CHALMERS
UNIVERSITY OF TECHNOLOGY

INFOTIV
INFOTIV
INFOTIV
INFOTIV

Autonomous Driving via Imitation Learning in a Small-Scale Automotive Platform

A Comparison Between BC, HG-DAgger, and the use of Various Inputs

Master's Thesis in Systems, Control and Mechatronics

Arvid Petersén
Johan Wellander

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024
www.chalmers.se

MASTER'S THESIS 2024

Autonomous Driving via Imitation Learning in a Small-Scale Automotive Platform

Arvid Petersén
Johan Wellander

Department of Electrical Engineering
Division of Signal processing and Biomedical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2024

Autonomous Driving via Imitation Learning in a Small-Scale Automotive Platform
A Comparison Between BC, HG-DAgger, and the use of Various Inputs
ARVID PETERSÉN, JOHAN WELLANDER

© ARVID PETERSÉN, 2024.
© JOHAN WELLANDER, 2024.

Supervisor: Hamid Ebadi, PhD in Computer Science, Infotiv AB
Examiner: Lars Hammarstrand, Department of Electrical Engineering, Chalmers

Master's Thesis 2024
Department of Electrical Engineering
Division of Signal processing and Biomedical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Image of autonomous platform generation 4. [1]. Reprinted with permission.

Typeset in LaTeX

Printed by Chalmers Reproservice Gothenburg, Sweden 2024

Autonomous Driving via Imitation Learning in a Small-Scale Automotive Platform
A Comparison Between BC, HG-DAgger, and the use of Various Inputs
ARVID PETERSÉN, JOHAN WELLANDER

Department of Electrical Engineering
Chalmers University of Technology

Abstract

In recent years, the advancement of autonomous driving (AD) technology has garnered significant interest. Traditionally, AD systems have relied on multiple submodules, each handling specific tasks such as perception, path planning, and vehicle control. However, an emerging alternative is the implementation of end-to-end systems, which directly process sensor input to predict vehicle control.

While both reinforcement learning (RL) and imitation learning (IL) are utilized in end-to-end AD systems, RL often finds its strength in simulated environments, where agents learn through exploration and failure. In contrast, IL, learning from an expert model or human, proves more suitable for real-world applications, requiring substantially less data.

This thesis presents an implementation of IL for achieving autonomous driving on a go-kart platform. Leveraging both behavioral cloning (BC) and Human Gated Dataset Aggregation (HG-DAgger), we compare the impact of using an interactive IL algorithm HG-DAgger compared to BC. Additionally, our research explores the use of different inputs, including color camera, stereo depth camera, IMU, and the position of ORB features. We also detail the development of a comprehensive software pipeline encompassing data collection, data formatting, model training, and go-kart control.

For evaluation, the go-kart was driven around a track for three laps using the trained BC and HG-DAgger models, and assessed based on *number of interventions required per lap, distance without accident, lap time, lap time deviation.* The results from the evaluation indicate an improvement in performance from using HG-DAgger over BC as well as an improvement from using a stereo depth camera or the position of ORB features as supplementary inputs to the color camera and IMU.

Keywords: Imitation Learning, Behavioral Cloning, Human Gated Dataset Aggregation, Autonomous Driving, ROS2.

Acknowledgements

First of all, we would like to extend our greatest thanks to our supervisor at Infotiv, Hamid Ebadi, for his significant engagement in our thesis. His constant support, willingness to discuss various concepts, and guidance throughout this project were invaluable.

We would also like to sincerely thank our academic supervisor, Lars Hammarstrand, for his patience, support, and dedication, providing us with the best opportunities to succeed in our thesis work. His invaluable advice and great ideas were crucial in ensuring the smooth and successful progression of this project.

A special thanks also goes to Infotiv for giving us the opportunity to work on this thesis, for their support, and for financing the project. They equipped us with all the necessary tools and provided support from start to finish, making this project possible.

Furthermore, we would like to extend our gratitude to Gokartcentralen in Kungälv for generously allowing us to drive, test, and validate our go-kart at their track. Without Gokartcentralen and the helpful individuals working there, this project would not have turned out as it did.

List of Acronyms

Below is the list of acronyms that have been used the most throughout this thesis listed in alphabetical order:

AD	Autonomous Driving
AP4	Autonomous Platform 4
BC	Behavioral Cloning
BRIEF	Binary Robust Independent Elementary Features
CAN	Controller Area Network
CNN	Convolutional Neural Network
CONCU	Connectivity Control Unit
DAgger	Dataset Aggregation
DNN	Deep Neural Network
DOF	Degrees of Freedom
ECU	Electronic Control Unit
E/E	Electrical/Electronic
FAST	Features from Accelerated Segment Test
FC	Fully Connected
GPIO	General Purpose Input/Output
HG-DAgger	Human Gated Dataset Aggregation
HLC	High-Level Control Computer
HWI	Hardware Interface Low-Level Computer
IL	Imitation Learning
IMU	Inertial Measurement Unit
ML	Machine Learning
NLL	Negative Log Likelihood
NN	Neural Network
ORB	Oriented FAST and Rotated BRIEF
PDF	Probability Density Function
RL	Reinforcement Learning
RPi	Raspberry Pi
ROS2	Robot Operating System 2
SAD	Sum of Absolute Differences
SPCU	Steering and Propulsion Control Unit
SSD	Sum of Squared Differences

Contents

Abstract **v**

Acknowledgements **vi**

List of Acronyms **vii**

List of Figures **xiii**

List of Tables **xv**

1 Introduction **1**
- 1.1 Background 2
- 1.2 Related work 2
- 1.3 Objective 3
  - 1.3.1 Research questions 3
- 1.4 Limitations 4
  - 1.4.1 Gokartcentralen 4
  - 1.4.2 Localization 4
  - 1.4.3 Maximum speed 4

2 Theory **5**
- 2.1 Software 5
  - 2.1.1 Robot operating system 2 5
    - 2.1.1.1 Communication 5
    - 2.1.1.2 Message types 7
    - 2.1.1.3 Packages 7
  - 2.1.2 Containers 8
    - 2.1.2.1 Docker 8
    - 2.1.2.2 Images 9
- 2.2 Machine learning for autonomous driving 9
  - 2.2.1 Behavioral cloning 9
  - 2.2.2 DAgger 9
  - 2.2.3 HG-DAgger 10
  - 2.2.4 Actor critic policy 11
  - 2.2.5 Convolutional neural network 12

2.2.6 Loss functions 13
2.3 Input data 14
2.3.1 ORB 15
2.3.2 IMU 16
2.3.3 Depth Camera 16
2.4 Hardware 18
2.4.1 Centralized E/E architecture 18

3 System Overview **19**
3.1 Hardware 19
3.1.1 Ninebot go-kart platform 19
3.1.2 Centralized E/E architecture 20
3.1.3 Central master computer & connectivity 20
3.1.4 SPCU 21
3.2 Software design overview 22
3.2.1 ROS 2 23
3.2.2 High-level computing unit 23
3.2.3 Hardware interface computing unit 24

4 Method **25**
4.1 Implementation of new hardware 25
4.1.1 Ensuring a modular system 25
4.1.2 Sensors 26
4.2 Imitation learning 26
4.2.1 Imitation library 26
4.2.2 Observations and actions 27
4.2.3 Behavior cloning network structure 28
4.2.4 HG-DAgger 29
4.2.5 Simulation 30
4.3 Data collection 30
4.3.1 Time synchronization 31
4.3.2 Data formatting 31
4.3.3 Continious data collection (HG-DAgger) 32
4.3.4 Real world data collection 32
4.3.4.1 Training phase 33
4.4 Data pre-processing 34
4.4.1 Color images 34
4.4.2 Depth images 34
4.4.3 ORBs 35

5 Evaluation **37**
5.1 Validation dataset 37
5.2 Evaluation measures 37
5.3 Implementation details 38
5.4 Experiments 38
5.5 Results 38
5.5.1 Repository and dataset 38

5.5.2 Validation dataset 38
5.5.3 Real-world experiments 39

6 Discussion **41**
6.1 Performance of IL models 41
6.2 Software pipeline 41
6.3 Conclusion 42
6.4 Future work 42
6.4.1 Localization 43
6.4.2 Understanding HG-DAgger decrease in performance 43
6.4.3 Explore the possibility to avoid obstacles 44
6.4.4 Address sensor shortcomings 44
6.4.5 Alternative IL algorithms and hybrid IL-RL models 45
6.4.6 Create a realistic simulation 46

Bibliography **47**

List of Figures

2.1 ROS 2 node interfaces: topics, services, and actions. 6
2.2 Visulization of twist mux 8
2.3 Visulization of twist stamper 8
2.4 Concept of convolution and pooling. [29] CC-BY-NC 12
2.5 Fully-connected layer. 13
2.6 IMU with three sensors. 16
2.7 Visualisation of stereo cameras 17

3.1 Ninebot go-kart. [1]. Reprinted with permission. 20
3.2 Overview of Centralized E/E architecture 20
3.3 Visualization of the central master computer setup and CONCU. [1]. Reprinted with permission. 21
3.4 A circuit diagram of the steering and propulsion control [1]. Reprinted with permission. 22
3.5 An overview of the software design of the AP4 [1]. Reprinted with permission. 23
3.6 Communication between HW and ROS. 23
3.7 Overview of Low-level Software implemented on the HWI unit. [1]. Reprinted with permission. 24

4.1 Visualization of modularity on AP4. [1]. Reprinted with permission. . 26
4.2 Structure of Behavioral Cloning Neural Network 28
4.3 Structure of CNN network 29
4.4 HG-DAgger overview 30
4.5 Structure of data collection. 31
4.6 Structure of HG-DAgger. 32
4.7 Gokartcentralen circuit. [44]. Adapted with permission. 33
4.8 View from depth camera in both grayscale and after application of a colormap. 35
4.9 ORB positions overlay on color image 35

5.1 Locations of interactions during validation tests. [44]. Adapted with permission. 40

6.1 Possible environment to train the AP4 for obstacle avoidance. 44
6.2 Field of view for the AP4 45

List of Tables

5.1 Loss for the models evaluated 39
5.2 Results from real-world experiments. 39

1

Introduction

In recent years, there has been a great interest in autonomous driving (AD) technology. AD holds the promise of enhancing safety, efficiency, and cost-effectiveness in transportation. To ensure a safe system, perception providing information about the surroundings is one of the key components. Autonomous vehicles achieve perception by employing diverse algorithms to analyze raw sensory data. Deep Neural Networks (DNNs) are commonly utilized for this purpose, given their demonstrated success in tasks like feature extraction, image recognition, and semantic segmentation [2].

Traditionally, AD systems have been designed based on multiple specific modules, each designed for a distinct purpose such as perception, path planning, and control. When integrated, these modules collectively form a comprehensive system capable of autonomous driving [3]. However, there has been a growing interest in end-to-end systems, where the entire system is based on a single module rather than multiple specialized ones. These end-to-end systems can directly map raw sensory data into control signals and have demonstrated promising performance across both common and rare driving scenarios [4],[5].

Moving away from manually designed systems and onward to end-to-end systems the state-of-the-art of decision making and path planning has changed. The focus has shifted toward training models via Imitation Learning (IL), utilizing large-scale datasets containing expert demonstrations [6].

Additionally, as the research is expanding and AD is becoming more common, the need for testing and validation of the systems is becoming more important. However testing a system on a full-size vehicle can be complicated, expensive, and time-consuming, making it harder for research projects at universities or smaller companies.

In the past few years autonomous race cars have been developed in various scales such as Formula Student Driverless [7] and Indy Autonomous Challenge [8]. This has shown that a reduced scale platform enables rapid development of both hardware and software while it also reduces the cost drastically.

To allow for more accessible and cost-effective testing compared to testing on a full-size vehicle, this thesis will employ an end-to-end approach, utilizing models trained with IL on a small-scale automotive platform. The number of sensors will be kept low exploring what kind of input could optimize the performance of AD.

Additionally, the platform is designed with a highly modular architecture combined with a centralized E/E architecture. This design facilitates seamless implementation and testing of various modules, including different sensors.

1.1 Background

This thesis builds upon the work of E. Magnusson and F. Juthe, who in 2023, in collaboration with Infotiv, developed a small-scale autonomous platform, known as Autonomous Platform 4 (AP4), to explore various autonomous driving (AD) technologies. They augmented a Ninebot go-kart with the necessary hardware and low-level algorithms, enabling future high-level AD functionalities. The implementation featured a centralized E/E architecture, with a central master computing unit responsible for managing control signals to the Electronic Control Units (ECUs), which handles vehicle operations [1].

The upcoming development will therefore concentrate on high-level driving algorithms integrated into the central master computing unit. As mentioned earlier, the approach will employ an end-to-end method utilizing IL, which involves training a DNN based on expert demonstrations. While there are various techniques to train a DNN for AD, such as reinforcement learning (RL), the challenges associated with applying RL in real-world training and the promising advancements in IL research have led to the adoption of the IL approach.

1.2 Related work

The approach in IL is to utilize datasets of expert demonstrations where the system tries to mimic the demonstration instead of having manually designed systems [4]. There are several different methods of IL and in a study from 2023, a benchmark comparison was performed of different IL-based policies for autonomous racing [5]. The IL algorithms tested were:

- Behavioral Cloning (BC)
- Data Aggregation (DAgger)
- Human Gated DAgger (HG-DAgger)
- Expert Intervention Learning (EIL)

but also in combination with the RL algorithm Proximal Policy Optimization (PPO)

- BC + PPO
- DAgger + PPO

- HG-DAgger + PPO
- EIL + PPO

The findings derived from the results demonstrate that interactive IL algorithms notably enhance the performance of autonomous racing. However, HG-DAgger in combination with PPO performed best in a real-world environment, proving the integration of RL and interactive IL surpasses the performance, exhibiting swifter convergence and enhanced stability across diverse scenarios [5].

Autonomous drone racing shares parallels with autonomous car racing, emphasizing the importance of rapid computational processes for path planning, obstacle avoidance, and lap time minimization. A recent study delved into end-to-end planning using feature-based imitation learning for autonomous drone racing [9]. By incorporating the positions of ORB features as additional inputs, they surpassed baseline methods relying solely on RGB images [9].

1.3 Objective

This thesis aims to implement AD on a small-scale automotive platform using IL, specifically HG-DAgger and BC. To achieve this, a complete pipeline is necessary, encompassing data collection, data formatting, training of IL models, and enabling these models to drive the AP4. Additionally, an investigation will be performed regarding whether the performance of AD can be improved by incorporating ORB features or a depth camera in the system. To achieve this the following research questions will be addressed in this thesis.

1.3.1 Research questions

- How can an effective software pipeline be implemented, allowing for data collection, training, and continuous improvement of IL model using HG-DAgger?
- Can the performance of an autonomous vehicle be improved with additional input in the form of a depth camera, alternatively using the same amount of sensors incorporating ORB features derived from the input of an RGB camera? Based on the metrics *number of interventions required per lap, distance without accident, lap time, lap time deviation.*
- Can an improvement in performance be seen driving the AP4 using HG-DAgger compared to driving the AP4 using BC? Based on the metrics *number of interventions required per lap, distance without accident, lap time, lap time deviation*, and can the same effect be observed with the incorporation of ORB-features and Depth image as supplementary input?

