TU WIEN Informatics

Autonomous Racing With Attention-Based Neural Networks

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Technische Informatik

eingereicht von

Felix Resch, BSc
Matrikelnummer 11777729

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.rer.nat. Radu Grosu
Mitwirkung: Univ.Ass. Dott.mag. Luigi Berducci

Wien, 30. November 2023

Felix Resch	Radu Grosu

TU WIEN Informatics

Autonomous Racing With Attention-Based Neural Networks

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Computer Engineering

by

Felix Resch, BSc
Registration Number 11777729

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.rer.nat. Radu Grosu
Assistance: Univ.Ass. Dott.mag. Luigi Berducci

Vienna, 30th November, 2023

Felix Resch	Radu Grosu

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Felix Resch, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 30. November 2023

Felix Resch

Acknowledgements

First and foremost, I have to thank my thesis advisors, Radu Grosu and Luigi Berducci. Without their assistance and dedicated involvement in the experimental phase and the writing of this thesis, I would have never finished finished this thesis. Thank you very much for your support over the past months.

I would also like to thank Daniel Scheuchenstuhl and Stefan Ulmer, with whom I collaborated to record the human attention data and discussed some ideas for imitating them. At this point, I also have to thank my colleague Monika Farsang, who helped me find the remaining problems in my approach.

Big thanks also got out to everyone who sacrificed their time to drive a small car around a track while wearing an eye-tracking device so that we could record their attention. In no specific order: Anna Sebor, Alex Führer, Till Abele, Florian, Anna, and Moritz Christamentl.

I also have to thank my friends and family, who had to endure my stressful behavior, especially at the end of my thesis. Your support meant a lot to me and helped me to focus on finishing this thesis. Thank you.

Kurzfassung

In dieser Arbeit untersuchen wir den Einfluss von menschlicher Aufmerksamkeit auf künstliche neuronale Netze im Kontext des autonomen Rennfahrens. Zu diesem Zweck trainieren wir eine Nachahmung der menschlichen visuellen Aufmerksamkeit und integrieren sie in einen modifizierten Steuerungsansatz für autonome Rennen.

Wir lernen eine Imitation der menschlichen Aufmerksamkeit aus Aufzeichnungen menschlicher Aufmerksamkeit mit einem supervised learning Ansatz für Heatmap-Regression. Nachdem wir die Nachahmung an realen Daten und in Simulationen evaluiert haben, verwenden wir sie, um Teilbilder der Eingabe für unseren Rennregler auszuwählen. Ziel dieser Teilbilder ist es, die Eingabegröße des Netzwerks zu reduzieren. Der in dieser Arbeit verwendete Renncontroller verwendet die Decision-Transformer-Architektur, die auf Generative Pre-Trained Transformern basiert, um High-Level-Aktionen zu generieren. Anschließend verwendet er *Pure Pursuit*, einen Standard-Pfadverfolgungsalgorithmus, um diese High-Level-Entscheidungen auszuführen.

Decision-Transformer verwenden eine Variante von Offline-Reinforcement Learning für das Training, welche große Datenmengen erfordert, die wir mit Simulationen erzeugen. Wir vergleichen zwei Auswahlverfahren für Teilbilder und einen Vollbildansatz hinsichtlich ihrer Rundenzeiten, ihres Fahrverhaltens und der Varianz der Aktionswerte. In diesem Vergleich schneidet die auf menschlicher Aufmerksamkeit basierende Auswahlmethode besser ab als die anderen Ansätze, da sie schnellere Rundenzeiten und eine geringere Varianz der Ausgabewerte erzielt, obwohl das Fahrverhalten manchmal unerwünscht ist.

Diese Arbeit hat gezeigt, dass es möglich ist, ein künstliches neuronales Netzwerk so zu trainieren, dass es die menschliche Aufmerksamkeit imitiert. Außerdem haben wir gezeigt, dass künstliche neuronale Netze, die menschliche Aufmerksamkeit erhalten, ihre Leistung verbessern und stabilere Vorhersagen machen können. Diese Arbeit hat auch einige der Mängel des derzeitigen Ansatzes aufgezeigt und neue Wege für die wissenschaftliche Erforschung eröffnet.

Abstract

In this thesis, we explore the influence of human attention on Artificial Neural Networks in the context of autonomous racing. To do so, we train an imitation of human visual attention and integrate it into a modified control approach for autonomous racing.

We learn a human attention imitation from recorded human attention data with a supervised learning approach for heatmap regression. After evaluating the imitation on real-world data and in simulation, we use it to select sub-images of the input to our racing controller to reduce the input size. The racing controller used in this thesis uses the Decision Transformer architecture based on Generative Pre-Trained transformers to generate high-level actions. It then uses *Pure Pursuit*, a standard path-tracking algorithm for executing those high-level decisions.

Decision Transformers use a variant of Offline Reinforcement Learning for training, requiring large amounts of data, which we generate with simulation. We compare two selection policies for input sub-images and a full-image approach regarding their lap times, driving behavior, and variance of action outputs. In this comparison, the human attention-based selection policy outperforms the other approaches, achieving faster lap times and less variance in output values, even though the driving behavior is sometimes undesirable.

In this thesis, we showed that training an Artificial Neural Networks to imitate human attention is possible. Furthermore, we showed that providing Artificial Neural Networks with human attention can improve their performance and lead to more stable predictions. This thesis also highlighted some of the shortcomings of the current approach and opened up new directions for scientific exploration.

Contents

CHAPTER 1

Introduction

Cyber-Physical Systems (CPSs) are systems comprised of physical hardware components controlled by cyber components, usually computers. Contemporary works use two different approaches to control CPSs. The first is analytical or optimal control, and the second uses heuristic methods, such as Artificial Neural Networks (ANNs), for traditional control methods and data-driven approaches. While optimal control strategies always require a physical model to achieve optimal control, heuristic approaches can either use a model or learn the physical characteristics of the controlled system.

The reliance on a physical model limits the accuracy of optimal control to the model's capabilities to characterize the real world. If the system is non-linear, the system's behavior can cause approximation problems if the problem is linearized or lead to infeasible computation times if the calculations become too complex. ANNs can learn and handle the non-linearities of systems, as ANNs build on non-linear, usually exponential, building blocks and, through their carefully crafted structure, allow for optimized acceleration on dedicated hardware such as graphics cards.

Modern ANNs can consist of billions of parameters, even for simple problems. For complex tasks, such as Natural Language Processing, modern ANNs can consist of hundreds of billions (Chat GPT 3, 175 billion[4]) or even trillions of parameters (Chat GPT 4, 1.76 trillion[34]). The functions in ANNs use these parameters to process information. Given the high number of parameters, which usually translates linearly to the number of functions, the number of computations necessary for a single prediction is also high.

Reducing the number of computations needed is desirable, especially in time-critical control problems. Simplification for optimal control strategies is usually achieved by further reducing the complexity of the model or by approximating some aspects of the physical characteristics. For ANNs, this reduction is only possible by simplifying network architectures or reducing the size of the network's input.

1.1 Motivation

To enhance the capabilities of neural networks, a common source of inspiration is nature. From early perceptrons over deep and Convolutional Neural Networks (CNNs) to *self-attention*-based networks, all those architectures find correspondences in biological neural networks. All those approaches have in common that they require the architecture to learn what elements of their sensory inputs are relevant and which are not.

But what if a neural network had access to this information during training? Human attention, a cognitive phenomenon deeply rooted in our biology, is a mechanism that allows us to focus on specific details while filtering out distractions. This thesis explores the possibilities of neural networks with a priori access to an attention mechanism that discerns relevant information from irrelevant information.

Case-Study Applying the methods developed in this thesis is necessary to test the claims this thesis makes. Autonomous racing presents a captivating and relevant case

study in the context of this thesis. Just as human drivers rely on their ability to swiftly allocate attention to critical cues, such as the trajectory of the race track, the positions of other vehicles, and potential hazards, autonomous racing vehicles require a comparable level of selective focus to navigate the complex and dynamic racing environment, with its high-dimensionality state space and requirements for high-frequency decision making.

Drawing insights from human attention research could enable ANN-based controllers to achieve human-like decision-making skills. Since these networks build on the provided attention mechanism, their training could improve as their perceptual understanding stems from human attention. Moreover, the fusion of human attention and ANNs could pave the way for safer and more efficient autonomous vehicles in racing and broader applications, contributing to the advancement of self-driving technologies.

Attention in Neural-Networks The most common mechanism in contemporary ANN models when discussing attention in ANNs is *self-attention*. *Self-attention* is a mechanism in transformers that enables the overall architecture to weigh the significance between elements within a given input sequence, fostering an understanding of contextual relationships and dependencies. Section 2.3 gives further details and explanations about self-attention and transformers.

In addition to *self-attention*, conventional ANNs also use inherent mechanisms similar to attention. Like biological neural networks, ANNs interpret sensory or input information and, in doing so, consider some parts of the input more relevant than others. For artificial neural networks, the value associated with the relevance of certain input parts is called *saliency*.

The *saliency* of ANNs is a way to interpret the way ANNs perceive information. A saliency map for an ANN indicates the relevance of each input modality. Unlike human attention, ANNs does not necessarily consider the indicative information as relevant but might also regard non-indicative information as more pertinent. While some ANNs perform well in optimal conditions, they might focus on parts of the input that are not primary indicators used by humans for specific tasks. A prominent example in autonomous driving shows that ANNs focusing on the road instead of the areas outside the road are more resilient to errors or noise[29].

1.2 Structure

This thesis describes a possible way to infuse state-of-the-art ANNs with human attention to reduce their complexity while maintaining or improving their prediction performance. To achieve this, we propose to learn an attention filter, a network that predicts Region of Interests (ROIs) in the form of attention heatmaps in images. Other parts of this thesis use those predictions as an attention mechanism.

Chapter 2 discusses prior and related work in this area of research and their applicability to this thesis. Chapter 3 describes the general approach of how this thesis includes human

attention in ANNs.

Chapter 4 details the recording process of ground truth human attention data and the training of human attention prediction. It details the pre-processing steps for the recorded data and statistics about the recorded data. Furthermore, it describes the structure of the ANN and the specialized loss function used during training.

Chapter 5 discusses the details of the simulation environment and the racing controller, trained using Offline Reinforcement Learning (Offline RL). The description of the simulation environment includes the state and action space of the environment and the track used for simulation. After discussing the simulation environment, section 5.2 describes the generation of the dataset for Offline RL. Section 5.3 gives an overview of the used ANN architecture, and section 5.4 describes the integration of human attention into the novel ANN.

Chapter 6 describes the training regimes and methods used to evaluate the models from Chapter 4 and 5, and Chapter 7 discusses the results of this thesis and possibilities for future work.

CHAPTER 2

Related Work

Developing novel control algorithms able to outpace other algorithms and safety mechanisms in autonomous racing has always driven innovation in this field of research. This chapter explores related work, giving an overview of the landscape of racing controllers and the critical aspect of safety within autonomous racing. It further delves into previous research into biological concepts such as attention and derived concepts such as *self-attention*. We also highlight neural network architectures used in medical imaging to imitate some of these biological concepts, as they focus more on prediction biological processes than conventional labeling architectures.

In autonomous racing algorithms, the primary challenge no longer revolves around the precision of control algorithms for high-speed trajectory tracking. Instead, the concern lies in effectively navigating the interactions with other, potentially slower-moving vehicles and unexpected obstacles on the racing track. Even when controllers excel in maintaining a racing line akin to driving on rails, the issue arises when the railroad(raceline) is obstructed, potentially resulting in degraded driving performance or collisions. As a result, we focus on research that exhibits the capability, or at the very least makes a concerted effort, to overtake slower vehicles and circumvent obstacles while leveraging optimized reference trajectories as a guiding principle.

High-speed autonomous racing requires a certain degree of performance on the computing platforms to perform control computations. Even though hardware performance on embedded platforms continues to evolve, most recent developments increase the number of parallel computations, not the single-core performance. With this restriction on computation frequency a critical factor remains the decision frequency of the controller, which must be sufficient to ensure timely responses in high-speed racing scenarios. The literature on racing algorithms presents diverging approaches, with some offering practical workarounds to meet real-time demands. In contrast, others predominantly serve as simulation-only, proof-of-concept ideas. Given our eventual goal of deploying the algorithms developed as part of this thesis on real-world 1:10 racing cars, we discuss the respective applicability of these approaches in real-world racing in this chapter.

As stated in the introductory section of this thesis, integrating a selective attention mechanism inspired by biological attention is the central focus of our research. We intend to use this concept of selective attention to shrink the overwhelming amount of information presented to ANNs. In this pursuit, we have turned to the domain of medical image processing, specifically ANNs designed for this task. While these networks primarily find their application in image segmentation, their capability to deal with the inherent uncertainty in biological processes, such as attention allocation, becomes evident.

An additional facet of biological inspiration that is currently impacting ANNs is the concept of *self-attention*. This concept is a foundational building block within transformer architectures, employed in various applications, spanning from natural language processing to control and visual tasks. *Self-attention* mechanisms equip transformers with the capacity to simultaneously process multiple data elements, for example, a time series of states, and effectively determine correlations between input elements. This ability to

uncover interdependencies among data elements significantly enhances their predictive performance.

This chapter discusses the related work used in this thesis. It starts with a discussion of autonomous racing approaches, with a small side-step to autonomous driving approaches focusing on overtaking. It continues with an overview of the literature on safety in autonomous driving and racing and sources on biological attention and *self-attention* as a critical mechanism in transformers, a modern approach for processing series data.

2.1 Autonomous Racing Approaches

In autonomous racing algorithms, it has become evident that non-reactive algorithms, which incorporate additional information about the track, outperform their purely reactive counterparts. Among the pioneering algorithms to embrace the concept of tracking reference trajectories was the *Pure Pursuit*[49, 13] algorithm. This straightforward yet practical approach involves tracking a predefined trajectory and generating steering and velocity commands accordingly. Following a reference trajectory requires pre-computed trajectories. A tool that can perform multiple different types of optimizations is a tool by the *TU Munich Formula Student* team[43, 21].

In more contemporary developments, algorithms that track trajectories embrace the paradigm of Model Predictive Control (MPC). MPC is a sophisticated control strategy that differs from traditional control methods in making decisions based on a predictive model of the system's behavior. MPC operates by repeatedly optimizing a control action over a finite horizon, considering the system dynamics, constraints, and a specified performance objective. This predictive approach allows MPC to find an optimal control sequence that minimizes a cost function, accounting for future system states. As the system evolves, MPC continually re-optimizes, making it adaptable to dynamic and uncertain environments. As most current developments use MPC in some form but also introduce other new techniques, we mainly discuss approaches using MPC in the rest of this section.

Li et al.[30] employ a non-linear MPC incorporating a mixed-integer quadratic non-linear model for the control strategy. This innovative approach has demonstrated promising outcomes, particularly in overtaking maneuvers. However, a notable limitation lies in the extended computational time required for optimization, rendering the approach impractical for real-world deployment beyond simulation environments.

Bhagarv et al.[3] employ a novel Model Predictive Contouring Control (MPCC) approach, which integrates offline learning to predict opportune areas on the racing track for overtaking maneuvers. By leveraging offline learning, their method identifies zones conducive to overtaking, which are subsequently executed by the MPCC controller, effectively combining predictive and reactive elements for safer racing. In a parallel development, Brüdigam et al.[5] expand upon the MPC framework for autonomous racing by incorporating a learning component. Their approach seeks to understand and predict

the behavior of opponent vehicles using Gaussian learning, enabling a more adaptive response to dynamic racing scenarios.

A recurring drawback highlighted in MPC implementations is the considerable increase in computational cost associated with the optimization process. MPC's iterative nature, which continually reevaluates control actions based on predictive models, demands significant computational resources, making it resource-intensive. In contrast, the *Pure Pursuit* algorithm offers a distinct advantage, relying on relatively simple computations. This computational efficiency allows *Pure Pursuit* to make timely decisions with lower overhead.

Weiss et al.[51] introduce a distinctive autonomous racing car control approach, deviating from the previously mentioned MPC trajectory-following paradigm. Instead, they condition an ANN to generate a Bezier curve, which a *Pure Pursuit* controller subsequently tracks. Their innovative method has been tested successfully in simulation and on 1:10 scale cars, demonstrating the required performance levels for autonomous driving.

2.1.1 Autonomous Driving Controllers with Overtaking Capabilities

To gain insights into state-of-the-art controllers capable of overtaking, we expanded our search to autonomous driving. Unlike in racing, where the whole track is available to all agents, autonomous road vehicles must navigate within the confines of traffic regulations and shared lanes. Consequently, the primary focus of research in autonomous driving has gravitated toward the strategic decision of when and how to switch lanes safely and efficiently, aligning with the demands of urban and highway environments.

As one of the earlier works, Shamir[42] delves into the intricacies of designing an optimal trajectory for a vehicle to execute a successful overtaking maneuver. Meanwhile, Mashadi and Maijidi[32] propose a unique approach focused on overtaking moving obstacles while adhering to a straight reference line. Their method leans on non-linear optimization techniques to determine the best path for successful overtaking. In a related domain, Usman and Kunwar[45] introduce the application of the Rendezvous-Guidance Technique. This technique intercepts another object to facilitate overtaking on a two-lane highway. Their work addresses the dynamics of trailing and leading vehicles, ensuring safe and efficient overtaking scenarios in a real-world, two-lane road setting.

Petrov and Nashashibi[36] propose a three-phase overtaking strategy on lane-based streets aligning with previous approaches in the field. They utilize a comprehensive kinematic model that optimizes a set of differential equations. In a parallel development, Murgovski and Sjöberg[33] present an innovative approach that transforms a general Quadratic Programming (QP) method into a convex QP framework, simplifying the optimization process for overtaking scenarios. Meanwhile, Coskun[12] addresses the complex challenge of overtaking on multi-lane roads, considering oncoming traffic. Their solution incorporates QP and the concept of reachable worst-case sets to reduce the state space to drivable maneuvers while employing model predictive reachability sets to estimate the behavior of surrounding vehicles.

In our survey, we also discuss machine-learning approaches for lane-based overtaking. Liu et al.[31] use Q-learning for high-level overtaking decision-making on a lane-based highway. Their research also involves a comparative analysis of various algorithms designed to optimize overtaking maneuvers, seeking to identify the most effective techniques in this context. On the other hand, Chai et al.[9] employ a fuzzy adaptive controller for overtaking and evading multiple objects while simultaneously addressing multiple partially contradicting objectives.

Bak et al.[2] undertake a comprehensive investigation into the evaluation and stress testing of a diverse range of autonomous racing algorithms, mainly focusing on overtaking strategies. Their study delves into the description and discussion of various algorithms designed for overtaking maneuvers. Among the approaches explored is the *Lane Switcher*, a straightforward algorithm that facilitates overtaking by switching between the main raceline and alternative lanes. They also introduce the *Frenet Planner*[52], a continuous variant of the *Lane Switcher*, which leverages the *Frenet* frame of the racetrack to select goal points for overtaking. Both algorithms employ a two-level controller system to follow the trajectories generated.

The approaches highlighted in this section share common characteristics that underpin their overtaking strategies. Primarily, they adopt discrete lanes for overtaking, except the *Frenet Planner*, which operates in a continuous space. Additionally, these approaches fall into two overarching categories: those employing optimal control methods with extensive optimization processes, entailing significant computational demands, and those integrating ANNs within a hierarchical controller structure. Given that the majority of existing works in the field of overtaking in autonomous driving rely on lane-based strategies and machine learning-driven approaches typically feature a two-level controller layout, we have chosen to leverage these well-established paradigms as the basis for our approach.

2.2 Safety in Autonomous Operations

In formulating our reward function, we aimed to incorporate not only fundamental metrics like track progress and speed but also a crucial consideration: the safety of all executed maneuvers. It is clear that rapid progress is desirable, yet an integral aspect of safety needs addressing, as controllers must avoid collisions. To effectively embed safety within our reward function, a clear and quantifiable definition of safety becomes essential. In this section, we delve into various approaches that propose definitions and methodologies for integrating safety as a critical concept within the reward function framework.

ISO 26262-2018[1], often called the *Functional Safety for Road Vehicles*, focuses on the safety of electrical and electronic systems within road vehicles. It introduces a structured framework for safety requirements and processes throughout the development lifecycle, incorporating risk assessment, hazard analysis, and safety goals. ISO 26262 emphasizes the importance of mitigating and controlling safety-related risks in the automotive domain and defines safety as *the absence of unreasonable risk*.

The J3016C[24] *Society of Automotive Engineers* standard, titled *Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems*, provides a comprehensive set of definitions and classifications for various levels of automation within automated driving systems. It outlines specific terminology and categories to facilitate clear communication and understanding of autonomous driving technology, assisting in the development and deployment of safe and standardized autonomous vehicles.

Both standards play vital roles in defining the safety and functionality of autonomous vehicles, with ISO 26262-2018 focusing on safety processes and risk management for electronic systems and J3016C offering standardized definitions for automation levels. However, both standards do not give a quantifiable definition of safety or risk.

Most approaches we mentioned rely on imposing substantial costs or assigning negative rewards for violations of safety constraints, particularly in Reinforcement Learning (RL) applications. However, Shalev-Shwartz et al.[41] have introduced an innovative paradigm termed *Responsibility-Sensitive Safety*, which focuses on limiting penalties to only the faults of the ego agent, loosening up the safety constraints. Going further, Huang et al.[23] have introduced risk-conditioned neural networks. These networks introduce an input parameter that specifies the allowable risk, replacing the need for large negative rewards or costs for risky actions.

Two predominant approaches have emerged to quantify safety within autonomous racing scenarios. The first approach involves assigning high costs or substantial negative rewards for unsafe behaviors, discouraging the system from engaging in risky actions. The second approach delegates the task of risk estimation to the controller, conditioning it to ensure that the system never exceeds a predefined risk threshold. Another interesting takeaway is that instead of enforcing safety constraints, it is possible to adopt a more lenient strategy by penalizing only the risky actions of the ego agent.

2.3 Attention & Self-Attention

As attention is a pivotal component of this thesis, we also surveyed the topic of attention and attention mechanisms in ANNs. Our exploration examines biological attention, seeking insights from nature's selective focus mechanisms. We then delve into work dedicated to working with biological processes to facilitate imitating attention in ANNs. Finally, we explore *self-attention* and its versatile applications in controller architectures, unveiling its potential for enhancing decision-making and information processing within autonomous systems.

In their seminal work, *Attention: Theory and Practice* Johnson and Proctor[27] offer a comprehensive exploration of attention, delving into the intricate mechanics of biological attention and its underlying principles. Their detailed insights shed light on how attention functions, providing a valuable foundation for understanding this fundamental cognitive process. Preceding this work, Kastner and Ungerleider's[28] research addressed the nuances of visual attention, distinctly categorizing it into *bottom-up* and *top-down*

mechanisms. Their discussions on visual attention, which forms a central focus in our thesis, offer invaluable insights into the factors that shape our visual perceptual experiences and the interplay between involuntary and voluntary attention.

