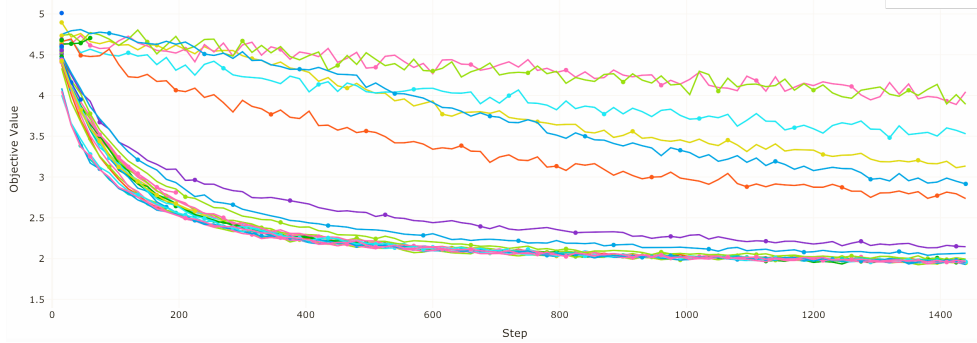


Figure 4.2: Loss function of the network over time

The image scale result is the most informative. At higher resolutions, it is expected that there is more detail available for the model to make a prediction, but the computational cost of processing the images is higher, limiting the batch sizes (and thus the total number of training samples the model can be exposed to in a given time window). In the experiment, the image scale produces a clear trend of increasing loss as the image resolution decreases, hence showing that the added spatial information is more valuable to the model than the increased exposure to new training samples.

Other papers in the literature, such as [21], have used significantly scaled down images (190x100 pixels) with good performance, suggesting that their network is able to learn the

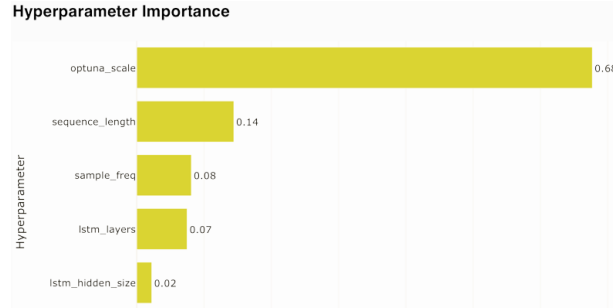


Figure 4.3: Hyperparameter importance

steering angle from relatively low-resolution images. However, the CARSIM environment in this study is dramatically simpler from a visual perspective than the *Gran Turismo 3* environment, and as such, this suggests the results of [21] may not be generalizable to more complex environments, and there is indeed significant value in spending computational resources on higher-resolution images.

Beyond this, the second strongest hyperparameter is the length of the sequence, but the relationship is inversed, with longer sequences producing worse performance. As the sequence length increases, the network is exposed to more temporal information, but the processing time of the network also increases, as it needs to feed each image in the sequence through the network. This reduces the number of training samples the network can be exposed to in a given time window. It appears that additional exposure to the spatial information is more valuable to the network than the added temporal information provided by a sequence length greater than 1. This is contrary to the conclusions of [19], which found an improvement in performance by using data sequences.

Interestingly, this behaviour is clearly visible in the loss function plot; the dense region on the bottom of the plot with many testing samples is the region where the sequence length is 1 (more trials are completed in this region due to the TPE algorithm selecting the sequence length of 1 more frequently). Above this region, there are distinct bands in which the sequence length is increases.

The behaviour of the short sequence models is reaching an asymptote around the 400min mark, while the longer sequence models are still improving. This suggests that the network may be able to achieve better performance with longer sequences, but the computational cost of processing the longer sequences necessitates a longer training window to realize these gains. Within the context of examining the ability to converge within this time window, the optimal sequence length is 1.

4.6.1 Pre-trained EfficientNetB0+LSTM Architecture

As an addendum to the combined optimization study, the pretrained EfficientNetB0 available in the PyTorch library is used as the CNN component of the network, with the classification head removed. The network is trained on the dataset with initial tests on a three-hour training window to evaluate best practices before extending to a 24-hour training window due to the limited time available for this study. The performance of the network is evaluated on the testing dataset, and the loss function of the network over time is plotted.