1.4 Limitations

During the project, several limitations affected what could be achieved. The main constraints were the limited opportunities to visit Gokartcentralen for testing, training, and evaluating the system, the lack of localization capabilities, and the maximum speed of the go-kart platform being 8 km/h.

1.4.1 Gokartcentralen

Throughout the project, testing, data collection for training IL models, and performance evaluation were conducted at Gokartcentralen in Kungälv. While this provided a consistent environment, for testing, training, and evaluation, it posed limitations due to restricted access. The track could only be used when customers did not use it, and the visits were limited to specific times. In total, three visits to the track were conducted, each visit starting at 11:30 and ending around 18:00 before most customers arrived, resulting in a cumulative time of 19.5 hours.

1.4.2 Localization

There was no localization implemented in the AP4 at the beginning of the project, and due to the limited time of the project, it was not deemed possible to implement both AD and localization. This meant that the approach used for AD could not rely on localization, limiting the approaches, that could be used. Even if IL is a suitable approach to use without localization the IL methods that could be used were also limited without localization.

1.4.3 Maximum speed

Due to a problem with connecting to the go-kart platform via the app provided by Ninebot, it was not possible to change the settings of the go-kart platform. This meant that the max speed of the AP4 throughout the project was 8 km/h instead of the actual max speed of the platform which was 24 km/h.

2

Theory

In this chapter, we describe the fundamental theory underpinning the existing platform and the theoretical framework necessary to comprehend the advancements made in the development of the AP4.

2.1 Software

The software structure of the AP4 is based on Robot Operating System 2 (ROS 2) which handles all the communication between the go-kart and the high-level computer. ROS 2 is one of the key components in this project to create a modular system since it enables communication between smaller software components and allows combinations of software using different languages.

2.1.1 Robot operating system 2

ROS was first developed in 2007 and became widely used within robotics and automatic systems with its modular architecture. ROS 2 is redeveloped from the ground to address the shortcomings of ROS, while keeping much of the structure and application programming interface that was used in ROS [10].

2.1.1.1 Communication

A ROS 2 program is separated into different processes for each runnable program, allowing for a modular system. Each running process is called a node. The nodes can communicate using topics, services, or actions depending on what type of communication is needed [10]. In Figure 2.1, communication between nodes is visualized.

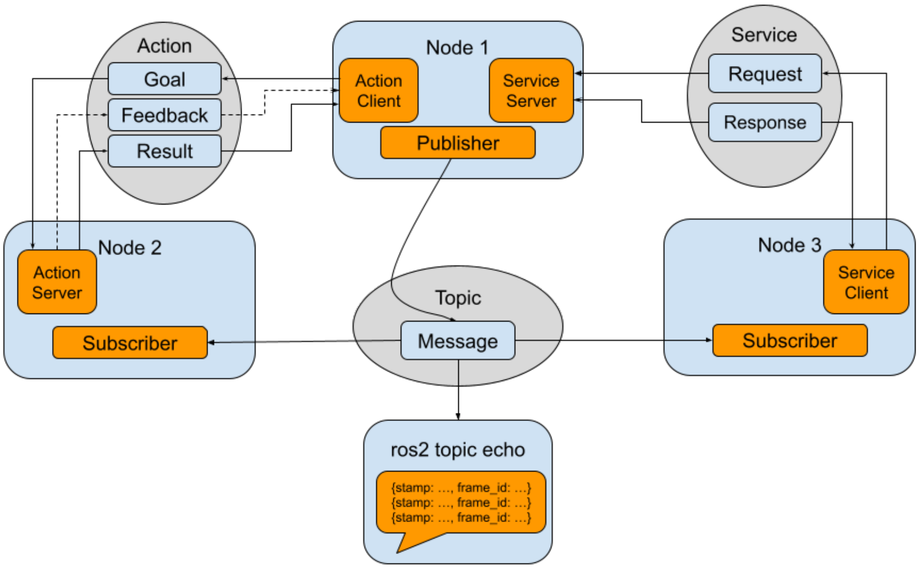

Action
Goal
Feedback
Result
Node 1
Action Client
Service Server
Publisher
Service
Request
Response
Node 2
Action Server
Subscriber
Topic
Message
Node 3
Service Client
Subscriber
ros2 topic echo
{stamp: ..., frame_id: ...}
{stamp: ..., frame_id: ...}
{stamp: ..., frame_id: ...}

Figure 2.1: ROS 2 node interfaces: topics, services, and actions.

Nodes within ROS 2 is an executable program running inside the application performing calculations [11]. In a system, many nodes can be utilized to calculate different tasks such as path planning or localization. All nodes combined will create a graph where they can communicate with each other using topics and services [11].

Topics is according to [10] the most common way for a user to communicate between nodes. Topics are an asynchronous message-passing framework. A node can publish messages to a topic and then that topic can be accessed by any node by subscribing to that topic. This creates a many-to-many communication structure, visualized in Figure 2.1.

Services is a request-response style pattern which means a node can request information from another node which is then required to give a response. This can be useful to ensure a task has been completed a visualisation of this can be seen in Figure 2.1.

Actions allow a node to set a goal for an action within another node. During the execution of the action, feedback is sent back describing the progress towards completing the goal. When the action is completed a result is returned describing the outcome of the action. A visualization of an action can be seen in Figure 2.1.

2.1.1.2 Message types

ROS 2 enables nodes to exchange messages of various types using different formats, offering flexibility when working with different kinds of data. The most common types used in this project can be seen down below.

Twist messages are useful for transmitting control signals or actions between devices. A twist message consists of two vectors: one for linear and one for angular motion, each containing three variables describing the directions in the x, y, and z axes. This message type is commonly employed for sending velocity commands to a system [12].

Stamped messages can contain different kinds of information. The key factor of these messages is that they have a timestamp included when the message was sent. The timestamp can then be used to pair incoming data from different nodes providing an option for synchronization.

Image messages are commonly used while streaming a video feed via ROS 2. The message type takes in an image as input and flattens it to a single vector containing the RGB values of each pixel.

2.1.1.3 Packages

Since ROS 2 is open source and widely used within various research areas, a lot of different packages exist. These packages provide a wide range of functionalities and tools that can be integrated into ROS 2 systems to enhance their capabilities and simplify development processes.

ROS bag is used to record data by subscribing to one or several topics within ROS. The data is stored in a bag file as it is received, facilitating efficient recording and subsequent playback. Additionally, the ROS bag tool includes an option to publish a simulated clock that corresponds to the time when the data was originally collected [13].

Twist mux provides a node within the package that can subscribe to one or several twist messages, multiplexing them based on a priority-based scheme. The node takes in N twist topics, decides what topic has the highest priority, and then sends out the message of the most prioritized one visualized in Figure 2.2 [14].

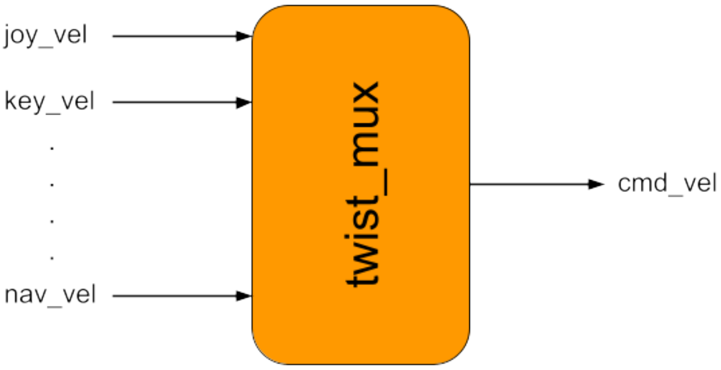


Figure 2.2: Visulization of twist mux

Twist stamper offers two nodes: one for adding a timestamp and another for removing a timestamp from a twist message. The timestamp addition node subscribes to a topic, timestamps the received message, and publishes the stamped message to a new topic. This process is visualized in Figure 2.3 [15].

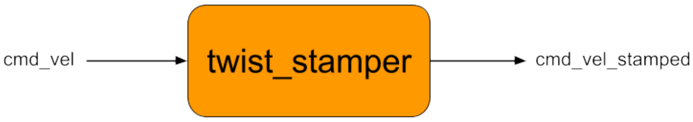


Figure 2.3: Visulization of twist stamper

Message filters is a package that offers various types of filters for ROS 2 messages. The specific filter used in this project is the approximate time synchronizer, which is designed to work with N messages and a predefined slop time. The time synchronizer function of this filter checks if the differences between the timestamps of the received messages lie within the specified slop time. If the messages are within the same period of time, the program continues processing them. Otherwise, the messages are discarded [16].

2.1.2 Containers

Containers are a package of software, packaging dependencies, and code together ensuring the software runs the same on all systems. The containers can then be run on a host operating system (OS) while each container runs its own isolated OS, with the possibility of running a different OS on each of the containers. The resources of the host OS are shared between the containers. With the use of containers creating containerized software, a modular software package can be developed [17].

2.1.2.1 Docker

There are multiple ways to achieve containerized software, one common way is the use of Docker. Docker is an open-source software that allows developers to build, run, and deploy programs inside something called a docker container which makes the programs executable in any environment [17].

2.1.2.2 Images

Images are the foundational component within a containerized architecture. An image can be seen as a blueprint and is a static file containing all the necessary software to construct a container. These files are immutable, meaning that the code cannot be changed after the image is created. Consequently, images can be deployed in any environment of a system [18].

2.2 Machine learning for autonomous driving

Machine learning (ML) is a part of artificial intelligence and computer science that imitates the way humans learn and gradually becomes better over time [19]. Machine learning algorithms used in AD are widely spread and can be used for tasks such as perception and lane-keeping. However, this project will mainly focus on IL implemented as an end-to-end solution for AD. The IL algorithms used in this project are BC and HG-DAgger based on Dataset aggregation (DAgger).

2.2.1 Behavioral cloning

Originating from the lane-keeping algorithm proposed in [20] and later the end-to-end obstacle avoidance algorithm outlined in [21], Behavioral Cloning (BC) has emerged as a fundamental technique in IL. BC, often utilized as a benchmark for more advanced IL models, trains a neural network on a dataset consisting of observations and corresponding expert actions. The primary aim is for the model to replicate the expert's behavior, enabling it to predict actions based on new observations. However, BC faces challenges such as distributional shifts between training and testing data, where the model may struggle to generalize effectively. Additionally, compounding errors pose a limitation, whereby one error can lead to subsequent errors as the model is only trained on previously encountered scenarios.

2.2.2 DAgger

Originally proposed in [22], Dataset Aggregation (DAgger) is an iterative algorithm that trains a deterministic policy. The algorithm is initialized to a policy, denoted as $\hat{\pi}_1$, for example, BC. It then utilizes an expert policy π^* to gather a dataset $\mathcal{D}$. Subsequently, this dataset is used to train a policy $\hat{\pi}_2$, and this procedure is iteratively repeated to reach a satisfactory policy. However, after the first iteration, the dataset collection is performed by both the expert policy π^* and the policy $\hat{\pi}_n$. This is achieved by randomly selecting which policy to use during each rollout of trajectories for T steps, with the probability of using the expert policy π^* denoted as β and the probability of using the trained policy $\hat{\pi}_i$ denoted as $(1 - \beta)$. These trajectories are then collected and aggregated into the original dataset $\mathcal{D}$, with actions determined by the expert policy π^* for all trajectories.

The parameter β is decreased with each iteration, thereby increasing the probability of using the trained policy $\hat{\pi}_i$. Specifically, $\beta_N = \frac{1}{N}\sum_{i=1}^{N}\beta_i \to 0$ as $N \to \infty$. This ensures that the trained policy increasingly relies on its own decisions rather than the expert's. The DAgger algorithm is outlined in Algorithm 1.

Algorithm 1 DAgger [22]

```
Initialize dataset $D \leftarrow \emptyset$
Initialize $\hat{\pi}_1$ to a policy
for $i = 1$ to $N$ do
    Let $\pi_i = \beta_i \pi^* + (1 - \beta_i)\hat{\pi}_i$
    Sample $T$-step trajectories using $\pi_i$
    Obtain dataset $D_i$ of visited states by $\pi_i$ and actions given by the expert
    Aggregate datasets: $D \leftarrow D \cup D_i$
    Train classifier $\hat{\pi}_{i+1}$ on $D$
end for
Return the best $\hat{\pi}_i$ based on validation
```

2.2.3 HG-DAgger

HG-DAgger aims to enhance IL models by enabling training with input from human experts, as originally proposed in [23]. While based on the DAgger algorithm, HG-DAgger introduces key differences that make it more suitable for human-gated training.

Both HG-DAgger and DAgger involve training a model using a combination of expert and novice policy data, with a gating function determining control of the agent. However, their approaches to gating differ significantly. In DAgger, the gating function probabilistically selects actions from either the expert or novice policy, based on a parameter $\beta \in [0, 1]$. Conversely, in HG-DAgger, the novice policy initially controls the agent, and the expert intervenes only when the novice's behavior becomes undesirable.

During HG-DAgger training, demonstrations are collected when the expert takes control of the novice. These demonstrations are then aggregated into the original dataset used to train the novice policy, and the policy is retrained on the aggregated dataset. This process is iteratively repeated until the policy meets a predefined level of performance. The HG-DAgger algorithm is outlined in algorithm 2.

By incorporating human guidance, HG-DAgger offers a robust approach to IL that adapts to expert input, ultimately producing more effective policies.

Algorithm 2 HG-DAgger [23]

```
procedure HG-DAgger ($\pi_H, \pi_{N_1}, \mathcal{D}_{BC}$)
$\mathcal{D} \leftarrow \mathcal{D}_{BC}$
for epoch $i = 1 : K$ do
    for rollout $j = 1 : M$ do
        for timestep $t \in T$ of rollout j do
            if expert has control then
                record expert labels into $\mathcal{D}_j$
            end if
        end for
        $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_j$
    end for
    train $\pi_{N_{i+1}}$ on $\mathcal{D}$
end for
return $\pi_{N_{K+1}}$
```

2.2.4 Actor critic policy

In both RL and IL, actor and critic policies are commonly utilized, each with its own set of strengths and weaknesses. One approach to leverage the strengths of both while mitigating some of their weaknesses is to employ a combined actor-critic policy [24].

An actor policy estimates the gradient of performance directly from the actor parameters. However, this method carries the risk of high variance in gradient estimation. Additionally, the actor policy does not consider previous estimations, updating the gradient independently with each policy parameter update, which may result in disregarding valuable historical information [24].

A critic policy approximates a value function and optimizes the Bellman equation, commonly used in algorithms such as Q-learning. The Bellman equation estimates the value of a state, which can be utilized in algorithms like Q-learning to combine the values of states visited by the agent's actions [25], [26]. Critic policies benefit from the ability to leverage previous information. Nonetheless, they also run the risk of not guaranteeing a near-optimal solution [24].