In the literature, researchers commonly describe visual attention as a process that involves directing our gaze to various points of interest through rapid eye movements known as *saccades*. These *saccades* are task-dependent, and researchers can track them as they move across different locations. As a result, the ROI consists of several points visited in quick succession. This dynamic necessitates a nuanced and fuzzy approach to handling ROIs, mirroring similar challenges encountered in other biological tasks for ANNs. Gros et al.[19] introduce a noteworthy technique for interpreting Magnetic Resonance Imaging (MRI) scans using an approach they call *soft segmentation*. Their approach employs ¸Adaptive Wing Loss[50] and a *U-Net*[38] architecture to handle fuzzy borders between relevant and non-relevant segments in MRI scans.

A similar method to top-down attention is a novel construct in ANNs called *self-attention*. *Self-attention* is a mechanism that allows ANNs to analyze input data while considering the relationships between different elements within the data. Initially introduced by Vaswani et al.[47] in natural language processing, *self-attention* has since found wide application in various domains. This higher-level cognitive processing capability enables networks to identify correlations and dependencies in data, leading to significant advancements in modern applications. Prominent examples include models like Generative Pre-trained Transformers (GPT), which have real-world applications such as ChatGPT[34], where *self-attention* enhances the understanding and generation of natural language text.

Beyond natural language processing, *self-attention* mechanisms have found relevance in various domains. In computer vision, Han et al.[20] offer a comprehensive survey of current applications of *Vision Transformers*, underscoring the versatility and expanding role of *self-attention* in processing visual data. Furthermore, Jaegle et al. present an innovative model known as the Perceiver[26, 25], which is grounded in the transformer architecture and offers a multi-modal input framework. This versatile model can effectively process different types of sensory inputs, highlighting the potential for incorporating various input modalities into transformer-based architectures.

Chen et al.[10] present the Decision Transformer, a novel control network based on the architecture of GPT. This innovative model employs multiple tokens per time step to encode essential information, including the state, reward-to-go, and actions for each time step. It subsequently generates tokens from these input modalities, predicting future outputs. Notably, in their study, the model is exclusively tasked with predicting actions while receiving ground truth data for the other modalities. They assess the efficacy of this approach through testing in both control tasks and Atari games.

While *self-attention* mechanisms are prevalent, there are relatively few approaches that leverage alternative forms of attention. Peng et al.[35] adopt an attention mechanism reminiscent of *self-attention*, complemented by 3D convolutions to tackle emotion recognition in speech data, effectively enhancing the model's ability to capture complex emotional

cues. On a different note, Yu et al.[53] introduce a unique approach by employing a separate network for attention, enabling their trajectory planning network to focus on the most pertinent surrounding vehicles selectively. In both cases, these approaches draw inspiration from simplified versions of biological attention mechanisms, aiming to refine their predictions and optimize their models for specific tasks.

In this survey of attention and *self-attention*, we found that there are only so many existing methodologies that use attention mechanisms in a way that aligns with the objectives of this thesis. While *self-attention* has gained prominence, it predominantly serves as a high-level method, endowing networks with the capacity to recognize and model correlations between input elements. However, the exploration of the effects of human attention on ANNs remains an area with limited prior research despite its potential to offer valuable insights into optimizing network performance based on human-like attention mechanisms.

CHAPTER 3

General Approach

This chapter describes the methodologies employed within this thesis, ascribing the specific challenges the research effort addresses. The all-encompassing challenge is to show that achieving convergence in ANNs is possible using focused and reduced input data. Convergence in the context of ANNs refers to the network's ability to reach a stable state where it can effectively learn and adapt its parameters to solve a given task. The fundamental difficulty lies in the unpredictability of this convergence process, with instances where it proves elusive or unattainable.

When ANNs encounter convergence issues, modifying the dataset is a common strategy to address the problem. The assumption is that providing more data can overcome these convergence challenges. To achieve this, ANNs receive more input modalities, sometimes even components of the unobservable state or extensive sensor readings. However, using extensive inputs for ANNs usually results in more complex networks that require more computational power, making them difficult to deploy on embedded platforms. Conversely, some approaches use less input data or simpler networks with similar prediction results, offering a potential solution. This observation prompts the following intriguing question:

Could ANNs achieve convergence with a reduced, more focused dataset?

In this thesis, we propose to answer this question by providing the ROIs within the input data relevant to an ANN. However, the ROIs require prior knowledge or an understanding of the information's significance for the model. A potential solution to this dilemma lies in biology, where biological systems employ attention mechanisms. These mechanisms naturally guide the system's focus to pertinent information, effectively simulating selective data preprocessing. It may be possible to replicate this biologically inspired strategy by incorporating attention mechanisms into ANNs, enabling the network to discern and emphasize crucial elements within the input data.

This thesis explores the possibilities of employing an a priori attention mechanism with a state-of-the-art controller for a non-trivial control problem. First, it describes the process of recording and imitating task-specific human attention with an ANN. Then, this thesis introduces a control approach, based on the existing controller, used for the underlying control problem and the novel integration of the *Human Attention Imitation* into the new approach.

3.1 Human Attention Recording & Imitation

Incorporating attention mechanisms into autonomous systems necessitates the creation of a standalone attention component. During the training phase, the need for such a component may appear less pressing, mainly when datasets tailored to the specific task allow the training of both the ANN and the attention mechanism. However, the practical challenge arises when deploying or employing these systems in RL settings. In these scenarios, prerecording human attention for all possible settings becomes an infeasible

Figure 3.1: VPS19 Smart Unit (left) and eye-tracking glasses (right) used for eye-tracking in this thesis. This eye-tracking device records the wearer's gaze and a video stream from the center of the field of view (light circle in the middle of the glasses).

task, rendering a standalone attention mechanism a crucial element to ensure the system's robustness and adaptability in real-world environments.

This thesis operates under the fundamental assumption that attention is inherently task-specific, tailoring its focus according to the requirements of the given task. To effectively explore the concept of attention, selecting a task in which some form of attention can be accurately recorded and analyzed is imperative. In this regard, the thesis confines its scope to the domain of visual attention, specifically delving into identifying and tracking ROIs within input images.

Capturing visual attention constitutes a pivotal aspect of this study. To ensure the effectiveness of the data collection process, it is imperative that human subjects can perform the designated task with minimal deviation from their typical performance, thereby preserving the naturalness of their responses. In addition to eye-tracking data, recording sensor data and human control inputs is essential for further machine-learning tasks. Furthermore, the data collection setup design should facilitate the reconstruction of pertinent information from the recorded sensor data.

The objective regarding this thesis's *Human Attention Imitation* component is the development of a task-specific attention mechanism derived from human attention recordings, intending to utilize it in subsequent downstream tasks. It is critical to underscore that the developed approach is a proof of concept. Consequently, expectations concerning its performance should remain tempered. The proposed approach will utilize ANNs to generate attention maps for input images akin to image segmentation. Through

this exploration, the thesis aims to validate the feasibility and potential effectiveness of leveraging attention mechanisms to improve the training and prediction performance of ANNs in various application domains.

3.1.1 Image Segmentation as Basis for Human Attention Imitation

Assigning values or classes to individual pixels is no new problem but has been researched extensively. There are two main approaches to processing images and creating different maps of values or classifications. Image segmentation aims to divide an image into meaningful segments or regions, and heatmap regression focuses on predicting the likelihood or confidence of the presence of specific objects or key points within the image. In some literature, people use heatmap regression and image segmentation interchangeably because heatmap regression can be considered a continuous specialization of image segmentation with only one class.

Image segmentation usually creates a mapping between pixels and classifications, resulting in a segmentation of the input, hence the name. Many applications rely on understanding the spatial layout of objects within an image. Medical imaging, autonomous vehicles, and object recognition are just a few examples of such applications. Implementations commonly output an image with N channels per pixel to indicate the classification score per pixel. The goal is to train the model to indicate only one class per pixel.

Heatmap regression, on the other hand, is a technique for localizing or detecting objects within an image by predicting a heatmap that highlights the ROIs. It focuses on predicting the likelihood or confidence of the presence of an object at each pixel. The output is a heatmap where higher values correspond to regions where the model believes an object is present. Many applications, such as human pose estimation, facial keypoint detection, and object localization, use heatmap regression. Implementations usually provide an output map with only a single channel per pixel to indicate the probability that a pixel contains the target.

As mentioned, the two approaches are similar, with the number of output channels per pixel and continuous and discrete outputs as the main difference. Therefore, ANN architectures for image segmentation are generally usable for heatmap regression. *Soft segmentation*[19] is an approach that uses this similarity, as it uses an architecture traditionally used for image segmentation to perform heatmap regression. This approach nevertheless needs adaptations to the training process to achieve the continuous probability scores desired in heatmap regression.

3.2 Autonomous Racing as Case Study

Autonomous driving and especially autonomous racing are currently at a level where elementary tasks, such as lane keeping or following a trajectory, even with error correction, are already well-researched. Currently, interactions with other vehicles, such as overtaking or even just driving on the same stretch of track, have become the research focus in these

areas. Most approaches rely on reactive methods to drive alongside another vehicle or to overtake it.

While these approaches show commendable results, they are often subject to indecision, especially in scenarios where the approaches have to decide between two seemingly equal options. In those cases, an approach may pick one fixed approach or oscillate between the two options with each execution of the control logic. While the first solution might yield suboptimal results, the latter can result in crashes when the algorithm does not decisively execute one option. This indecisiveness in controllers highlights the need for a high-level decision-making component.

The unresolved problem of high-level decision-making and the fast-paced environment of autonomous racing make it an interesting case study for our proposed approach. Performing timely decision-making on constrained hardware is a task that could benefit from the reduced complexity of ANNs with fewer input data.

At the same time, the task of autonomous racing provides a straightforward opportunity for the recording of human attention and decision-making processes. Instead of full-sized vehicles, this thesis employs a smaller platform to reduce the high costs associated with operating and adapting conventional cars. In this thesis, we use the *F1Tenth* platform[16], which offers a scaled-down yet highly functional alternative to full-sized cars. It encompasses a complete software and hardware stack, making the platform conducive to experimental research in autonomous racing. Furthermore, the ease of integration for remote control systems simplifies developing and testing autonomous algorithms and recording human driving behavior.

Additional sensors, such as cameras, can be seamlessly incorporated into the *F1Tenth* software and hardware stack. Integrating cameras into the platform allows for establishing camera streams directed to remote locations where human drivers can remotely control the vehicle, facilitating the recording of human attention during autonomous racing experiments. The *F1Tenth* platform offers flexibility regarding control mechanisms, as it accommodates both joypad-based control interfaces and more immersive setups, such as steering wheels and pedals.

Many simulators exist for autonomous racing and driving scenarios. Given the research objective of employing RL for racing control, leveraging simulation environments becomes an imperative strategy. Simulated environments facilitate the development and training of RL-based racing controllers and provide a controlled and reproducible setting for experimentation. Extensive literature and resources are available in the broader autonomous driving and racing field, offering valuable insights and established methodologies.

A substantial body of literature on autonomous driving emphasizes the utility of lane-based information for dynamic driving decisions. Autonomous driving algorithms in this literature adhere to predefined lanes and make agile decisions based on them, such as evasive or overtaking maneuvers. This thesis intends to employ a hierarchical approach to decision-making, recognizing that more intricate and context-aware decisions may not meet the timing requirement of a fast-paced domain like autonomous racing. It

Figure 3.2: *F1Tenth* racing car from Vienna Technical Universities team used at the 11th *F1Tenth* Grand Prix at ICRA 2023 in London. The car performs all calculations with the onboard computation unit and uses LIDAR and inertial sensors for navigation.

Figure 3.3: Simulated RGB camera image of an *F1Tenth* car in the simulator maintained by Brunnbauer and Berducci[6].

will use a hierarchical planner, with a high-level planning module orchestrating complex decision-making processes and a low-level controller responsible for lane adherence.

Initially, the technical approach in this study involves the implementation of a pre-existing state-of-the-art control methodology tailored for autonomous racing applications. Subsequently, this thesis augments the established control framework by integrating a *Human Attention Imitation* component. A randomized control approach is employed for comparative analysis to assess *Human Attention Imitation*'s impact on autonomous racing systems' performance. Specifically, this thesis compares the outcomes of the autonomous racing system with *Human Attention Imitation* against a control scenario with randomized attention allocation.

The primary emphasis of this thesis centers on conducting a qualitative comparison rather than quantitative analysis, with a particular focus on eschewing traditional metrics such as lap times. Instead, the thesis seeks to establish a compelling proof of concept, highlighting the feasibility of using a selection mechanism to train and operate ANNs on reduced input data.

CHAPTER 4

Human Attention Imitation

This chapter describes the development of the *Human Attention Imitation* used in simulation tasks to approximate human attention. We describe the data recording process for the human attention data and then detail the architecture and loss function used for training the neural network.

In the section about the recording process, we also discuss the pre-processing steps undertaken to enable the *Human Attention Imitation* to train on the recorded data. This discussion includes details about transformations and synchronization of different data streams and statistics about the recorded data. Finally, we also describe the attention aggregation used to derive ROIs from the recordings of gaze traces.

While discussing the ANN architecture used for the *Human Attention Imitation* we also describe the loss function employed during training for this model. We also include a short comparison and rationale for why we selected this loss function as the loss function used for this model.

4.1 Human Attention Measurement Setup

We asked human drivers to drive remotely controlled model cars on different tracks to record attention data for our imitation of human attention, further referred to as the *Human Attention Imitation*. The drivers steer an *F1Tenth* car with a camera mounted at the front while an eye-tracking device records their gaze.

Software Stack *F1Tenth*[16] is a racing series for autonomous racing cars at the 1:10 scale. The cars are based on the chassis of the *Traxxas Slash Ultimate* and use an *NVIDIA Jetson Xavier NX*[44] computing unit for autonomous control. Additional sensors, such as a *Hokuyo UST-10LX LIDAR*[46] sensor and two inertial measurement units, can be used for autonomous control.

The *F1Tenth* software stack utilizes the open Robot Operating System (ROS)[39] as its foundation. ROS organizes various components into nodes that communicate through message-passing over named channels known as topics. ROS uses the same communication channels to realize service calls (a request message followed by a response message) and actions (similar to a service call, but with messages sent during the processing).

On top of the Inter-Process Communication (IPC) framework, ROS provides an extensive standard library with common tools used in robotics. The standard library includes handling log messages, diagnostics, a transformation tree implementation that handles static and dynamic transformation between physical frames, and many other packages.

The nodes can issue static or updates to dynamic transformations via ROS messages. The transformation tree library listens to these messages and builds the transformation tree in each component that requires it.

The base software stack, developed at the University of Pennsylvania, implements the nodes for communication with standard sensors and input devices. It also contains

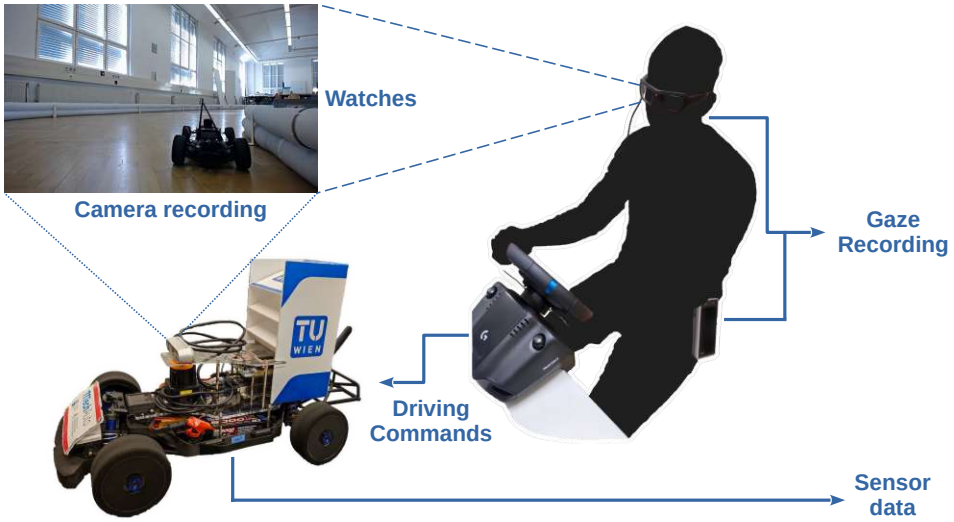


Figure 4.1: Recording setup used for human attention data recordings. A human driver, wearing an eye-tracking device, remotely controls the 1:10 scale *F1Tenth* car and watches a transmitted camera stream from the car.

low-level actuator controllers and primitive safety features like a deadman switch. We implemented control and planning nodes as part of this thesis or used existing implementations if they existed.

Setup Figure 4.1 shows the setup of the human attention recording process. A human driver is seated in front of a table with a screen, a steering wheel, and pedals. The drivers can control the car using either the steering wheel or a Gamepad. At the same time, the screen shows the live camera feed from the car's mounted camera.

We used a laptop as a central device to control all connected devices via WLAN. This laptop sends the steering commands from the steering commands and pedals or the gamepad to the motor controller on the car. In turn, it receives and displays the video stream from the front camera of the car. To minimize bandwidth usage and potential packet loss, the *Jetson Xavier NX* encodes the video stream before transmitting it over the network. The encoding step uses *NVIDIA's* hardware-accelerated video encoder to reduce the latency and size of the video stream.

Another critical aspect that we needed to address is the latency of the transmitted control inputs and the video stream. Brunnström et al.[7] have shown that even slight delays of 20ms, in addition to the intrinsic system delays, can cause decreased operator performance or even nausea in remote control tasks. During initial tests, we noticed that the latency was generally in acceptable ranges but sometimes exceeded the mark of 20ms. These exceedances happened, for example, when the car was driving fast or something obscured the line of sight between the WLAN Access Point and the car.

Recorded Data	Format	Storage	Time Frame	Spatial Frame
Driver's View	MKV/MP4	*VPS 19*	*VPS 19*	Driver's View
Gaze points	TSV	*VPS 19*	*VPS 19*	Driver's View
Car's Video	MP4	*Jetson Xavier NX*	Car Camera	Car Camera
Depth Video	rosbag	*Jetson Xavier NX*	Car ROS	Car Camera
Driving controls	rosbag	*Jetson Xavier NX*	Car ROS	N/A
LIDAR scans	rosbag	*Jetson Xavier NX*	Car ROS	Car Lidar
IMU 0	rosbag	*Jetson Xavier NX*	Car ROS	Car IMU 0
IMU 1	rosbag	*Jetson Xavier NX*	Car ROS	Car IMU 1
Raw motor commands & Diagnostics	rosbag	*Jetson Xavier NX*	Car ROS	N/A

Table 4.1: Recorded data streams with storage location and spatial and temporal frames. This table lists each recorded stream with the storage file format, where it is stored, and the temporal and spatial frame of the data.

To cope with these occasional high-latency transmissions or packet drops, which negatively impacted the human driver's ability to control the vehicle, we upgraded the networking hardware to a *Netgear WAX630E* access point. This tri-band enterprise access point offered the necessary bandwidth and signal strength to achieve low-latency transmissions for the whole area of the track.

Gaze Recording A *VPS 19 System*[48] eye-tracking device records the gaze of the human drivers and their view while driving. Simultaneously, the *Jetson Xavier NX* on the car records the driving inputs of the human driver, the transmitted video stream, a depth video stream, and sensor data from inertial measurement units.

Table 4.1 shows the data streams recorded on the two devices used for recording with their respective time and spatial frames. According to its documentation, the *VPS 19 System* data (driver's view & gaze points) are synchronized and use the same spatial frame for recording, as they use the same sensor.

Similarly, data recorded by the ROS data recording utility `rosbag` has a timestamp based on the same clock since all software components are running on the same physical device. We added all sensors to the ROS transformation tree, including their static transformations in the recorded data. The transformation tree includes the spatial frame of the mounted camera, and the time frames of the car's video stream and the depth video stream should be the same.

Recording Variations We recorded data in different rooms with different lighting conditions and floors for higher variation in recorded data. Using different rooms also allowed us to vary the layout and complexity of the track. All these variations led to

changes in recording quality. While some recordings only had limited use due to lighting conditions, some tracks proved too tricky to drive for collecting reliable attention data, as the drivers moved too much and the recorded images of the driver's view became blurry.

In this study, we selected a small set of drivers and tried to keep the entry requirements as minimal as possible to record a dataset with diverse skill levels. Each driver had at least one year of driving experience and some experience in racing setups, such as kart racing or car racing games.

Simulators The *F1Tenth* community commonly adopts a simulator for multi-agent racing, which can simulate most vehicle sensors in real-time. However, it cannot provide visual inputs or the 2D depth measurements we planned to use in this thesis. For this reason, we do not use it in this thesis. Brunnbauer et al.[6] maintain another simulator based on *PyBullet*[8], a faithful physics simulation library. This simulator can simulate visual inputs and 2D depth measurements at the cost of real-time capabilities, which makes recording with this simulator infeasible.

As there was no feasible way to record simulated data with human attention, and large-scale RL tasks are infeasible on real-world cars, the attention filter needs the capability to predict human attention in simulated and real-world recordings.

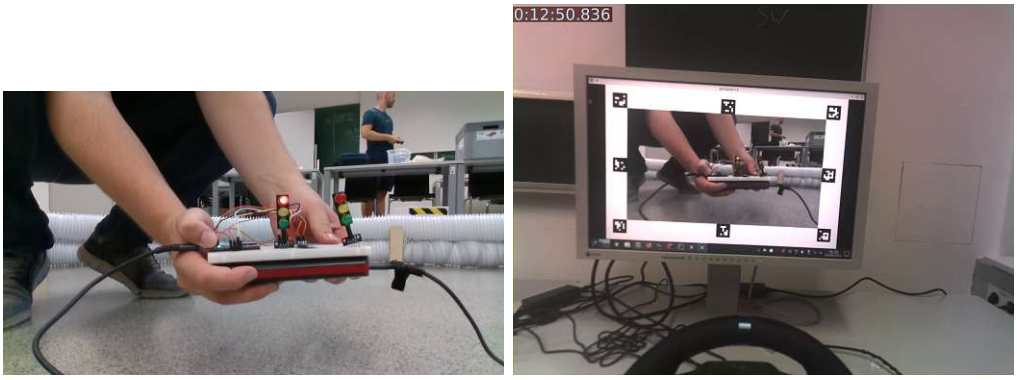
4.1.1 Data Synchronization

Since we use the onboard computing unit and *VPS 19 System* to record the data and both systems record in different time and spatial frames, the time and spatial frames of the individual data streams are different as well. Before the downstream tasks could use the recorded data, we needed to synchronize the data and define the transformations between the different spatial frames.

In order to synchronize the data streams, which consist of gaze points, driver's view from the eye-tracking device, video stream, and sensor recordings (including the depth video stream), we have incorporated synchronization points into the recording or used distinct points in the existing recordings.

For visual recordings, we used a small traffic light with a simple sequence of lights to find the matching frames in both the eye-tracking system recording and the video stream from the car's camera. An FPGA with a high-frequency quartz oscillator generated the light sequence on a small LED traffic light. Each sequence element was active for precisely the time of two frames, and the camera sensors recorded them in a maximum of three frames. The frames with the sequence in each video stream mark the synchronization points between the car's stream and the driver's view.

We used a similar technique for the sensor readings and the video stream from the car's camera. Instead of the traffic light, we used the first jerk of the car when the driver started driving. This motion is visible on the video stream, and the inertial measurement units observe it as an acceleration peak. Since we recorded all sensor measurements in



(a) Front camera on car. (b) Drivers view.