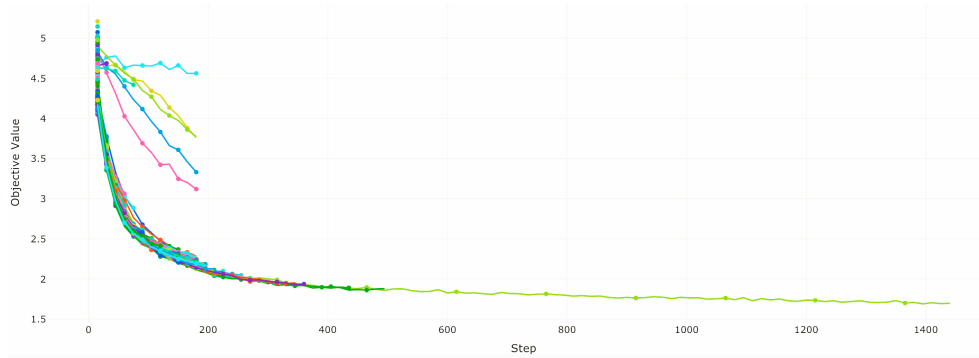


Figure 4.4: Loss function of pretrained CNN network over time

The initial results show that the pretrained EfficientNetB0 model is able to achieve a lower loss than the best CNN-only model in the same time window, achieving 1.696 loss on the initial 24-hour trial. This model appears to be similarly sensitive to the image scale factor, and also achieves this result with a single-shot sequence length of 1. However, given the limited time available for this study, the results are not conclusive, and further investigation is required to determine the optimal hyperparameters for the pretrained EfficientNetB0 model. More results in this area are expected in future work, but due to time constraints, are not conducted inside the scope of this thesis.

Chapter 5

Summary and Conclusions

5.1 Applicability to Other Kinematic Systems

The review in the case selection of other kinematic systems shows some of the underlying kinematic principles are shared between different problem areas. In particular, models of friction on unknown terrains, and trajectory optimization based on holonomic constraints, are common to other problems; in gait recognition, the friction model is necessary to ensure the robot does not slip, and the current speed of the robot informs how quickly it can change direction, enforcing a holonomic constraint.

However, there is a significant difference in the demands of the control systems of each kinematic system. In particular, the speed in which decisions must be made vary dramatically for applications such as gait recognition. The 200ms timeframe restriction for this network structure is likely unworkable for such a task, as the robot would lose balance in a fraction of that time. At the same time, the problem of balance maintenance likely has less need to process image data compared to the autonomous vehicle problem.

The concept of separation of control and perception hence becomes more important in these contexts. It may be that for certain applications, such as gait recognition, the use of a simplified model purely taking sensor data with the aim of ensuring stability and the direction of movement is more appropriate, with a separate model for longer-term path planning providing the desired trajectory. This differs from the autonomous vehicle problem, where the combination of perception and control fits within the timing constraints.

Considering segmented models, the objective optimization approach described in this literature is likely highly applicable to trajectory planning models. Considering the problem

of gait planning, it is likely that the application of guiding a robot to follow a planned trajectory could use similar network structures to derive the list of actions to take, with these actions then being executed by a separate control system. In this sense, the model is generalizable to other kinematic systems.

Irrespective of the ability to generalize the specific model to other kinematic systems, the objective approach to network design is applicable to other kinematic systems, and is a useful tool for training the highest-quality models possible within a given inference time and computational budget. Hence, the methodology of this thesis is likely to be applicable to other kinematic systems, even if the specific network structures are less so.

5.2 Limitations and Future Work

The presuppositions of the study include:

- Inference time of 200ms being a hard limit for the network structure;
- The combination of [CNN](#) and [LSTM](#) being the best network structure for the problem;
- The use of a single camera for perception; and
- The use of a single network structure for both perception and control.

The presupposition that 200ms is the upper bound on inference time is largely arbitrary based on existing good results in the literature. Within the created framework, the 200ms limit can instead be treated as a hyperparameter that is chosen by the optimizer to address the trade-off between inference time and model accuracy. This would allow for the creation of models that are more complex (and potentially accurate) at the expense of inference time, or vice versa.

A network structure other than the combined [CNN-LSTM](#) network, or a network with additional layers at different locations in the network, might improve the performance of the model. Restricting the network structure to this form is informed by the existing literature in an effort to reduce the search space, but it is possible that a different network structure could provide better results.

The use of a single camera for perception is a limitation of the memory extraction technique used for the *Gran Turismo 3* simulator. It would require a more complex interface

with the game engine to extract different camera information, increasing the complexity of the simulation interface. This is a limitation of the approach and a trade-off of using a commercial game engine as a simulation environment.

The use of a single network structure is a presupposition taken to increase similarity to existing literature, aiming for the objective of extension with objective parameter selection techniques. The feature detection and control networks could be separated, with some technique for road extraction and other key feature detection being used to inform the perception network, providing more interpretability of the model. However, the human cost of this approach is likely to be high, as it would require a significant amount of manual labelling of the data.