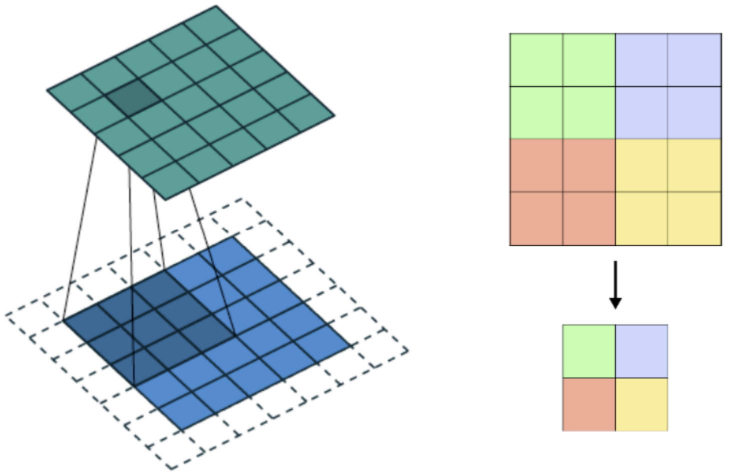
The combined actor-critic policy works by letting the critic learn a value function from the states visited, which is then used for updating the parameters of the actor policy. This approach allows the critic policy to converge to a near-optimal solution and facilitates faster convergence for the actor policy, making it a powerful framework for RL and IL [24].

2.2.5 Convolutional neural network

Convolutional neural networks (CNNs) are commonly used for feature extraction from images, object detection, and pattern recognition. A CNN is built by three types of layers: convolutional layer, pooling layer, and fully connected (FC) layer [27].

The convolutional layer constitutes a fundamental component of CNNs. Assuming the input is a three-dimensional image representing an RGB image with dimensions height, width, and depth, the raw image input undergoes processing using a kernel, also known as a filter. The kernel traverses across the pixels of the image, computing the scalar product of the pixels and the kernel, as illustrated in 2.4a. The interval over which the kernel moves across the image is referred to as the stride, determining both the overlap between the kernel and the image and the size of the subsequent layer [28]. A lower stride value yields a larger output size, while a higher stride value produces a smaller output size. Consequently, neurons in the subsequent layer only receive information from the corresponding part of the input image, resulting in a significant reduction in the number of connections between neurons [28].

The pooling layer aims to down-sample the dimensions further, reducing the complexity and number of parameters within the activation function [28][27]. The most common pooling function is called max-pooling which returns the maximum value within the specific sub-region and is visualized in Figure 2.4 (b).



(a) Convolution **(b)** Pooling

Figure 2.4: Concept of convolution and pooling. [29] CC-BY-NC

FC layers in a CNN resemble traditional artificial neural networks, where all nodes in the current layer are connected to every node in both the previous and next layers, illustrated in Figure 2.5 [28]. This component of a CNN is dedicated to performing classification tasks. However, FC layers are computationally expensive, and efforts are often made to reduce the number of connections and nodes [28].

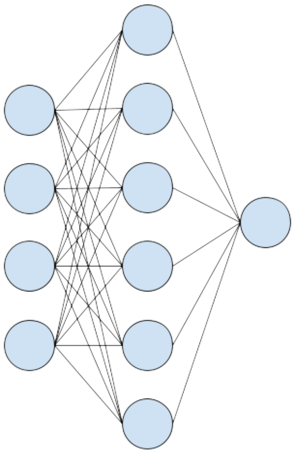


Figure 2.5: Fully-connected layer.

2.2.6 Loss functions

In ML, a loss function, also referred to as a cost function or objective function, serves as a metric to evaluate how effectively an ML model's predictions align with the actual values. Essentially, it quantifies the disparity between the predicted output and the true output for a given set of input data. Loss functions are pivotal in training ML models, as they facilitate the optimization process by furnishing a signal for adjusting the model's parameters to minimize error.

There are several ways of calculating the loss while training a neural network. The methods used in this thesis are called L2 loss, cross-entropy loss, and negative log-likelihood (NLL) loss.

The L2 loss function serves to minimize the error by calculating the sum of the squared differences between the true values y and the predicted values $\hat{y}$ [30]. The L2 loss is calculated as:

$$\text{L2LossFunction } = \sum_{i=1}^{n} (y - \hat{y})^2 \,. \tag{2.1}$$

The NLL loss function aims to maximize the likelihood of predicting the correct labels by minimizing the negative log-likelihood of the predicted probabilities. It quantifies the error between the predicted probabilities and the true labels of the target. The NLL loss function is computed differently depending on whether the task is a continuous or a discrete classification task. In the discrete domain, the minimization of NLL loss is computed as:

$$NLL(y, \hat{y}) = -\sum_{i=1}^{n} \left(y_i \log \hat{y}_{\theta,i} + (1 - y_i) \log \left(1 - \hat{y}_{\theta,i}\right) \right), \tag{2.2}$$

where y represents the true target labels taking the value 0 or 1 and $\hat{y}$ represents the predicted probability distribution for each label [31].

In the continuous case, this approach is not possible, instead, a probability density function (PDF) is utilized. The PDF is a function *f(x)* over the sample space S, where $S \subseteq \mathbb{R}$, and represents the probability of a random continuous variable x being within a certain interval. This probability can be obtained as:

$$\int_a^b f(x), \mathrm{d}x = \mathrm{P}[a < X \leq b], \tag{2.3}$$

where a and b are real numbers and the PDF has to fulfill the following constraints:

- $f(x) \geq 0$ for all $x \in \mathbb{R}$
- $\int_{-\infty}^{\infty} f(x)\mathrm{d}x = 1$.

Meaning that the PDF is always positive for all x in S and that the integral over S will always be equal to one [32].

Letting the sample vector S represent all independently drawn samples from the distribution of the PDF denoted as P, giving $S = \left(z^0, z^1, \ldots, z^{|\mathcal{S}|-1}\right)$. So that $z^i = \left(z_0^i, z_1^i, \ldots, z_{l-1}^i\right)$ where l is the number of random variables available. The sample vector is then used to find a probability distribution $\hat{P}$ as an approximation of the true probability distribution P. The NLL loss is then defined as:

$$NLL(S \mid \hat{P}) = -\sum_{i=0}^{|\mathcal{S}|-1} \log\left(\hat{P}\left(z^i\right)\right). \tag{2.4}$$

Taking the negative logarithm of the predicted probability penalizes the model more heavily for low-confidence incorrect predictions, as the negative log function increases sharply as the predicted probability approaches zero. Conversely, it rewards the model for confident and accurate predictions, as higher probabilities result in lower loss values [33].

The cross-entropy loss, also called log loss is very similar to NLL where it is also used to measure the discrepancy between predicted probabilities and true labels. The definition is the same as in NLL; however, the implementation of cross-entropy loss applies a softmax activation and a log transformation, while NLL does not [34].

2.3 Input data

Training a DNN using IL necessitates annotated datasets comprising inputs from sensors coupled with actions executed by the expert driver. The input data for the AP4 encompasses an RGB camera, depth camera, inertial measurement unit (IMU), and an additional input providing the location of key points extracted from the RGB camera feed, known as ORB features.

2.3.1 ORB

Oriented FAST and Rotated BRIEF (ORB) is a 2D object recognition system within the field of computer science. ORB was first introduced in 2011 by Rubee et al. and developed in the OpenCV lab as an alternative to traditional methods such as SWIFT or SURF [35]. As the name indicates, ORB is built by two different methods: FAST and BRIEF, but with some improvements for ORB.

FAST is the method used to enable keypoint detection in real-time systems matching visual features. FAST takes one parameter, an intensity threshold between a center pixel and those in a Bresenham circle around it [36]. Consider a pixel p with a circle of 16 pixels surrounding it. If p should be considered as a keypoint, there has to exist at least N, in this case, 12, pixels surrounding it with a greater difference in intensity than the set threshold [37]. To make the algorithm fast, the comparison is only done with four of the pixels. If at least three of the four pixels are above or below the threshold, p is considered as a keypoint. However, FAST does not provide a certainty that the detected points are indeed a corner. To address this, ORB includes a Harris corner measure to select keypoints [35]. The Harris corner measure evaluates the FAST keypoints based on the local intensity gradients in the image. Keypoints with higher Harris corner scores are more likely to be true corners and are then selected as a true point of interest [35]. Furthermore, FAST does not produce multi-scale features. To overcome this, a scale pyramid is created from the original image, FAST features are calculated, and the resulting keypoints are filtered using the Harris corner measure [38]. This method effectively produces keypoints at varying scales, enabling feature detection across different levels of detail within the image [35].

BRIEF is the second part of ORB which stands for Binary Robust Independent Elementary Features. BRIEF is a binary feature descriptor that generates a string for each keypoint in an image. The way it operates within ORB is by comparing intensity values of pixel pairs surrounding each keypoint and encoding these comparisons into binary strings [35].

To enhance the robustness and effectiveness of BRIEF, a technique called steered BRIEF is introduced [35]. This approach involves adapting BRIEF descriptors based on the orientation of image patches surrounding each keypoint. By leveraging the patch orientation θ and corresponding rotation matrices, a steered version of the BRIEF operator is constructed. A $2 \cdot n$ matrix S is introduced for each feature set of n binary tests at a location (x_n, y_n), such that

$$\mathbf{S} = \begin{pmatrix} \mathbf{x}_1, \ldots, \mathbf{x}_n \\ \mathbf{y}_1, \ldots, \mathbf{y}_n \end{pmatrix}. \tag{2.5}$$

To construct the steered BRIEF descriptor, the original descriptor matrix S is multiplied by the rotation matrix R_θ, resulting in $\mathbf{S}_\theta = \mathbf{R}_\theta \mathbf{S}$. This transformation allows BRIEF descriptors to be adjusted based on the orientation of keypoints, resulting in descriptors that are more invariant to image rotations and transformations [35].

For the intensity p at position x the steered BRIEF descriptor is:

$$g_n(\mathbf{p}, \theta) := f_n(\mathbf{p}) \mid (\mathbf{x}_i, \mathbf{y}_i) \in \mathbf{S}_\theta. \tag{2.6}$$

As long as the orientation of the keypoints is consistent, the correct set of points from S_θ will be used to compute the descriptors [35].

Combining FAST and BRIEF with additional adjustments enables real-time calculation of keypoints that are invariant to rotations within the image.

2.3.2 IMU

IMU stands for inertial measurement unit, which provides information about velocity, acceleration, and orientation. Earlier models of IMUs consisted of two types of sensors: accelerometers and gyroscopes. Accelerometers measure inertial acceleration, while gyroscopes measure angular rotation. Newer versions of IMUs often include a magnetometer, which measures the magnetic bearing direction. This type of sensor enhances the readings by decreasing the drift issue from the gyroscope. Most often, each sensor has three degrees of freedom (DOF) in the x, y, and z axes. Combining the sensors results in an IMU having up to six or nine DOF, depending on the number of sensors utilized [39].

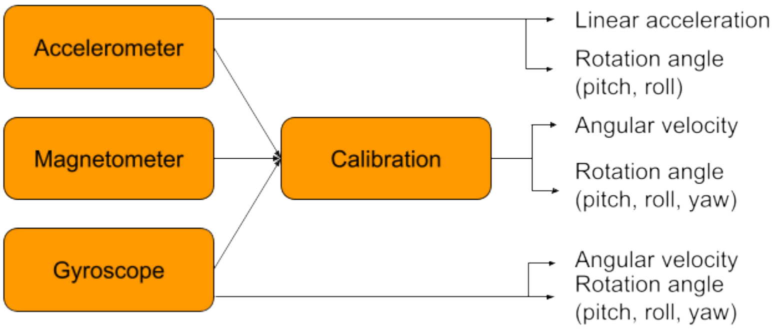

Accelerometer
Magnetometer
Gyroscope
Calibration
Linear acceleration
Rotation angle (pitch, roll)
Angular velocity
Rotation angle (pitch, roll, yaw)
Angular velocity
Rotation angle (pitch, roll, yaw)

Figure 2.6: IMU with three sensors.

2.3.3 Depth Camera

A stereo-depth camera allows for the estimation of depth and the creation of an image with each pixel-value representing the distance from the camera to the object [40]. A stereo-depth camera works by utilizing two lenses sitting at a distance from each other. In Figure 2.7, a visualization of a stereo-depth camera setup can be seen.

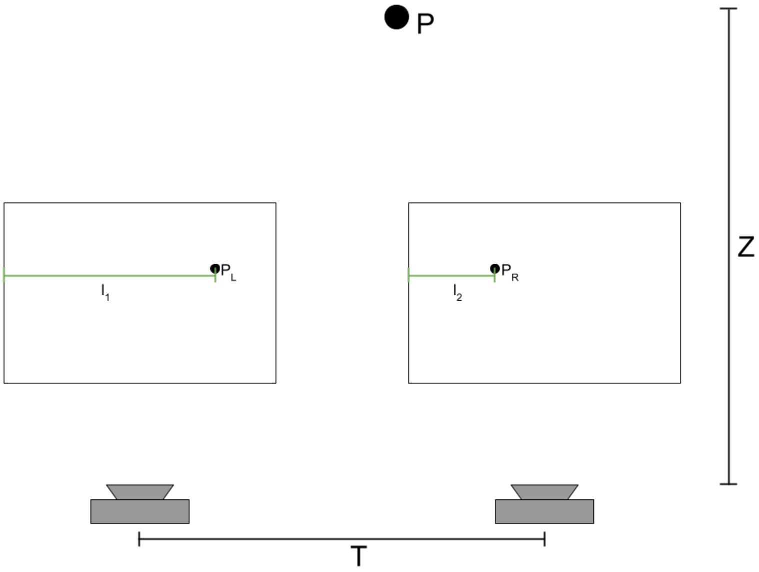


Figure 2.7: Visualisation of stereo cameras

Let,

P be a point whose distance needs to be determined.
Z be the distance from the camera lenses to the point in the z-direction.
T be the distance between the two camera lenses.
P_L and P_R be the points as seen in the left and right images captured by the cameras, respectively.
l_1 and l_2 be the distance from the left side of the image to the point in the captured frame in the left and right images, respectively.

The distance from the camera to point P is calculated from the disparity between the two points P_L and P_R and the distance between the lenses T, along with the focal length of the camera f and the physical size of the pixel in the camera sensor d. The disparity, defined as $D = l_1 - l_2$, allows the calculation of the distance Z as follows:

$$Z = \frac{f}{d} \times \frac{T}{D}. \tag{2.7}$$

To calculate the depth from matching points in the left and right images, the corresponding points between the images must be determined. This is achieved by examining a point in one image and searching for the same point in the other image within a certain neighborhood. Matching patches of pixels can be determined using methods such as Sum of Squared Differences (SSD);

$$\text{SSD}(\text{win}_L, \text{win}_R) = \sum_x \sum_y \left(I_{\text{win}_L}(x, y) - I_{\text{win}_R}(x, y)\right)^2, \tag{2.8}$$

and Sum of Absolute Differences (SAD);

$$\mathrm{SAD}(\mathrm{win}_L, \mathrm{win}_R) = \sum_x \sum_y \left| I_{\mathrm{win}_L}(x, y) - I_{\mathrm{win}_R}(x, y) \right| . \tag{2.9}$$

2.4 Hardware

The small-scaled autonomous platform developed during this thesis is based on a go-kart from Ninebot. The hardware on the go-kart was implemented by E. Magnusson and F. Juthe in last year's master's thesis in collaboration with Infotiv [1]. The design features a centralized E/E architecture, with a central master computer responsible for executing all algorithms and processing calculations.