Figure 4.2: Frames of a synchronization point between the driver's view and the recording from the camera on the car. The traffic light displays a light sequence to synchronize the data streams.

one synchronized data stream, we matched the peak's timestamp with the motion's frame to synchronize those streams.

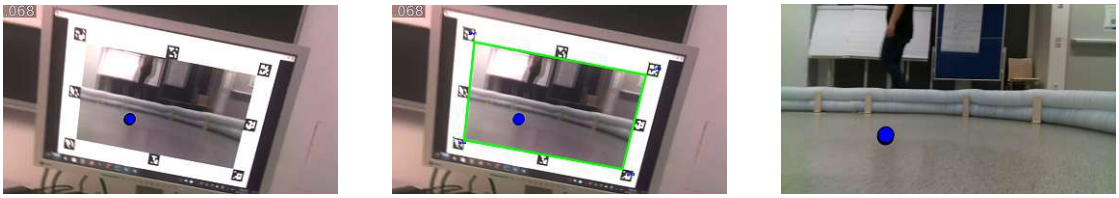
4.1.2 Gaze-Point Transformation

To determine the transformation from the driver's view to the car's stream displayed on the screen, we placed eight *ArUco*[18] markers around the video stream. We used *OpenCV's*[22] *objdetect* library to detect the markers in the driver's view recorded by the *VPS 19 System*. It then estimates a pose of the car's video frame in that recording, determining the perspective transformation between the two video frames. Finally, it transforms the gaze points from the *VPS* frame to the car's video frame.

OpenCV's objdetect library allows defining *ArUco* boards, patterns of individual *ArUco* markers. Each marker encodes its ID, and the board defines the location of each marker. The library can perform pose estimation on boards using the correspondences of image pixels and the board's definition using OpenCV's Perspective-N-Point solver. Detecting a whole board provides more stability for pose estimation than individual markers, especially if not all markers are visible.

Perspective-N-Point Problem	
Input:	A set of 3D points, their 2D projection correspondences in an image and the calibration parameters of the camera that took the image
Output:	Estimated pose of the 3D points in the frame of the camera

Figure 4.3 shows the process of video processing and gaze point transformation. *ArUco* marker detection finds the individual *ArUco* markers in the driver's view video, and



(a) Driver's view with individual markers

(b) Board in the *VPS 19 System's* video stream

(c) Individual gaze points in the car's video

Figure 4.3: Video alignment and transformation process. The alignment process detects the *ArUco* markers in the driver's view and then calculates the transformation between the car's video and the driver's view to transform the recorded gaze points.

pose estimation estimates the board's orientation in the frame. Based on the estimated pose, we transform the recorded gaze points into the car's frame with the perspective transform.

4.1.3 Post-Processing and Data Aggregation

In this section, we outline the steps involved in preparing the collected data for training purposes. Our first step is to eliminate sections of the video recording that do not contain driving or footage of drivers crashing. For crashes, we remove a more extended part of the recording before the collision to prevent models from training on crashes. Our second step is to discard frames that do not contain any usable data.

We filter out irrelevant data in our system by discarding frames and associated data if the transformed gaze points lie outside the scope of the recorded video or if there are no gaze points to be tracked, such as when the driver is blinking. We also discard frames that contain visible compression artifacts or show transmission errors, as they can affect the accuracy of our attention filter.

Even if the artifacts do not appear in the recorded stream of the car's video, they can still impact the driver's reaction and potentially lead to undesired behavior in the attention filter.

We ensured that there was sufficient data available for different track sections by categorizing each recording based on the track type. The classification included straight sections, left or right sweepers (curves), left or right hairpins, and chicanes. Figure 4.4 shows an exemplary classification for the track used in other tasks in this thesis. Generally, we consider primarily straight sections, even when they contain slight bends, as straight sections. We identify corners with a right angle or below as sweepers, and we identify corners above a right angle as hairpin curves. Chicanes are a combination or sequence of curves.

These classifications are never definite, and the classification of this dataset also depends on when the driver starts to maneuver for a specific type of corner. It is also debatable

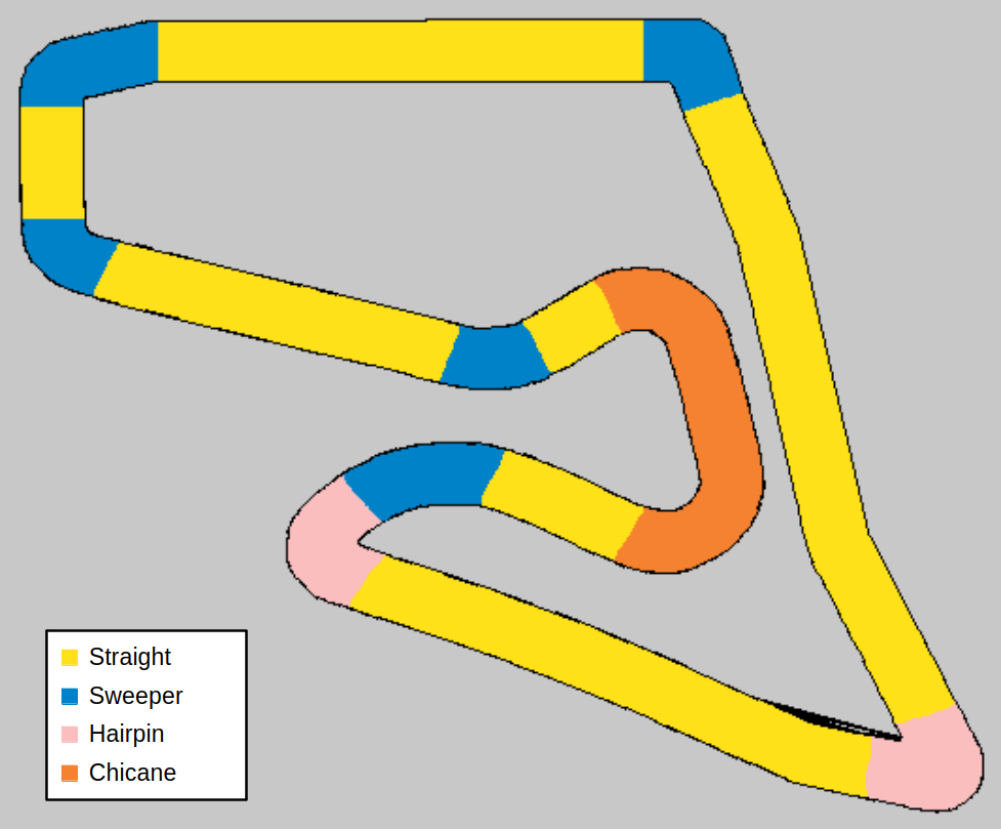


Figure 4.4: Track with colored section types. The classification is not definite, especially since the border between two sections is often debatable.

Section Type	Total Frames	Frames with Gaze Points	Usability Rate
Straight	21,111	13,144	62.26%
Sweeper	19,467	10,847	55.72%
Hairpin	16,456	7,279	44.23%
Chicane	5,904	1,475	24.98%
Total	**62,938**	**32,745**	**52.03%**

Table 4.2: Recorded frames in different track types and the frames per type with gaze points. The usability rate gives the rate of frames with gaze points compared to the total frames.

whether a set of corners is simply close to each other or if it is a chicane. We, therefore, focused more on the distribution of straight sections, sweepers, and hairpins as they show different types of behavior over the time horizons considered in this thesis.

Table 4.2 shows the number of frames in the relevant sections of the video recordings, the number of frames with estimated gaze points that our tool could transform into the

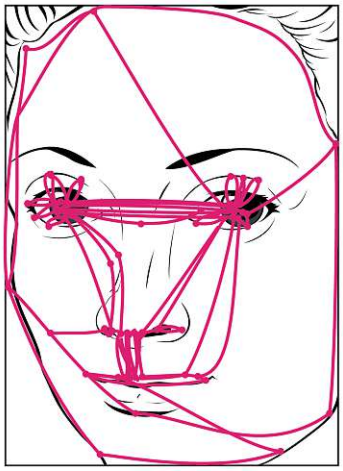


Figure 4.5: Typical focus points when looking at a human face, commonly known as *saccades*. These movements are rapid movements that move the *fovea*, the area of the eye with the highest resolution, to watch one specific point in the field of view.[11]

car's video stream, and the rate of usable frames with gaze points. As it is possible to mirror images to obtain a left sweeper image from an image of a right sweeper, this table shows the aggregated numbers for sweepers and hairpins.

Human eyes can only focus on one narrow point at a time. Therefore, for human brains to see an object as a whole, the eyes focus on many different points of a single object. The movements the focal points of the eyes make are called *saccades*.[11] Figure 4.5 shows typical *saccades* for analyzing a human face, which shows that the individual points the eyes are looking at vary greatly in location, even though only one region of interest, the face, exists in the image.

To achieve smoother changes between frames and, by doing so, obtain more straightforward training data, we construct attention maps for each frame. In this map, pixels get assigned higher values if they are visited more often. The *VPS 19 System* eye-tracking device also visualizes the *saccades* this way.

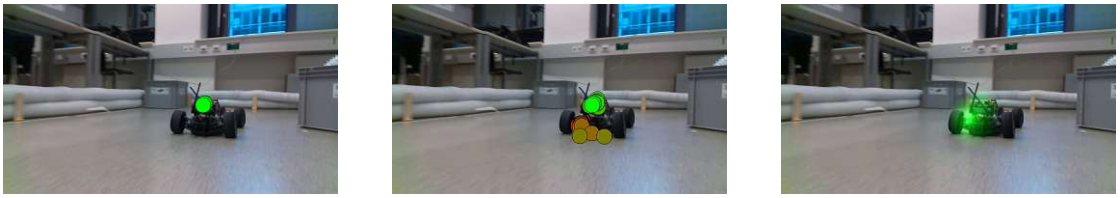
$$\mathbf{AHM}_t = \tag{4.1}$$

$$\sum_{t'=0}^{15} Blur\left(Circles\left(GP_{t-t'}, 5\right), 25\right) + \tag{4.2}$$

$$\sum_{t'=1}^{5} Blur\left(Circles\left(GP_{t+t'}, 5\right), 25\right) \tag{4.3}$$

$$\overline{\mathbf{AHM}}_t = Norm(\mathbf{AHM}_t) \tag{4.4}$$

Equations 4.1 to 4.4 explain how we generate the attention heatmap for a frame from



(a) Gaze points for the current frame.

(b) All gaze points considered.

(c) Resulting attention heatmap.

Figure 4.6: Gaze point aggregation steps to obtain attention fields. We aggregate the sets of gaze points for 15 time steps in the past and 5 in the future, apply Gaussian blur, and normalize the result to generate attention heatmaps.

the set of gaze points GP_t, which gives the set of gaze points at time t or an empty set if t is smaller than 0 or greater than the number of frames. The function *Circles* draws a circle for a set of points with the radius given in the second argument on an image the size of the original video stream. We then apply a Gaussian blur with the function *Blur* to soften the generated heatmaps.

For each time step, we aggregate 15 frames from the past and 5 frames from the future and add them together. By doing so, we intend to train the *Human Attention Imitation* to anticipate future changes in the ROIs. In the final step, we normalize the aggregated weights **AHM** to get the resulting normalized attention heatmap $\overline{\mathbf{AHM}}$.

Figure 4.6 shows the aggregation process for a single frame. We start with the gaze points for a single frame, as shown in Figure 4.6a, and then add the other sets of gaze points in the second step, displayed in Figure 4.6b. The last figure, Figure 4.6c, shows the result of aggregation and normalization. The network can now train to focus on the vehicle instead of predicting one specific point that changes rapidly.

4.2 Human Attention Imitation Architecture

We consider the problem of generating attention values for individual pixels in an image as a variation of heatmap regression. To represent attention maps, we use normalized heatmaps with values between 0 and 1. A value of 0 indicates low relevance of the pixel, and a value of 1 indicates high relevance. In contrast to segmentation, heatmap generation generates values between the boundaries, representing continuous relevance values between low and high relevance.

Gros et al.[19] developed *SoftSeg*, a soft segmentation approach, to achieve continuous segmentation distributions for medical imaging. The network for predicting attention uses a model similar to their approach. Instead of using binary masks as labels, the network trains on continuous attention heat maps, constructed as described in section 4.1.3. *SoftSeg* uses the *U-Net* architecture, a network architecture commonly used for image segmentation.

When dealing with time series data, most use some form of Recurrent Neural Networks (RNNs), for example, Long Short-Term Memorys (LSTMs). Those networks process all frames of the time series to predict the result for the last frame while they maintain some form of internal state or feed their output back in as additional input. This recurrence allows RNNs to track the state over multiple frames, which, to guarantee stability, needs to have a damping factor applied in each iteration.

This damping factor has the effect that the information for the first frame has only limited influence on the final prediction. Especially during optimization, this problem leads to minor or no changes for earlier frames, as their respective gradients can become too small due to floating point representation issues. This problem is known as the vanishing gradient problem in RNNs.

In this thesis, we mitigate the vanishing gradient problem by employing a single-shot approach that predicts an attention heatmap based only on a single frame. To achieve this, we train the *Human Attention Imitation* on the aforementioned aggregated attention heatmaps. As the aggregated heatmaps already contain additional temporal information, the model needs no additional frames to predict relevant data. An additional benefit is that due to the constructed similarities between two adjacent frames, the network trains on less erratic training data.

4.2.1 U-Net neural network architecture

Ronneberger et al.[38] first introduced *U-Net* as an architecture for medical image segmentation. It consists of three stages: a downsampling stage, an intermediary convolutional stage, and an upsampling stage. The downsampling stage decreases the image size while increasing the feature size with convolution. In the upsampling stage, up-convolution layers decrease the number of features and increase the image's size back to the original size.

The downsampling stage consists of N downsampling sub-stages. Each substage consists of three convolutional layers with Rectified Linear Unit (ReLU) activation. The downsampling sub-stages feed their output to the next sub-stage through a max-pool layer.

The last downsampling stage feeds its output into the intermediary[1] stage. This stage consists of three convolutional layers with ReLU activation. The last convolutional layer feeds its output to the first upsampling sub-stage.

The upsampling stage consists of N sub-stages. Each sub-stage consists of three convolutional layers with ReLU activation. In addition to the output of the prior sub-stage, each sub-stage also uses the output of its corresponding downsampling stage. Up-convolution layers connect the individual sub-stages. After the last up-sampling sub-stage, a 1x1 convolution layer constructs the segmentation map.

[1]Some literature also calls this the bottleneck stage, as it is the stage of the architecture with the smallest size but with the most features.

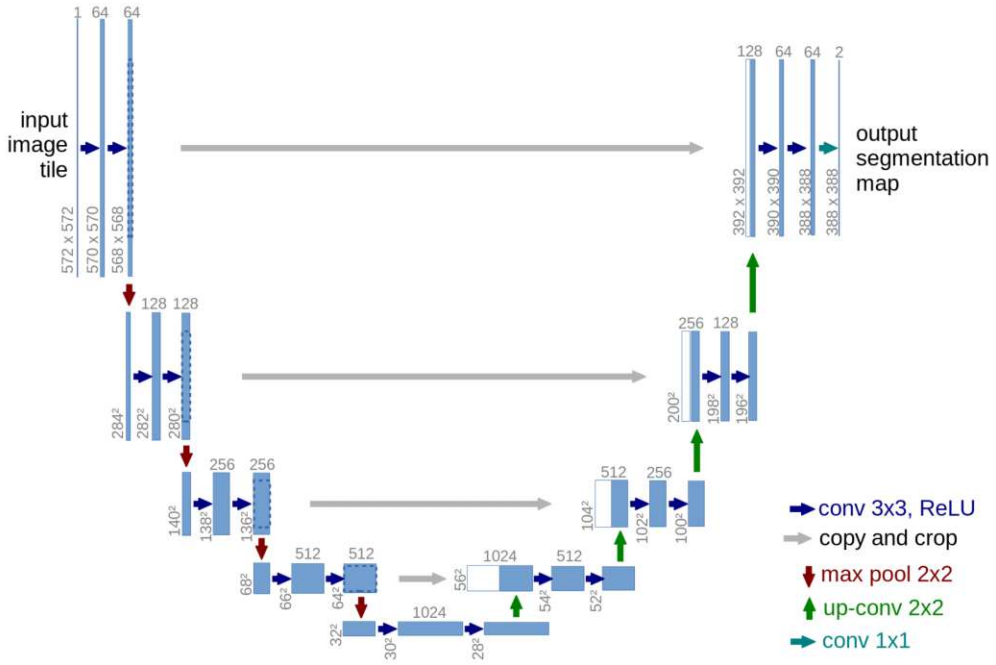


Figure 4.7: Architecture of the U-Net neural network architecture for image segmentation.[38]

For the *U-Net* architecture, the number N represents the so-called *depth* of the architecture. The architecture is, by design, symmetrical, such that the outputs and the inputs of the corresponding sub-stages of the down- and upsampling stages match. Figure 4.7 shows the standard *U-Net* architecture with a depth of 4 initially proposed by Ronneberger et al..

Table 4.3 contains the hyperparameters used for the architecture of the *Human Attention Imitation*. The network uses a depth of four and accepts images with one channel. After the first encoder block, the architecture uses 64 features, followed by increases of a factor of 2 in each downsampling stage. The bottleneck block uses 512 input and 1024 output features. The decoder blocks then decrease the feature size again by a factor of 2, and the final classifier block collapses the 64 features into one final feature.

4.2.2 Loss Function

In this section, y denotes the ground truth, and $\hat{y}$ denotes the prediction made by a network. All operations are scalar or element-wise operations, and loss functions sum pairwise over the elements of the ground truth and prediction vectors.

$$\text{CrossEntropy}(y, \hat{y}) = -y * \log(\hat{y}) \tag{4.5}$$

Depth	4
Input Dimension	128 x 256 x 1
Downsampling Stages	64
	128
	256
	512
Bottleneck	In: 512, Out: 1024
Upsampling Stages	1024
	512
	256
	128
Classifier	128, 64, 1
Output Dimension	128 x 256 x 1

Table 4.3: Hyperparameters of the *Human Attention Imitation* architecture used in this thesis. The depth is the number of down- and upsampling stages and the input and output dimensions state the size of the input and output frames, respectively. The other entries give the number of features at each point in the architecture.

Ronneberger et al. used cross-entropy loss for training the standard *U-Net* architecture in their work. Cross-entropy loss is a loss function usually used for classification problems. Equation 4.5 shows the definition of cross-entropy loss. Assuming that $y, \hat{y} \in [0, 1]$, the function will give a zero loss for all cases where the target label is 0 and a logarithmic error value for the class that that pixel represents.

This loss function is undesirable for heatmap regression, as it generally does not give zero losses for identical values. Take $y = 0.5, \hat{y} = 0.5$ as an example. The loss for these identical values is 0.346574 and, therefore, non-zero. We, therefore, need to look into other loss functions that result in zero loss values for equal values while retaining the classification ability that cross-entropy offers.

Faced with a similar problem in detecting brain tumors in MRI scans, Gros et al.[19] experimented with different standard loss functions. To detect brain tumors in MRI scans, they used *U-Net* and tried loss functions, such as cross-entropy or dice score. Due to the MRI's limited resolution, the cells at the edges of the tumors show up as a mixture of cancerous and healthy cells, which is difficult to discern in binary segmentation maps. They noticed that the sharp borders learned with those functions fail to represent those overlapping areas. Similarly, attention heatmaps should give continuous values for the relevance of each pixel in an image.

To avoid those sharp edges, they use Adaptive Wing Loss (AWL), first proposed by Wang et al.[50], to achieve smoother predicted areas. AWL is an adaptation of wing loss to improve the learning of heat maps. Feng et al.[17] proposed wing loss for landmark localization. Its gradient is smooth and continuous in the case of $|y - \hat{y}| = \omega$ but not if $y - \hat{y} = 0$, where the gradient reaches a high magnitude, making training difficult.

Equation 4.6 defines wing loss. It uses a threshold ω to differentiate between the case of pixels with significant errors and pixels with low errors. The function uses the value ϵ and the constant $C = \omega - \omega \ln(1+\omega/\epsilon)$ to assure continuity of the gradient at $|y-\hat{y}| = \omega$. Feng et al. used $\epsilon = 2$ in their experiments, as a too-low value of ϵ can lead to the issue of an exploding gradient.

$$\text{Wing}(y, \hat{y}) = \begin{cases} \omega \ln\left(1 + \left|\frac{y-\hat{y}}{\epsilon}\right|\right) & \text{if } |y-\hat{y}| < \omega \\ |y-\hat{y}| - C & \text{otherwise} \end{cases} \tag{4.6}$$

AWL mitigates the discontinuity at $y - \hat{y} = 0$ by including the target value to scale the loss value in the foreground case, making the loss function continuous and smooth for $y - \hat{y} = 0$. It also treats the case of minor errors differently, depending on whether the pixel is a foreground (high value) or background (small value) pixel. To scale the influence of foreground pixels, the case where $|y-\hat{y}| < \omega$ is augmented by an exponential term $\alpha - y$.

$$\text{AWing}(y, \hat{y}) = \begin{cases} \omega \ln\left(1 + \left|\frac{y-\hat{y}}{\epsilon}\right|^{\alpha-y}\right) & \text{if } |y-\hat{y}| < \theta \\ A|y-\hat{y}| - C & \text{otherwise} \end{cases} \tag{4.7}$$

Equation 4.7 shows the function used for AWL. The values α, ω, ϵ and θ are positive values. AWL uses θ as an additional threshold to switch between the two cases of the loss function. α is a value slightly larger than 2[2] and ω and θ are values between 0 and 1. The terms $A = \omega(1/(1+(\theta/\epsilon)(\alpha-y)))(\alpha-y)((\theta/\epsilon)(\alpha-y-1))(1/\epsilon)$ and $C = (\theta A - \omega \ln(1+(\theta/\epsilon)\alpha - y))$ assure continuity of the gradient for $y - \hat{y} = \theta$.

The exponential term of the first case makes the gradient continuous in the case of $y - \hat{y} = 0$ as the exponent is always greater than one, which means that the error converges fast to 0. The exponential term will be more significant for $y \to 0$, reducing the error even more.

As AWL is state-of-the-art for heatmap regression, this thesis uses it for training the attention filter.

[2]For details see the rationale in Wang et al. [50]

CHAPTER 5

Racing Controller

This chapter illustrates the implementation details of the racing controller ANN used in this thesis. It delves into the autonomous racing system, exploring the intricate details of the racing controller. It begins by providing an overview of the simulation environment and the foundational lane-based low-level controller, *Pure Pursuit*, the algorithm controlled by the high-level controller.

Before describing the network architecture of the high-level controller, this chapter presents the dataset generated for Offline RL training. It includes insights into its creation, state-action distribution, and formulation of the reward function that guides the learning process.

After introducing the prerequisites for the training process, this chapter characterizes the neural network model used in the high-level controller, discussing its architecture and characteristic components. This chapter also discusses the adaptations and enhancements implemented to include human-like attention in the model to reduce complexity and improve performance.

5.1 Simulation Environment & Low-Level Controller

Brunnbauer et al.[6] developed the simulation environment used in this study, which uses the *Bullet*[14] physics simulator. *Bullet* is a physics simulation engine focusing on faithful mechanical simulation and sensor emulation. The environment closely emulates the real-world behavior of *F1Tenth* cars at the expense of real-time simulation. It allows replicating scenarios involving single or multiple agents, allowing up to four cars simultaneously.