Further to these limitations, there are additional areas in which the project could be extended. It would be useful to test performance on a variety of different simulation environments to test the generalizability of the model to driving as a whole. The impact of [IL](#) and [RL](#) on optimal network structures is also a future expansion of interest, and the infrastructure has been created to allow for this as a future extension.

Finally, only preliminary results on the impact of using pre-trained [CNN](#) layers are presented. A more thorough investigation of the impact of pre-trained layers on the performance of the network would be useful, as well as the impact of different pre-trained layers on the performance of the network. This would provide more insight into techniques for reducing the training time for control models, but is left as a future extension due to training time constraints.

5.3 Conclusion

In conclusion, following a background analysis of the limitations of existing control systems for kinematic system control, and a literature review of the existing state of the research regarding the application of [ANNs](#) to the control and perception problem, a novel dataset and simulation environment based on the *Gran Turismo 3* game engine was created. This dataset was used to train a combined [CNN-LSTM](#) network to control a simulated vehicle in the game environment, through an objective optimization approach to network design. The resulting network structure provides insights into the optimal network structure for the problem, in particular highlighting the relative higher importance of visual information over temporal information in providing good control outputs. Preliminary results on the impact of using pre-trained [CNN](#) layers is presented, showing that the use of pre-trained layers can improve the performance of the network. The generalizability of the conclusions

to other kinematic systems is discussed, and the limitations of the study and future work are presented.

References

- [1] Gerhard Dohrn-van Rossum. *History of the Hour: Clocks and Modern Temporal Orders*. University of Chicago Press, 1996.
- [2] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, December 1943.
- [3] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.
- [4] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, 1969.
- [5] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.
- [6] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
- [7] Jacques Denavit and Richard S Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *Journal of Applied Mechanics*, 1955.
- [8] Pushpendra050. File:classic dh parameters convention.png — wikimedia commons, the free media repository, 2024. [Online; accessed 17-August-2024].
- [9] Faraz Torabi, Garrett Warnell, and Peter Stone. Recent advances in imitation learning from observation, 2019.
- [10] Quanyi Li, Zhenghao Peng, and Bolei Zhou. Efficient learning of safe driving policy via human-ai copilot optimization, 2022.

- [11] Markus Kuderer, Shilpa Gulati, and Wolfram Burgard. Learning driving styles for autonomous vehicles from demonstration. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2641–2646, 2015.
- [12] S. V. Kruglikov. *On the Separation Principle in Guaranteed Control*, pages 240–250. Birkhäuser Boston, Boston, MA, 1991.
- [13] Luis Alberto Rosero, Iago Pachêco Gomes, Júnior Anderson Rodrigues da Silva, Carlos André Przewodowski, Denis Fernando Wolf, and Fernando Santos Osório. Integrating modular pipelines with end-to-end learning: A hybrid approach for robust and reliable autonomous driving systems. *Sensors*, 24(7), 2024.
- [14] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ”why should i trust you?”: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’16, page 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery.
- [15] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2014.
- [16] Kashyap Chitta, Aditya Prakash, Bernhard Jaeger, Zehao Yu, Katrin Renz, and Andreas Geiger. Transfuser: Imitation with transformer-based sensor fusion for autonomous driving, 2022.
- [17] Andrew Lobbezoo, Yanjun Qian, and Hyock-Ju Kwon. Reinforcement learning for pick and place operations in robotics: A survey. *Robotics*, 10(3), 2021.
- [18] Justin Fu, Sergey Levine, and Pieter Abbeel. One-shot learning of manipulation skills with online dynamics adaptation and neural network priors, 2016.
- [19] Simge Nur Aslan, Recep Özalp, Ayşegül Uçar, and Cüneyt Güzeliş. New CNN and hybrid CNN-LSTM models for learning object manipulation of humanoid robots from demonstration. *Cluster Computing*, 25(3):1575–1590, June 2022.
- [20] Hesham M. Eraqi, Mohamed N. Moustafa, and Jens Honer. End-to-End Deep Learning for Steering Autonomous Vehicles Considering Temporal Dependencies. *CoRR*, abs/1710.03804, 2017. arXiv: 1710.03804.
- [21] V. Rausch, A. Hansen, E. Solowjow, C. Liu, E. Kreuzer, and J. K. Hedrick. Learning a deep neural net policy for end-to-end control of autonomous vehicles. In *2017 American Control Conference (ACC)*, pages 4914–4919, May 2017. Journal Abbreviation: 2017 American Control Conference (ACC).