2.4.1 Centralized E/E architecture

In a centralized E/E architecture, ECUs function primarily as zone controllers or edge nodes. Their main role is to interface with sensors and actuators, possessing limited computing capabilities. Conversely, all complex functionalities, including the execution of driving algorithms and other computational tasks, are handled by a high-performance computing unit. This unit processes the necessary data and sends control signals back to the ECUs, directing vehicle operation [1].

3

System Overview

In this chapter, we provide an overview of the AP4 as constructed in the previous year's master thesis [1], encompassing both its software and hardware components. The software is based on ROS 2 while the hardware is based on a centralized E/E architecture allowing for control of the AP4 through ROS 2.

3.1 Hardware

The AP4 utilizes a go-kart platform from Ninebot as its base. To allow for autonomous driving, additional hardware has been installed, enabling control over the go-kart's steering and throttle. The system comprises a high-level control computer (HLC) for processing and decision-making, a hardware interface low-level computer (HWI) for direct device control, a CAN network for communication among components, and various sensors to gather environmental data.

3.1.1 Ninebot go-kart platform

The AP4 is based on a go-kart kit by Ninebot. The go-kart kit consists of an electric segway which is used for propulsion of the go-kart. The segway is combined with a go-kart kit by Ninebot, allowing for steering and propulsion by a driver sitting in the driver's seat. The go-kart kit can be seen in Figure 3.1. A centralized E/E Architecture has been implemented on the go-kart kit to allow for control of the go-kart kit's propulsion and steering.

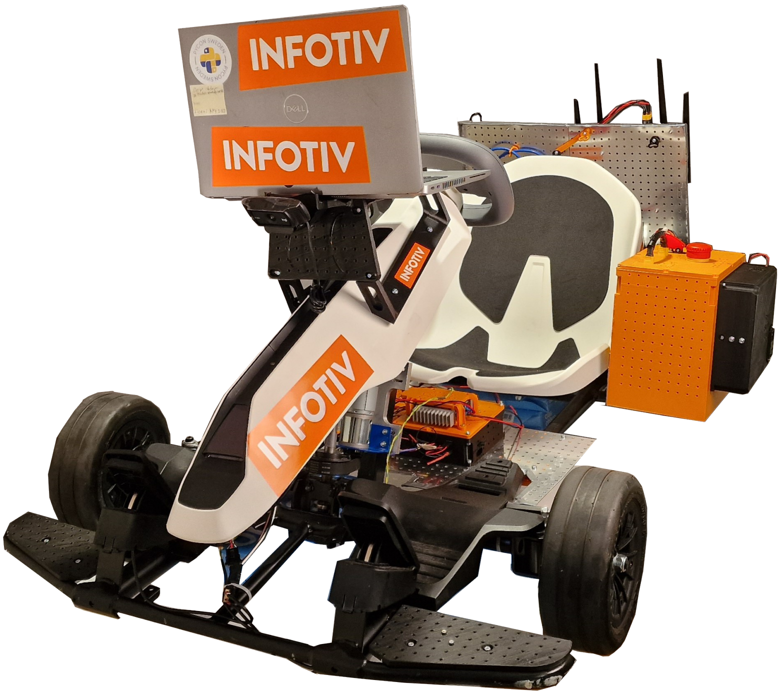


Figure 3.1: Ninebot go-kart. [1]. Reprinted with permission.

3.1.2 Centralized E/E architecture

The hardware of the AP4 is based on a centralized E/E architecture with one central master computer connected to multiple Electronic control units (ECUs), with each ECU responsible for its own area. The currently implemented ECUs can be seen in Figure 3.2 along with the connection between the Central master computer and the ECUs.

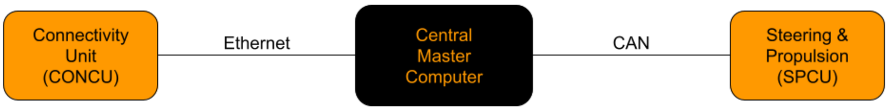


Figure 3.2: Overview of Centralized E/E architecture

3.1.3 Central master computer & connectivity

The central master computer comprises two distinct hardware components linked by the Connectivity Control Unit (CONCU). This split is necessitated by the lack of GPIO connections in most off-the-shelf laptops. To address this limitation, a Raspberry Pi 4b serves as the Hardware Interface Computer (HWI), used for connections to the other ECUs via CAN, while a more powerful High-Level Control Computer (HLC) assumes responsibility for higher-level control tasks. The HLC, a laptop running Ubuntu 22.04, orchestrates the AP4's operations. The CONCU, which consists of a router enabling both Ethernet and WiFi communication between the hardware components, facilitates seamless communication between the HLC and HWI. Figure 3.3 provides a visualization of the central master computer setup and CONCU, illustrating the hardware components and connectivity infrastructure.

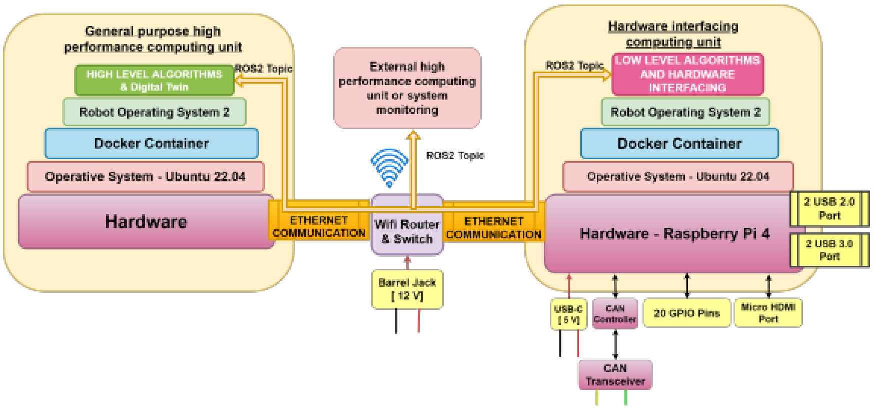

General purpose high performance computing unit
HIGH LEVEL ALGORITHMS & Digital Twin
ROS2 Topic
Robot Operating System 2
Docker Container
Operative System - Ubuntu 22.04
Hardware
External high performance computing unit or system monitoring
ROS2 Topic
ETHERNET COMMUNICATION
Wifi Router & Switch
ETHERNET COMMUNICATION
Barrel Jack [12 V]
Hardware interfacing computing unit
ROS2 Topic
LOW LEVEL ALGORITHMS AND HARDWARE INTERFACING
Robot Operating System 2
Docker Container
Operative System - Ubuntu 22.04
Hardware - Raspberry Pi 4
2 USB 2.0 Port
2 USB 3.0 Port
USB-C [5 V]
CAN Controller
20 GPIO Pins
Micro HDMI Port
CAN Transceiver

Figure 3.3: Visualization of the central master computer setup and CONCU. [1]. Reprinted with permission.

3.1.4 SPCU

The steering and propulsion control unit (SPCU) is responsible for converting steering and propulsion signals to physical control of the AP4. For propulsion, this is achieved by regulating the signals sent to the Ninebot Segway, bypassing those from the brake and throttle pedal, and directly sending its signal. For steering control, a DC motor is connected to the steering rod, and the SPCU converts steering signals into signals that control the DC motor. However, this means that it is not possible to steer the AP4 via the steering wheel while the system is turned on. Instead, steering signals are sent from the SPCU to the DC motor. A circuit diagram illustrating the connections and components between the propulsion, steering, and SPCU can be seen in Figure 3.4.

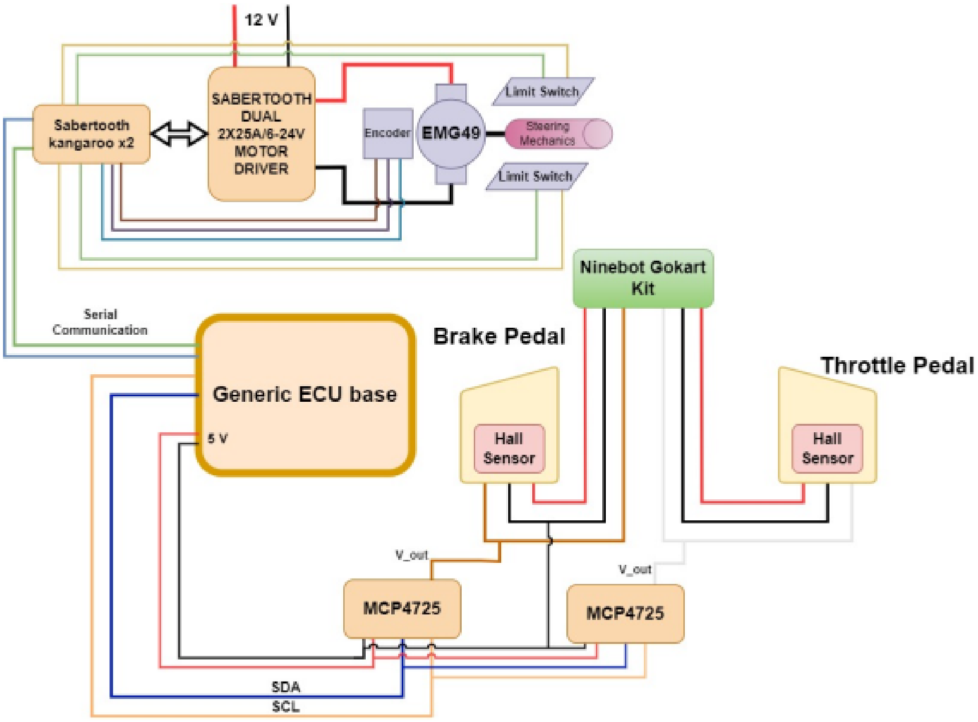

12 V
SABERTOOTH DUAL 2X25A/6-24V MOTOR DRIVER
Sabertooth kangaroo x2
Encoder
EMG49
Limit Switch
Steering Mechanics
Limit Switch
Ninebot Gokart Kit
Serial Communication
Generic ECU base
5 V
Brake Pedal
Hall Sensor
Throttle Pedal
Hall Sensor
V_out
MCP4725
V_out
MCP4725
SDA
SCL

Figure 3.4: A circuit diagram of the steering and propulsion control [1]. Reprinted with permission.

3.2 Software design overview

The implementation of an autonomous system requires multiple software components working together. Where the need for the software components is different, with some hardware interfacing software being time-critical while a lot of the other software not being time-critical. The software is therefore split into three software components running on different machines, the components being a hardware interfacing machine, a low-level hardware interfacing component, and a high-level control software component. This design can be seen in Figure 3.5 and further explained in this section.

To allow for modular software and communication between the different software components ROS 2 is used. As ROS 2 is well documented and has a large active community, the use of ROS 2 allows this to be leveraged to allow for new development on the AP4 to be done with relative ease. ROS 2 is structured by nodes and topics, enabling communication between different hardware components where the data can be published and subscribed to.

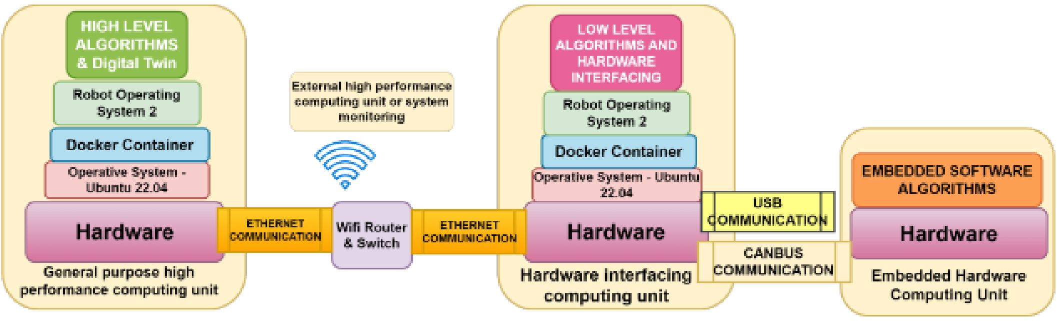

HIGH LEVEL ALGORITHMS & Digital Twin
Robot Operating System 2
Docker Container
Operative System - Ubuntu 22.04
Hardware
General purpose high performance computing unit
External high performance computing unit or system monitoring
ETHERNET COMMUNICATION
Wifi Router & Switch
ETHERNET COMMUNICATION
LOW LEVEL ALGORITHMS AND HARDWARE INTERFACING
Robot Operating System 2
Docker Container
Operative System - Ubuntu 22.04
Hardware
Hardware interfacing computing unit
USB COMMUNICATION
CANBUS COMMUNICATION
EMBEDDED SOFTWARE ALGORITHMS
Hardware
Embedded Hardware Computing Unit

Figure 3.5: An overview of the software design of the AP4 [1]. Reprinted with permission.

3.2.1 ROS 2

In AP4 all communication between the high- and low-level computers is handled by ROS 2. The inputs from the sensors, the actions made to control the car, and the decisions made by the model are published to the network allowing the HW components to work together and exchange data. The overall structure of the communication handled by ROS 2 is illustrated in Figure 3.6.

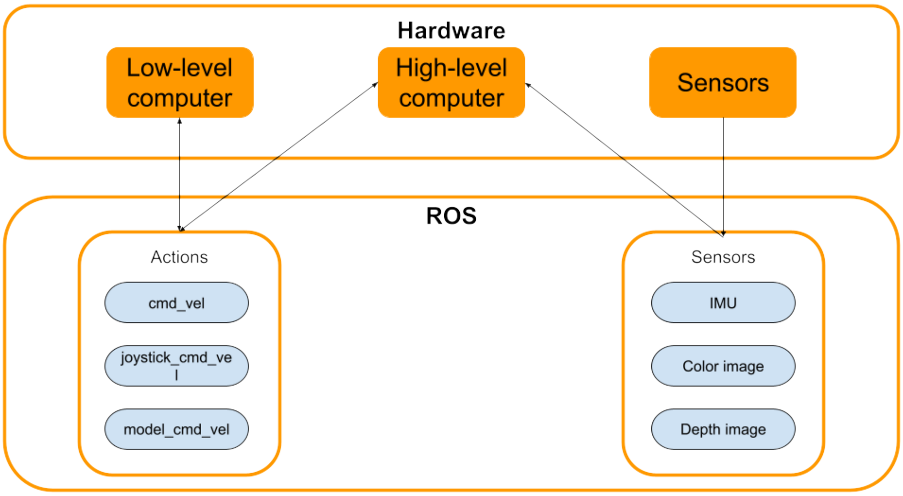

Hardware
Low-level computer
High-level computer
Sensors
ROS
Actions
cmd_vel
joystick_cmd_vel
model_cmd_vel
Sensors
IMU
Color image
Depth image

Figure 3.6: Communication between HW and ROS.

3.2.2 High-level computing unit

The Software of the High-Level Computing Unit hosts the high-level software that is responsible for the collection of data, the camera, and the autonomous control of the AP4. The high-level software is based on ROS 2 allowing for the software to be modular and to use standardized packages within ROS 2. The high-level software allows for the reading of sensor data and implementation of autonomous drive which is the aim of this project further described in Chapter 4.