A necessary extension to the simulator is the generation of depth images because all components in this thesis require them. Leveraging the capabilities of the *Bullet* simulator to generate depth images, this addition required minimal modifications to the existing simulation environment. Nevertheless, creating depth images is computationally expensive, further degrading simulation performance.

Figure 5.1 depicts the track layout employed within the simulation environment. The arrow indicates the direction of travel. This track layout offers diverse challenges, featuring corners of varying difficulty levels in both directions. This thesis uses the track segment classification definitions used by Bhargav et al.[3].

Specifically, corners 1, 7, and 8 are conventional right sweepers, requiring precise control during high-speed turns. Corners 2 and 3 are classified as hairpins, demanding sharp and careful maneuvering. Corners 4, 5, and 6 collectively form a chicane, testing the controller's ability to navigate a sequence of quick directional changes.

An essential step to achieve comparable visual inputs between simulated data and real-world recording involved adapting the camera parameters to align with those of the original cameras. The field of view and the maximum depth are crucial parameters for this thesis. By ensuring that these parameters closely match those of the real-world

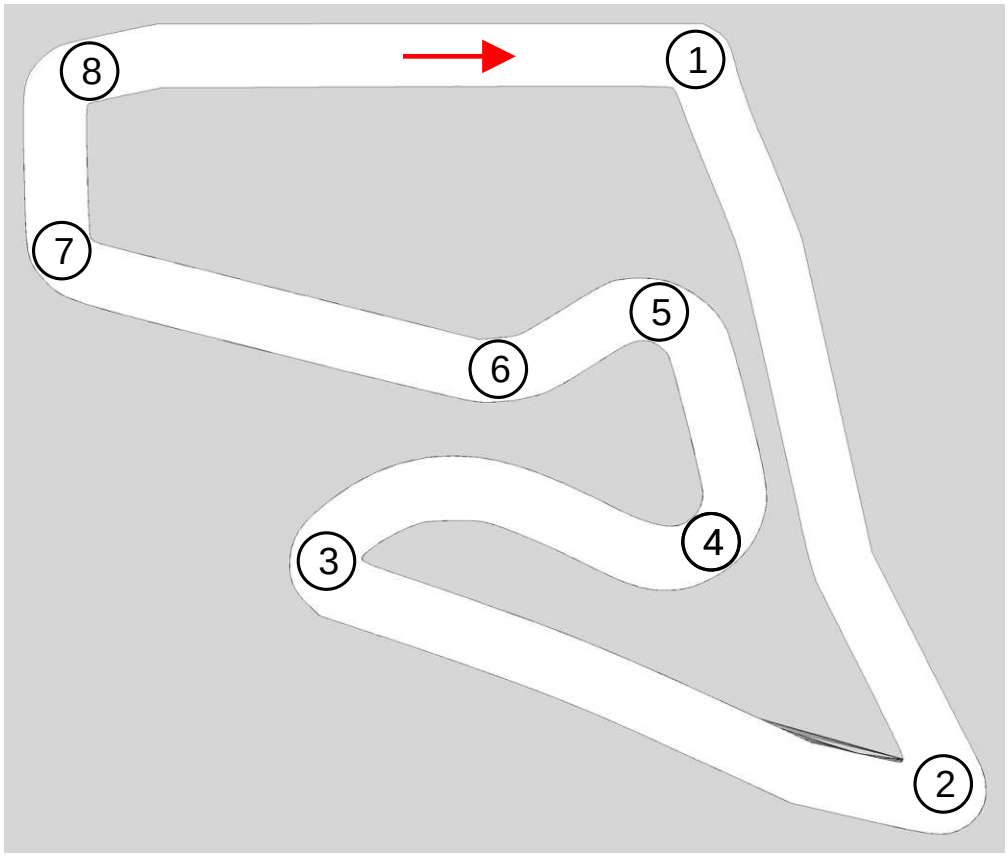


Figure 5.1: Track layout used in the simulation environment for all tasks in this thesis. The red arrow indicates the driving direction, and the numbers denote the relevant corners.

cameras, the simulation environment achieves a more faithful representation of the real world.

Figure 5.2a presents an RGB image, while Figure 5.2b showcases the corresponding depth image captured from a different virtual camera but in the same position on the car. By already providing the depth image at the size required in downstream tasks, the system optimizes computational efficiency, as the computation of depth images is comparably expensive.

Table 5.1 on the observation space of the simulation environment provides a comprehensive insight into the measurements beyond the cameras provided by the simulation environment. The simulation environment derives the measurements in the observation space from the internal simulation state by adding small amounts of noise. In addition to the already mentioned visual inputs, it provides the controller with the car's pose, velocity, and acceleration. The simulation environment employs ray marching, the same technique used to calculate the depth image, to create the lidar scan measurement.

(a) RGB image (b) Depth image

Figure 5.2: Visual inputs provided by the simulation environment. Both pictures show the same scene but with slightly different fields of view, and the depth image in each pixel shows the distance from the sensor to the first solid object. In the depth image, lighter values indicate higher distances.

Key	**Description**
`pose`	Holds the position (‘x‘, ‘y‘, ‘z‘) and the orientation (‘roll‘, ‘pitch‘, ‘yaw‘) in that order.
`velocity`	Holds the ‘x‘, ‘y‘ and ‘z‘ components of the translational and rotational velocity.
`acceleration`	Holds the ‘x‘, ‘y‘ and ‘z‘ components of the translational and rotational acceleration.
`lidar`	Lidar range scans.
`rgb_camera`	RGB image of the front camera.
`hd_camera`	High-definition RGB image of the front camera.
`depth_camera`	Depth image of the front camera.

Table 5.1: Complete observation space of the simulation environment.[6]

Key	**Description**
motor	Throttle command. If negative, the car accelerates backwards.
speed	Normalized target speed.
steering	Normalized steering angle.

Table 5.2: Action space of the simulation environment. The environment accepts either *motor* or *speed* actions and always requires a *steering* action.[6]

Table 5.2, describing the action space of the simulator, provides a comprehensive overview of the available control actions of the simulator. The table outlines the available actions to the controller, including motor control, either throttle (acceleration) or speed control, and steering inputs. Using either speed or acceleration control at a time is possible. *Normalized* for the action values signifies that each control output uses a standardized scale where the maximum value corresponds to 1.0, and the minimum value corresponds to -1.0.

The environment simulates 50 time steps per simulated second, providing a high-frequency environment. The controllers in this thesis primarily process information from a subset of sensors: pose, velocity, and the images from `depth_camera`. We use the additionally added `hd_camera` measurement for visualization tasks that use the high-resolution images from this sensor.

5.1.1 Low-level Controller

The high-level controller oversees the vehicle's navigation by assigning it to one of three lanes and dynamically adjusting the low-level controller's *Lookahead Distance* and *Speed Factor*. At the core of the low-level controller lies the *Pure Pursuit* algorithm, a lane-tracking algorithm. With the *Speed Factor*, the high-level controller can scale the reference speeds from the trajectories used by the low-level controller.

Pure Pursuit[49, 13] is a fundamental algorithm in autonomous vehicle navigation designed to achieve accurate path tracking. In this method, a vehicle selects the closest point outside the *Lookahead Distance* along the reference trajectory, known as the *Lookahead Point*, and navigates to the *Lookahead Point*. Depending on the *Lookahead Distance*, the vehicle might skip intermediate points and cut corners or try to navigate to a point that is too close and overshoot that point, leading to oscillations. This thesis uses the implementation by Zheng[54] in the simulation environment.

Despite its effectiveness in many scenarios, *Pure Pursuit* has some drawbacks. One limitation is its sensitivity to changes in the path's curvature, as it may struggle to accurately track highly curved or sharp turns, leading to overshooting or undershooting the desired path. Additionally, conventional *Pure Pursuit* uses a fixed *Lookahead Distance*, which may only be suitable for some situations, as it can result in difficulties when navigating through complex environments with changes in the path. Therefore, while *Pure Pursuit* is a valuable path-tracking technique, its limitations necessitate considering alternative control algorithms in particularly complex and dynamic environments.

Here, the high-level controller plays a critical role in utilizing the tracking capabilities of the *Pure Pursuit* algorithm while adding a layer of adaptability. This controller ensures the vehicle can navigate without over- or undershooting by dynamically controlling variables such as the *Lookahead Distance*, selected *Lane*, and *Speed Factor*. Adjusting the *Lookahead Distance* allows the system to anticipate and react to changes in the makeup of the track. Moreover, the high-level controller's control over the *Speed Factor* enables the system to adapt seamlessly to varying track layouts.

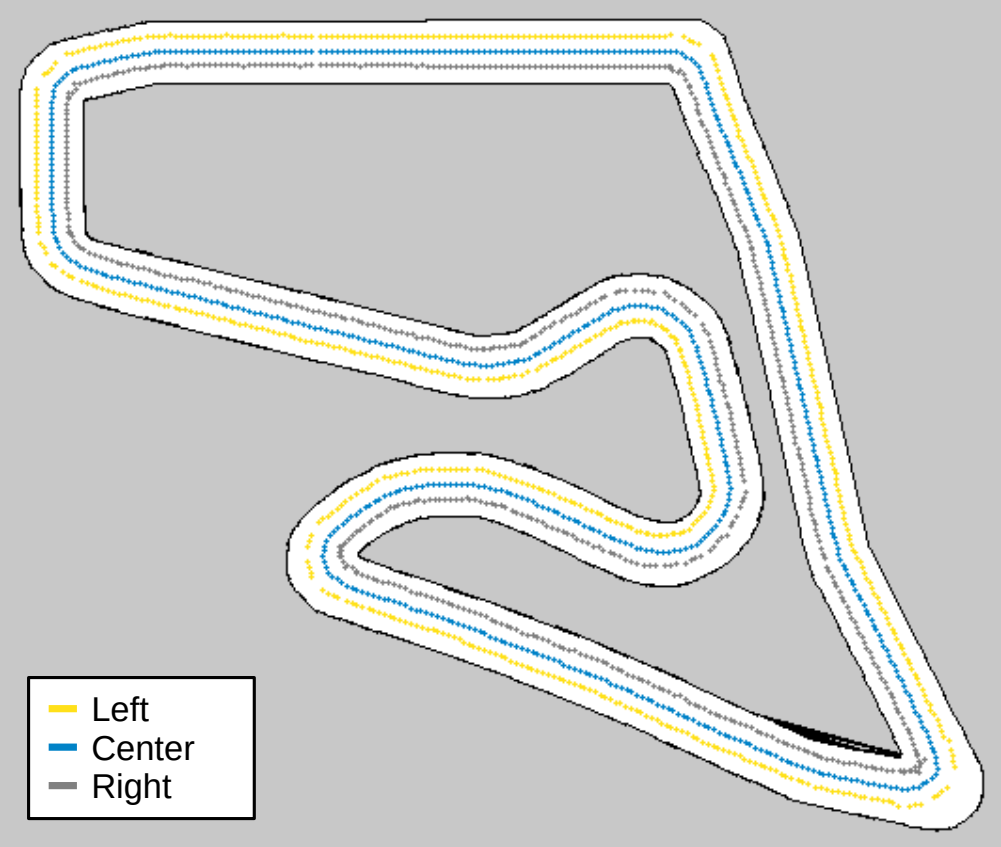


Figure 5.3: Lanes available to the high-level controller on the simulated track.

The *Pure Pursuit* algorithm operates using three lanes, as shown in Figure 5.3. It maintains a constant velocity within each lane, simplifying the control process. We apply a fixed offset in their respective directions to derive the left and right lanes from the centerline.

In steep corners, the individual waypoints of the inner lane are in the wrong order. This reordering happens when the normals from the centerline intersect at a point closer to the centerline than the offset. This reordering can lead to undesirable results for some control strategies, such as MPC. However, in the case of *Pure Pursuit*, since the points affected by the reordering are close together, the selection of the waypoints is only marginally influenced.

5.2 Dataset

Offline RL requires a set of experiences of the system to learn from. These experiences usually consist of traces recorded in the target environment, either based on an already established strategy or using a random strategy to explore the state and action space.

The distribution of states and actions is essential, as, in contrast to conventional RL, no additional exploration of the environment is possible in pure Offline RL.

This thesis investigates two possible dataset generation approaches to ensure a diverse state-action distribution. One approach is pure random generation, and the other uses an expert reference line with some noise to generate the dataset.

The *Random* generation changes the current strategy with a 5% chance. It picks the lane, *Lookahead Distance* and *Speed Factor* with uniform distributions; apart from the lanes, the distributions range from 0.0 to 2.0. This randomization leads to many crashes, especially in corners. This strategy records four cars on the track, potentially capturing evasive maneuvers and overtaking. The four cars start randomly on the track to ensure that exploration covers the whole track equally.

Expert dataset generation uses a reference trajectory optimized by the minimum curvature optimization implementation by Heilmeier et al.[21]. During driving, it uses the lane closest to the nearest point on the reference trajectory and normally distributed *Speed Factor* and *Lookahead Distance*. The distribution for the *Lookahead Distance* has a mean of 0.6, a standard deviation of 0.3, and the *Speed Factor* a mean of 0.5 and a standard deviation of 0.2. The car starts at the beginning of the lap for this generation strategy, as crashes in corners are rare.

Both strategies terminate after the car collides with a wall for a second or 50 simulation steps or if it finishes a lap. Each dataset consists of 250 traces generated by either strategy. Traces generated by the random strategy generate four different trace files for each simulation, one for each of the four cars.

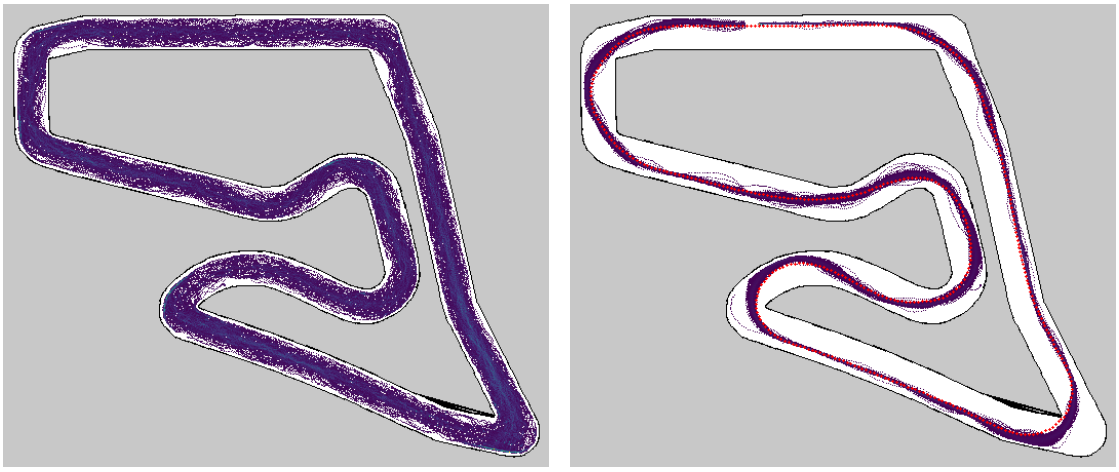
5.2.1 State-Action Distribution

In Offline RL, the effectiveness of learned policies depends on the distribution of the underlying dataset. Based on the strategies for generating the datasets, it is unsurprising that the datasets derived from random exploration and expert demonstrations exhibit divergent distributions in general. As expected, the *Lane* distribution in the *Expert* strategy favors the left lane before the others, matching the distribution of the expert trajectory.

Figure 5.4a visually illustrates that the *Random* generation strategy effectively explores a significant portion of the track area. In contrast, Figure 5.4b shows that the *Expert* dataset adheres to the reference minimum curvature trajectory.

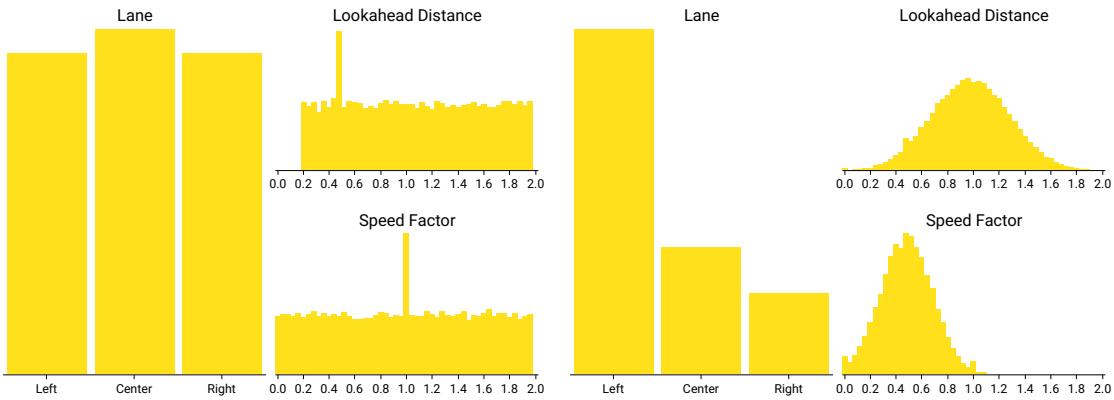
It is important to note that while the *Random* generation approach explores more states compared to the *Expert* approach, individual traces within this dataset might encounter early terminations due to collisions. Conversely, although exploring less track area, the *Expert* generation strategy shows robust exploration along the reference line.

Figure 5.5a shows the action distributions derived from the *Random* generation strategy. As expected from uniformly generated data, all values show almost uniform distributions,



(a) Traces of *Random* dataset. (b) Traces of *Expert* dataset with expert line.

Figure 5.4: All traces of both generation strategies plotted on the track used in the simulation. The red line in the *Expert* dataset indicates the reference trajectory.



(a) Distribution of *Random* dataset. (b) Distribution of *Expert* dataset.

Figure 5.5: Action distributions of both generated datasets. The selected *lanes* are discrete values, while *Lookahead Distance* and *Speed Factor* are continuous values, only shown in discrete buckets for visualization purposes.

except for singular peaks, which represent the starting configurations in the randomized generation strategies.

However, a stark contrast emerges in the *Expert* dataset, as shown in Figure 5.5b. The *Expert* dataset demonstrates a different *Lane* selection behavior, with approximately two-thirds of its actions using the left lane, closely aligning with the reference line. The other control values exhibit normalized distributions, again with slight peaks in the initial states of the simulation.

	Random	**Expert**
Traces	250	250
Total Steps	169,513	189,736
Speed Factor Distribution	$\mu = 1.00, \sigma = 0.57$	$\mu = 0.50, \sigma = 0.20$
Lookahead Dist. Distribution	$\mu = 1.09, \sigma = 0.52$	$\mu = 1.00, \sigma = 0.30$
Strategy Distribution	L = 32.5%	L = 62.6%
	C = 35.0%	C = 22.8%
	R = 32.5%	R = 14.6%
Best Lap Time	N/A	1:33.62 min

Table 5.3: Distributions and statistics for both generated datasets.

Table 5.3 provides a comprehensive summary of statistics of the two distinct datasets, *Random* and *Expert*, each containing 250 traces. These traces contain many recorded steps, with the *Random* dataset totaling 169,513 steps and the *Expert* dataset 189,736. Notably, the two datasets exhibit different distributions in all parameters.

The *Random* dataset displays a higher mean *Speed Factor* of 1.00 with a standard deviation of 0.57. The *Expert* dataset has a lower mean *Speed Factor* of 0.50 with a standard deviation of 0.20. Regarding *Lookahead Distance*, the *Random* dataset has a mean of 1.09 with a standard deviation of 0.52, while the *Expert* dataset exhibits a mean of 1.00 with a standard deviation of 0.30.

The distribution of driving strategies in both datasets differs significantly, with *Expert* traces predominantly following the left lane (62.6%) compared to the more evenly distributed strategies in the *Random* dataset (32.5% left, 35.0% center, and 32.5% right).

The *Random* dataset initiates its traces from random starting positions to cover the entire length of the track, as the randomized driving commands lead to frequent collisions. These frequent collisions result in the absence of completed laps and the inability to establish a comparable best lap time, as no full trace exists in the *Random* dataset from the starting point used in the *Expert* dataset to the finish line. Conversely, the *Expert* dataset comprises mostly full traces, enabling the determination of a trace with the best complete lap time of 1:33.62 minutes.

After initial tests with both datasets and considering that the original implementation of the racing controller ANN performed better with an *Expert* dataset, this thesis continued work with the *Expert* dataset and discarded the *Random* dataset for further tasks. Training on the *Random* dataset led to policies that either produced a trivial solution (fixed lane and low-level controller parameters) or policies with high variance that did not converge.

5.2.2 Reward Function

Reinforcement Learning (RL) iteratively updates a policy model, possibly represented by a neural network, using a *reward* function. This function returns a numerical value r_t

after the agent takes an action in the environment, which signals how good the agent's action was. The cumulative, possibly discounted sum of all rewards over one series of interactions with length T in an environment for time t is called the *return* R_t.

$$R_t = \sum_{t'=t}^{T} \gamma^{t'} r_{t'} \tag{5.1}$$

With $\gamma = 1.0$ the cumulative sum is undiscounted, and the return function is the sum over all future rewards. In this case, the function is called *return to go* $\hat{R}_t$. Unlike the *reward*, which only represents local results, the *return* contains all possible penalties and rewards, making it a valuable tool for environments with sparse rewards and long sequences.

$$\hat{R}_t = \sum_{t'=t}^{T} r_{t'} \tag{5.2}$$

To design a *reward* function for an autonomous agent, achieving a balance that incentivizes the desired behavior while penalizing undesirable actions is essential. In autonomous racing, finishing a lap without colliding with a wall constitutes the desired behavior; thus, the *reward* function should assign positive rewards for this achievement.

Conversely, crashing into a wall, representing undesirable behavior, should result in significant penalties to discourage such actions. However, each time step without collision should also incur a slight penalty to encourage efficient and speedy performance. This subtle penalty for inactivity helps motivate the agent to complete laps quickly. By carefully designing the *reward* function, we can guide the agent to learn the optimal racing behavior that balances speed, accuracy, and safety.

Contrary to common practice, we set the *reward-to-go* to a fixed value when the vehicle collides with a wall or another vehicle. Without this adaptation, the learning process would rarely achieve a sensible policy. As every trace ends once the vehicle crosses the finish line, the *reward-to-go* also effectively assumes a constant value for states where the vehicle finishes a lap. It is possible to define the *reward-to-go* function used in this thesis as a recursive function instead of an undiscounted sum of rewards.

$$\hat{R}_t = \begin{cases} 1.000 & \text{if finished} \\ -5.000 & \text{on collision} \\ \hat{R}_{t+1} - 1 & \text{otherwise} \end{cases} \tag{5.3}$$

Equation 5.3 shows the recursive definition of the *reward-to-go* function for each time step. When the controller has finished a lap or crashed, the *reward* in the terminal conditions is a constant value. Since trace generation does not necessarily stop when the agent collides with the wall, leaving in traces that contain *touching*, short contact with the wall,

which the agent can recover from, allows the controller to learn recovery while including undesirable outcomes and the events leading up to those events. As mentioned earlier, each step that does not end in a terminal condition is penalized with a low penalty to incentivize the learning of a faster controller.