- [22] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey, 2019.
- [23] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016.
- [24] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search, 2019.
- [25] Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search, 2019.
- [26] James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 115–123, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.
- [27] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proceedings of the 24th International Conference on Artificial Intelligence, IJCAI’15*, page 3460–3468. AAAI Press, 2015.
- [28] Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022.
- [29] T. N. Fatyanosa and M. Aritsugi. Effects of the Number of Hyperparameters on the Performance of GA-CNN. In *2020 IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT)*, pages 144–153, December 2020. Journal Abbreviation: 2020 IEEE/ACM International Conference on Big Data Computing, Applications and Technologies (BDCAT).
- [30] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.
- [31] Carnegie Mellon University. Cmu graphics lab motion capture database, 2003. Accessed: 2024-08-17.

- [32] Peng Wang, Xinyu Huang, Xinjing Cheng, Dingfu Zhou, Qichuan Geng, and Ruigang Yang. The apolloscape open dataset for autonomous driving and its application. *IEEE transactions on pattern analysis and machine intelligence*, 2019.
- [33] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.
- [34] Eric Rohmer, Surya P. N. Singh, and Marc Freese. V-rep: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326, 2013.
- [35] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.
- [36] Quentin Gallouédec, Nicolas Cazin, Emmanuel Dellandréa, and Liming Chen. panda-gym: Open-Source Goal-Conditioned Environments for Robotic Learning. *4th Robot Learning Workshop: Self-Supervised and Lifelong Learning at NeurIPS*, 2021.
- [37] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5(47), October 2020.
- [38] A. Goldberger, L. Amaral, L. Glass, J. Hausdorff, P. C. Ivanov, R. Mark, J. E. Mietus, G. B. Moody, C. K. Peng, and H. E. Stanley. Physiobank, physiotoolkit, and physionet: Components of a new research resource for complex physiologic signals. *Circulation [Online]*, 101(23):e215–e220, 2000.
- [39] Zhaoming Xie, Xingye Da, Buck Babich, Animesh Garg, and Michiel van de Panne. Glide: Generalizable quadrupedal locomotion in diverse environments with a centroidal model, 2022.
- [40] Florian Fuchs, Yunlong Song, Elia Kaufmann, Davide Scaramuzza, and Peter Dürri. Super-human performance in gran turismo sport using deep reinforcement learning. *IEEE Robotics and Automation Letters*, 6(3):4257–4264, 2021.
- [41] Jordan Greer. Dr. kazunori yamauchi gives lecture on gran turismo’s driving physics and production, Oct 2017.

- [42] Dark Byte. Cheat engine, Feb 2023.
- [43] Rain and Abhishek Thakur. Ra1nty/dxcam: A python high-performance screen capture library for windows using desktop duplication api.
- [44] DocSkellington xTVaser, pocokhc. Add input recording/playback functionality 2412, 2019.
- [45] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.
- [46] Ilya Loshchilov and Frank Hutter. Cma-es for hyperparameter optimization of deep neural networks, 2016.
- [47] Google Trends. Google trends: Pytorch, tensorflow. <https://trends.google.com/trends/explore?date=today%205-y&q=%2Fg%2F11gd3905v1,Tensorflow&hl=en>, 2024. Accessed: 2024-08-29.
- [48] Shashank Shekhar, Adesh Bansode, and Asif Salim. A comparative study of hyperparameter optimization tools, 2022.
- [49] Dan Wang, Junjie Wen, Yuyong Wang, Xiangdong Huang, and Feng Pei. End-to-End Self-Driving Using Deep Neural Networks with Multi-auxiliary Tasks. *Automotive Innovation*, 2(2):127–136, June 2019.

Glossary

Backpropagation A fundamental algorithm in the field of artificial neural networks, used to train models by iteratively adjusting the weights of the network to minimize the difference between the predicted output and the true output, making the network more accurate over time [2](#)

Configuration Space A mathematical concept that describes the set of all possible configurations of a system. In the context of kinematic systems, configuration space contains the set of all possible input configurations of the system, such as joint angles or end effector positions. It also includes the set of all possible obstacles or constraints that the system must navigate around [14](#), [21](#), [22](#), [24](#), [25](#)

Inference The process of using a trained model to make predictions on new, unseen data [3](#)

One-Shot Learning A type of machine learning where the model is trained on a single example of each class, rather than a large dataset. Often used in situations where a pre-existing model is fine-tuned to a new task with limited data [14](#)

Supervised Learning A type of machine learning where the model is trained on a labeled dataset, meaning each input data point is paired with the correct output. The model learns to map inputs to outputs and is used to make predictions on new, unseen data based on this learned relationship [2](#)

Training The process of adjusting the weights of a model to minimize the difference between the predicted output and the true output, making the model more accurate over time [3](#)