3.2.3 Hardware interface computing unit

The Hardware Interface Computing Unit hosts the low-level software which serves as the intermediary between the High-Level Software, and the Embedded Software, tailored specifically for the hardware of the AP4 and executed on the Raspberry Pi (RPi) mounted on the AP4. An overview of the low-level software architecture can be observed in Figure 3.7.

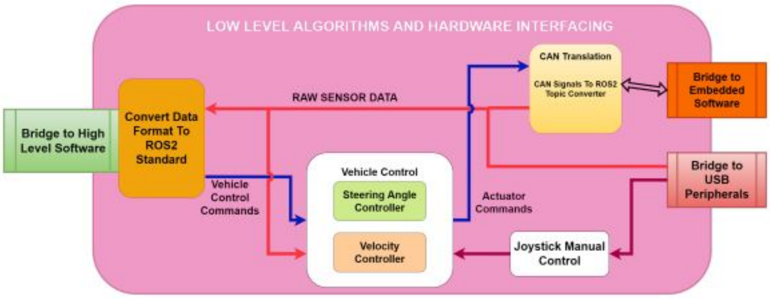

LOW LEVEL ALGORITHMS AND HARDWARE INTERFACING
CAN Translation
CAN Signals To ROS2 Topic Converter
Bridge to Embedded Software
RAW SENSOR DATA
Bridge to High Level Software
Convert Data Format To ROS2 Standard
Bridge to USB Peripherals
Vehicle Control
Steering Angle Controller
Velocity Controller
Vehicle Control Commands
Actuator Commands
Joystick Manual Control

Figure 3.7: Overview of Low-level Software implemented on the HWI unit. [1]. Reprinted with permission.

The low-level software includes a module, designed to facilitate communication between the HLC and the ECUs on the AP4 using ROS 2. This module comprises two primary components: a standard ROS 2 package and a custom-developed package.

The standard ROS 2 package initializes two nodes—one for receiving and another for sending CAN messages over the CAN bus. The custom package manages the exchange of data between ROS 2 topics and the CAN bus. It decomposes incoming CAN frames into separate topics, which are then published to ROS 2. Conversely, when a message is received from ROS 2, it is encoded and transmitted over the CAN bus to communicate with the ECUs.

4

Method

In this chapter, the additions made to the AP4 during this project to achieve autonomous driving using BC and HG-DAgger are covered. The additions consist of the implementation of new hardware, recording, synchronizing data, data formatting, as well as the implementation of the IL pipeline. These additions together with the existing system covered in Chapter 3 form the foundations for the implementation of AD.

4.1 Implementation of new hardware

To reach the goal of implementing AD in AP4, there was a need for additional hardware, especially sensors. When implementing new hardware on the go-kart the standard implementation of hardware established in [1] was followed. This ensures that any further development of the platform down the line can continue working on the platform without the need to understand multiple different frameworks and standards.

4.1.1 Ensuring a modular system

The standard for ensuring a modular hardware design established during last year's master thesis project is visualized in Figure 4.1. A modular design based on an aluminum sheet with a rectangular pattern with holes is used. The dimensions of the aluminum sheet are 500mm long, a width of 250mm, and a thickness of 1.5 mm. The hole pattern has holes of 4 mm in diameter and is evenly distributed in a rectangular pattern of 15 mm between each hole. This hole pattern is set as a standard for the complete AP4 [1].

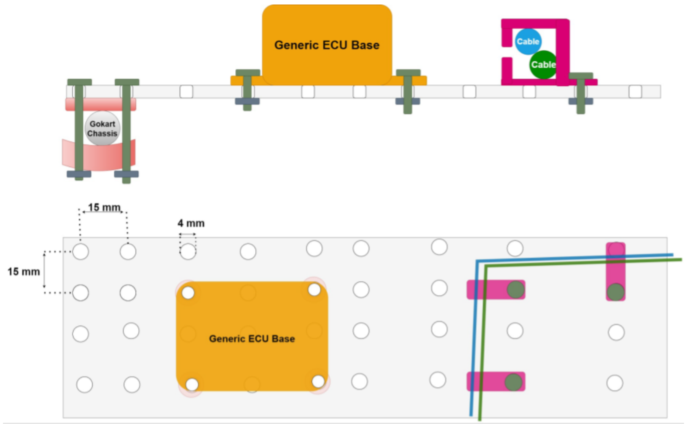


Figure 4.1: Visualization of modularity on AP4. [1]. Reprinted with permission.

The standard that was established in the previous master thesis project was followed during this project as well. This was done to avoid any unnecessary confusion for upcoming projects on the AP4 down the line.

There is also an established standard for ECU casings, with holes aligning with the aluminum sheets. When adding new sensors or components, this design is to be used if there is a need for an ECU as this will both save time, avoid the need to design a new ECU casing, and have a clear standard for future development.

4.1.2 Sensors

A stereoscopic camera of the model OAK-D from Luxonis was implemented on the AP4 [41]. The OAK-D has a built-in 9-axis IMU, an RGB camera as well as two grayscale cameras, which can be used to get depth images. The OAK-D camera was implemented via USB directly into the HLC Computer, while an open-source ROS 2 wrapper and camera drivers provided by Luxonis were used for the camera.

4.2 Imitation learning

To make the AP4 autonomous IL was used. The implementation of the IL algorithm was done by learning from human drivers, driving around a track. Learning to match the actions taken by the driver to the observations recorded by the sensors of the AP4.

4.2.1 Imitation library

The implementation of IL leveraged the imitation learning library Imitation [42], which is constructed on top of the RL library Stable-Baselines3 [43]. This choice was made due to the robustness and versatility offered by the Imitation library. Utilizing this framework enabled the development of IL models in a modular fashion, allowing for easy customization and adaptation to various tasks and environments.

One of the key advantages of the Imitation library is its flexibility. It provides a range of pre-implemented algorithms and functionalities tailored specifically for IL tasks. Moreover, the library seamlessly integrates with Stable-Baselines3, enabling future improvements with its extensive set of RL algorithms and tools.

In addition to its modular design, the Imitation library offers comprehensive documentation and support, making it straightforward to implement and experiment with different IL methodologies. This facilitated our development process and allowed us to focus on the specifics of our research objectives without being encumbered by the intricacies of low-level implementation details.

4.2.2 Observations and actions

The observations were structured in a dictionary format which allows for changing what input data is used. The inputs that are used for observations are color image, depth image, ORB locations, and IMU. However, not all inputs are used at all times, instead, three different combinations of inputs are used. The combinations are;

- Color image and IMU
- Color image, depth image, and IMU
- Color image, ORB locations, and IMU

The ORB locations are structured in a matrix the size of the image inputs, with ones at the locations of ORBs and zeros otherwise. The IMU data consisting of acceleration, a and angular velocity, ω in x, y, and z direction, is structured as follows:

$$\begin{bmatrix} a_x & a_y & a_z & \omega_x & \omega_y & \omega_z \end{bmatrix},$$

while the image data consists of RGB images of size 120 x 160.

The three observation spaces in turn look as follows:

- color image: $(120 \times 160 \times 3)$, IMU: (1×6)
- color image: $(120 \times 160 \times 3)$, ORB positions: (120×160), IMU: (1×6)
- color image: $(120 \times 160 \times 3)$, depth image: $(120 \times 160 \times 3)$, IMU: (1×6)

The action space of the system consists of throttle and steering angle resulting in an action space of $\mathbf{u} = [\bar{v}, \bar{\Omega}]$, with $\bar{v} \in [-1, 1]$ and $\bar{\Omega} \in [-1, 1]$. The actions are defined within $[-1, 1]$ to maintain a normalized action space. These actions are then scaled to match the inputs used in the AP4.

4.2.3 Behavior cloning network structure

The IL in this project was implemented using BC as well as HG-DAgger, which is also based on BC. To train the BC model, demonstrations in the form of observations and actions were captured from driving the AP4 around the go-kart track at Gokartcentralen in Kungälv. The captured demonstrations were then fed into the BC-network, seen in Figure 4.2, where features are first extracted and concatenated into a feature vector. The features extracted by the feature extractor were then used as the input for the actor-critic part of the network. The actor trains on predicting the correct actions, while the critic gives feedback on the predicted actions to the actor part of the network.

To allow for the different types of inputs and to allow for changing the inputs used, the neural network used in the BC model was set up as in Figure 4.2.

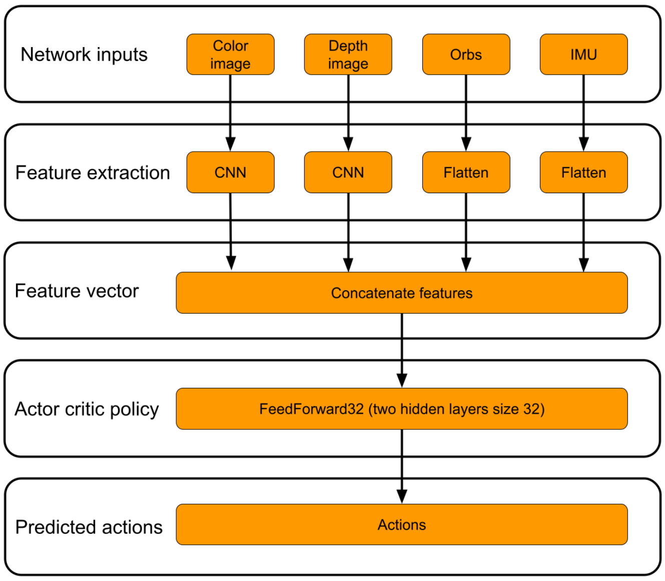

Network inputs
Color image
Depth image
Orbs
IMU
Feature extraction
CNN
CNN
Flatten
Flatten
Feature vector
Concatenate features
Actor critic policy
FeedForward32 (two hidden layers size 32)
Predicted actions
Actions

Figure 4.2: Structure of Behavioral Cloning Neural Network

The inputs are structured in a dictionary format and then using a combined extractor the features from each of the inputs are extracted and then concatenated into one feature vector. This allows for changing which inputs are used without changing the structure of the network. In the case of ORBs and IMU, the data is flattened directly, while in the case that the input is an image, a CNN is used for the feature extraction resulting in a flattened vector. The structure of the CNN used for feature extraction can be seen in Figure 4.3.

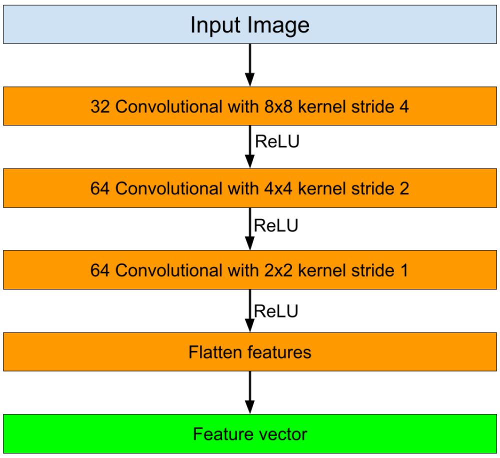

Input Image
32 Convolutional with 8x8 kernel stride 4
ReLU
64 Convolutional with 4x4 kernel stride 2
ReLU
64 Convolutional with 2x2 kernel stride 1
ReLU
Flatten features
Feature vector

Figure 4.3: Structure of CNN network

The concatenated feature vector is then fed into an actor-critic policy that trains to determine the correct action from the feature vector. Since the input to the actor-critic policy consists of already extracted features, the policy includes two fully connected layers of size 32 for both the actor and the critic.

4.2.4 HG-DAgger

To address the shortcoming of BC generalizing poorly, an HG-DAgger model was implemented. The implemented HG-DAgger algorithm can be seen in Figure 4.4. The HG-DAgger model is first initialized using BC trained on a dataset of demonstrations. To make a more robust model the AP4 is driven by the model until an expert driver deems the model to behave undesirably and the expert driver takes control. When the expert driver has taken control the actions of the expert driver along with the observations are recorded and aggregated into the initial demonstration dataset. When finished driving, the model is retrained on the aggregated dataset and is then repeated until the model behaves desirably. This creates a more robust model that can adapt to new environments by learning from an expert's actions.

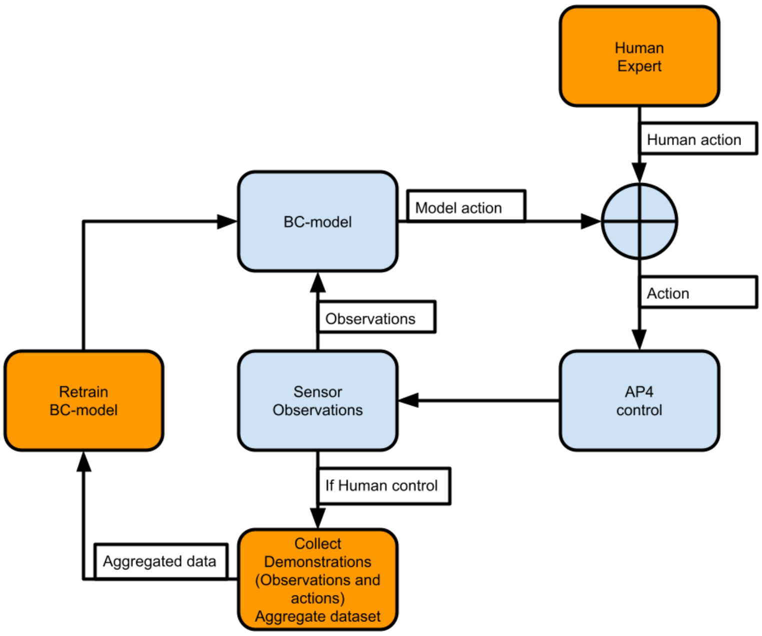

Human Expert
Human action
BC-model
Model action
Action
Observations
Retrain BC-model
Sensor Observations
AP4 control
If Human control
Aggregated data
Collect Demonstrations (Observations and actions) Aggregate dataset

Figure 4.4: HG-DAgger overview

4.2.5 Simulation

A simulation was set up to allow for the evaluation and development of the IL algorithms independently from the whole system. The simulation was done in Donkey Car Simulator allowing demonstrations to be collected from driving a car on a track in simulation using an Xbox controller. The demonstrations collected in the simulation consisted of synced actions and observations for the complete system, with the actions being throttle and steering angle, while the observations consisted of only RGB images, unlike the real-world data.

The demonstrations were then used to train a BC model, which could be tested in the simulation allowing for evaluation and tweaking of the BC model.

4.3 Data collection

To be able to train a neural network to learn the behavior of an expert driver, data collection was a vital part. All the inputs from sensors and the actions made by the driver to control the car had to be saved and synchronized to match each other in different situations. To increase effectiveness, the method to start up the vital parts of the data collection was automated by a single start script where the structure can be seen in Figure 4.5.