This *reward-to-go* function is comparably simple and derives the *reward* values from the traces in the dataset and the planned control frequency of the high-level controller. The fastest-finishing traces take approximately $5,000$ time steps at a control frequency of $10\ Hz$ and a simulation frequency of $50\ Hz$. It, therefore, takes the agent approximately 1000 control steps to finish a lap. Given that each control step without collision or finishing a lap has a penalty of -1, a fast lap would result in an episode return of 0. The collision penalty is an arbitrary negative value, lower than minus the maximum achievable reward, so no amount of reward can nullify the penalty of a collision.

5.3 Decision Transformer

Decision Transformers (DTs), first introduced by Chen et al.[10], represent an exciting advancement in artificial intelligence and RL, offering a novel approach to complex control problems. This concept leverages the power of GPTs by encoding various system modalities and feeding them to the transformer to predict control actions. These modalities encapsulate various aspects of the system and consist of *reward-to-go*, *state*, and *action* tuples. By doing so, DTs aim to synthesize the actions to achieve a specific target return.

This innovation draws from the transformer's ability to process sequential data efficiently. Each modality is assigned a single token, allowing DTs to simultaneously handle and fuse multiple information streams. This fusion enables the model to maintain a holistic understanding of the system's state, its ongoing pursuit of rewards, and the actions it has taken in the past. It is worth noting that having access to multiple time steps is crucial for making informed decisions, particularly in environments with long time horizons and intricate inter-modality dependencies.

DTs operate over multiple time steps, including temporal and inter-modal dependencies in the decision-making process. This consideration of history enables them to forecast subsequent actions while striving to maximize the expected *return*. They have the potential for applications across various domains, from robotics and autonomous systems to game-playing and autonomous driving.

The training process for DTs resembles a supervised learning paradigm more than traditional RL. This disparity stems from the fact that, while the training conditions the network on the *reward* signal, it employs a loss function to guide the training process, optimizing the predictions made by the network. By employing ample training data, the idea is that the generative capabilities in GPT-based architectures will enable DTs to extrapolate behavior for previously unseen situations.

The DT architecture encapsulates each trace as a series of *reward-to-go*, *state*, *action* tuples, forming a structured representation. These trace tuples are subsequently passed as tokens to the transformer, serving as the input for the transformer's processing. This representation enables the DT to make informed decisions based on the contextual information contained within the sequence of tuples.

$$\tau = \left(\hat{R}_1, s_1, a_1, \hat{R}_2, s_2, a_2, ..., \hat{R}_T, s_T, a_T\right) \tag{5.4}$$

The DT architecture uses a flat representation of the trace to encode the modalities for processing with the underlying transformer. Equation 5.4 defines the representation for a trace τ with length T.

Figure 5.6 illustrates the general architecture of DTs. The trace undergoes embedding, with each modality receiving its dedicated and temporal embedding. The underlying causal transformer then processes the input tokens and produces a series of output tokens. This architecture only uses the action tokens of the generated output tokens after post-processing them with a linear layer.

5.3.1 Hyperparameters

Along with traditional hyperparameters, such as the size and depth of linear or convolutional network components, transformers and the DT architecture introduce two new relevant hyperparameters. The first is the *Context Length*, which defines how long the sequence of input tokens is, and the second is the *Embedding Size*, which defines the size of the internal token representation.

In the context of GPT and similar transformer-based models, *Context Length* refers to the number of previous tokens or words the model considers when generating predictions for the next token in a sequence. It represents the extent of the model's memory or the historical context it uses to understand and generate coherent and contextually relevant output. As the DT architecture uses 3-tuples to represent each time step, the context length is three times the number of time steps the DT considers.

Embedding Size, in turn, refers to the dimensionality or size of the embeddings used to represent words, tokens, or subword units in the model's vocabulary. The transformer maps each token in the input sequence to a dense vector in a high-dimensional space, and the *Embedding Size* determines the length or dimension of these vectors.

5.3.2 Adaptions for Autonomous Racing

In order to apply the DT architecture to autonomous racing, several modifications are necessary to meet the demands of this domain. In this thesis, a significant difference from the original architecture lies in the action space. While the original paper uses only continuous or discrete actions, our action space consists of one discrete and two continuous actions. Those actions are the parameters for the low-level *Pure Pursuit*

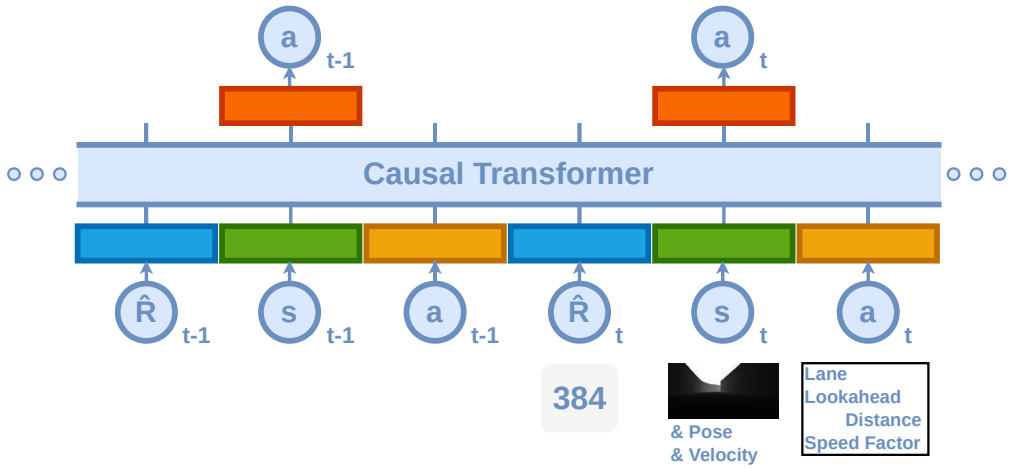


Figure 5.6: Architecture of DT used in this thesis.

controller. Moreover, the state space consists of a depth image and data about the vehicle's pose and velocity.

Figure 5.6 visualizes the adapted structure of the DT for autonomous racing. As already mentioned, certain key elements remain consistent with the original framework. The *reward-to-go*, denoted as $\hat{R}$, remains unchanged. Changes exist in the state representation, which now encompasses a depth image and essential data on the vehicle's pose and velocity. Furthermore, the action space, used as input and output, now contains a selected *Lane* (left, center, or right), the *Lookahead Distance*, and *Speed Factor*.

These significant changes in the autonomous racing adaptation also necessitate rethinking how to encode information for the DT architecture. Figure 5.7 shows how the implementation transforms the values obtained from the simulation into tokens to process with the GPT backend. Since the DT is an autoregressive transformer, we feed the generated tokens back into the transformer for inference.

Just as in the original DT architecture, linear layers transform the *return-to-go* value into a vector as wide as the *Embedding Size* of the DT. The mixed action space, on the other hand, needs more differentiated handling. A linear layer processes the continuous actions, *Lookahead Distance* and *Speed Factor*. The discrete action signals undergo embedding of the numerical *Lane* value and then Hyperbolic Tangent (TanH) activation. We employ a three-layer CNN with Rectified Linear Unit (ReLU) activation for depth images, followed by a linear layer and TanH activation.

The encoding network adds timestamp embeddings to the embedded values to obtain the input tokens. We also add the differently embedded modalities together for the actions and observations. This way, we stick with the three-token-per-time-step layout while including more than three modalities.

The DT produces action tokens, which subsequently serve as inputs for two separate

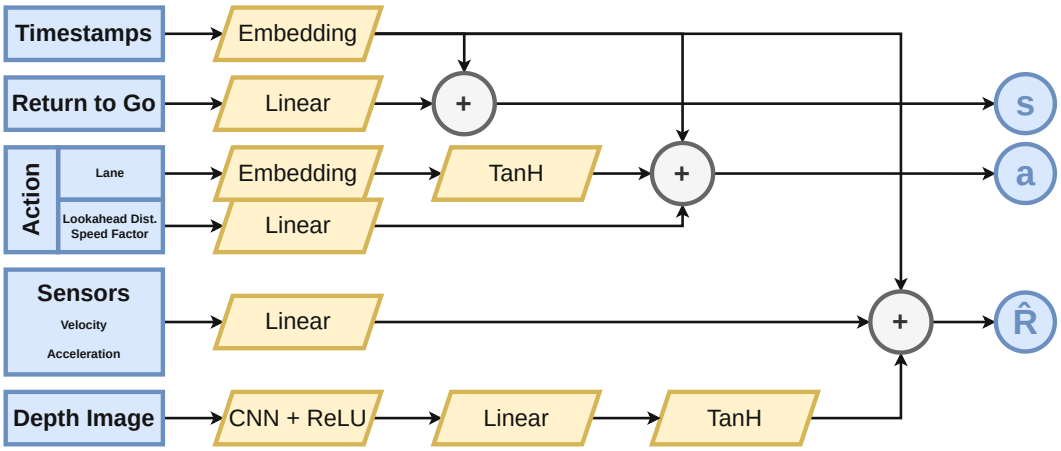


Figure 5.7: Embedding strategies for the individual modalities passed into the DT. The left shows the different values obtained from the simulator, and the middle part visualizes the different layers to process the input values and obtain the tokens for the underlying GPT network.

networks responsible for predicting discrete and continuous actions. The ReLU activation function transforms the outputs of both sub-networks to the individual action predictions. Other tokens generated by the transformer, which would correspond to predicted returns-to-go and state predictions, remain unused in this implementation.

In contrast to the original implementation, the approach for autonomous racing diverges from absolute time embeddings, opting for a strategy where time embeddings are purely relative, assigning time 0 the first time step the DT receives and time context_length/3−1 to the most recent step. Doing so encourages the model to focus on temporal relationships between modalities rather than the embedded time values. The time values for all three tokens of a time step are the same.

5.4 Decisions Transformer with Human Attention

In order to reduce the size of the DT while maintaining or even improving network performance, this thesis uses smaller images as input to the network. Instead of scaling the images to a smaller size, the network operates on small parts of the image chosen by a selection policy. The selection policy must either provide all the essential information to the network or the DT must be able to perform comparable inferences with reduced data, possibly deducing missing information from the whole provided time frame.

This thesis uses a 4-by-2-grid of the depth image generated by the simulation environment as a basis for the selection strategy. Selection strategies provide grid coordinates in each frame as the partial image for usage in the DT. Figure 5.8 shows the grid on top of a depth image.

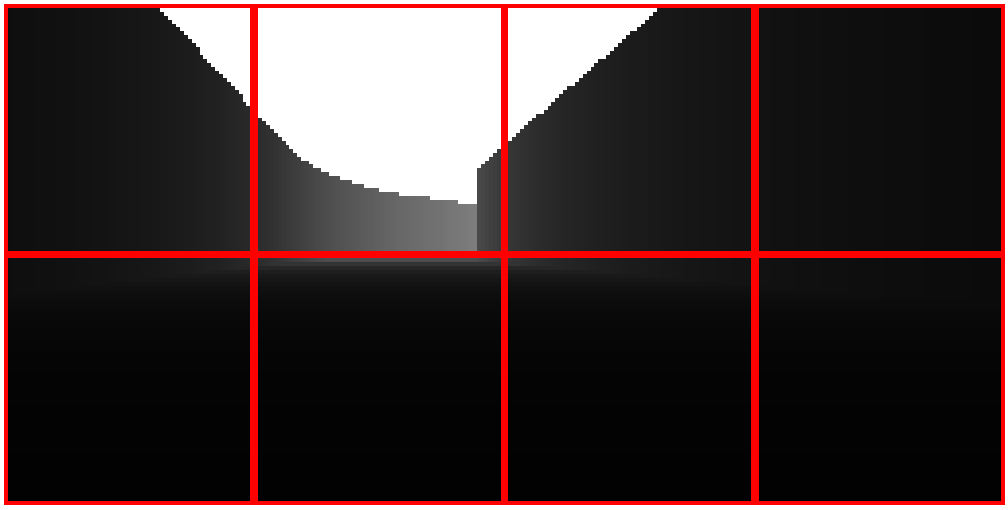
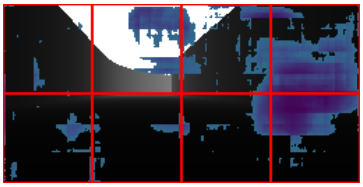
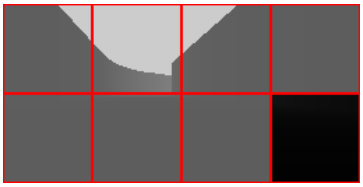


Figure 5.8: Selection grid placed over a depth image from the simulator.

(a) Human attention prediction on depth image.



(b) Partial grid over human attention prediction.



(c) Selected area of the depth image, selected by the policy.

Figure 5.9: Illustration of the *Human Attention Imitation*-based selection policy for partial images.

The reduction in the size of the input image has two effects on the complexity of the network. Firstly, we can reduce the complexity of the CNNs for image embedding, a reduction that applies tenfold, once for each considered time step. We can also use a smaller *Embedding Size* for the tokens, as the partial images contain less information than the full image.

One of the selection policies we use a selection policy guided by the *Human Attention Imitation*, introduced in Chapter 4. With this policy, we incorporate human attention into the network. The attention-based selection policy identifies the grid element within the input image with the highest cumulative attention score. Even though predicting the attention map for each input image introduces additional overhead in the inference process, this thesis only compares the complexity and size of the task-specific network, not the whole stack's inference time, to investigate the influence of human attention in ANNs.

Figure 5.9 shows how the *Human Attention Imitation*-based selection policy determines the partial image for inference. In subfigure 5.9a, the top 25% of the attention heatmap

is displayed over the depth image. Subfigure 5.9b provides a visual depiction of the selection grid on top of the attention scores. Subfigure 5.9c highlights the selected grid element based on the attention-driven selection policy. It is worth noting that, owing to the fixed nature of the grid, the selected grid elements often fail to contain all ROIs within the input data, which is a limitation of the selection process.

To verify that the *Human Attention Imitation*-based selection policy benefits the ANN architecture and not simply profits from the correlative properties of *self-attention* in the GPT backend, we use a random selection policy as a baseline. We use the same *Embedding Size* and similar hyperparameters to train the network with both selection policies for good comparison results. Only the approach without a selection policy, which uses the entire depth image, uses a different *Embedding Size*.

This thesis refers to the *Human Attention Imitation*-based selection policy as *attention (att)* and the random policy as *random (rand)*. The network using the full images is referred to as *none*, as the network operates without any selection policy.

CHAPTER 6

Training & Evaluation

This chapter delves into the critical aspects of training regimes and evaluation strategies for the networks presented in the preceding chapters. The exploration encompasses the *Human Attention Imitation*, trained using supervised learning, and the DT architectures, which use Offline RL. In the case of supervised networks, training and evaluation phases were performed in lock-step to gauge the training progress during the training process. The Offline RL DT networks operate in a fundamentally different but technically similar paradigm. It uses methods commonly used in supervised learning but evaluates network performance differently than supervised learning by running simulations with the current model to determine its performance.

To evaluate the performance of the high-level control networks, we undertake a quantitative and qualitative analysis, comparing the behaviors of models trained using the two different partial image selection policies and investigating the impact of these policies on network performance and the stability of their outputs. This chapter aims to provide valuable insights into the influence of the different partial image selection strategies on these networks' training progress and evaluation performance, contributing to a deeper understanding of their capabilities and limitations.

6.1 Human Attention Imitation

The *Human Attention Imitation* developed in this thesis uses a supervised learning approach to optimize the network. In a supervised approach, following each training epoch, an evaluation epoch evaluates the network's performance by testing the current model against a subset of the dataset distinct from the data used for training. This step plays a crucial role in gauging the network's progress, helping to ascertain the point at which it operates effectively without falling prey to overfitting. Overfitting, a common issue, is often indicated by the training loss decreasing while the evaluation loss increases. This continual assessment ensures that the network maintains a robust and generalizable performance.

As the network uses real-world data to train but finds application in simulation environments, this section also contains an informal argument that the *Human Attention Imitation* can perform adequately in simulation. We have outlined the reasons for not recording in simulation in Chapter 4. However, the absence of labels in simulation makes providing a more formal or analytical argument challenging.

6.1.1 Training

Table 6.1 is a reference for the training parameters and hyperparameters utilized during the development of the model. The training process spans 30 epochs, with each epoch processing data in batches of 16. We use dataset partitioning to ensure robust model performance by evaluating the network on data previously unseen, with 80% of the data allocated for training and the remaining 20% for evaluation purposes. The model optimization process uses the AWL function, detailed in Section 4.2.2.

Learning Rate	$1 \cdot 10^{-6}$
Epochs	30
Batch Size	16
Training Partition	80% (26,667 frames)
Loss Function	*Adaptive Wing Loss*

Table 6.1: Training parameters for the *Human Attention Imitation* network.

The training process for our network uses a dataset containing a total of 33,334 usable frames, resulting in a training partition with 26,667 frames. We use an AMD Ryzen 1800X machine with an NVidia GTX1080 graphics card and 64 GB of memory for training. To enhance the efficiency of both the training and evaluation phases, we used *CUDA*[15] hardware acceleration, leveraging the power of GPU parallelism. Our network implementation uses the *PyTorch*[37] framework and builds upon Nikhil Tomar's *U-Net* implementation[40].

Figure 6.1 visualizes the progression of the training loss during training. Each epoch's performance is represented by error bars, illustrating the maximum, minimum, and mean loss values achieved during that training phase. A notable observation is that, after approximately ten epochs, the training loss exhibits only marginal decreases. This behavior is a significant indicator that the network has achieved convergence.

6.1.2 Evaluation

Following each training epoch, we evaluate the network against a separate evaluation dataset, maintaining consistency with the training batch size to achieve comparable loss values. The evaluation process centers around monitoring the evaluation loss, a metric for gauging the selection of the loss function and the network's performance. In addition to the numeric metric of the loss value, visual comparisons offer valuable qualitative insights to verify that the loss function matches the problem and that the model predicts desirable results.

Figure 6.1 represents the relationship between the evaluation loss and training loss throughout the network's training process. As a general guideline, when the training loss steadily decreases while the evaluation loss rises, checking if the network starts to overfit becomes necessary. After approximately 10 epochs, the evaluation loss closely aligns with the training loss, indicating a good balance between the network's capacity to fit the training data and its generalization abilities.

In the subsequent epochs, the evaluation loss remains relatively stable, with only slight fluctuations. However, a small spike in the evaluation loss is evident in the final epoch, suggesting a potential early indication of overfitting. To verify that the network is not overfitting, we visually compared labels from the evaluation dataset with the predictions from the network.

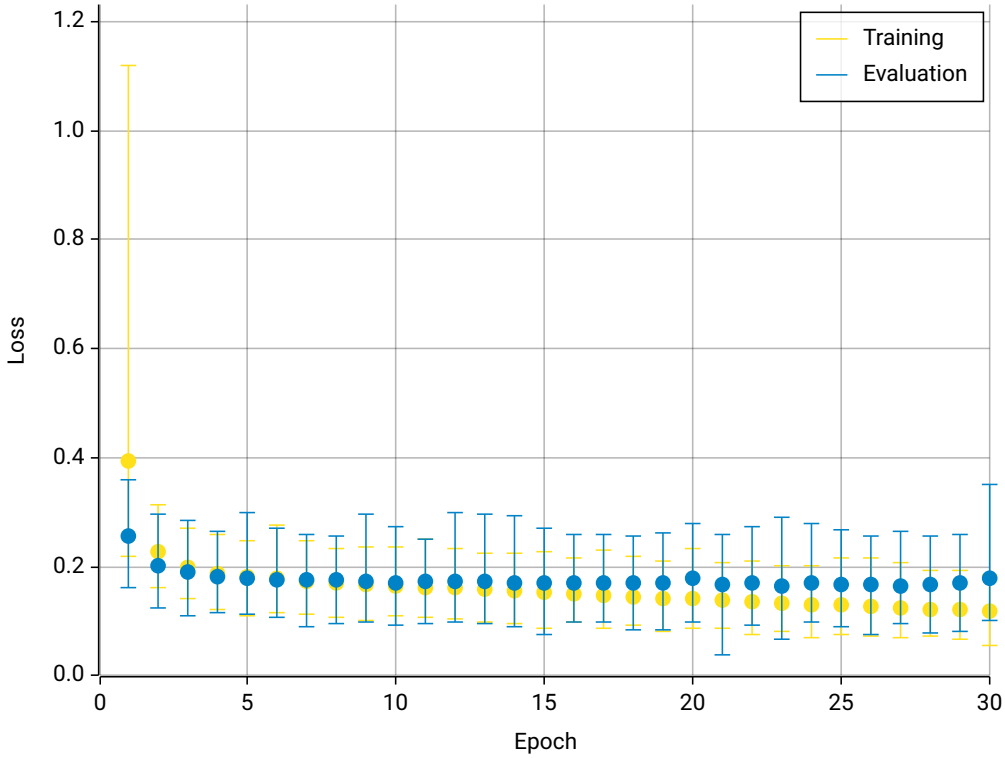


Figure 6.1: Training and evaluation losses of the *Human Attention Imitation* network. The elements in the figure mark the maximum (top bar), minimum (bottom bar), and mean (center circle) loss values for each epoch. The training (yellow) and evaluation (blue) loss values show early convergence of the network after about ten epochs.

In Figure 6.2, we present four representative samples from the evaluation dataset. The top row of the figure shows the RGB images; even though the network operates on the corresponding depth images, the RGB images are more accessible to interpret than their depth counterparts. The second row consists of the aggregated attention maps used as labels during the training process. Lastly, the third row shows the network's predictions, offering a direct visual comparison between the predicted attention maps and the actual labels.

We additionally conduct visual comparisons of dataset labels and network predictions. These comparisons focus on whether the peaks observed in the labels are visible in the predictions and whether any additional peaks are close to those labeled. Lastly, we check that the predictions accurately indicate the most pronounced areas in the ground-truth data. For instance, samples **1** and **2** in the evaluation exemplify well-matching instances where the predictions closely align with the labeled peaks. Sample **3** provides an example where not all labeled peaks appear in the prediction as pronounced as in the labels. In the case of sample **4**, the prediction demonstrates a notable shortfall, rendering it unusable due to its inability to effectively capture ROIs in the input.

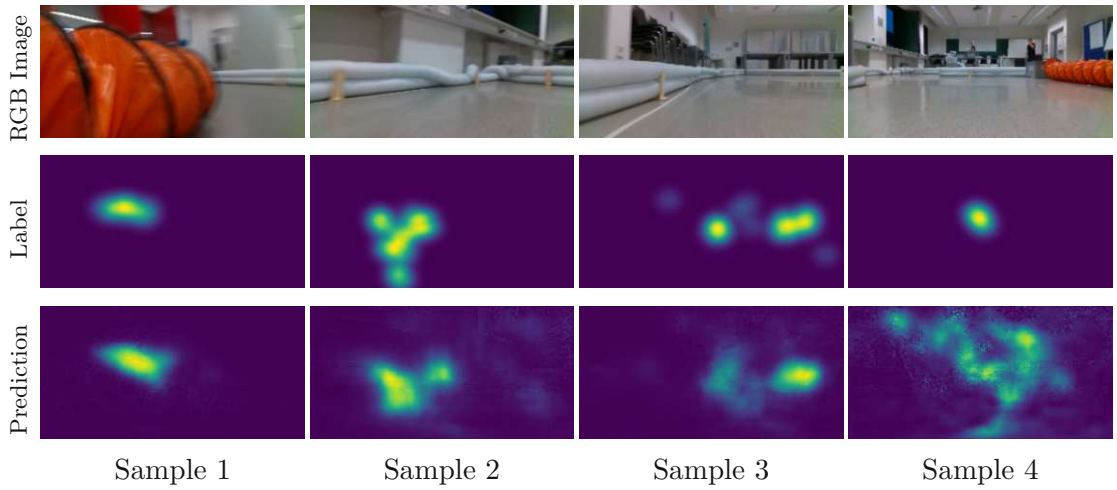


Figure 6.2: Predictions by the *Human Attention Imitation*. For each sample, we show the RGB image (top) corresponding to the input depth image, the ground truth heatmap (middle) from the dataset, and the prediction(bottom). While samples 1, 2 and 3 represent good predictions, sample 4 shows a failure case.