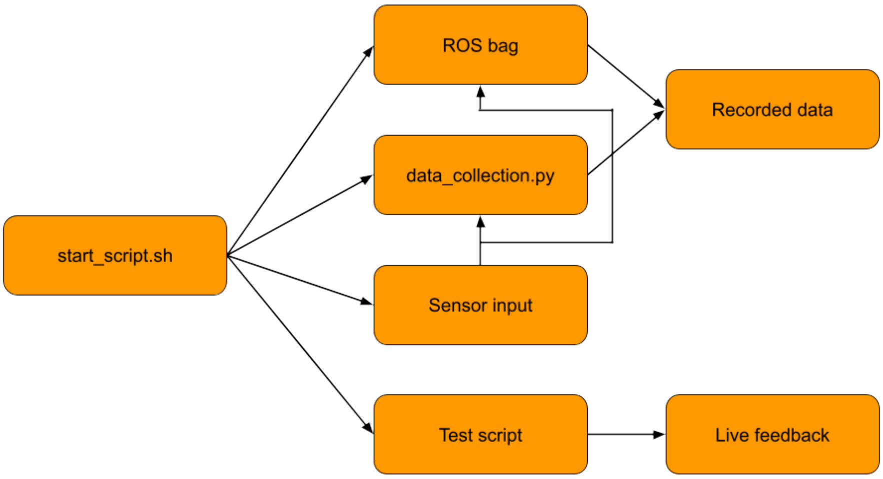


Figure 4.5: Structure of data collection.

The data collection process involves synchronizing sensor observations with the actions performed by the expert driver, utilizing message timestamps. Data is continuously saved to a pickle file after capturing 3000 pictures ensuring that the high-level computer does not run out of memory. Simultaneously all raw data published to ROS 2 is captured by ROSbag as a backup. Additionally, a test script runs in the background, providing users with real-time feedback on network communication between the HLC and the HWI, as well as detecting any potential loss of nodes or topics within ROS 2.

4.3.1 Time synchronization

The time synchronization is handled by the HLC which subscribes to the desired topics of interest. To ensure accurate synchronization between observations and actions recorded at slightly different time stamps, a slop time of 0.05 seconds was implemented. This allows messages published within 0.05 seconds of each other to be appropriately paired together, aligning observations with the corresponding actions performed by the expert driver.

The purpose of the slop time is to accommodate slight variations in message timestamps while still preventing mismatches, such as pairing an outdated action with a recent observation or vice versa. This interval, though arbitrary, is deemed reasonable, considering that data is saved at a frequency of 10 Hz.

Moreover, information is only recorded and saved to a pickle file when all topics are actively published and the timestamps of the messages fall within the slop time.

4.3.2 Data formatting

The received data is directly placed into a dictionary consisting of color images, depth images, IMU data, and actions within the same timestamp. A dictionary enables the possibility of using the same data files combined with easy access to varying input combinations to train different models. Additionally, all data is transformed into a Pandas data frame before saving it as a pickle file.

4.3.3 Continious data collection (HG-DAgger)

When a BC model is trained with the desired input and is set up to take control of the car, the data recording methodology shifts. Instead of continuously collecting all observations and actions during driving, data recording is restricted to instances when expert actions are received. This process is illustrated in Figure 4.6.

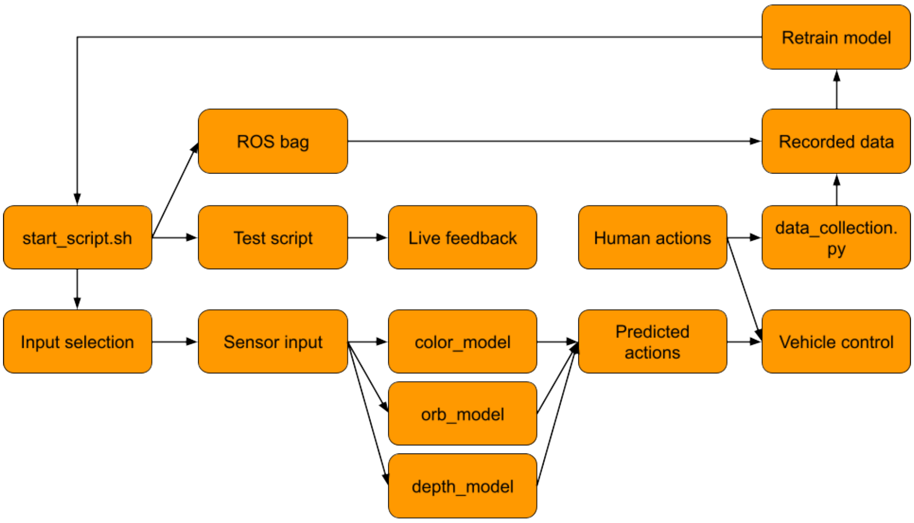

Retrain model
ROS bag
Recorded data
start_script.sh
Test script
Live feedback
Human actions
data_collection.py
Input selection
Sensor input
color_model
Predicted actions
Vehicle control
orb_model
depth_model

Figure 4.6: Structure of HG-DAgger.

The actions derived from both the model predictions and the Xbox controller inputs are published to ROS 2, where the twist mux node determines which commands should be executed. The twist mux is configured to prioritize commands from the Xbox controller, ensuring that the go-kart follows the model predictions until input is received from an expert driver. Upon receiving a message from an expert driver, the data collection initiates, gathering sensor observations combined with corresponding actions and aggregating them with the old dataset.

4.3.4 Real world data collection

To keep the environment consistent and to delimit the training and validation process, the data collection was made at Gokartcentralen. The track is 650 meters long and consists of a combined indoor and outdoor track which tests the AP4 of different road surfaces, tight turns in both directions, lightning, and weather conditions. The circuit can be seen in Figure 4.7 together with the optimal racing path.

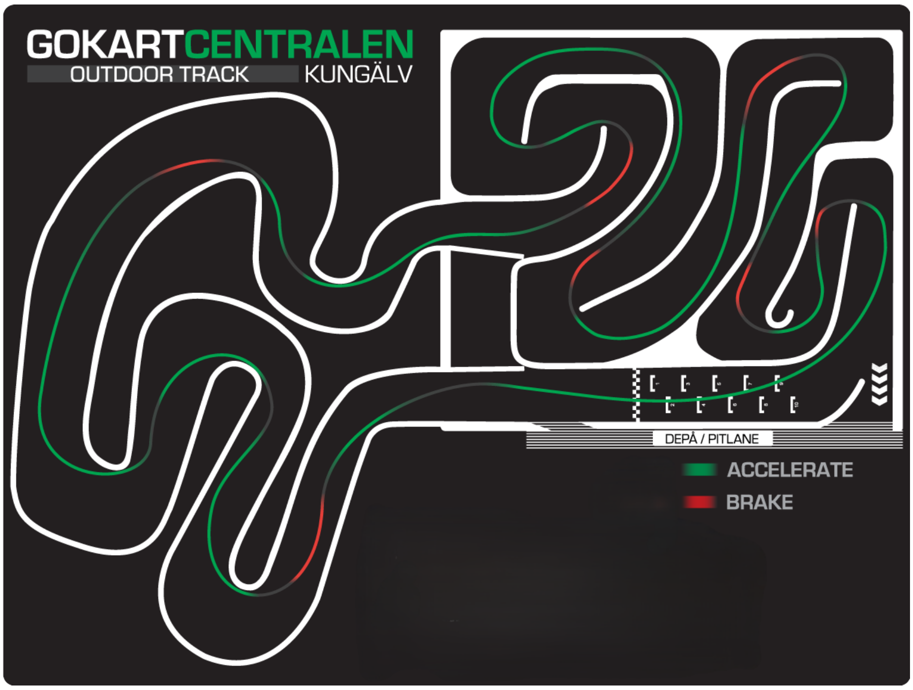


Figure 4.7: Gokartcentralen circuit. [44]. Adapted with permission.

The initial data collection consists of six laps driven around the track by an expert driver, yielding approximately 20000 observations and actions. These data points are distributed over a distance of 3900 meters, equating to roughly five observations and actions per meter.

Due to the automated pipeline, the data collection was started by a single script giving information about the progress and formatting the data for the imitation learning phase.

4.3.4.1 Training phase

The initial dataset was utilized to train various models with different inputs, ensuring consistent starting conditions for performance evaluation.

When the three models were trained, the data collection was started using DAgger. One at a time, the models were set up to drive the AP4 autonomously where it predicted the actions based on the observations made by sensors. When the AP4 did not follow the optimal path or was about to crash into the barriers, a human expert took over the control. During autonomous driving, no data is recorded. However, upon detection of signals from a human expert, data collection is promptly initialized, and the actions are then saved together with the observations ready to be aggregated into the initial dataset. Since the different models might have varying flaws in different parts of the circuit, the dataset with corrected actions was saved to a folder specified for the specific model.

After the models were driven two laps around the track with new data recorded at places where the AP4 did not drive as expected, the data was aggregated into the old dataset and retrained accordingly to algorithm 2.

These steps could then be iterated, collecting new data and updating the policies. However, due to the limitation of time on the test track, the process was only repeated two times for each model.

4.4 Data pre-processing

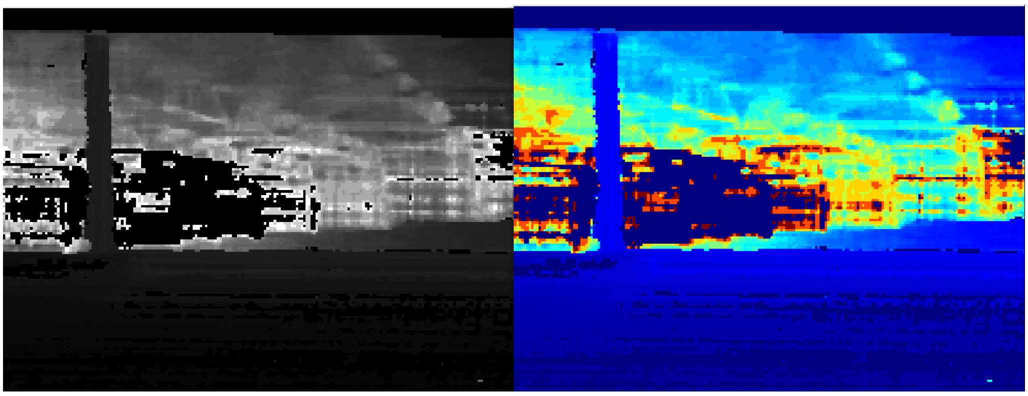
To allow for the data captured when driving the AP4 to be used for training the IL algorithms, the data had to be pre-processed. The pre-processing of the data consisted of converting the data format of the images from ROS 2 images to numpy arrays, calculating positions of ORB features, reformating the IMU data, and lastly structuring all the data into transitions which is used for training IL algorithms in the Imitation learning library. This is all explained further in this section.

4.4.1 Color images

The expected input for color images to the BC model is a matrix of size $\mathcal{I}$ (120 × 160 × 3), while the images are recorded as $\mathcal{I}(156 \times 208 \times 3)$ and flattened into a vector. The images are therefore first reformatted into a matrix and then downscaled to match the input size of $\mathcal{I}$ (120 × 160 × 3).

4.4.2 Depth images

As for the color images, the depth images are also extracted as a vector which then is reformated into a matrix. The depth image is however initially in grayscale, but is converted into an RGB image using a colormap to match the action space of the color image as well as RGB images being the expected input to the CNN used for feature extraction. In Figure 4.8a, a depth image can be seen before being converted to RGB, while in Figure 4.8b, the same image can be seen after being converted.



(a) Grayscale **(b)** Colormap

Figure 4.8: View from depth camera in both grayscale and after application of a colormap.

4.4.3 ORBs

The computation of ORBs was done from the color images using the Open CV library's *ORB_create.detect(color image)*. The function then detects a given amount of ORB features along with their position. The positions of the ORB features were then structured into a matrix the size of the color image, with one representing an ORB feature while zeros representing no ORB feature.

In Figure 4.9 an example where the positions of the ORB features are added as an overlay to the color image can be seen.

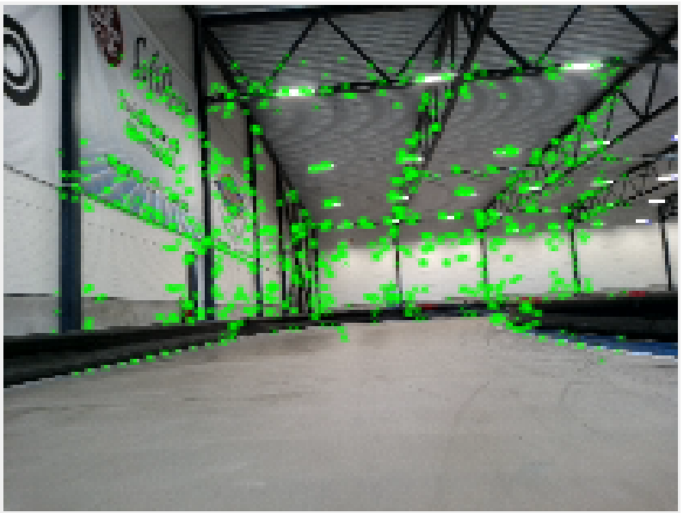


Figure 4.9: ORB positions overlay on color image

5

Evaluation

In this chapter, we delve into the methods employed to evaluate the system's performance. These methods encompass assessing the individual IL models themselves using validation datasets, wherein losses are presented. Furthermore, our evaluation extends to the comprehensive analysis of the entire system. This involves observing the AP4 in action as it navigates a go-kart track, enabling us to gauge its performance firsthand. Along with the methods used for evaluation, the results of the evaluation are also presented.

5.1 Validation dataset

The validation dataset includes pre-recorded observations with corresponding actions and real-world testing conducted at the GokartCentralen circuit. The pre-recorded dataset was utilized for testing different parameter values during training and consisted of one lap around the track, while real-time data was used to evaluate the driving performance of the various models.

5.2 Evaluation measures

Real-world testing of the AP4 was essential for validating the system's effectiveness and reliability under practical conditions. To gain a comprehensive understanding of how various inputs affected performance, it was crucial to select measurable and relevant tests that provided insights into the go-kart's behavior. A test protocol was developed and utilized during validation, focusing on the following measurable parameters:

- Lap time
- Lap time deviation
- Number of interventions per lap
- Distance traveled without interactions

All models, trained using both BC and HG-DAgger, were tested by having the AP4 drive around the track for three laps. The results were recorded in the protocol. Additionally, a map of the track, as shown in Figure 4.7, was printed. Whenever a human interaction was performed, the corresponding location on the track was marked. This was done to assess if there were specific sections of the track where the models encountered more or fewer challenges.

5.3 Implementation details

During training, various parameters were adjusted to optimize performance. Each model underwent training and was saved after 10 epochs with a batch size of 64 and a learning rate set to 0.001, utilizing the Adam optimizer from PyTorch. The loss function comprised L2 and entropy losses, both weighted at 0.001, alongside NLL loss weighted at 1.

5.4 Experiments

The experiments performed during the validation process were to let the AP4 drive around the track autonomously while measuring the lap time, lap time deviation, number of interventions and distance traveled without interventions. The objective was to assess if the driving performance could be enhanced using different inputs and HG-DAgger compared to BC.

The initial phase of the experiment involved AD by the AP4 using models trained exclusively with BC. The results from this phase were then used as a baseline for comparison with the second part, where the AP4 utilized the HG-DAgger algorithm. During the validation stage, the AP4 only received human actions when it was on the brink of a collision, rather than for path adjustment utilized in the training phase.

5.5 Results

In this section, the results from the evaluation are presented. The results consist of a table for the losses calculated from the validation dataset and a table for the performance from the real-world test. Furthermore, maps over each test run are presented, marked with the positions of the interactions during the test drives.

5.5.1 Repository and dataset

The repository containing the developed code from this thesis can be found in [45], and the dataset with all the recorded data is available in [46].

5.5.2 Validation dataset

In Table 5.1, the loss results for each trained model based on the validation dataset recorded at Gokartcentralen are presented. The losses are calculated by comparing the predicted actions with the actions performed by the human driver. The data column is the number of demonstrations used for training each of the models.

Table 5.1: Loss for the models evaluated

Model Type	Model	Data	L2 Loss	Entropy Loss	NLL
Color	`BC color`	21120	0.0890	0.0022	-1.6102
	`HG color 1`	22069	0.1190	0.0020	0.0467
	`HG color 2`	22858	0.1035	0.0017	-0.1989
ORB	`BC ORB`	21120	0.3331	0.0033	2.9339
	`HG ORB 1`	21803	0.3825	0.0031	3.6199
	`HG ORB 2`	22166	0.3021	0.0027	1.8326
Depth	`BC depth`	21120	0.0839	0.0022	-1.6556
	`HG depth 1`	21514	0.0874	0.0019	-1.5751
	`HG depth 2`	21941	0.0634	0.0016	-1.5140

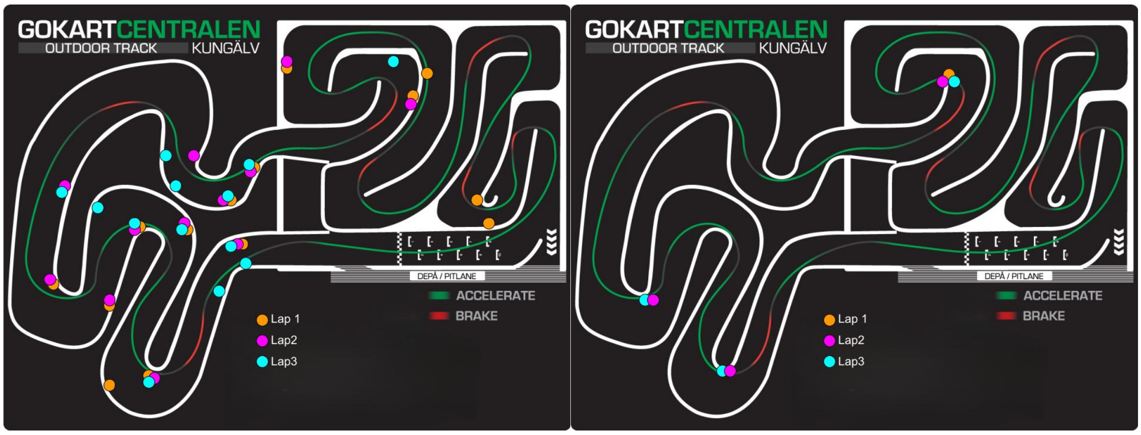
5.5.3 Real-world experiments

The real-world experiments were carried out letting the AP4 drive autonomously over three laps. In Table 5.2 the average result can be seen. The table includes the number of demonstrations used to train each model, seen in the data column.

Table 5.2: Results from real-world experiments.

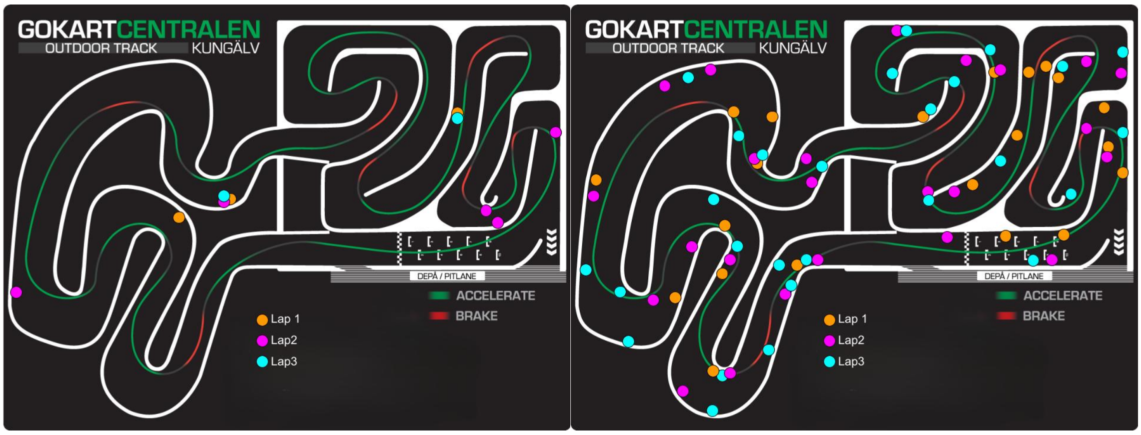
Model	**Data**	**Lap time** [s] ↓	**Laptime deviation** [s] ↓	**No. interventions** ↓	**Maximum Distance** [m] ↑
BC-Color	21120	408,58	22,13	13	250,89
HG-Color	22858	333,54	**0,6**	2,33	321,78
BC-Depth	21120	378,47	8,37	3,33	296,0376
HG-Depth	21941	466,24	27,86	22,33	72,94
BC-ORB	21120	369,74	4,76	6,67	154,39
HG-ORB	22166	**326,34**	3,20	**1**	**519,04**

In Figure 5.1 the locations of all interactions done during the validation test drive can be seen. The locations are marked with orange dots for the first lap, pink for the second, and blue for the third. This is done in separate sub-figures for each of the HG-DAgger and BC models respectively.


GOKARTCENTRALEN
OUTDOOR TRACK
KUNGÄLV
DEPÅ / PITLANE
ACCELERATE
BRAKE
Lap 1
Lap2
Lap3

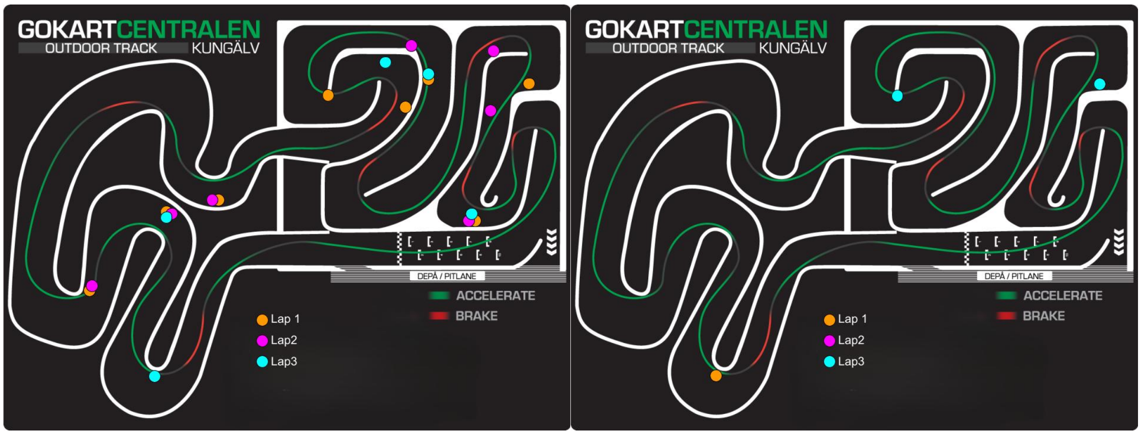
(a) Behavioral Cloning Color

(b) HG-DAgger Color


GOKARTCENTRALEN
OUTDOOR TRACK
KUNGÄLV
DEPÅ / PITLANE
ACCELERATE
BRAKE
Lap 1
Lap2
Lap3

(c) Behavioral Cloning Depth

(d) HG-DAgger Depth


GOKARTCENTRALEN
OUTDOOR TRACK
KUNGÄLV
DEPÅ / PITLANE
ACCELERATE
BRAKE
Lap 1
Lap2
Lap3

(e) Behavioral Cloning ORB

(f) HG-DAgger ORB

Figure 5.1: Locations of interactions during validation tests. [44]. Adapted with permission.

6

Discussion

6.1 Performance of IL models

By reflecting on the results obtained from the evaluation of the system discussed in subsection 5.5.3, it is evident that overall, our efforts have yielded success. However, it's noteworthy to address the unexpected findings regarding the HG-DAgger model utilizing depth input. While the BC-model incorporating depth input emerged as the top performer among BC-model variants, the HG-DAgger model witnessed a surprising decrease in performance, contrary to the observed improvements seen in other HG-DAgger variants compared to their corresponding BC models.

From the results, it can also be seen that the models with ORB feature positions as inputs performed better than the pure RGB model counterparts both for BC and HG-DAgger, which suggests that the use of ORB feature positions as a supplementary input has a positive effect in AD scenarios.

6.2 Software pipeline

The implemented software pipeline now allows for initial data to be collected from driving the AP4 platform, with the collected data directly being formatted into pickle files which are used for training BC models. The training can then be started for one of the three input formats resulting in a trained BC model.

The BC model can then be driven around the track autonomously and in the case of a near collision, a human expert can take control. If a human expert takes over the control, data is collected and formatted into pickle files. This data is then aggregated into the original data set allowing an HG-DAgger model to be trained. This process can then be repeated for several iterations until a satisfactory performance has been reached.

During the training of the models at Gokartcentralen in Kungälv the software pipeline proved to be effective allowing for multiple iterations of HG-DAgger to be trained during one visit. Encompassing, data collection, data formatting, training, and running the models. There was however a bottleneck in the time it took to train the models, taking between 20-40 minutes depending on the inputs used for the model.

6.3 Conclusion

The objective of this thesis was to implement an effective software pipeline allowing for data collection, data formatting, and setting up the structure for IL. It also includes training end-to-end models based on the recorded data with various inputs and the implementation of AD using the trained models. The IL algorithms implemented during this project are HG-DAgger as well as BC as a baseline. The models were implemented with RGB images and IMU as input, as well as two models with additional input. One with the positions of ORB features as an extra input and one with depth image as an extra input.

Except for the HG-DAgger model with depth image as a supplementary input, there was a clear improvement in performance after two iterations of HG-DAgger compared to the base BC model, for both the pure RGB model and the model with supplementary ORB features positions. It can therefore be concluded with some certainty that the use of HG-Dagger had a positive effect on the performance compared to pure BC. Even if further research is needed to understand why there were such unexpected results from HG-DAgger model with depth input.

It can also be concluded that an effective software pipeline has been implemented. Allowing for easy data collection, data formatting, training of models and iteratively retraining of the models using HG-DAgger.

When it comes to concluding the effect of using ORB feature positions and depth images as supplementary, the result is a bit more inconclusive. As the models with ORB feature positions outperformed the RGB model counterpart, both for the BC and HG-DAgger model, it can be concluded that the use of ORB-feature positions had a positive effect on performance compared to the use of only RGB-image as input. However, the effect of the depth image as a supplementary is less conclusive. Looking at the BC models, the model with depth input was the top performer and was expected to be the best performer for the HG-DAgger model as well, but instead, we saw unexpected behavior. Even if the expectation still holds it can not be concluded with the current result and further research would be needed, to understand why the HG-DAgger model with depth input behaved as it did.

6.4 Future work

Both of the developed hardware and software structures of the AP4 are highly modular enabling future enhancements such as discovering new algorithms within the HLC and adding additional sensors to the go-kart. In this section some thoughts on how the development of the AP4 could proceed based on the insights made during this thesis.

6.4.1 Localization

Localization is a critical aspect of AD, enabling vehicles to navigate efficiently from point A to point B in urban environments. Similarly, in the specialized context of autonomous racing, while the vehicle may not need to determine its global position on Earth, localization remains critically important. For racing vehicles, precise localization relative to the racetrack is essential for optimizing lap times and increasing speeds. By accurately understanding its position on the track, an autonomous racing car can adhere to the most optimal racing line, effectively balancing speed and handling to maximize performance.

Localization of autonomous vehicles can typically be achieved using traditional methods such as GPS or LiDAR, which constructs a 3D point cloud of the environment. However, regarding go-kart tracks most often located indoors, GPS connectivity might be insufficient due to poor satellite reception which excludes the use of a GPS. While LiDAR remains a viable option as an additional sensor to the system, the wish to keep the cost down and the number of sensors low, alternative methods are available for this.

One such solution to localize the vehicle on the track would be to utilize the already implemented ORB features in the system. This could be done by utilizing the existing open-source library called ORB SLAM3 [47]. ORB SLAM3 effectively and accurately maps the environment pinpointing the position of the ORB features and builds a 3D map using an IMU and a monocular, stereoscopic, or RGB-D camera.

By constructing a 3D map of the environment and incorporating the predefined optimal track path, the HLC could be fine-tuned. The predictions of the next action would still be made based on the visual input from the cameras, but by incorporating the actual position of the vehicle given by localization relative to the optimal path, the steering actions could be tweaked by a closed-loop control system.

6.4.2 Understanding HG-DAgger decrease in performance

At the current state, the aggregation of more data to the model utilizing depth image as supplementary input resulted in a large drop off in performance. As can be seen in Figure 5.1c, Figure 5.1d, along with Table 5.2. The reason for this drop-off in performance is not fully understood and an investigation of the reason behind the drop-off would lead to a better understanding of both the input data and the HG-DAgger implementation. This would allow for the construction of a more robust system and allow further improvements to be made.

One possible approach to better understand the decrease in performance would be to rerun the last iteration of HG-DAgger for the depth camera, collecting new data and retraining the model. If this were to result in an improved model, it would be of interest to compare the new data collected to the data which was collected in this project to understand what the difference in the data was.

6.4.3 Explore the possibility to avoid obstacles

During this project, the aim was for the AP4 to autonomously drive around the track. It would however be of interest to explore if the implemented software pipeline and IL algorithms would transfer well to other scenarios if trained on those instead. One such scenario would be to train the system on obstacle avoidance instead. This could be done by training the AP4 on weaving through a grid pattern of cones to avoid hitting cones which can be seen in Figure 6.1.