We perform a systematic visual categorization for 50 randomly sampled elements drawn from the evaluation dataset. We consider 27 samples as *Matching*, denoting instances where the model's predictions meet all evaluation criteria, exemplified by the well-matching samples **1** and **2**. Fifteen samples are considered part of the *Not Exactly* category, representing cases where the model's predictions slightly differ from the desired outcomes, as seen in sample **3**. Additionally, six samples provide unusable results, which result in them ending up in the *Unusable* category, such as in sample **4**. Notably, two samples are labeled as *Not Applicable* as they lacked ground truth labels for evaluation. We must highlight that we conservatively employ classification, resulting in 42/50 (87.5% of applicable samples) samples providing acceptable results.

The evaluation of the *Human Attention Imitation* network has affirmed its satisfactory performance. The convergence of loss values between the training and evaluation sets indicates that the network has reached a stable point, and visual verification has shown that it can generate reliable predictions. Visual comparisons underscore the suitability of the selected loss function for effectively addressing the problem at hand, further validating the network's effectiveness in imitating human attention.

6.1.3 Real-to-Sim Gap

The final challenge for the *Human Attention Imitation* lies in bridging the gap between real-world data used for training and the simulated environment used by the racing controller. It is crucial to ensure that the prediction capabilities of the imitation network can translate to the simulated environment.

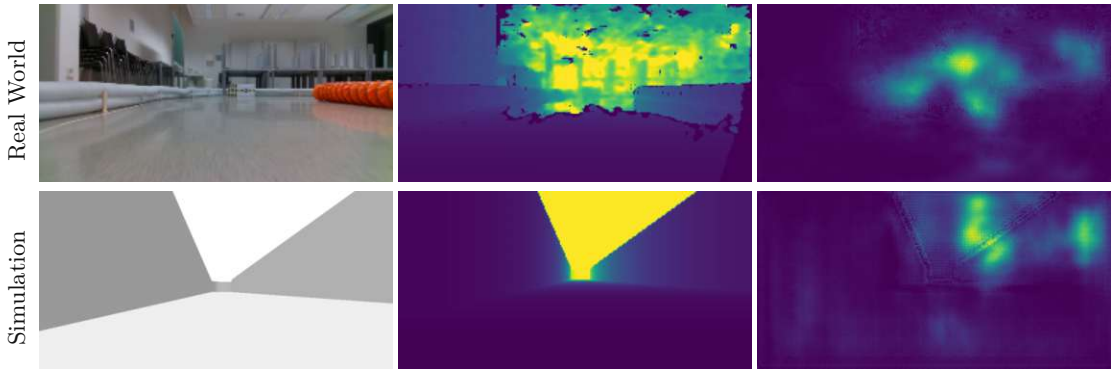


Figure 6.3: Similarity between real-world (top) and simulated (bottom) driving scenarios, associated sensor data, and predicted attention heatmaps. Despite the differences in inputs, both predictions focus on the inner track boundary.

To verify this, We visually compare predictions of similar situations in simulated and real-world data to provide an informal argument that the attention filter also works with simulated data. Most of the predictions used for the comparison are close to attention heatmaps in recorded data for similar situations. Even with noise added to the sensor data by the simulation environment, the simulated data is more precise and has fewer measurement artifacts. There are also some slim deviations in the setup of the tracks, especially the height of the track boundaries.

Figure 6.3 presents analogous driving scenarios in real-world and simulated data, highlighting the network's ability to generalize to simulation. The situations used for the comparison differ in some key aspects, but the *Human Attention Imitation* provides comparable predictions. In the real-world example, the vehicle is closer to the corner when compared to the simulated frame. The figure showcases three key elements: the RGB image, the depth image, and the prediction generated by the *Human Attention Imitation*.

While depth images for real-world and simulated data have a broader field of view than the RGB image, the predictions mostly align. Both predictions highlight areas at the top of or slightly extend over the inner boundary. This consistent behavior observed across many randomly picked scenarios verifies the ability of the attention imitation model to transfer well to simulation.

6.2 Overtaking Controller

In contrast to the *Human Attention Imitation*, the racing controller adopts a RL framework for training. RL traditionally interacts with the environment to record traces with reward scores and optimizes the network to achieve the highest reward scores. This method faces

data collection and evaluation challenges with possibly unsafe policies for some physical systems, mainly if used at their physical limits.

This thesis employs two key measures to work around these challenges. First, simulation for RL training removes the need to record real-world data, provided that the simulation accurately emulates real-world dynamics and scenarios. Second, we employ Offline RL, allowing data to be generated once and subsequently loaded, which significantly eases the computational burden, especially when employing resource-intensive simulations.

Chen et al.[10] employ a novel approach to perform Offline RL to condition the GPT-based DT on the *return-to-go* parameter. In this approach, training relies on a loss function to guide the learning process instead of traditional RL methods that use rewards or episode returns for optimization. This approach is closer to supervised learning than RL.

Despite the deviations from traditional RL in the training process, the evaluation phase relies on conventional RL techniques. Following each training epoch, a simulation environment evaluates the network's performance using the network as the control policy. The system records key performance indicators during those evaluation runs to determine the performance of the current policy.

Possible metrics in autonomous racing include the progress made on the track before any potential crashes occur and the lap times achieved. Additionally, the variance in the control outputs generated by the trained policy is an essential comparative measure, ensuring that the network's decision-making remains consistent and reliable across diverse scenarios.

This section describes the training regime for the racing controller implementation and the deviations for the two implementations based on partially selected sub-images. It also describes the different combinations of training and hyperparameters used to get the complete image-based DT to work. After details about training, the section gives an overview of the evaluation process and the results obtained from the evaluation.

6.2.1 Training

Unlike traditional Offline RL methodologies, the DT adopts a supervised learning approach. The primary objective of this training process is to condition the network to predict actions that will yield a specific *return-to-go* outcome in a given state. Supervised learning approaches require a loss function to guide the optimization process. This loss function requires careful selection to facilitate effective training.

The state space of the DT in this thesis incorporates two distinct action types: discrete *Lane* selection and continuous parameters for low-level controller adjustments. We employ a hybrid loss function to train the model, utilizing Cross-Entropy (CE) loss for the discrete actions and Mean Squared Error (MSE) loss for the continuous actions. Equation 6.1 formally defines this hybrid loss function. In this equation, **a** denotes the ground truth

Training Parameters	
Learning Rate	$5 \cdot 10^{-9}$
Weight Decay	$6 \cdot 10^{-6}$
Epochs	15
Steps per Epoch	$1,000$
Warmup Steps	$2,000$
Batch Size	32
Hyperparameters	
Context Length	30
Embedding Size	2048
Execution Frequency	$10\ Hz$
Partial Image Size	N/A
Image Embedding Layers	$256x128x1 \rightarrow 31x63x32 \rightarrow$ $14x30x64 \rightarrow 12x28x64$
Trainable Parameters	195.297.452

Table 6.2: Training parameters and hyperparameters for the full-image DT network.

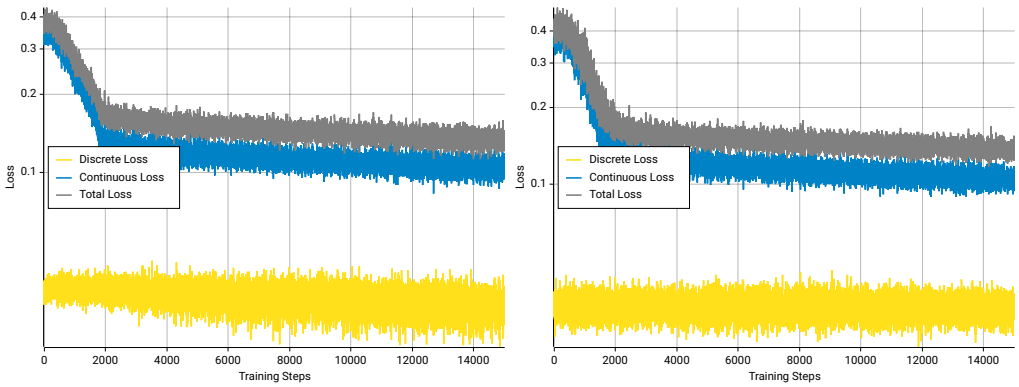
actions and $\hat{\mathbf{a}}$ the predicted actions, $\mathbf{a}_c$ and $\hat{\mathbf{a}}_c$ denote the continuous actions, $\mathbf{a}_d$ and $\hat{\mathbf{a}}_d$ the discrete actions.

$$L(\mathbf{a}, \hat{\mathbf{a}}) := CrossEntropy(\mathbf{a}_d, \hat{\mathbf{a}}_d) + MSE(\mathbf{a}_c, \hat{\mathbf{a}}_c) \tag{6.1}$$

Typical use cases for CE loss include classification problems in which it calculates the loss between a one-hot encoded prediction with a reference class index. This thesis uses the PyTorch implementation of the cross-entropy loss function. In addition to working well with the rest of the framework, the *PyTorch* implementation offers the option to assign different weights to each class, which helps when dealing with unbalanced datasets.

The assigned weights help prevent the model from overly fixating on a single class, ensuring a more balanced and accurate learning process if the class distribution in the dataset is unbalanced. We chose the class weights as the inverse of the class distribution within the dataset. The weights assigned to the left, center, and right lane classes are $\frac{1.0}{62.6}$, $\frac{1.0}{22.8}$, and $\frac{1.0}{14.6}$, respectively.

For the CE loss function, classes refer to the different categories or labels the model predicts. This loss function offers the flexibility to assign different weights to each class, which proves particularly valuable when dealing with unbalanced datasets. The assigned weights help prevent the model from overly fixating on a single class, ensuring a more balanced and accurate learning process. In this specific implementation, we chose the class weights as the inverse of the class distribution within the dataset. For instance, the weights assigned to the left, center, and right lane classes are $\frac{1.0}{62.6}$, $\frac{1.0}{22.8}$, and $\frac{1.0}{14.6}$, respectively.



(a) Trace that managed 70.86% of a lap. (b) Trace that managed 65.51% and had a lower projected lap time.

Figure 6.4: Training losses for the full-image DT models. The total loss (gray) is the sum of the loss values for the continuous (blue) and discrete actions (yellow).

Table 6.2 presents an overview of the training parameters used for the full-image DT model. It shows various hyperparameters for training and fine-tuning the model. These parameters include the learning rate, weight decay, number of epochs, warmup steps, context length, batch size, and *Embedding Size*. The learning rate is $5 \cdot 10^{-9}$, indicating the step size for optimization during training. Weight decay, which controls regularization, is $6 \cdot 10^{-6}$.

The model undergoes training for 15 epochs, with a warmup phase spanning 2,000 steps. The context length is 30, determining the historical data considered in each training sub-trace, while a batch size of 32 defines the number of sub-traces processed in each training iteration. Lastly, the *Embedding Size* is 2048, which impacts the size of the embedded representations used by the model.

Since the simulation environment runs at 50 Hz, but the network operates at only 10 Hz, inference only uses every fifth frame from the dataset. This choice gives the model a temporal window of 10 frames, equivalent to 1 second. Each time step uses three tokens for encoding, so the network must process 30 tokens, requiring a context length of 30. The dataset has 1,164,448 recorded frames, translating to 232,988 decision steps used for training. In each training epoch, the 1,000 training steps result in training on 32,000 traces. The warmup phase, during which the learning rate steadily increases, persists for two epochs, allowing the optimizers to collect statistical information and yield better optimization results.

Figure 6.4 shows the training loss dynamics for two full-image DT networks, which are the ones used in the evaluation. One of the networks manages to drive 70.86% of the track but takes considerable time (Fig. 6.4a). The other trace is faster but only manages 65.51% (Fig. 6.4b). The figure shows the total loss and the loss split into two distinct components: the loss associated with discrete and continuous actions. While

	rand	*att*
	Training Parameters	
Learning Rate	$\mathbf{1 \cdot 10^{-8}}$	$\mathbf{5 \cdot 10^{-9}}$
Weight Decay	$6 \cdot 10^{-6}$	$6 \cdot 10^{-6}$
Epochs	15	15
Warmup Steps	2,000	2,000
Steps per Epoch	1,000	1,000
Batch Size	32	32
	Hyperparameters	
Context Length	30	30
Embedding Size	512	512
Execution Frequency	10 Hz	10 Hz
Partial Image Size	64	64
Image Embedding Layers	$64x64x1 \rightarrow 15x15x32 \rightarrow 7x7x64$	$64x64x1 \rightarrow 15x15x32 \rightarrow 7x7x64$
Trainable Parameters	12.743.884	12.743.884

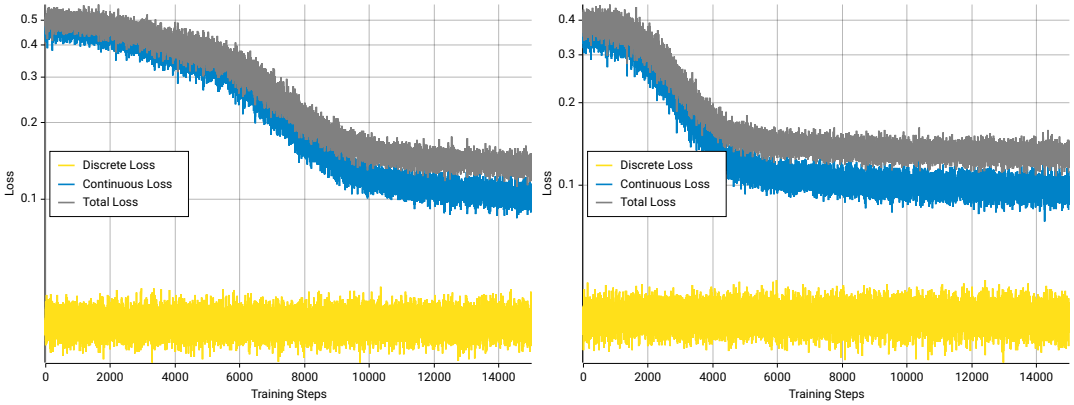
Table 6.3: Training parameters for the partial-image DT networks. Both networks use a smaller *Embedding Size*, resulting in a smaller CNN for embedding the input images and fewer trainable parameters. The learning rate is the only difference in training parameters between the two networks indicated in bold.

the continuous loss values show an initial relatively fast convergence, the losses for the discrete actions only decrease slightly, and the noise even increases over time.

The different behavior of the loss functions, observable in both traces, indicate the challenges in optimizing mixed action spaces. As no evaluation phase checks the loss values for a separate dataset, it is challenging to detect overfitting or other issues with training. Interestingly, this behavior is observable in both loss histories, possibly indicating structural problems with this thesis's mixed action space approach.

Table 6.3 provides an overview of the training parameters for both partial-image DT models, with *Human Attention Imitation*-based and the random selection policy. Most parameters for training are equal for both implementations; only the learning rate is different for the approaches. The *Human Attention Imitation*approach uses a learning rate of $5 \cdot 10^{-9}$ while the model with the random selection policy uses a slightly higher learning rate of $1 \cdot 10^{-8}$. Both use an *Embedding Size* of 512 and partial images of size 64 x 64. The CNN to embed the images uses two layers with size transitions progressing from $64x64x1$ to $15x15x32$ and ultimately to $7x7x64$. The remaining parameters are the same as in the full-image DT.

Figure 6.5 shows the training losses for two partial-image DT networks using the random selection policy. The network that managed to drive a full lap exhibits an initial higher training loss of 0.5 that gradually converges to an approximate loss of 0.1. The faster network starts with a lower loss value of about 0.35 but similarly converges to a loss of



(a) Trace that managed a full lap.

(b) Trace that managed 89.55% of the track in a faster projected lap time than the full lap.

Figure 6.5: Training losses for the partial DT models with random sub-image selection. The total loss (gray) is the sum of the loss values for the continuous (blue) and discrete actions (yellow).

approximately 0.1. Both networks converge at a comparable loss level, suggesting they reach a similar state during training.

The continuous action loss, in the same fashion as for the full-image DT, plays a dominant role in the total loss throughout training. Again, the discrete action loss remains relatively stable and consistent across the training process. This continued discrepancy in loss trends might indicate that combining different action spaces requires different optimizers or a joint loss function that can deal with discrete and continuous values.

For the final network configuration, Figure 6.6 shows the training losses of the network that managed to drive a full lap while simultaneously driving the fastest lap time with the *Human Attention Imitation*-based sub-image selection policy. The losses for continuous and discrete action values are similar to the other network configurations, with the discrete loss values remaining constant with considerable noise and the continuous loss values converging after approximately 7000 training steps. Like the random selection policy network that can drive a whole lap, the continuous loss values start at approximately 0.35 and converge to 0.1.

The consistently low final loss value of approximately 0.1 and the converging trend observed in the training process signal that the networks have undergone effective training, successfully adapting to their respective tasks. The small changes to the discrete action loss values indicate that the employed hybrid loss function is suboptimal, and other solutions might yield better results. However, it is worth noting that identifying overfitting is difficult without evaluation losses.

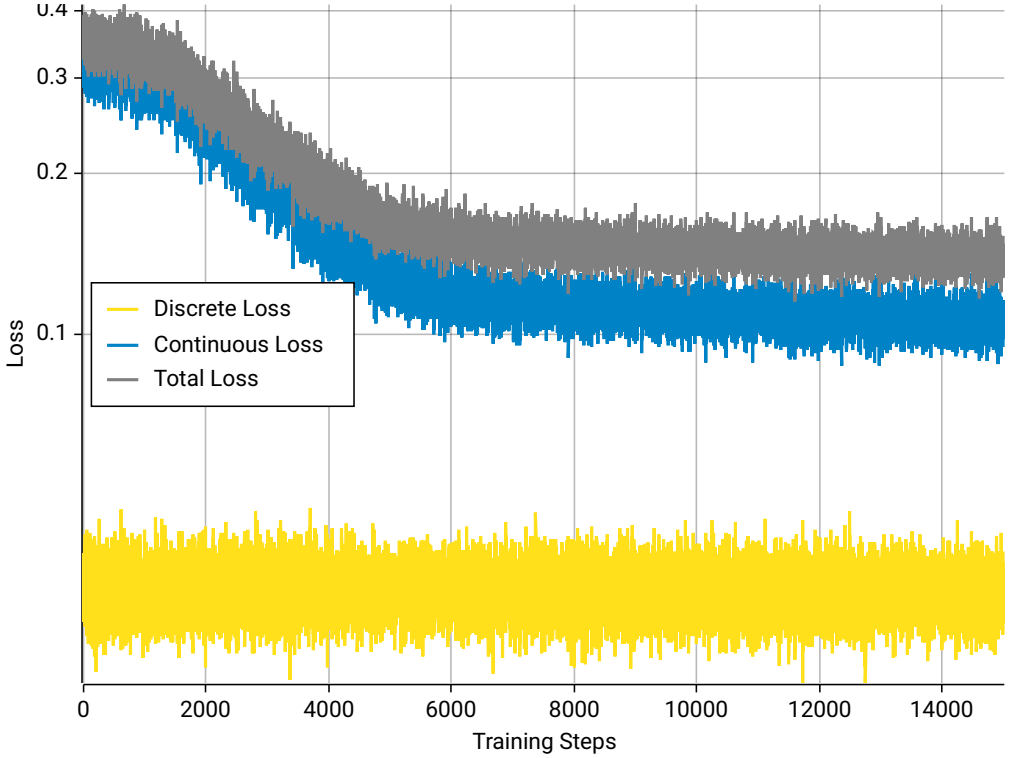


Figure 6.6: Training losses for the partial DT model with *Human Attention Imitation*-based sub-image selection policy. The total loss (gray) is the sum of the loss values for the continuous (blue) and discrete actions (yellow).

6.2.2 Evaluation

We conduct a systematic evaluation to determine which approach is the most effective among the three different ones we propose. Six models are trained for these three approaches, with prior fine-tuning of the training parameters to ensure optimal learning conditions.

We systematically choose two representative runs from each pre-processing approach from the various training attempts conducted. One is selected based on its ability to complete a full lap, indicating the model's capacity for safe driving, and the other for its speed.

Within each selection policy, we carefully pick one attempt that manages to produce the fastest run that completes a lap. Additionally, we select the quickest run that reached significant progress, at least 50%, for the speed-oriented selection. We used the projected lap time, calculated as $\frac{\text{run time}}{\text{progress}}$, to determine the speed of traces unable to complete laps.

Table 6.4 shows the selected runs used in the comparison. Under the *att* selection policy, the fastest network also completes a whole lap. We, therefore, selected only one trace to represent this selection policy. Notably, most of the more successful runs of all selection

Name	Sel. Policy	Att.	Epoch	Progress	Time	Proj. Time
Best	*att*	6	1	**100%**	**2:03.06 min**	**2:03.06 min**
Furthest	*none*	1	11	70.86%	2:46.82 min	3:55.41 min
Fast	*none*	4	7	65.51%	2:07.70 min	3:14.92 min
Full	*rand*	2	9	**100%**	4:05.40 min	4:05.40 min
Fast	*rand*	5	3	89.55%	2:36.96 min	2:55.27 min

Table 6.4: Selected evaluation runs for the comparison. The *Selection Policy*, *Attempt*, and *Epoch* uniquely identify each trace. Each evaluation run has a *Name* for identification unique per *Selection Policy*. *Progress* and *Time* give the maximum progress and the running time the evaluation run achieved. *Projected Time* is the estimated lap time used for traces that failed to finish a lap. Bold values, if present, indicate the best value in each column.

policies exhibit the best performance after only a few training epochs, with a total of 15.

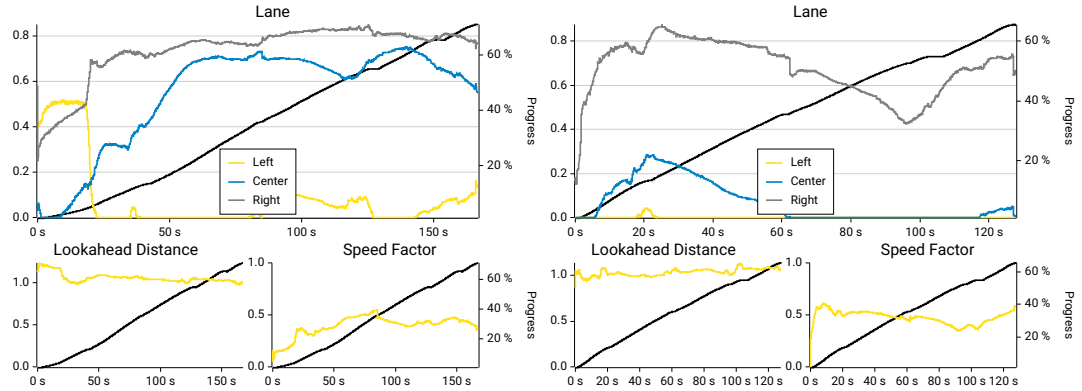
None of the six networks trained on full images accomplished a whole lap. We suspect the causes might be diverse, ranging from too little *Embedding Size* to overfitting. Consequently, we use the runs that achieve the most progress, providing a basis for comparison with runs that achieved complete laps.

This section of the thesis focuses on evaluating and comparing the performance of individual selection policies. It delves into a detailed analysis of their respective outcomes and offers a comparative assessment. The comparison uses three metrics: relative progress, driven path, and the variance of control outputs.

In our performance discussion, we undertake a detailed analysis of the model's actions, encompassing three primary components: the selected *Lane*, *Lookahead Distance*, and *Speed Factor*. These actions are crucial in determining the high-level controller's behavior and decision-making. To gain deeper insights into the actual behavior of the controller, we also investigate the effective actions. The high-level controller derives the effective *Lane* by applying `argmax` to the continuously valued output vector. Comparing these values provides a more accurate representation of the controller's choices as it represents the selected *Lane*.

Full image Decision Transformer

None of the six networks trained on full images demonstrates the capability to complete whole laps over a wide array of different attempts at hyperparameter configurations. These attempts involved experimenting with *Embedding Sizes* of 1024 and 2048; however, we did not pursue further exploration of larger *Embedding Sizes* to avoid excessive model sizes. We tested an array of learning rates, including $1 \cdot 10^{-6}$, $1 \cdot 10^{-7}$, $6 \cdot 10^{-8}$, $5 \cdot 10^{-8}$, $1 \cdot 10^{-8}$, $5 \cdot 10^{-9}$, $1 \cdot 10^{-9}$, and $1 \cdot 10^{-10}$, to optimize the network, but neither of them yielded a policy able to complete a whole lap. Additionally, we tried variations in epoch



(a) Evaluation actions for the trace with the furthest progress.

(b) Evaluation actions for the trace with the fastest progress.

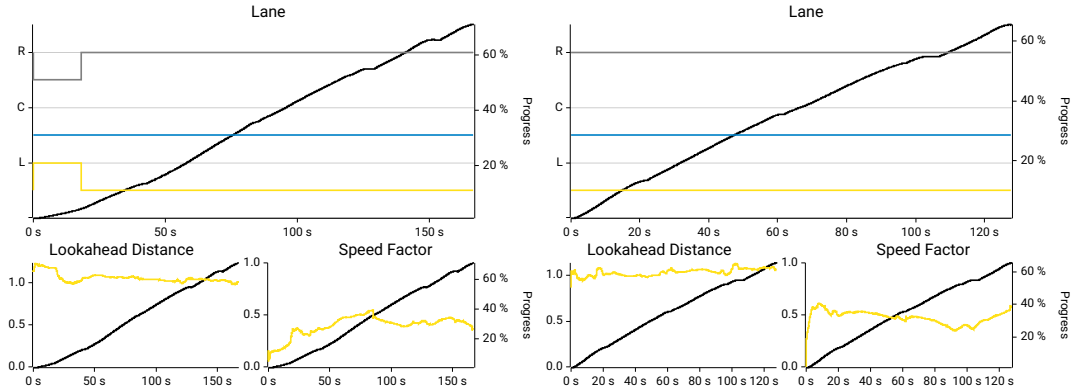
Figure 6.7: Action outputs by the full-image DT implementation. The upper graph shows the continuous outputs for the selected *Lane* (yellow: left, blue: center, gray: right) over the progress on the track (black). The lower left plot shows the *Lookahead Distance*, and the lower right shows the *Speed Factor* in each evaluation run.

sizes, the presence or absence of weight decay, and different configurations for warmup steps to find possibly working configurations.

Among the various configurations evaluated, the settings outlined in Table 6.2 emerged as the most successful in terms of both progress and projected lap time. Five of the six runs accomplish more than 50% of a lap, with two successfully navigating the most challenging section of the track, the steepest corner, where the other policies crash. It is worth noting, however, that even the network that achieves the most progress crashes at the second left corner. Similarly, the swiftest run terminates at the first left corner after a collision.

Figure 6.7 shows the continuous output action values generated by the networks, with 6.7a showcasing the trace with the most progress, denoted as *Furthest*, and 6.7b featuring the fastest trace, in the following called *Fast*. Notably, the *Furthest* trace exhibits a notable behavior where it initially employs the left (yellow) lane and subsequently switches to the right (gray) lane. This behavior contrasts with the *Fast* trace, which predominantly adheres to the right lane. A noteworthy observation is that, in the *Furthest* trace, the continuous output actions for *Lane* selection consistently display relatively high values for the center lane despite not being utilized due to the higher values for the right lane.

The values predicted for the center lane might indicate that the network would prefer to drive in the middle between the right and the center (blue) lane, with a slight preference for the right lane. The action space allowing driving between lanes would require a different design - continuous instead of discrete - to drive in the middle of multiple lanes but could warrant further exploration.



(a) Effective evaluation actions for the *Furthest* trace.

(b) Effective evaluation actions for the *Fast* trace.

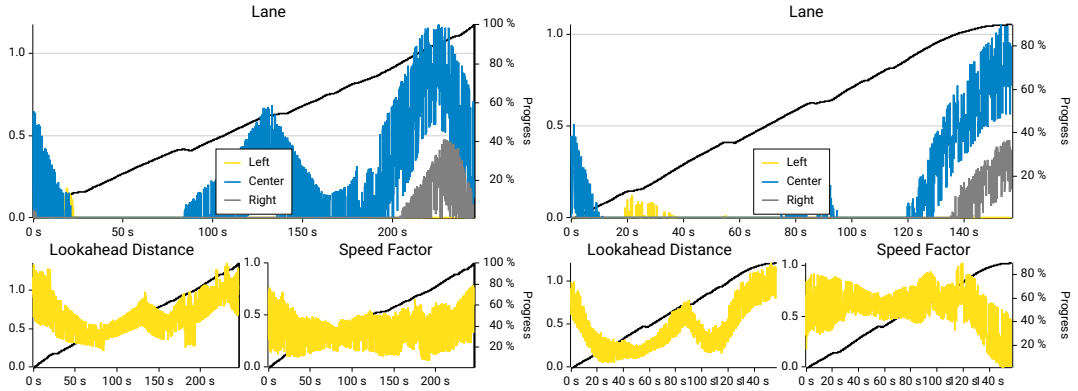
Figure 6.8: Effective action outputs by the full-image DT implementation. The controller uses the *argmax* function to determine the effective *Lane* used in the low-level controller. The upper half shows the currently selected *Lane* over time and the track progress (black). The lower plots show the *Lookahead Distance* and *Speed Factor*.

The *Lookahead Distance* for both traces remains relatively constant, while the *Speed Factor* of the *Furthest* trace experiences a minor initial increase, followed by modest fluctuations. In contrast, the fast trace demonstrates an early rise in the *Speed Factor*, with subsequent fluctuations similar to those of the *Furthest* trace.

The observed values in both traces exhibit relatively low variance, and the *Lane* selection values within both traces demonstrate distinct responses to corners, reflected by transient flat sections in the black progress line. Even though the continuous values show reactions to approaching corners, the selected *Lane* remains the same. This behavior is further illuminated in Figure 6.8, highlighting the values effectively employed by the low-level controller. Here, a prominent preference for the right lane becomes evident.

The behavior exhibited by the *Furthest* trace before encountering the first corner is promising. It initiates in the outer (left) lane with a lower speed and subsequently transitions to the right lane, accelerating as it navigates out of the corner. This behavior closely aligns with the outcome of minimum curvature optimization, but after the corner, the model changes to a constant policy after this corner. The initial good driving suggests that the network could achieve better results through further refinements in the network structure and training strategies.

While the full-image DT demonstrates promising behavior in one of the challenging corners, its inability to generalize and complete a full lap throughout this thesis suggests the presence of potential limitations. These issues could arise from insufficient *Embedding Size* or other training complexities. However, it is noteworthy that the model displays low variance in its control outputs, indicating its capacity to learn a stable and consistent decision-making policy.



(a) Evaluation actions for the *Full* trace. (b) Evaluation actions for the *Fast* trace.

Figure 6.9: Action outputs by the partial-image DT implementation using *rand*. The upper graph shows the continuous outputs for the selected *Lane* (yellow: left, blue: center, gray: right) over the progress on the track (black). The lower left plot shows the *Lookahead Distance*, and the lower right shows the *Speed Factor* in each evaluation run.

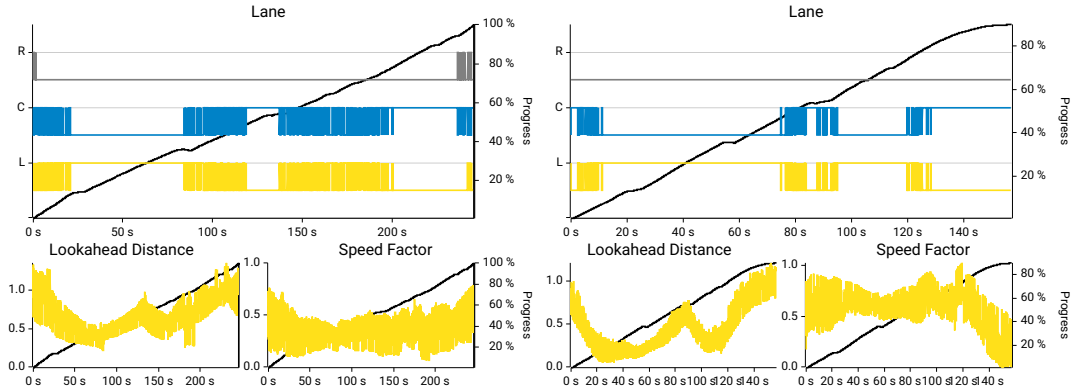
Random Selection Policy

Among the networks trained using the random sub-image selection policy, 5 out of 6 demonstrated the ability to navigate more than 50% of the track. Three of these networks navigate over 80% of the course, while 2 out of the six networks encounter crashes in the last corner. Remarkably, one network excels, completing a full lap, while the fastest network drives 89.55% of the track. During the dial-in phase, we explored various training parameter configurations, and while some did yield complete laps driven, the configuration detailed in Table 6.3 consistently produces good training results.

Figure 6.9 shows the output actions of the networks with 6.9a illustrating the trace that completed a full lap and 6.9b depicting the fastest trace. Both networks prefer the center (blue) lane as their primary choice. Nevertheless, there is a shared minor peak in the right (gray) continuous *Lane* output towards the end of the trace, while neither network uses the left (yellow) lane much.

The *Lookahead Distance* and *Speed Factor* exhibit similar behavior in both traces, adjusting dynamically to changes in trajectory. While the *Speed Factor* for the faster trace remains consistently higher than that for the full lap, it drops at the end of the trace, where the fast trace effectively halts. The *Lookahead Distance* decreases as both networks approach corners, utilizing more dynamic driving behavior in the low-level controller, and increases on straight sections, enabling more stable driving.

The high variance in the control outputs complicates discerning which *Lane* is active. However, Figure 6.10 presents the effective actions and provides a clearer view of the currently used *Lane*. Surprisingly, it reveals that both traces spend a substantial amount of time in the left lane. Transitions between lanes also exhibit considerable noise, making



(a) Effective evaluation actions for the *Full* trace.

(b) Effective evaluation actions for the *Fast* trace.

Figure 6.10: Effective action outputs by the partial-image DT implementation using *rand*. The upper half shows the currently selected *Lane* over time and the track progress (black). The lower plots show the *Lookahead Distance* and *Speed Factor*.

the network's *Lane*switching behavior somewhat erratic. The full lap trace displays higher noise during *Lane* changes, while the fast trace demonstrates shorter noise during transitions between lanes.

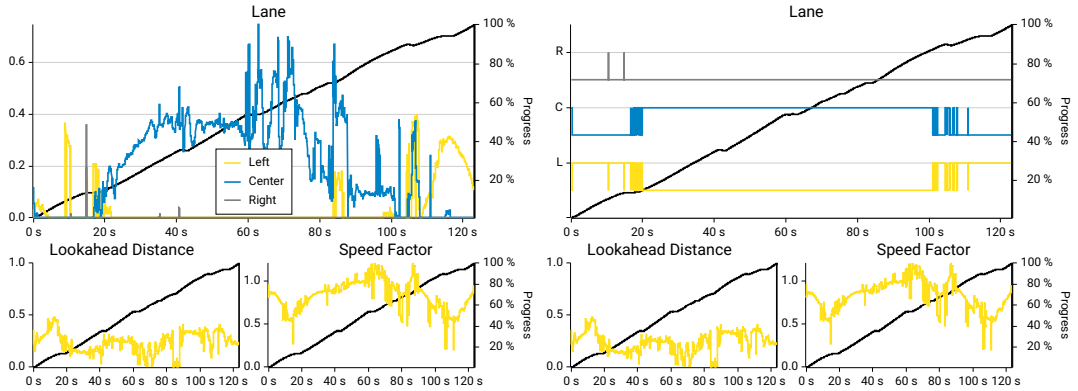
The unexpectedly high use of the left lane stems primarily from the implementation details of the *PyTorch argmax* function. This implementation returns the index of the first occurrence of the maximum value, which, in the case of an all-zero vector, is the index 0, resulting in the left lane, which uses index 0 in this thesis' implementation.

Despite the high variance in the control outputs, the action trend remains remarkably consistent. This consistency underscores the effectiveness of the context-aware GPT architecture employed by the DT. Despite the high variance in individual actions, the network's ability to maintain a cohesive overall course showcases the robustness and adaptability of this approach.

Random sub-image selection emerges as a viable tool to reduce the complexity of the DT architecture, enabling the learning of fast and effective policies capable of driving complete laps. Despite the inherently random nature of the inputs, consistent trends in output actions point to the DT's capacity to establish meaningful temporal connections. This ability is showcased in the network's learned policy for controlling *Lookahead Distance* and *Speed Factor*, which aligns closely with the intuitive behavior exhibited by human drivers, exemplifying the adaptability and decision-making capabilities of the DT.

Human-Attention Imitation Selection Policy

Among the networks trained with *Human Attention Imitation*-based sub-image selection, 4 out of 6 can navigate more than 50% of the track. However, 2 out of the six networks



(a) Evaluation actions for the trace using the *att* selection policy.

(b) Effective evaluation actions for the trace using the *att* selection policy.

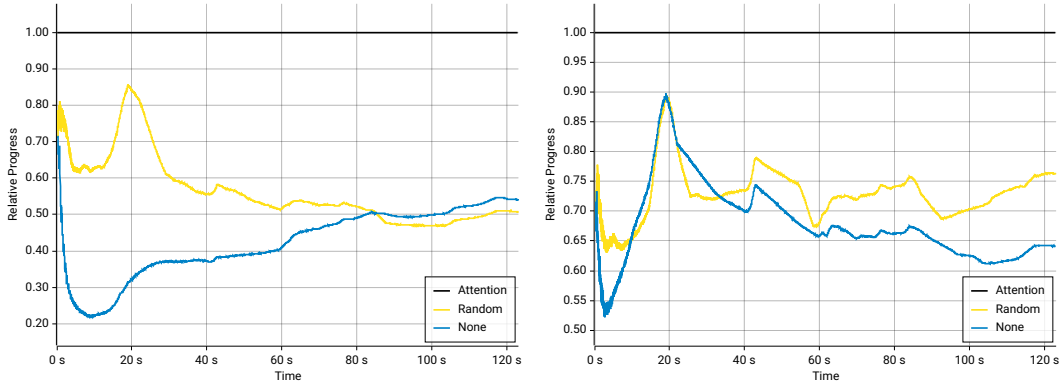
Figure 6.11: Output and effective output actions for the partial-image DT network using the *att* selection policy. The upper graph shows the continuous outputs for the selected *Lane* (yellow: left, blue: center, gray: right) over the progress on the track (black). The lower left plot shows the *Lookahead Distance*, and the lower right shows the *Speed Factor* in each evaluation run.

fail to initiate driving, while another two encounter crashes at the steepest corner of the track. 2 of the six networks complete a full lap. One of these successful laps achieves remarkable speed, outperforming the other full trace by 66 seconds. Consequently, we consider only the fastest network that drives an entire lap for comparison, denoted as *Best*.

Figure 6.11a presents the output actions of the attention-based network, showing distinct *Lane* selection outputs. The trace begins by driving on the left (yellow) lane and subsequently transitions to the center (blue) of the track, maintaining this central lane for most of the course, with a brief return to the left lane towards the end. The *Lookahead Distance* is consistently low, even on straight sections, resulting in some oscillation on straight sections. At the same time, the *Speed Factor* exhibits a slight reduction before corners yet consistently remains high for most of the track, decreasing during the final few corners and increasing at the end of the trace.

The network consistently generates low-variance outputs, even after just one training episode, underlining its stability and reliability in decision-making. Figure 6.11b shows the effective actions executed by the low-level controller. Compared to the random selection policy, the intervals marked by noise during *Lane* changes are shorter, highlighting the network's more precise *Lane*switching behavior. Even though this model achieves the fastest lap times of all trained models, its outputs differ from the values in the minimum curvature trajectory.

The network achieves fast lap times despite driving a trajectory different from the expert dataset. The low variances observed in its control outputs indicate that the network has



(a) Relative progress for the traces of complete laps.

(b) Relative progress for the traces that drive fast and manage considerable progress.

Figure 6.12: Relative progress comparisons between *att* (black), *rand* (yellow), and *none* (blue) evaluation runs.

successfully learned a stable and reliable policy. Furthermore, as evidenced by the *Lane* output changes, the network's adaptive behavior during corners suggests that it reacts to visual inputs.

Comparative Analysis

We conduct a comprehensive evaluation across different criteria to determine the effectiveness of the three selection policies for partial image inputs to a network. In this evaluation, we compare relative progress values among evaluation traces to assess their performance. Visual comparisons of the paths driven by different policies offer valuable insights into their decision-making and ability to react to visual cues. Furthermore, we analyze the variance in the output behaviors of different networks to assess their stability.

Calculating relative progress involves setting a baseline trace as a reference, in this case, the *Best* trace from the *Human Attention Imitation*-based network. For the other two approaches, we divide the reported progress by the reported progress of the baseline at each time step. If the resulting value is less than 1, it signifies that the other trace is slower than the baseline trace, and conversely, if it is greater than 1, it indicates that the other trace is faster.

Figure 6.12 compares the *Human Attention Imitation*-based trace with the traces that completed a lap, as shown in Figure 6.12a, and the fast traces, shown in Figure 6.12b. In both comparisons, the attention-based trace consistently outperforms the others, finishing its laps before the other traces either complete a lap or encounter simulation terminations. These premature terminations occur primarily due to crashes or the car stopping to drive.

The full-image DT starts slower when comparing complete laps, while the random selection-based network quickly reaches a level of performance close to the baseline. This

behavior changes as the random approach slowly falls more and more behind the baseline. At the same time, the full-image approach catches up with the random approach and eventually overtakes it. Despite this, the full-image DT maintains only a slight lead, with approximately 55% at the end, over the random approach, which reaches about 50% by the time the attention-based approach finishes.

This pattern slightly changes when comparing the fastest traces. Both the random selection and full-image networks show an initial dip in progress. While the full-image DT initially manages to catch up to the random selection network, the random approach continuously drives slightly faster than the full-image approach. The full-image DT manages around 65% of the track, while the random selection policy achieves 75% before the attention-based approach finishes the lap.

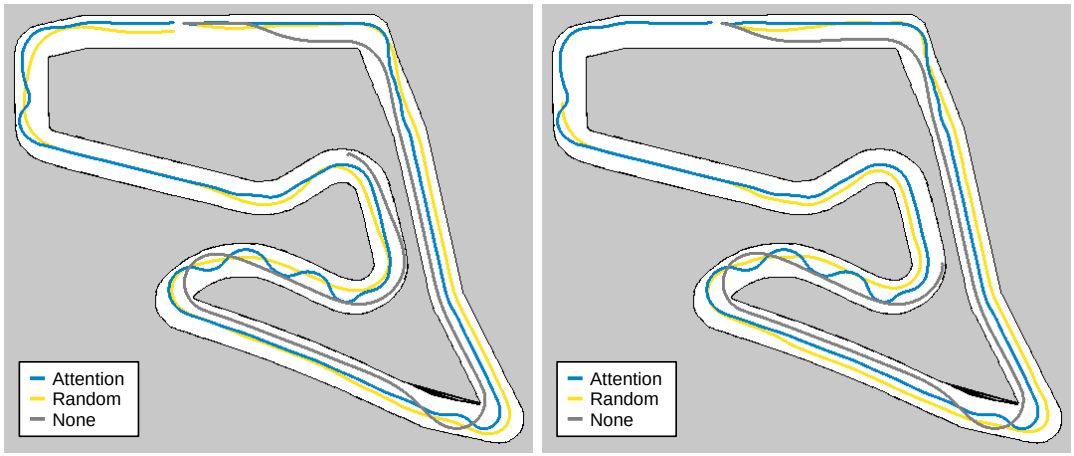
The comparison of relative progress among different selection strategies shows that the attention-based sub-image selection policy yields the fastest full lap, showcasing its usefulness in autonomous racing. In contrast, the random sub-image selection drives slower and with more variance in the output values, likely due to the higher variance in the input images. Given that the full-image approach manages to catch up with the random selection network in the full-lap comparison, improving the architecture's performance appears possible.

In autonomous racing, the pursuit of speed must go hand in hand with safety. We investigate the trajectories that the various networks drive to assess the safety of the driving behavior. By examining these paths alongside the actions the networks execute, we gain valuable insights into their ability to control the low-level controller effectively. Behaviors like oscillating, overshooting, or cutting corners are undesirable while driving through corners with as little way traveled as possible, and smooth steering is desirable.

Figure 6.13 shows the paths driven by the models under comparison. Figure 6.13a illustrates the paths of the networks completing total laps, while Figure 6.13b showcases the paths of the fastest traces. The attention-based trace (blue) is the same in both plots, as it was both the fastest and the trace that completed a whole lap. It drives with little overshooting and follows the selected *Lane* closely. However, oscillations in the path occur in two areas, notably after the steepest corner and between the final two corners. These oscillations stem from the low lookahead values of the network outputs in these areas.

In the case of complete laps, the random (yellow) strategy closely adheres to the lanes with minimal overshooting or corner-cutting and without oscillations. In stark contrast, this category's full-image (gray) DT network tends to overshoot significantly. This overshooting behavior eventually results in a crash towards the middle of the track. We attribute these issues to a high *Speed Factor* and a low *Lookahead Distance*.

In the comparison of the fastest laps, a similar pattern is visible. The random *Fast* trace demonstrates *Lane* adherence but slightly cuts corners, even more so than the random *Full* trace. On the other hand, the full-image DT continues to exhibit an overshooting tendency, eventually leading to a crash. This observation underscores a key



(a) Models that drive full laps or the furthest for their respective selection policy.

(b) Models that drive fast lap times and a significant portion of the track.

Figure 6.13: Traces of evaluation runs plotted on a track map. The recording for the *att* (blue) trace is the same, as this trace was the fastest recorded and completed a whole lap.

point—high speed is not always advantageous, as safety and stability are also paramount in autonomous racing scenarios.

The comparison of the paths taken by the networks yields several notable findings. While the attention-based network may exhibit impressive speed, it oscillates significantly, behavior generally considered undesirable. In contrast, the random selection network demonstrates an ability to closely follow the lanes on the track and react to changes in the track. On the other hand, the full-image DT encounters challenges in learning a dynamic and adaptive strategy, possibly due to an inadequate *Embedding Size* to process the complete image or issues stemming from the training process.

We suspect the underlying GPT network combines an internal representation of the entire image based on the random snippets it receives. Even though this representation would have a coarse temporal resolution, with this internal representation, a model could interpret the whole scene without receiving it as input.

Finally, we compare the output variances of the networks under comparison. This comparison provides valuable insights into the stability of the network's policy. We compute the variance about a moving average μ_i over 50 time steps (equivalent to 10 control steps). To calculate the variance of a set of values, we use the following formula:

$$\text{Variance } \sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_i)^2 \tag{6.2}$$

where N represents the length of the trace, x_i are individual data values, and μ_i signifies the moving average.

	att	*rand*		*none*	
		full	**fast**	**furthest**	**fast**
Left Lane	$1.87 \cdot 10^{-3}$	$9.97 \cdot 10^{-5}$	$8.01 \cdot 10^{-5}$	$1.92 \cdot 10^{-4}$	$1.24 \cdot 10^{-3}$
Center Lane	$3.47 \cdot 10^{-3}$	$1.75 \cdot 10^{-2}$	$5.40 \cdot 10^{-3}$	$4.09 \cdot 10^{-5}$	0.00
Right Lane	$1.07 \cdot 10^{-4}$	$1.78 \cdot 10^{-3}$	$8.50 \cdot 10^{-4}$	$8.44 \cdot 10^{-5}$	$8.87 \cdot 10^{-4}$
Lookahead Dist.	$2.07 \cdot 10^{-3}$	$1.31 \cdot 10^{-2}$	$9.94 \cdot 10^{-3}$	$3.11 \cdot 10^{-5}$	$7.89 \cdot 10^{-4}$
Speed Factor	$4.19 \cdot 10^{-3}$	$1.51 \cdot 10^{-2}$	$1.74 \cdot 10^{-2}$	$5.02 \cdot 10^{-5}$	$1.51 \cdot 10^{-3}$

Table 6.5: Variance for action output values in the evaluation runs under comparison. As the *Lane* selection values for some traces are 0.0 for long stretches, variances of those selection values are less informative.

Table 6.5 presents the variance values for each action output, providing insights into the stability of each network's policy. The variance comparison predominantly focuses on the *Lookahead Distance* and *Speed Factor*. While the variance for the *Lane* actions is less informative. As both random approaches output zero values for the action output values for the left lane, the resulting zero variance influences the averaging in the variance calculation.

The analysis of variance values reveals notable differences between the various sub-image selection policies. While the random selection policy demonstrates similar variance values for the *Lookahead Distance* and *Speed Factor*, the faster full-image DT displays higher variance in the *Lookahead Distance* than in the *Speed Factor*. Furthermore, the *Speed Factor* variance in the fastest full-image DT is approximately 2.5 orders higher than that of the furthest trace.

The variance values of the attention-based network are roughly an order of magnitude smaller than those of the random selection network. Additionally, the full-image DT exhibits smaller variance values than the *Human Attention Imitation*-based network, underscoring the differences in the stability and adaptability of these sub-image selection policies.

The recorded *att* trace, captured after a single training epoch, stands out for its lower variance than the random networks trained over 3 and 9 epochs. This observation highlights the attention-based network's capability to learn a more stable policy early in its training process. Nevertheless, we expected the increased variance in the random networks, given that they inherently receive more variable inputs due to the random selection of sub-images. Furthermore, the full-image DT employed more training epochs (7 and 11) to achieve comparable variance.

The lower variance in the *att* trace, captured after a single training epoch, compared to the variance of the random networks trained over 3 and 9 epochs, shows the influence of *Human Attention Imitation*on a network's capability to learn a more stable policy early in its training process. Nevertheless, we expected the increased variance in the random networks, given that they inherently receive more variable inputs due to the random

selection of sub-images. Furthermore, the full-image DT employed more training epochs (7 and 11) to achieve comparable variance.

One intriguing conclusion we dare to draw is that the attention-based network exhibits a variance profile akin to that of the full-image DT trained over seven episodes. This insight underscores the potential of *Human Attention Imitation*to enhance network performance, presenting an avenue for further exploration and development in the domain of ANNs.

CHAPTER 7

Conclusion

In this chapter, we summarize the contributions made in this thesis. We develop a novel architecture inspired by human attention for Reinforcement Learning (RL). The proposed approach developed within this thesis consists of two major components: an ANN designed to replicate human attention patterns, called *Human Attention Imitation*, and a GPT-based racing controller. The *Human Attention Imitation* predicts ROIs in depth images. The racing controller leverages the Decision Transformer (DT) architecture to control the vehicle. In the experimental section, we evaluate the advantages and limitations of each approach, considering different pre-processing approaches for selecting either partial or complete images as inputs.

To imitate human attention, we train an ANN on real-world data recorded from human drivers' remote-controlling F1Tenth miniature cars in single and multi-agent scenarios with obstacles on the track. The racing controller, trained instead in simulation, produces control predictions based on depth images, vehicle state, return-to-go signal, and prior actions. To train it, we designed a reward to prioritize minimum lap time without explicitly encouraging smoothness in control. In fact, the racing agent achieves impressive lap times by executing precise, though occasionally counterintuitive, control actions.

To assess the influence of the selected portion of the input image, we compare three distinct pre-processing strategies: we consider the complete image as input, we select a random portion of the image, and we use the *Human Attention Imitation* to select the relevant portion. During training, we observed that the full-image approach exhibits convergence after fewer training steps than the partial-image strategies. The partial-image methods, owing to their smaller network size, require significantly less computational time for training but need more iterations to converge. They also learn more intricate policies than the full-image approach, which predominantly learns rudimentary strategies. In contrast, despite exhibiting slower driving performance, the random approach showed better performance in high-level control, allowing more effective tracking of available lanes. Apart from the random approach driving slower, we also suspect that this observation suggests that the underlying GPT architecture can effectively reconstruct essential elements of the complete image over multiple time steps, facilitating a broader understanding of the entire scene.

This thesis successfully demonstrates the feasibility of training a driving agent by leveraging a human-inspired attention model. Applying this imitation model achieves comparable and often superior performance to several baselines, suggesting that a more selected attention mechanism has a large potential for learning control applications.

7.1 Future Work

The findings of this thesis open up further avenues for scientific exploration. These either focus on further exploring the architectures and methods used in the thesis or on new concepts and understandings gained from those architectures.

Exploring additional mechanisms to provide ROIs to ANNs would be interesting. These

mechanisms could utilize properties of the network, such as *saliency*, or be outputs of the same network or a different network, trained in tandem. It would also be interesting to try different approaches for the DT approach, including different methods to tokenize data or changes in hyperparameters. Interpreting inter-modal correlations could offer valuable insights into the inner workings of the network, and using a separate evaluation dataset to interpret the training progress better.

In addition, we intend to investigate if the randomized images lead to an internal representation of the scene in the random selection approach. Further, we would like to find out if supplying a low-resolution overview image and a high-resolution partial image of the ROI would benefit the system even more. The low-resolution overview would then correspond to low-resolution areas in human eyes, which enable reflexive reactions, also called bottom-up attention, while the high-resolution smaller image would match the concept of the fovea, the high-resolution area of our eye, which is used for task-specific attention.

List of Figures

List of Tables

Acronyms

ANN Artificial Neural Network. ix, xi, 2–4, 6, 8–12, 14–17, 19, 22, 36, 43, 49, 50, 73, 76

AWL Adaptive Wing Loss. 11, 33, 34, 52

CE Cross-Entropy. 57, 58

CNN Convolutional Neural Network. 2, 47, 49, 60

CPS Cyber-Physical System. 2

DT Decision Transformer. ix, xi, xiii, 11, 45–48, 52, 57–73, 76, 77, 79–81

GPT Generative Pre-trained Transformers. ix, xi, 11, 45–48, 50, 57, 67, 71, 76

IPC Inter-Process Communication. 22

LSTM Long Short-Term Memory. 31

MPC Model Predictive Control. 7, 8, 40

MPCC Model Predictive Contouring Control. 7

MRI Magnetic Resonance Imaging. 11, 33

MSE Mean Squared Error. 57

Offline RL Offline Reinforcement Learning. ix, xi, 4, 36, 40, 41, 52, 57

QP Quadratic Programming. 8

ReLU Rectified Linear Unit. 31, 47, 48

RL Reinforcement Learning. 10, 14, 17, 25, 41, 43, 45, 56, 57, 76

RNN Recurrent Neural Network. 31

ROI Region of Interest. 3, 11, 14–16, 22, 30, 50, 54, 76, 77

ROS Robot Operating System. 22, 24

TanH Hyperbolic Tangent. 47

Bibliography

[1] ISO/TC 22/SC 32. *ISO_26262-1:2018*. Dec. 2018. URL: https://effects.austrian-standards.at/action/effectsDownload/648953/4/ISO_26262-1_2018_2018_12_17_en.pdf (visited on 01/04/2022).

[2] Stanley Bak et al. "Stress Testing Autonomous Racing Overtake Maneuvers with RRT". In: *arXiv:2110.01095 [cs]* (Oct. 3, 2021). arXiv: 2110.01095. URL: http://arxiv.org/abs/2110.01095 (visited on 12/16/2021).

[3] Jayanth Bhargav et al. "Track based Offline Policy Learning for Overtaking Maneuvers with Autonomous Racecars". In: *arXiv:2107.09782 [cs]* (July 20, 2021). arXiv: 2107.09782. URL: http://arxiv.org/abs/2107.09782 (visited on 12/16/2021).

[4] Tom B. Brown et al. *Language Models are Few-Shot Learners*. July 22, 2020. DOI: 10.48550/arXiv.2005.14165. arXiv: 2005.14165[cs]. URL: http://arxiv.org/abs/2005.14165 (visited on 08/08/2023).

[5] Tim Brüdigam et al. "Gaussian Process-based Stochastic Model Predictive Control for Overtaking in Autonomous Racing". In: *arXiv:2105.12236 [cs]* (May 25, 2021). arXiv: 2105.12236. URL: http://arxiv.org/abs/2105.12236 (visited on 12/16/2021).

[6] Axel Brunnbauer and Luigi Berducci. *racecar_gym*. Version 0.0.1. original-date: 2020-08-15T08:26:44Z. Oct. 5, 2023. URL: https://github.com/axelbr/racecar_gym (visited on 10/07/2023).

[7] Kjell Brunnström et al. "Latency impact on Quality of Experience in a virtual reality simulator for remote control of machines". In: *Signal Processing Image Communication* 89 (Nov. 1, 2020), p. 116005. DOI: 10.1016/j.image.2020.116005.

[8] *Bullet Real-Time Physics Simulation | Home of Bullet and PyBullet: physics simulation for games, visual effects, robotics and reinforcement learning*. Mar. 21, 2022. URL: https://pybullet.org/wordpress/ (visited on 11/05/2023).

[9] Runqi Chai et al. "Multiobjective Overtaking Maneuver Planning for Autonomous Ground Vehicles". In: *IEEE Transactions on Cybernetics* 51.8 (Aug. 2021). Conference Name: IEEE Transactions on Cybernetics, pp. 4035–4049. ISSN: 2168-2275. DOI: 10.1109/TCYB.2020.2973748.

[10] Lili Chen et al. *Decision Transformer: Reinforcement Learning via Sequence Modeling*. June 24, 2021. DOI: 10.48550/arXiv.2106.01345. arXiv: 2106.01345[cs]. URL: http://arxiv.org/abs/2106.01345 (visited on 03/16/2023).

[11] OpenStax College. *Illustration from Anatomy & Physiology, Connexions Web site. http://cnx.org/content/col11496/1.6/, Jun 19, 2013.* URL: https://commons.wikimedia.org/wiki/File:1607_Saccadic_Movements.jpg (visited on 07/19/2023).

[12] Serdar Coskun. "Autonomous overtaking in highways: A receding horizon trajectory generator with embedded safety feature". In: *Engineering Science and Technology, an International Journal* 24.5 (Oct. 1, 2021), pp. 1049–1058. ISSN: 2215-0986. DOI: 10.1016/j.jestch.2021.02.005. URL: https://www.sciencedirect.com/science/article/pii/S2215098621000343 (visited on 12/16/2021).

[13] R Craig Coulter. "Implementation of the Pure Pursuit Path Tracking Algorithm". In: (Jan. 1992).

[14] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. 2016. URL: http://pybullet.org.

[15] *CUDA Toolkit - Free Tools and Training*. NVIDIA Developer. URL: https://developer.nvidia.com/cuda-toolkit (visited on 10/19/2023).

[16] *F1TENTH*. URL: https://f1tenth.org/ (visited on 07/18/2023).

[17] Zhen-Hua Feng et al. "Wing Loss for Robust Facial Landmark Localisation with Convolutional Neural Networks". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). Salt Lake City, UT, USA: IEEE, June 2018, pp. 2235–2245. ISBN: 978-1-5386-6420-9. DOI: 10.1109/CVPR.2018.00238. URL: https://ieeexplore.ieee.org/document/8578336/ (visited on 07/30/2023).

[18] Sergio Garrido-Jurado et al. "Automatic generation and detection of highly reliable fiducial markers under occlusion". In: *Pattern Recognition* 47 (June 1, 2014), pp. 2280–2292. DOI: 10.1016/j.patcog.2014.01.005.

[19] Charley Gros, Andreanne Lemay, and Julien Cohen-Adad. "SoftSeg: Advantages of soft versus binary training for image segmentation". In: *Medical Image Analysis* 71 (July 1, 2021), p. 102038. ISSN: 1361-8415. DOI: 10.1016/j.media.2021.102038. URL: https://www.sciencedirect.com/science/article/pii/S1361841521000840 (visited on 07/30/2023).

[20] Kai Han et al. "A Survey on Vision Transformer". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.1 (Jan. 2023). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 87–110. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2022.3152247.

[21] Alexander Heilmeier et al. “Minimum curvature trajectory planning and control for an autonomous race car”. In: *Vehicle System Dynamics* 58.10 (Oct. 2, 2020). Publisher: Taylor & Francis, pp. 1497–1527. ISSN: 0042-3114. DOI: 10.1080/00423114.2019.1631455. URL: https://www.tandfonline.com/doi/full/10.1080/00423114.2019.1631455 (visited on 10/07/2023).

[22] *Home*. OpenCV. URL: https://opencv.org/ (visited on 07/18/2023).

[23] Xin Huang et al. “Risk Conditioned Neural Motion Planning”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). ISSN: 2153-0866. Sept. 2021, pp. 9057–9063. DOI: 10.1109/IROS51168.2021.9636201.

[24] *J3016C: Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles - SAE International*. URL: https://www.sae.org/standards/content/j3016_202104/ (visited on 11/03/2021).

[25] Andrew Jaegle et al. *Perceiver IO: A General Architecture for Structured Inputs & Outputs*. Mar. 15, 2022. DOI: 10.48550/arXiv.2107.14795. arXiv: 2107.14795[cs,eess]. URL: http://arxiv.org/abs/2107.14795 (visited on 04/11/2023).

[26] Andrew Jaegle et al. *Perceiver: General Perception with Iterative Attention*. June 22, 2021. DOI: 10.48550/arXiv.2103.03206. arXiv: 2103.03206[cs,eess]. URL: http://arxiv.org/abs/2103.03206 (visited on 04/11/2023).

[27] Addie Johnson and Robert W. Proctor. *Attention: Theory and Practice*. Google-Books-ID: YTAoEX4LiAUC. SAGE, 2004. 489 pp. ISBN: 978-0-7619-2761-7.

[28] S. Kastner and L. G. Ungerleider. “Mechanisms of visual attention in the human cortex”. In: *Annual Review of Neuroscience* 23 (2000), pp. 315–341. ISSN: 0147-006X. DOI: 10.1146/annurev.neuro.23.1.315.

[29] Mathias Lechner et al. “Neural circuit policies enabling auditable autonomy”. In: *Nature Machine Intelligence* 2.10 (Oct. 2020). Bandiera_abtest: a Cg_type: Nature Research Journals Number: 10 Primary_atype: Research Publisher: Nature Publishing Group Subject_term: Computer science;Information technology;Software;Statistics Subject_term_id: computer-science;information-technology;software;statistics, pp. 642–652. ISSN: 2522-5839. DOI: 10.1038/s42256-020-00237-3. URL: https://www.nature.com/articles/s42256-020-00237-3 (visited on 11/03/2021).

[30] Nan Li et al. “Autonomous racecar control in head-to-head competition using Mixed-Integer Quadratic Programming”. In: (2021), p. 6.

[31] Liangkai Liu et al. “Computing Systems for Autonomous Driving: State of the Art and Challenges”. In: *IEEE Internet of Things Journal* 8.8 (Apr. 2021). Conference Name: IEEE Internet of Things Journal, pp. 6469–6486. ISSN: 2327-4662. DOI: 10.1109/JIOT.2020.3043716.

[32] B. Mashadi and M. Majidi. “Global optimal path planning of an autonomous vehicle for overtaking a moving obstacle”. In: *Latin American Journal of Solids and Structures* 11.14 (2014), pp. 2555–2572. ISSN: 1679-7825. DOI: 10.1590/S1679-78252014001400002. URL: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S1679-78252014001400002&lng=en&tlng=en (visited on 12/16/2021).

[33] Nikolce Murgovski and Jonas Sjöberg. “Predictive cruise control with autonomous overtaking”. In: *2015 54th IEEE Conference on Decision and Control (CDC)*. 2015 54th IEEE Conference on Decision and Control (CDC). Dec. 2015, pp. 644–649. DOI: 10.1109/CDC.2015.7402302.

[34] OpenAI. *GPT-4 Technical Report*. Mar. 27, 2023. DOI: 10.48550/arXiv.2303.08774. arXiv: 2303.08774[cs]. URL: http://arxiv.org/abs/2303.08774 (visited on 08/08/2023).

[35] Zhichao Peng et al. “Speech Emotion Recognition Using 3D Convolutions and Attention-Based Sliding Recurrent Networks With Auditory Front-Ends”. In: *IEEE Access* 8 (2020). Conference Name: IEEE Access, pp. 16560–16572. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2967791.

[36] Plamen Petrov and Fawzi Nashashibi. “Modeling and Nonlinear Adaptive Control for Autonomous Vehicle Overtaking”. In: *IEEE Transactions on Intelligent Transportation Systems* 15.4 (Aug. 2014). Conference Name: IEEE Transactions on Intelligent Transportation Systems, pp. 1643–1656. ISSN: 1558-0016. DOI: 10.1109/TITS.2014.2303995.

[37] *PyTorch*. URL: https://www.pytorch.org (visited on 10/19/2023).

[38] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. Number: arXiv:1505.04597. May 18, 2015. DOI: 10.48550/arXiv.1505.04597. arXiv: 1505.04597[cs]. URL: http://arxiv.org/abs/1505.04597 (visited on 11/02/2022).

[39] *ROS: Home*. URL: https://www.ros.org/ (visited on 07/19/2023).

[40] *Semantic-Segmentation-Architecture/PyTorch/unet.py at main · nikhilroxtomar/Semantic-Segmentation-Architecture · GitHub*. URL: https://github.com/nikhilroxtomar/Semantic-Segmentation-Architecture/blob/main/PyTorch/unet.py (visited on 10/19/2023).

[41] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. “On a Formal Model of Safe and Scalable Self-driving Cars”. In: *arXiv:1708.06374 [cs, stat]* (Oct. 27, 2018). arXiv: 1708.06374. URL: http://arxiv.org/abs/1708.06374 (visited on 02/28/2022).

[42] T. Shamir. “How should an autonomous vehicle overtake a slower moving vehicle: design and analysis of an optimal trajectory”. In: *IEEE Transactions on Automatic Control* 49.4 (Apr. 2004). Conference Name: IEEE Transactions on Automatic Control, pp. 607–610. ISSN: 1558-2523. DOI: 10.1109/TAC.2004.825632.

[43] TUM-Institute of Automotive Technology. *Introduction*. original-date: 2019-05-21T08:29:33Z. Nov. 1, 2023. URL: https://github.com/TUMFTM/global_racetrajectory_optimization (visited on 11/02/2023).

[44] *The World's Smallest AI Supercomputer*. NVIDIA. URL: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-xavier-nx/ (visited on 07/18/2023).

[45] Ghumman Usman and Faraz Kunwar. "Autonomous vehicle overtaking- an online solution". In: *2009 IEEE International Conference on Automation and Logistics*. 2009 IEEE International Conference on Automation and Logistics. ISSN: 2161-816X. Aug. 2009, pp. 596–601. DOI: 10.1109/ICAL.2009.5262854.

[46] *UST-10LX :: Sentek Hokuyo*. URL: https://hokuyo-usa.com/products/lidar-obstacle-detection/ust-10lx (visited on 07/18/2023).

[47] Ashish Vaswani et al. "Attention is All you Need". In: 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA, p. 11.

[48] *VPS 19 – the simplest Smart Glasses in the World*. VIEWPOINTSYTEM. URL: https://viewpointsystem.com/en/products/ (visited on 07/18/2023).

[49] Richard Wallace et al. "First Results in Robot Road-Following." In: Jan. 1, 1985, pp. 1089–1095.

[50] Xinyao Wang, Liefeng Bo, and Li Fuxin. *Adaptive Wing Loss for Robust Face Alignment via Heatmap Regression*. May 19, 2020. DOI: 10.48550/arXiv.1904.07399. arXiv: 1904.07399[cs]. URL: http://arxiv.org/abs/1904.07399 (visited on 07/30/2023).

[51] Trent Weiss, Varundev Suresh Babu, and Madhur Behl. "Be'zier Curve Based End-to-End Trajectory Synthesis for Agile Autonomous Driving". In: (), p. 10.

[52] Moritz Werling et al. "Optimal trajectory generation for dynamic street scenarios in a Frenét Frame". In: *2010 IEEE International Conference on Robotics and Automation*. 2010 IEEE International Conference on Robotics and Automation. ISSN: 1050-4729. May 2010, pp. 987–993. DOI: 10.1109/ROBOT.2010.5509799. URL: https://ieeexplore.ieee.org/document/5509799 (visited on 11/02/2023).

[53] Jian Yu et al. "A Dynamic and Static Context-Aware Attention Network for Trajectory Prediction". In: *ISPRS International Journal of Geo-Information* 10.5 (May 2021). Number: 5 Publisher: Multidisciplinary Digital Publishing Institute, p. 336. DOI: 10.3390/ijgi10050336. URL: https://www.mdpi.com/2220-9964/10/5/336 (visited on 12/16/2021).

[54] Billy Zheng. *f1tenth_planning*. original-date: 2021-05-26T16:10:43Z. Sept. 6, 2023. URL: https://github.com/f1tenth/f1tenth_planning (visited on 10/07/2023).