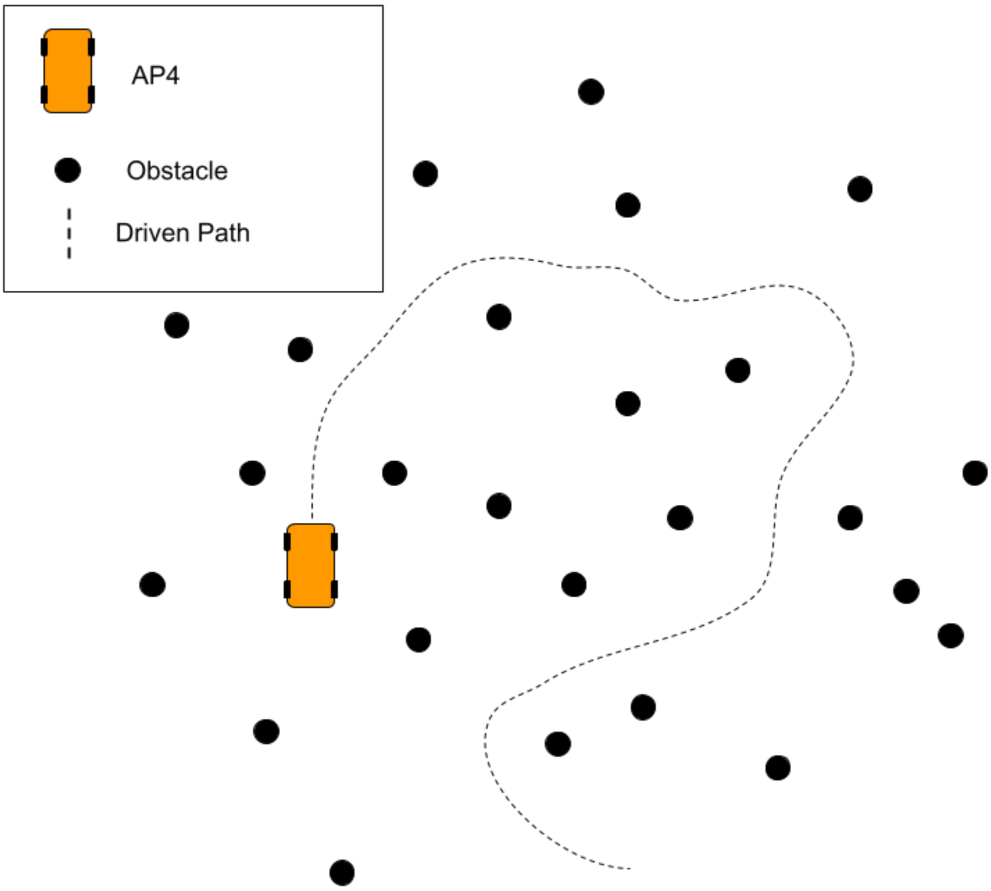


Figure 6.1: Possible environment to train the AP4 for obstacle avoidance.

6.4.4 Address sensor shortcomings

During real-world test and validation at Gokartcentralen it was noticed that in some locations of the track, often after a corner, the AP4 was driving very close to the barriers of the track. Due to the visual input to the system being directed directly forward with a limited field of view, there is no input of the environment reaching the sides of the AP4, visualized in Figure 6.2.

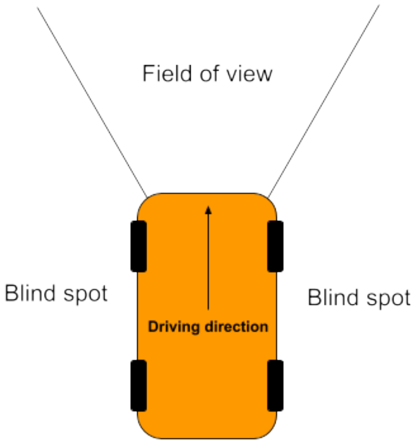


Figure 6.2: Field of view for the AP4

To address this issue a wide-angle camera could be used instead, increasing the field of view up to 180°. Alternatively, more sensors could be added directed to the sides of the AP4, covering the blind spots.

6.4.5 Alternative IL algorithms and hybrid IL-RL models

While the scope of this project primarily focused on the implementation of IL using BC and HG-DAgger, there are intriguing avenues for future research in enhancing autonomous driving systems.

One such avenue involves exploring alternative IL algorithms beyond BC and HG-DAgger. With the established modular software pipeline for IL, investigating the performance of alternative IL algorithms or refining existing ones presents an opportunity for improving the capabilities of autonomous driving systems.

Additionally, the combination of IL and RL represents another promising area for future investigation. Models can be trained initially using IL and then further refined through RL, resulting in a hybrid approach that offers greater robustness than pure IL models while requiring less training data compared to pure RL models.

Another compelling strategy proposed in [48] involves integrating IL and RL by enabling the model to predict forthcoming states and iteratively refine its predictions using RL. This approach holds the potential for enhancing the adaptability and performance of autonomous driving systems in complex real-world environments.

Exploring the potential of alternative IL algorithms and hybrid IL-RL models represents a promising direction for future research, offering opportunities to develop more resilient and efficient autonomous driving algorithms.

6.4.6 Create a realistic simulation

To allow for the implementation and faster testing of IL and RL models it would be beneficial to have a robust simulation. A simulation that is similar enough to the real world would allow for the exploration of training in simulation and then driving in the real world. This is an interesting area to explore if it is done successfully, it would open many opportunities for testing different models quickly, as well as using methods such as pure RL, more suitable for simulation in a physical system.

Bibliography

[1] E. Magnusson and F. Juthe, “"design of a modular centralized E/E and software architecture for a small-scale automotive platform",” 2023. [Online]. Available: `http://hdl.handle.net/20.500.12380/307245`.

[2] D. Parekh, N. Poddar, A. Rajpurkar, *et al.*, “A review on autonomous vehicles: Progress, methods and challenges,” *Electronics*, vol. 11, no. 14, 2022, ISSN: 2079-9292. DOI: `10.3390/electronics11142162`. [Online]. Available: `https://www.mdpi.com/2079-9292/11/14/2162`.

[3] L. Le Mero, D. Yi, M. Dianati, and A. Mouzakitis, “A survey on imitation learning techniques for end-to-end autonomous vehicles,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 9, pp. 14 128–14 147, 2022. DOI: `10.1109/TITS.2022.3144867`.

[4] L. Le Mero, D. Yi, M. Dianati, and A. Mouzakitis, “A survey on imitation learning techniques for end-to-end autonomous vehicles,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 9, pp. 14 128–14 147, 2022. DOI: `10.1109/TITS.2022.3144867`.

[5] X. Sun, M. Zhou, Z. Zhuang, S. Yang, J. Betz, and R. Mangharam, “A benchmark comparison of imitation learning-based control policies for autonomous racing,” in *2023 IEEE Intelligent Vehicles Symposium (IV)*, 2023, pp. 1–5. DOI: `10.1109/IV55152.2023.10186780`.

[6] L. Le Mero, D. Yi, M. Dianati, and A. Mouzakitis, “A survey on imitation learning techniques for end-to-end autonomous vehicles,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 9, pp. 14 128–14 147, 2022. DOI: `10.1109/TITS.2022.3144867`.

[7] SAE. “Formula sae.” Accessed: January 30, 2024. (2024), [Online]. Available: `https://www.fsaeonline.com/`.

[8] I. A. Challenge. “Indy autonomous challenge.” Accessed: January 30, 2024. (2024), [Online]. Available: `https://www.indyautonomouschallenge.com/`.

[9] H. X. Pham, M. Heiss, D. Tran, M. A. Nguyen, A. Q. Nguyen, and E. Kayacan, “Orb-net: End-to-end planning using feature-based imitation learning for autonomous drone racing,” in *ISR Europe 2023; 56th International Symposium on Robotics*, 2023, pp. 16–21.

[10] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, “Robot operating system 2: Design, architecture, and uses in the wild,” *Science Robotics*, vol. 7, no. 66, eabm6074, 2022. DOI: `10.1126/scirobotics.abm6074`. [Online].

Available: `https://www.science.org/doi/abs/10.1126/scirobotics.abm6074`.

[11] O. Robotics. “Ros 2.” Accessed: April 18, 2024. (2018-12-04), [Online]. Available: `https://wiki.ros.org/Nodes`.

[12] O. Robotics. “Ros 2 twist messages.” Accessed: April 19, 2024. (2018), [Online]. Available: `https://abedgnu.github.io/Notes-ROS/chapters/ROS/00_getting_started/overview.html`.

[13] O. Robotics. “Bags.” Accessed: April 23, 2024. (2022-02-15), [Online]. Available: `https://wiki.ros.org/Bags`.

[14] O. Robotics. “Twist$_mux$.” Accessed: April 23, 2024. (2018-09-13), [Online]. Available: `https://wiki.ros.org/twist_mux`.

[15] J. Newans. “Twist$_stamper$.” Accessed: April 23, 2024. (2021-01-29), [Online]. Available: `https://github.com/joshnewans/twist_stamper`.

[16] O. Robotics. “Message$_filters$.” Accessed: April 23, 2024. (2024-04-16), [Online]. Available: `https://github.com/ros2/message_filters/tree/rolling`.

[17] A. Mouat, *Using Docker: Developing and Deploying Software with Containers.* O’Reilly Media, 2015, pp. 3–11, ISBN: 9781491915929. [Online]. Available: `https://books.google.se/books?id=wpYpCwAAQBAJ`.

[18] Aqua. “Container images: Architecture and best practices.” Accessed: May 3, 2024. (2021), [Online]. Available: `https://www.aquasec.com/cloud-native-academy/container-security/container-images/`.

[19] IBM. “What is machine learning?” Accessed: January 19, 2024. (2023), [Online]. Available: `https://www.ibm.com/topics/machine-learning`.

[20] D. A. Pomerleau, “Alvinn: An autonomous land vehicle in a neural network,” in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 1, Morgan-Kaufmann, 1988. [Online]. Available: `https://proceedings.neurips.cc/paper_files/paper/1988/file/812b4ba287f5ee0bc9d43bbf5bbe87fb-Paper.pdf`.

[21] Y. Lecun, U. Muller, J. Ben, E. Cosatto, and B. Flepp, “Off-road obstacle avoidance through end-to-end learning.,” 2005-01.

[22] S. Ross, G. Gordon, and D. Bagnell, “A reduction of imitation learning and structured prediction to no-regret online learning,” in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, G. Gordon, D. Dunson, and M. Dudík, Eds., ser. Proceedings of Machine Learning Research, vol. 15, Fort Lauderdale, FL, USA: PMLR, 2011-11–13 Apr, pp. 627–635. [Online]. Available: `https://proceedings.mlr.press/v15/ross11a.html`.

[23] M. Kelly, C. Sidrane, K. Driggs-Campbell, and M. J. Kochenderfer, “Hg-dagger: Interactive imitation learning with human experts,” in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8077–8083. DOI: `10.1109/ICRA.2019.8793698`.

[24] V. Konda and J. Tsitsiklis, “Actor-critic algorithms,” in *Advances in Neural Information Processing Systems*, S. Solla, T. Leen, and K. Müller, Eds., vol. 12, MIT Press, 1999. [Online]. Available: `https://proceedings.neurips.cc/`

`paper_files/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf`.

[25] R. Bellman, *Dynamic Programming.* Princeton, NJ: Princeton University Press, 1957. [Online]. Available: `https://gwern.net/doc/statistics/decision/1957-bellman-dynamicprogramming.pdf`.

[26] C. J. C. H., “Learning from delayed rewards,” *PhD thesis, Cambridge University*, 1989. [Online]. Available: `https://cir.nii.ac.jp/crid/1570291224996580224`.

[27] K. O’Shea and R. Nash, *An introduction to convolutional neural networks*, 2015. arXiv: `1511.08458 [cs.NE]`.

[28] S. Albawi, T. A. Mohammed, and S. Al-Zawi, “Understanding of a convolutional neural network,” in *2017 International Conference on Engineering and Technology (ICET)*, 2017, pp. 1–6. DOI: `10.1109/ICEngTechnol.2017.8308186`.

[29] W. Commons, *File:convolutionandpooling.svg — wikimedia commons, the free media repository*, [Online; accessed 25-April-2024], 2024. [Online]. Available: `https://commons.wikimedia.org/w/index.php?title=File:ConvolutionAndPooling.svg&oldid=868000694`.

[30] A. Shekhar. “What are l1 and l2 loss functions?” Accessed: May 3, 2024. (2019), [Online]. Available: `https://amitshekhar.me/blog/l1-and-l2-loss-functions`.

[31] A. D. Tovar. “Negative log likelihood explained.” Accessed: May 7, 2024. (2019), [Online]. Available: `https://medium.com/deeplearningmadeeasy/negative-log-likelihood-6bd79b55d8b6`.

[32] N. University. “Probability density function.” Accessed: May 8, 2024. (), [Online]. Available: `https://www.ncl.ac.uk/webtemplate/ask-assets/external/maths-resources/statistics/distribution-functions/probability-density-function.html`.

[33] P. A. Bosman and D. Thierens, “Negative log-likelihood and statistical hypothesis testing as the basis of model selection in ideas,” in *Proceedings of the Tenth Dutch–Netherlands Conference on Machine Learning. Tilburg University*, 2000.

[34] R. Lau. “Cross-entropy, negative log-likelihood, and all that jazz.” Accessed: May 8, 2024. (2022), [Online]. Available: `https://towardsdatascience.com/cross-entropy-negative-log-likelihood-and-all-that-jazz-47a95bd2e81`.

[35] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “Orb: An efficient alternative to sift or surf,” in *2011 International Conference on Computer Vision*, 2011, pp. 2564–2571. DOI: `10.1109/ICCV.2011.6126544`.

[36] E. Rosten and T. Drummond, “Fusing points and lines for high performance tracking,” in *Tenth IEEE International Conference on Computer Vision (ICCV’05) Volume 1*, vol. 2, 2005, 1508–1515 Vol. 2. DOI: `10.1109/ICCV.2005.104`.

[37] D. G. Viswanathan, “Features from accelerated segment test (fast),” in *Proceedings of the 10th workshop on image analysis for multimedia interactive services, London, UK*, 2009, pp. 6–8.

[38] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden, "Pyramid methods in image processing," *RCA engineer*, vol. 29, no. 6, pp. 33–41, 1984.

[39] N. Ahmad, R. A. R. Ghazilla, N. M. Khairi, and V. Kasi, "Reviews on various inertial measurement unit (imu) sensor applications," *International Journal of Signal Processing Systems*, vol. 1, no. 2, pp. 256–262, 2013, [Online; accessed 25-April-2024].

[40] V. Safin, "Distance estimation," *Medium*, 2022. [Online]. Available: `https://medium.com/analytics-vidhya/distance-estimation-cf2f2fd709d8`.

[41] Luxonis. "Oak-d." Accessed: January 8, 2024. (Accessed: 2024-01-08), [Online]. Available: `https://shop.luxonis.com/collections/oak-cameras-1/products/oak-d`.

[42] A. Gleave, M. Taufeeque, J. Rocamonde, *et al.*, *Imitation: Clean imitation learning implementations*, arXiv:2211.11972v1 [cs.LG], 2022. arXiv: `2211.11972 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/2211.11972`.

[43] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: `http://jmlr.org/papers/v22/20-1364.html`.

[44] Gokartcentralen, *Kungälvs inne/utomhusbana*, [Online; accessed 16-May-2024], 2024. [Online]. Available: `https://gokartcentralen.se/info/banan/`.

[45] A. Petersén and J. Wellander. "Autonomous platform." Accessed: June 10, 2024. (Accessed: 2024-06-10), [Online]. Available: `https://github.com/infotiv-research/autonomous_platform`.

[46] A. Petersén and J. Wellander. "Autonomous platform gokartcentrallen imitation learning dataset." Accessed: June 10, 2024. (Accessed: 2024-06-10), [Online]. Available: `https://huggingface.co/datasets/hamidebadi/autonomous_platform_gokartcentrallen_imitation_learning_dataset`.

[47] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM3: an accurate open-source library for visual, visual-inertial and multi-map SLAM," *CoRR*, vol. abs/2007.11898, 2020. arXiv: `2007.11898`. [Online]. Available: `https://arxiv.org/abs/2007.11898`.

[48] P. Cai, H. Wang, H. Huang, Y. Liu, and M. Liu, "Vision-based autonomous car racing using deep imitative reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7262–7269, 2021. DOI: `10.1109/LRA.2021.3097345`.

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY