Diploma thesis

Behavioural-Cloning-Based Path Planning for Autonomous Student Formula

Roman Šíp

Vedoucí: doc. Ing. Karel Zimmermann, Ph.D.

Fakulta Elektrotechnická

Katedra kybernetiky

May 24, 2024

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Šíp Roman**	Personal ID number:	**492273**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Open Informatics**		
Specialisation:	**Computer Vision and Image Processing**		

II. Master’s thesis details

Master’s thesis title in English:

Behavioral-Cloning-Based Path Planning for an Autonomous Student Formula

Master’s thesis title in Czech:

Plánování pro studentskou autonomní formuli založené na podmíněném behaviorálním klonování

Guidelines:

1. Procedurally generate virtual race tracks according to the Formula Student rules [4].
2. Estimate the optimal trajectory of each track according to [3].
3. Train a deep learning model that predicts the segment of optimal trajectory from top-view images containing spatial detections of race cones. Use previously generated optimal tracks as the ground truth.
4. Propose and implement conditional deep learning model [2], and suggest various conditionings such as turn types that can be provided in advance according the FS rules. [4]. Examples includes prior turn types (e.g. left, right, hairpin, chicane) and their approximate location.
5. Experimentally evaluate proposed pipeline with respect to the existing solution [3] that follows the center line. The evaluation should contain average speed improvement and analysis of robustness with respect to cone detection failures.
6. Evaluate its performance in terms of reliability and performance.

Bibliography / sources:

[1] Bojarski, Mariusz, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, et al. “End to End Learning for Self-Driving Cars.” arXiv, April 25, 2016. http://arxiv.org/abs/1604.07316.
[2] Codevilla, Felipe, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. “End-to-End Driving via Conditional Imitation Learning.” arXiv, March 2, 2018. http://arxiv.org/abs/1710.02410.
[3] Horáček, Michal. “Nalezení Nejrychlejší Trajektorie pro Autonomní Studentskou Formuli,” June 9, 2022. https://dspace.cvut.cz/handle/10467/101617.
[4] https://www.formulastudent.de/fileadmin/user_upload/all/2024/rules/FS-Rules_2024_v1.0.pdf

Name and workplace of master’s thesis supervisor:

doc. Ing. Karel Zimmermann, Ph.D. Vision for Robotics and Autonomous Systems FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **07.02.2024** Deadline for master's thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

doc. Ing. Karel Zimmermann, Ph.D.	prof. Dr. Ing. Jan Kybic	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 24, 2024, Roman Šíp ....

Signature

Used Software

In accordance with the methodological instructions, the following software was used during the development of this thesis:

- GitHub Copilot as a programming assistant system[1]
- Claude as a chatbot for answering technical questions[2]

[1] `https://copilot.github.com/`
[2] `https://claude.ai/`

Acknowledgment

I want to thank my thesis supervisor doc. Ing. Karel Zimmermann, Ph.D. for going through the whole evolution process of this thesis with me.

Next, I want to thank my teammates from team eForce, namely: David Korčák, Ondřej Kuban, Marko Matoničkin, Michal Horáček, Erik Doležal and Oliver Vidriczkó for helping with this thesis in one way or another.

Lastly, I thank my family for financially supporting my lifestyle, while writing this thesis.

Annotation

This thesis presents a novel, neural-network based path planning solution for an autonomous Formula Student car. In the competition, the car autonomously drives on racing tracks marked by traffic cones. Departing from traditional geometric and rule–based approaches, we use behavioral cloning to train a neural network to predict optimal racing line by learning to imitate expert demonstrations. Our method consists of synthetically generating a dataset of virtual tracks and scenarios, including right-angle turns, chicanes and hairpins, each paired with the optimal racing line. Using a simulator, we collect expert trajectories of the car driving on these tracks, which serve as training data for our neural network path planner model. This model, designed as a sequence–to–sequence architecture with Gated Recurrent Units (GRUs), learns to map scene state information (traffic cone detections) to a sequence of path points, representing the path the car should follow. Our approach incorporates a conditioning mechanism that provides the network with high–level tokens, which let the network know about the type of the upcoming turn. This enables the model to anticipate certain track features and execute complex maneuvers, such as setting up early for a turn to be able to drive through it smoothly and at higher speeds, mimicking the way human pilots drive. Furthermore, we demonstrate the model's ability to drive the skidpad discipline, an 8–shaped track, in which the car has to turn based on the context of the drive, typically requiring the use localization and mapping algorithms. We show that our model learns to predict the correct path using only local information. We evaluate the proposed method both in simulations and using a physical car. We show, that the car is able to drive at higher speeds, when using our model, compared to baseline classical planning algorithm.

Key-words: Autonomous driving, Formula Student Driverless, Neural Networks, Path Planning, Behavioural Cloning, End-to-End Learning

Anotace

Tato diplomová práce představuje algoritmus pro plánování trasy pomocí neuronové sítě pro autonomní formuli účastnící se kategorie Driverless v univerzitni soutěži Formula Student. V této soutěži, plně autonomně řízené formule jezdí po závodních okruzích vyznačených pomocí dopravních kuželů. Historicky bylo plánování trasy, po které má auto jet, prováděno pomocí jednoduchých geometrických předpokladů a ručně vytvořených pravidel pro plánování středové čáry trati. Tato práce představuje metodu plánování trasy založenou na imitaci experta. Vygenerujeme dataset virtuálních tratí a specifických zatáček (pravoúhlé, šikany, hairpin, atd.) spolu s jejich optimálními závodními trasami. Uvnitř simulátoru auto virtuálně projíždí tyto tratě, aby zaznamenalo expertní trajektorie. Poté natrénujeme neuronovou síť, aby tyto expertní trajektorie imitovala a naučila se mapování z vizuálních vstupů auta na předpovídání správné trasy, bez jakéhokoli explicitního odhadu mapy nebo ručně vytvořených pravidel. Síť se naučí vlastní časovou reprezentaci scény, což jí umožňuje plánovat na základě kontextu nedávných pozorování. Model podmiňujeme tokenem, který mu dává informace o typu zatáčky, jež je před ním. To umožňuje modelu provádět složité manévry, jako je nadjetí si před zatáčkou pro plynulý průjezd. Demonstrujeme také schopnost využítí plánovacího modelu pro disciplínu skidpad, osmičkovou trať. Na skidpadu se musí auto rozhodovat na základě kontextu jízdy, což obvykle vyžaduje použití algoritmů pro mapování a lokalizaci. Ukazujeme, že plánovač založený na neuronové síti se naučí naplánovat správnou trasu podle podmiňovacího vektoru. Představenou metodu vyhodnotíme jak v simulacích, tak na fyzickém autě, přičemž ukazujeme, že neuronová síť může překonat referenční klasickou metodu plánování.

Klíčová slova: Autonomní řízení, Formula Student Driverless, Neuronové sítě, Plánování cesty, Imitační učení

0 Contents

0 List of Figures

0 List of Tables

1 Introduction

This thesis proposes a novel approach for solving the problem of path planning for autonomous racing in the context of the Formula Student competition. In this section, we first give a brief introduction to the Formula Student competition with a focus on the driverless part of the competition. Subsequently, we give the motivation for the thesis and problem statement. We then introduce the team eForce, for which this thesis was developed and the describe the specifics of the car setup and the autonomous system. Finally, we outline the structure of the thesis.

1.1 Formula Student

Formula Student is an international engineering competition, where university teams of students build formula-style race cars. During summer, there are several competitions held around the world. At the competitions, the teams are evaluated based on the race performance of their car, as well as on their engineering skills reflected in the car's design.

The competition, which started in 1981, has since evolved to accommodate the advancements in automotive technology. When the competition began the cars used combustion engines, but in 2010 the competition introduced a new class for electric cars. In 2016, the competition introduced a new discipline for autonomously piloted cars. Since 2022, all formula cars are required to be be capable of being piloted by a human and to race autonomously. In this thesis, we focus only on the autonomous racing part of the competition.

In the driverless part of the competition, there are four different dynamic disciplines, namely: acceleration, skidpad, autocross and trackdrive. Although they are similar to their piloted versions, they differ in several key ways.

Firstly, the traffic cones used to denote the tracks are colored based on their placement. There are four distinct types of traffic cones, as shown on Figure 1. The blue and yellow ones are used to denote the general part of the track, placed with a rule that yellow and blue cones are respectively on the right and left sides of the road from the perspective of the driving direction. The big orange cones are placed around the start and finish line. The regular orange cones are used to denote braking areas in the skidpad and acceleration disciplines.

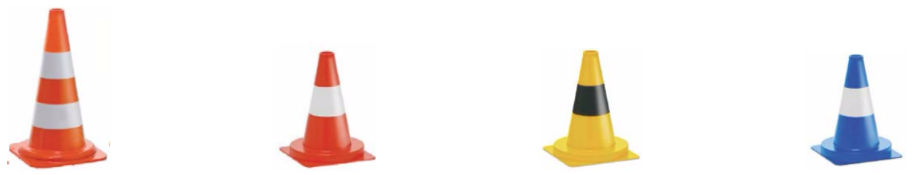


Figure 1: Traffic cones used in the Formula Student Driverless competition.

1.1.1 Autocross

The autocross discipline takes place on a circuit with a maximum length of 500 meters. The track is marked with yellow, blue and big orange traffic cones and is built according to these rules: [7]

- Straights: No longer than 80 m
- Constant Turns: up to 50 m diameter
- Hairpin Turns: Minimum of 9 m outside diameter (of the turn)
- Miscellaneous: Chicanes, multiple turns, decreasing radius turns, etc.
- The minimum track width is at least 3 m
- The length of one lap is approximately 200 m to 500 m.
- Yellow and blue cones mark the left and right boundary of the track respectively.

The goal of the discipline is for the car to drive a single lap around the track as fast as possible. The car cannot drive off track and is penalized for hitting the traffic cones.

1.1.2 Trackdrive

The trackdrive typically takes place on the exact same track as the autocross, with the only difference being that the car has to drive 10 laps around the track instead of just one. This tests the car's physical endurance and the robustness of the algorithms driving the car. It also gives the teams the opportunity to deploy more advanced algorithms, such as simultaneous localization and mapping (SLAM) to build a map of the track during the first lap and then use this map to optimize the racing line for the later laps. [8]

1.1.3 Acceleration

The acceleration discipline consists of a 175 meter long straight track. It consists of two parts. The first 75 meters is acceleration zone, through which the car has to drive as fast as possible. The remaining 100 meters is the braking zone, where the car has to safely come to a stop without hitting the traffic cones or driving off track. This discipline tests the car's acceleration and braking capabilities, while also testing the robustness of the control algorithms at keeping the car centered on the track.

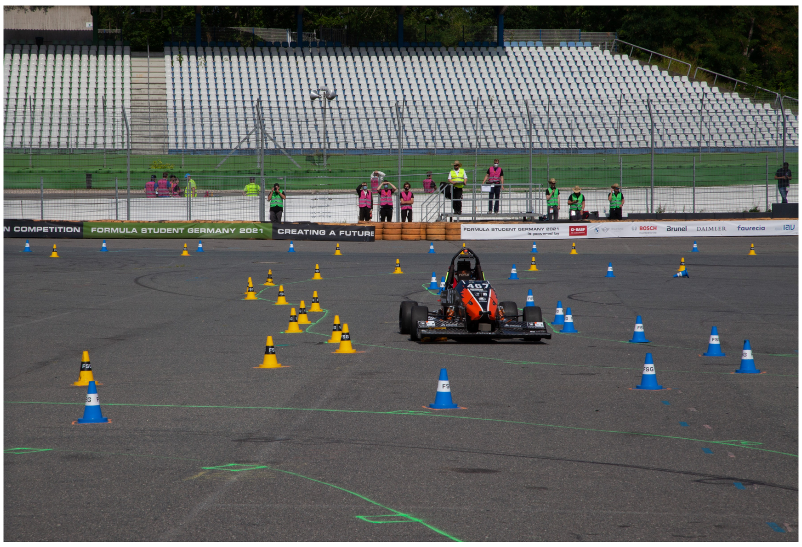


Figure 2: The DV.01 car of team eForce driving the Trackdrive discipline in Formula Student Germany 2022 competition.

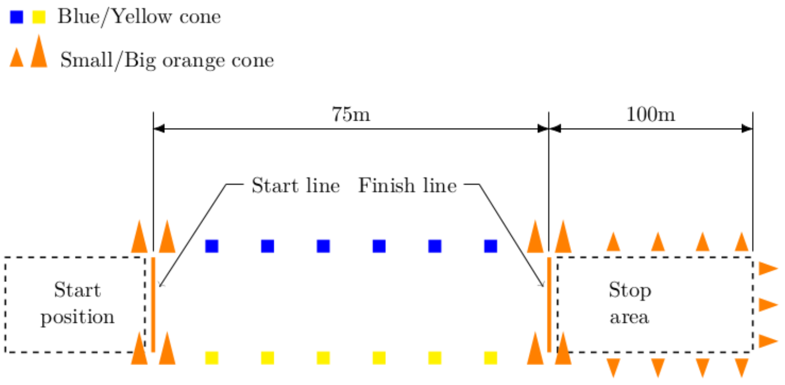


Figure 3: The acceleration track.

1.1.4 Skidpad

The skidpad discipline takes place on a track made of two circles with a 15 meter diameters. The challenge is that the car first has to drive two laps around the right circle, then two laps around left circle and finally stop in the middle of the track. From a physics perspective, this discipline tests that car's cornering capabilities at the limit of the tire's grip. From an algorithmic perspective, there is a challenge in planning the correct path when the car is in the middle of the track, because depending on the context, the car has to drive either to the left or to the right. Since the track layout is known before-hand, teams usually solve it with a localization algorithm such as Monte Carlo localization (MCL) or particle filter. [9]

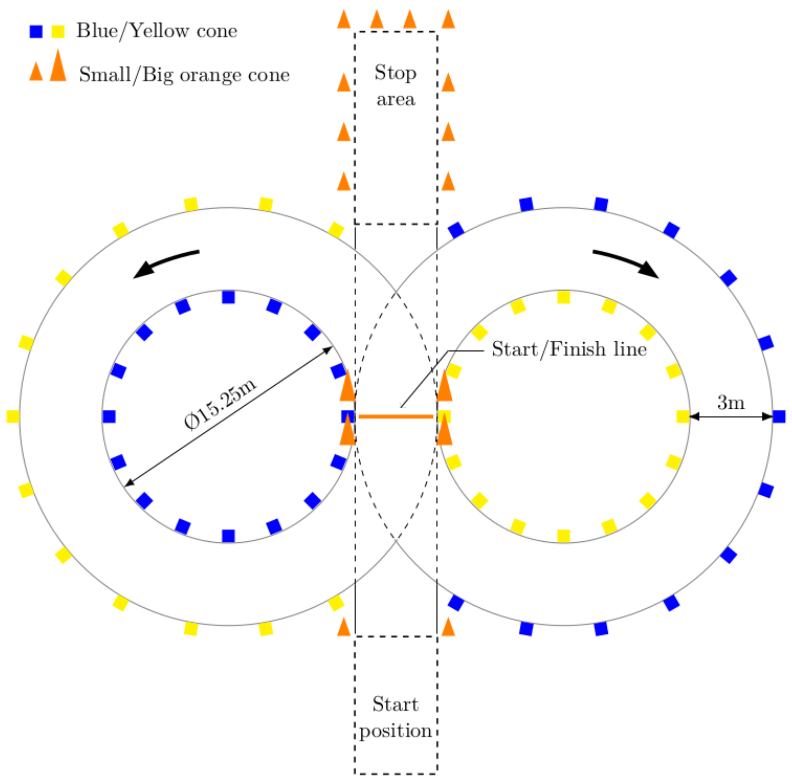


Figure 4: The skidpad track.

1.2 Motivation

1.2.1 Autocross racing line

Since the inception of the Formula Student Driverless competition in 2016, teams have been trying to solve the problem of planning the optimal path for the car to follow. Most solutions do however only concern themselves with planning the center line of the track, citing safety and minimization of the risk of driving off track or into the traffic cones which denote the track.[8][10]

The only discipline, in which certain teams attempt to drive an improved racing trajectory is the trackdrive. Trackdrive consists of 10 laps, giving the teams opportunity to build a close to perfect map of the track during the first lap using some kind of a SLAM algorithm. This map is subsequently used as input in a racing line optimization problem. This optimization results in some sense optimal path for the car to follow, taking into in the criterion function things such as the car's driving model and in-track safety margins. Following such a trajectory allows the car to complete the laps with lower lap times, increasing the chance of winning the discipline.

The other free lap discipline is the autocross. It is driven on the same track as the trackdrive with only change being, that only a single lap is driven. This makes the strategy of mapping, localization and racing line optimization not possible. This leaves most teams being able to only drive the center line in autocross with the exception of teams with very high end LiDAR sensors, which are in some cases able to map the whole track very early in the lap.

One of the main points of focus of this thesis is to provide a solution to the problem of planning the optimal path for the car to follow in the autocross discipline. We believe it is possible to drive along a better path than the center line without needing any special mapping or perfect information about the track. Before the autocross discipline takes place, the teams are given the opportunity to walk the track to get a feeling for it. In the piloted version of the competition, the pilots heavily rely on this track walk to be able to drive fast on their first attempt.

One of the goals of this thesis is to provide a solution, which in some sense mimics the way a pilot uses the information from the track walk to then be able to drive fast on his first attempt. We believe, that by just knowing the types of turns on the track and their order, the car can drive faster than the center line approach.

The inspiration for our approach came from the way rally drivers drive. The rally races are typically driven on a very large track, which the drivers have never driven on before. The drivers have a co-driver, who has a map of the track and conveys key information about the road ahead to the driver. Due to the intensity of rally racing, the co-driver doesn't have a time to convey the exact position and shape of every turn, but instead conveys the type of the turn ahead using a predefined set of turn types and names.

1.2.2 Skidpad without localization

As described before, the skidpad is an 8-shaped track, whose layout is known beforehand. Historically, teams have solved the problem of driving the skidpad by using a localization algorithm, which is able to determine the car's position on the track. When the car's position is known, the correct path is easy to determine as a crop of the global path, known before hand, which is closest to the car's position.

The potential problem with this approach is that the localization error directly translates to the error in the planned path. As in way, the car is driving blindfolded, only knowing its position estimate. If the estimate is wrong, the car will drive right through the cones or off track.

One of the goals of this thesis is to provide a solution, which is able to plan the correct path for the car to follow with only local detections of the cones, without the need to rely on perfect localization. Even more than in the autocross discipline, driving the skidpad is highly dependent on the context of the drive. As the car has to take left, right or straight path in the middle of the track, depending on the current progression of the drive.

Learning to drive the skidpad using only local data and temporal context learned from expert trajectories is another goal of this thesis. This new approach could potentially be more robust to localization errors and and would be more similar to the way a human driver would drive the skidpad.

1.3 Problem statement

As mentioned in the previous section, this thesis aims to enable car to drive a faster path than the center line in the autocross discipline, in which it must drive a single lap on an unknown track.

We propose to solve this problem using a neural network trained on a large dataset of expert trajectories. The basic idea is to divide to enumerate the types of turns on the track, that the car can encounter. We then generate a dataset of tracks containing these turns and virtually drive expert trajectories on them. This will provide us with a dataset of expert trajectories of how to optimally drive through every type of turn.

In the next step, we will train a neural network to imitate these expert trajectories. During training, we will condition the neural network model on the type of the turn the expert trajectory is driving through. This will allow the model to learn how to act to set itself up for the turn ahead, even before it sees the turn. An example of this would be to hug the right side of a road before a left turn, to be able to take the turn in a sharper way with minimal steering input.

When the model is trained to drive through individual turns, we can then use it to drive a whole track, which is just a sequence of individual turns. For a given track, we will collect the types of turns and their lengths during the trackwalk and then feed this information to the model before the attempt. Using this information, we will be able to change the model's conditioning on the fly, allowing it to exhibit complex driving behavior, such as setting up for the turn ahead and then aggressively cutting the corner.

Similarly, we will use the same approach to drive the skidpad track without the need for localization. We will divide the track into individual turns and generate expert trajectories of the car driving through them. Then we will train a model to imitate these expert trajectories. Then, during a full skidpad drive, we will condition the model on the type of turn it should take in the middle intersection.

1.4 Team eForce

Team eForce is a student team from the Faculty of Electrical Engineering of the Czech Technical University in Prague. The team was founded in 2012 as the first Czech team that built an electric formula car. In 2020, we started to participate in the Formula Student Driverless competition by rebuilding the 2018 piloted car to have autonomous capability by adding sensors such as camera and lidar, electrical steering and the emergency brake system. In 2023, the team's FSE.12 car was its first car built for both autonomous and piloted racing. This thesis is a part of the team's effort to improve the autonomous racing capabilities of the car and to compete in the Formula Student Driverless competition. We now give a brief overview of the car and then describe its autonomous system.[11]

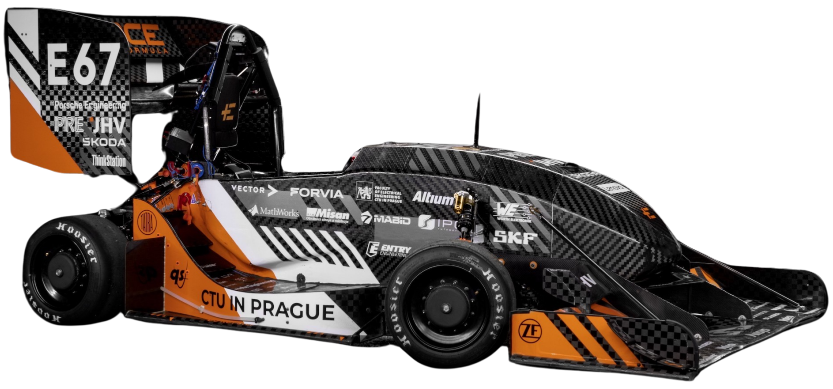


Figure 5: The FSE.12 car is was team eForce car of the 2023 season and is was used for testing the algorithms in this thesis.

1.4.1 Car setup

This thesis is developed for the team's FSE.12 car, which is a 4-wheel drive electric formula car. It is equipped with a 35 kW motor in each wheel. The car is powered by a 600V, 7.45 kWh battery, built to last for the duration of the endurance discipline, which is 22 km long. The car is weighs approximately 200kg, has a wheelbase of 1.2 meters and is 2 meters long.

For the autonomous racing, the car has a power steering system controlled by the car's computer over a CAN bus. For the perception, the car has a single camera mounted at the top of the main hoop of the car. The camera is a ZED 2 stereo camera with a 110 degrees field of view, although only one of the camera lenses is used for driving. There is also an Inertial Navigation System (INS) unit inside the car, which can be used to get real-time position and orientation of the car. There are two CAN buses in the car connecting all the electric control units and sensors. The main computer is a Zotac Zbox with an Intel i7 CPU and a Nvidia RTX 2080 GPU.

1.4.2 Autonomous system

The autonomous system runs on an in-house developed microsevice architecture similar to the Robot Operating System (ROS). The individual nodes and their services are written in Python and communicate using the ZeroMQ (ZMQ) messaging protocol. The system is divided in to several nodes, each responsible for a different part of the system. Generally, the nodes are divided into three types of nodes: IO nodes, processing nodes and actuator nodes. The whole system diagram is shown in Figure 6.

The CAN nodes and the perception nodes are the IO nodes, they are responsible for watching the car's sensors and when data is available, processing it and sending it forward. The mission node is a processing node. It is responsible for the decision making of the car. Algorithms for path planning, mapping and localization and control are implemented there. The steering, motor and can sender nodes are actuator nodes. The steering node implements the low-level control of the car's

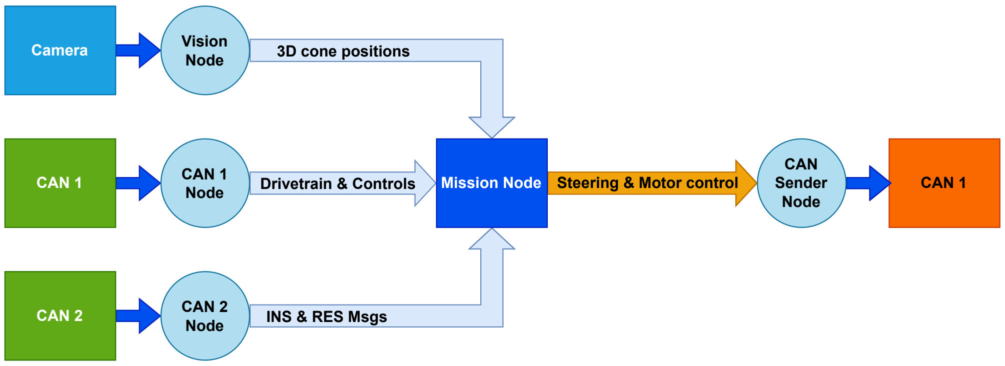


Figure 6: Diagram of individual processing nodes in the micro–service architecture of the car's autonomous system.

steering system, translating steering angle setpoint into increment level commands of the steering stepper motor. The motor node is responsible for the control of the car's motors. It receives speed and steering setpoints and translates them into motor torque commands. It implements advanced control algorithms such as torque vectoring and traction control. Finally, the can sender node is responsible for sending status messages to the car's CAN bus, which are then displayed on the car's dashboard.

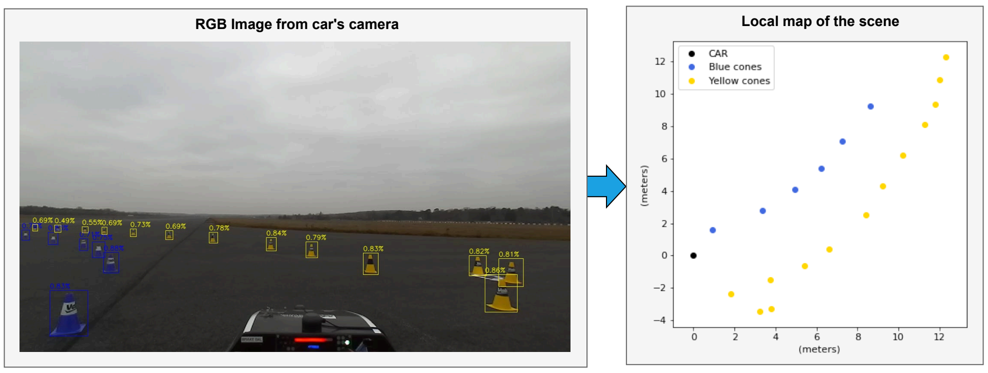


Figure 7: Detection of traffic cones in the image using a YOLO detector and subsequent projections of the bounding boxes into a map of the scene.

From the algorithmic perspective, the system can be divided into three main parts: perception, planning and control. Calling the parts more specifically to the context of our system, it would be: traffic cone detection, path planning, path tracking and actuation control.

The traffic cone detection is done by taking the image from the camera and running it through a YOLO (You Only Look Once) object detection neural network.[12] YOLO is a real-time object detection neural network, using a single convolutional neural network pass to detect objects in the image. The used instance of the YOLO network was designed and trained as a part of author's bachelor thesis.[13]

The network detects the traffic cones as bounding boxes in the image. The bounding boxes are then transformed into a 3D position in the car's coordinate system by projecting the bounding box onto the ground plane using a pre-computed homography matrix. This homography matrix is recomputed at the start of each racing session, as the camera's position and orientation can change between sessions and even small changes can have a big impact on the accuracy of the detection.

The path planning part of the system receives the 3D positions of the traffic cones and their respective types and outputs a path for the car to follow. The path is represented as a sequence of waypoints. The current implementation of the path planner is a classical geometric path planner. It uses assumptions about the yellow and blue cones being placed on the left and right side of the track respectively and the assumption that the cones come in pairs.[8] The basic idea of the path planner is to find the closest blue and yellow pair of cones and compute their mid-point. The next mid-point is computed in the similar way, only we start from previous mid-point and search for the next closest pair of cones. This way, the path planner creates a sequence of mid-points, which approximate the center line of the track. One of the goals of this thesis is to replace this path planner algorithm with a neural network based solution.

Subsequently, the planned path is passed to the path tracking algorithm, which is responsible for keeping the car on the planned path. The path tracking algorithm is a simple carrot on a stick algorithm, which picks a point on the path in front of the car and tries to steer the car towards it, by minimizing the angle between the car's heading and the point on the path. [14]

1.5 Outline

We give outline of following chapters:

In section 2, related work, we first give an introduction to behavioural cloning, its history and its challenges. Next, we give a brief introduction to neural networks, especially recurrent neural networks and their advanced versions LSTM and GRU. Lastly, we talk about sequence to sequence (seq2seq) models and how they work.

In section 3, method, we describe our solution the problem of using behavioural cloning for path planning in the context of the Formula Student Driverless competition. We first describe the data generation process, first by generating virtual tracks and then driving expert trajectories on them. Subsequently, we describe the neural network model is trained to imitate the expert trajectories.

In section 4, experiments, we present results of experiments we conducted. First, we evaluate the performance of the model without any temporal or conditioning context. Next, we show results of experiments with the model conditioned on the type of the turn ahead and its learnt temporal context. We also evaluate the system's performance on a physical car driving on a real track. Lastly, we evaluate the model on the skidpad track.

In section 5, conclusion, we summarize the motivation, method and the results of the thesis. We then point out and discuss the limitations and challenges of the proposed approach.

2 Related Work

2.1 Imitation learning

The term imitation learning refers to a type of learning methods where agents learn to perform tasks by observing an expert demonstration. It includes methods such as apprenticeship learning, inverse reinforcement learning and behavioral cloning. It has been used in a variety of fields such as robotics, self-driving cars and natural language processing. [4][15][16][17]

2.2 Behavioral cloning

In this thesis, we focus on behavioral cloning, which is a type of imitation learning where we directly learn to approximate the function mapping observations to actions. For example, in the context of self-driving cars, a typical learning task would be to learn to map a camera image to the corresponding steering angle. In the self-driving car's context, these state-action pairs are typically collected by using a ground truth vehicle. A human drives the vehicle, while at every timestep recording the state (ie. camera image) and the action (ie. steering angle). [3]

Formally, given a dataset of state-action pairs:

$$\mathcal{D} = \{(s_1, a_1), (s_2, a_2), \dots, (s_N, a_N)\} \tag{1}$$

We optimize the criterion function J with respect to the parameters θ of the mapping function F_θ, which maps states to actions.

$$\min_\theta J(\theta) = \sum_{i=1}^{N} L(F_\theta(s_i), a_i) \tag{2}$$

Given that the dataset $\mathcal{D}$ is rich enough to cover the entire state space, the mapping function F_θ is expressive enough and if the assumption that a mapping function exists, that can map states to actions holds, then learned function F will be able to perform the task it was trained on, ie. drive a car. [3]

2.2.1 History of behavioral cloning

Behavioral cloning has been used in the context of self-driving cars for a long time. Its use originated in 1988, when Dean Pomerleau developed ALVINN (Autonomous Land Vehicle In a Neural Network).[1] ALVINN was a Chevrolet van, equipped with RGB camera, laser range finder, computer and steer by wire capabilities. The 30x32 input image and 8x32 laser scan were used as input for a 3-layer neural network consisting of less than 100 neurons in total. The network was trained on 1200 pairs of image and scan vector of a given scene and the corresponding steering angle. The

Figure 8: ALVINN [1]

achieved performance was comparable to the State of the Art traditional algorithms at the time. From the 1988 paper: *"Performance of the network to date is comparable to that achieved by the best traditional vision-based autonomous navigation algorithm at CMU under the limited conditions tested. Specifically, the network can accurately drive the NAVLAB at a speed of 1/2 meter per second along a 400 meter path through a wooded area of the CMU campus under sunny fall conditions."*

Figure 9: DAVE [2]

More than 15 years later, in 2004, LeCun et al. developed a 50cm off-road RC car called DAVE.[2] The RC car was equipped with two RGB cameras producing 149x58 resolution images and a steering rack. It was also equipped with a radio, sending sensory data to a remote computer for processing and receiving steering commands. The robot's 6-layer convolutional neural network was trained on 95,000 frames of images and corresponding steering angles. The data was recorded in several different types of environment such as in a forest, on a parking lot and in a grass field. The main goal of the training was to teach the robot to avoid obstacles and drive forward as long as possible. The robot managed to drive at 2 m/s and avoid a diverse range of obstacles such as table chairs, trees and human legs.

Another 12 years later, in 2016, Bojarski et al. from NVIDIA, utilizing modern developments in hardware and deep learning algorithms created a system called DAVE-2.[4] Similarly to ALVINN, it was regular car equipped with an RGB camera, a computer and steer by wire capability. This time, the task was to drive on real public roads. The team collected 72 hours of driving data at various locations in United States. They put emphasis on having for diverse set of weather and lighting conditions. Similarly to the original DAVE, DAVE-2 uses a convolutional neural network to output the steering angle. However, the DAVE-2's PilotNet has 8 layers and around 4-times as many parameters. They evaluated the network's performance based on the percentage of time the system drove autonomously without the human driver having to take control. In their test drives they managed to drive autonomously 98% of the time.

Today, companies like Comma.ai and Tesla are using behavioral cloning to train self-driving cars. [18]

2.2.2 Challenges and limitations

When training model with the approach of behavioural cloning, there are several issues one might run into. Usually, when training a model with supervised learning, we make an assumption about the data being i.i.d. (independently and identically distributed). For behavioral cloning, this assumption generally holds during training, but it doesn't hold during deployment. It doesn't hold, because the state at time $t+1$ depends on the action taken at time t. This means that the distribution we are sampling our states from during deployment is different from our train and holdout test set used for learning.

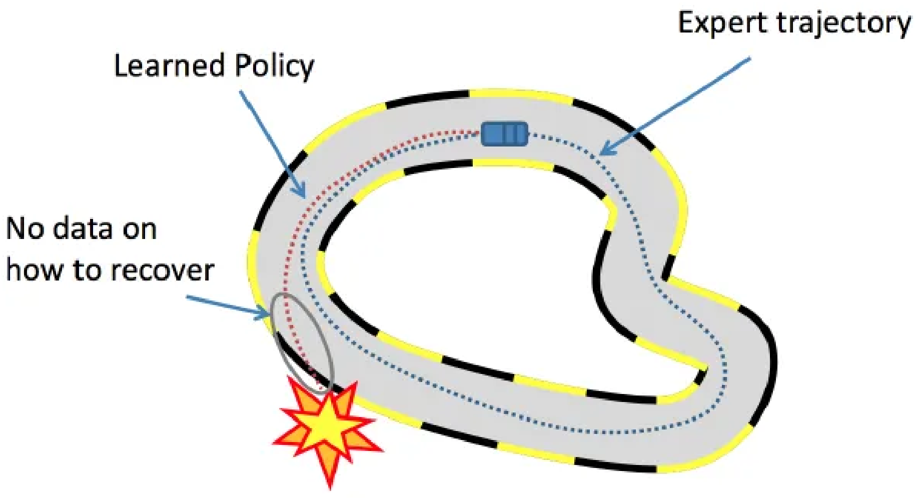


Figure 10: Out-of-distribution problem in behavioural cloning. [3]

We look at the Figure 10 sketch. The model, trained on the expert trajectory, outputs a steering angle which is close to correct, but inevitably with some error ϵ.

$$steering\_angle_t^* = F_\theta(s_t) + \epsilon \tag{3}$$

This has the effect, that the car leaves the expert trajectory on the track and the model is out-of-distribution in the sense, that the training set didn't contain similar situations.

One approach to solving this problem, would be to contain all possible situations in the dataset. This is practice is not possible due to financial and time constraints. And in the case of recording data on public roads even dangerous and illegal.

A more practical approach, is to slightly offset the car from the expert trajectory, to "teach it to recover". In the 1988 ALVINN experiments, they would place the van on the edge of the road at different angles and ground truth the steering angle so that it would recover to the center. This strategy wouldn't be possible for the 2016 NVIDIA DAVE-2 project, since it as trained on driving data collected on public roads. Their approach was, using computer vision, offset the translation and rotation of the car in the image and adjust steering angle accordingly. This approach is depicted on Figure 11 and a similar approach is utilized in this thesis.

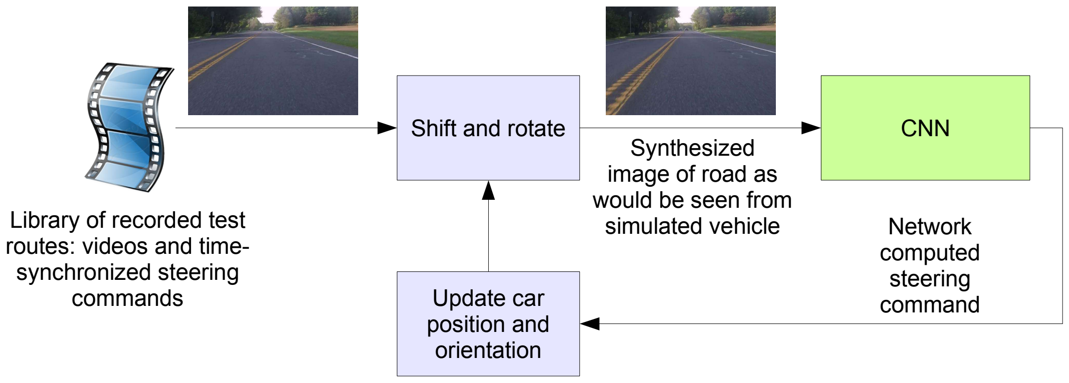

Library of recorded test routes: videos and time-synchronized steering commands
Shift and rotate
Synthesized image of road as would be seen from simulated vehicle
CNN
Network computed steering command
Update car position and orientation

Figure 11: Camera image augmentation to simulate out-of-distribution situations in the dataset. [4]

2.3 Conditional imitation learning

Another assumption that is made in the standard formulation of behavioral cloning, is that a function mapping the perception data to actions exists. This assumption holds for simple situations, such as trying to stay centered on the highway. However, when the car reaches an intersection, there are suddenly multiple possible trajectories, that are correct in different contexts. Depending on the final destination, it might be correct to turn left or to turn right.

In the paper End-to-end Driving via Conditional Imitation Learning (2018) by Codeville et al., they expand the behavioral cloning formulation in Equation 2, to include a high level command c_t, as shown on Figure 12 diagram.[5] The c_t can represent commands such as "turn right at the next intersection" or even the maximum allowed speed in the area. This command is given as ground truth during training, but can be arbitrarily chosen during deployment.

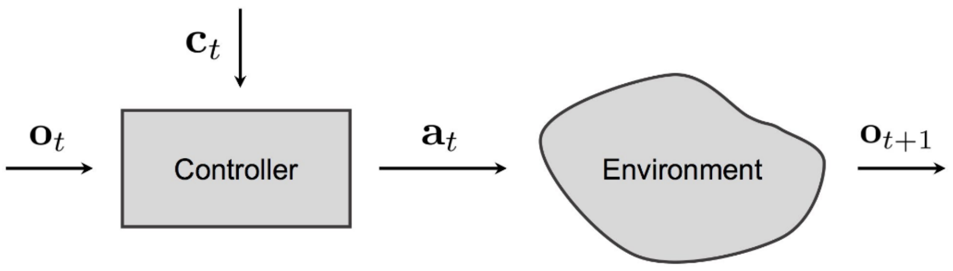


Figure 12: Conditional imitation learning. [5]

2.4 Neural Networks

Neural networks are a popular machine learning model, which is capable of learning complex functions from data.[19] They are in some sense similar to the human brain, that they consist of interconnected layers of neurons, which model a function mapping inputs to outputs.

Neural networks were first introduced in the 1940s, but have especially gained popularity in 2010s and 2020s. In 2012, the AlexNet, a convolutional neural network based image classifier, won the ImageNet competition by a large margin, which started a new era for neural networks.[20] Since then, neural networks have been used with great success in variety of fields such as computer vision, natural language processing and robotics. The recent success of neural networks is typically attributed to the increase of available data, due to the internet and the increase in computational power, mainly due to the development of GPUs.[21]

We now give a brief overview of the most common types of neural networks. We mainly focus on recurrent neural networks and their variants, as they are used in this for the path planning model presented in this thesis.

2.4.1 Feedforward Neural Networks

Feedforward neural networks are the vanilla neural network architecture. The network is composed of fully connected layers of neurons, where each neuron is connected to every neuron in the previous layer. Mathematically a fully–connected layer can be expressed as:

$$a_t = \sigma(Wa_{t-1} + b) \tag{4}$$

where a_{t-1} are the input activations, W is the matrix representing the weights of the connections, b is the bias vector and σ is the sigmoid activation function. In practice, networks are composed of multiple such fully–connected layers.

These networks are trained using the backpropagation algorithm, which is a gradient descent based algorithm for teaching the network to minimize a loss function. The loss is computed as the difference between the network's output and the ground truth label. We compute the gradient of the loss with respect to the network's parameters and update the parameters in the opposite direction of the gradient, to minimize the loss.[22]

2.4.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a type of neural network used for modeling data with variable–sized inputs. Since the feedforward neural networks have a fixed input and output size, they are not the best choice for modeling data of variable size. RNNs are able to model sequences of data, such as language, time series data or signals.

The key idea behind RNNs is that they use a fully–connected network, which receives the input vector and its previous output, called the hidden state or context vector. This way, to process a sequence of length T, the network is run T times, each time receiving the current input vector x_t and the context vector h_{t-1} and outputting the context vector (h_t). The output of the network is the final context vector h_T. This way, the network is able to compress the information about the entire sequence into a fixed–size vector. This vector can then be used as input for following fully–connected layers, whose output could be a classification or regression result.

Mathematically, a single iteration of an RNN can be expressed as:

$$h_t = \sigma(W_h h_{t-1} + W_x x_t + b) \tag{5}$$

2.4.3 Long Short-Term Memory (LSTM)

While RNNs are able to model sequences of data, they have been shown to have problems with retaining information over long sequences. This can lead to poor performance on tasks such as next word prediction, where the correct prediction might depend on a word from several paragraphs back. The Long Short-Term Memory (LSTM) network was introduced to address this issue. [23]

The LSTM network is a type of an RNN. It extends the RNN by adding several terms to the hidden state computation. The LSTM has not only the hidden state h_t, but also introduces the cell state c_t, which is able to store information over long sequences. The LSTM contains three so called gates, which are neural network layers tasked with controlling the flow of information in the cell state.

The three gates are the forget gate, the input gate and the output gate. The forget gate decides, based on the current input and the previous hidden state, which information to forget from the cell state. It achieves this by outputting a vector of values between 0 and 1, which is then multiplied element–wise with the cell state. The other two gates, the input and output gate, decide which information to add to the cell state and pass to the output.

The LSTM network has been shown to perform better than RNNs on tasks that require modeling long–term dependencies.

2.4.4 Gated Recurrent Units (GRU)

The Gated Recurrent Unit (GRU) is a type of RNN, which is very similar to the LSTM network. Introduced by Cho et al. in 2014, the GRU network is also able to model long–term dependencies in the data, while being of simpler design than the LSTM network. [24]

The GRU network only has two gates compared to the LSTM's three. It also merges the cell state and the hidden state into a single vector. This makes the GRU network simpler have the same interface as a regular RNN, while still being able to model long–term dependencies. When compared to the LSTM network, both networks have been shown to perform similarly.[25]

The GRU networks are used in this thesis for processing of input sequences of traffic cone detections.

2.5 Sequence to Sequence models

Sequence to Sequence models (seq2seq) are a type of neural network architecture, which is used for mapping variable length input sequences to variable length output sequences. Typical use cases for seq2seq models are would be machine translation and speech transcription.[26]

The seq2seq model consists of two parts, the encoder and the decoder, both of which are typically RNNs or their advanced variants such as LSTMs or GRUs. The encoder processes the input sequence and outputs a fixed–size context vector. This context vector, which contains the information about the entire input sequence, is then passed to the decoder.

The decoder is an RNN, which takes the context vector h_t, outputs the next context vector h_{t+1}, but also outputs the prediction for the next element in the output sequence. Typically, this prediction is then also fed back to the decoder together with context vector.

In this thesis, we use the seq2seq encoder–decoder architecture to model the path planning problem. Where the input sequence is the list of detected traffic cones in the scene and output is a sequence of 3D path points.

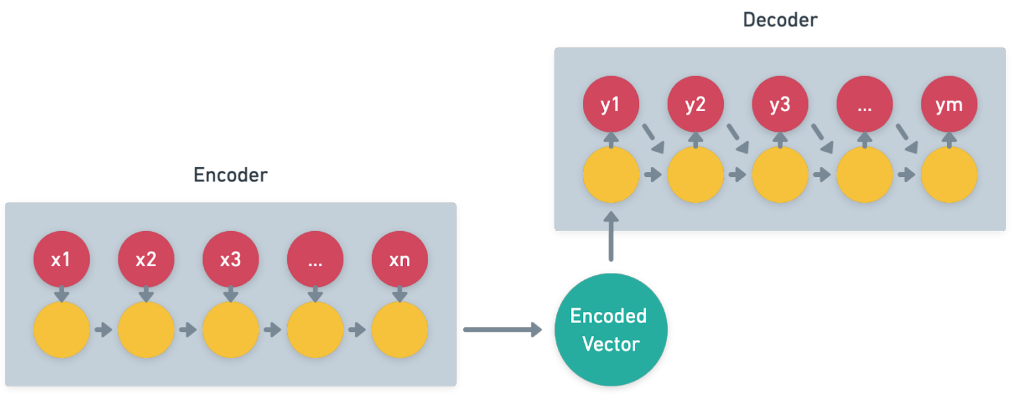


Figure 13: The sequence to sequence model. The input sequence is iteratively turned into an encoded vector, which is then passed to the decoder, which uses to iteratively generate the output sequence.[6]

2.6 Racing line optimization

The racing line optimization is a well–known problem in the field of autonomous racing. It is the problem of finding the optimal trajectory for a car to follow, in order to minimize lap time. There are several approaches to solving this problem including geometric methods, optimal control optimization and even reinforcement learning.[27][28][29]

In this thesis, we use a gradient descent based optimization method to find the in–some–sense optimal racing line, which was presented in the bachelor thesis by Michal Horáček "Finding the Fastest Trajectory for Autonomous Student Formula".[8].

In the mentioned work, the car's trajectory is represented as a sequence of 2D points, which are connected, forming a piecewise linear path. The algorithm starts with an initial center line. For each point on the path, a unit orthogonal vector is computed. Then, in the direction of this orthogonal vector, each path point is moved by distance t, which is limited by the track boundaries.

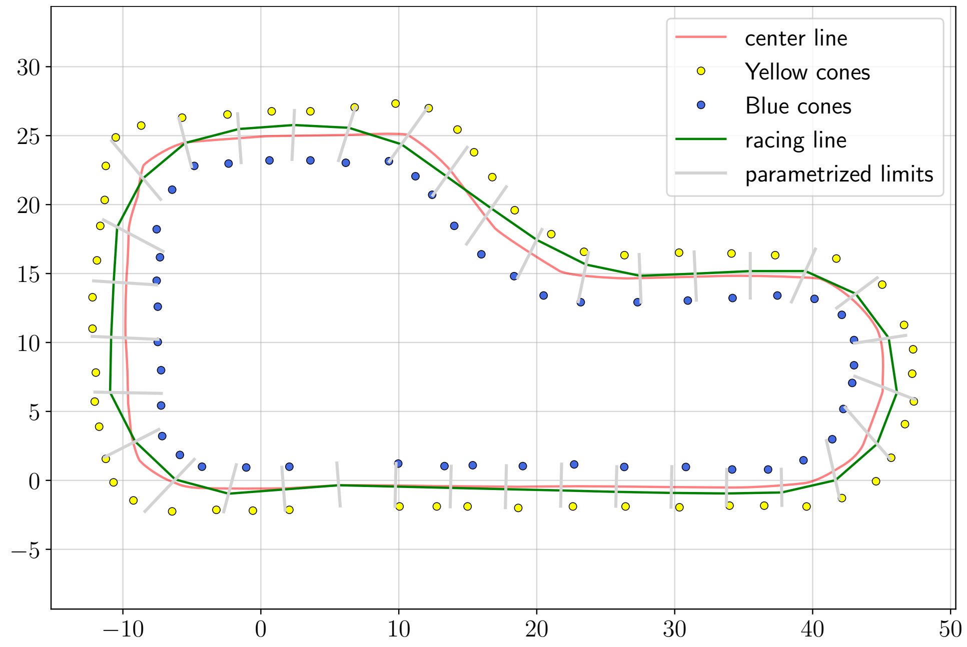


Figure 14: Track optimization process.

In Figure 14 we can see the visualization of this process. Inside the track denoted by blue and yellow traffic cones, car trajectories are visualized. The red line represents the initial center line and the green line the optimized racing line. The grey lines represent the orthogonal vectors, which are used to parameterize the path points and move them in the direction that minimizes a given loss function.

There are several loss function that can be used to optimize the racing line. The thesis proposed three approaches: minimizing the overall path length, minimizing the path curvature and directly optimizing the lap time. For the optimization of the racing line for the path spanning the entire track, the lap time objective was shown to be the most effective. However, for optimizing only a part of the track, the curvature objective could be more suitable, as it is the objective that also maximizes the potential speed of the car.

In this thesis, we use the racing line optimization algorithm for estimating the optimal path for individual track segments consisting of a single turn. Which are then used to generate training data for a behavioral cloning based path planning model.

3 Method

In this section, we describe the methods used for generating synthetic data of different types of race tracks and using them to train a neural network for planning a path. In the data generation process, four types of tracks are generated: general tracks, chicane tracks, right-angle tracks and hairpin tracks. The chicane, right-angle and hairpin tracks consist of only a single turn, while the general tracks consist of a full circuit with multiple turns. The general tracks contain the center line of the track as the ground truth path, while the other turn-specific tracks contain a curvature optimized path.

Subsequently, these tracks, together with their ground truth paths, are used to generate clips of the car driving on the track. These clips are composed of pairs of vision inputs, which are traffic cone detections and the optimal path to be planned by the model. We then train a sequence-to-sequence model to predict the optimal path given the traffic cone detections on the dataset of clips of driving.

3.1 Track generation

Before explaining the track generation process of different types of tracks, we give a definition of what we mean by a track. A track is a set of traffic cone positions and list of path points. A traffic cone is represented by its 2D position in the world frame and its type, which makes it a triple (x, y, t), where x and y are the 2D position of the cone and t is the type of the cone. Where $t \in \{0, 1, 2, 3\}$, where 0 is a yellow cone, 1 is a blue cone and 2 is an orange cone and 3 is a big orange cone. In terms of matrices, the cones are represented by a matrix of size $N \times 3$, where N is the number of cones in the track.

The track path is represented by a list of points in the world frame, where each point is a pair of (x, y) coordinates. The path points in the list are ordered as the car drives along the path from start to finish. The path is represented by a matrix of size $M \times 2$, where M is the number of path points.

A track is then a tuple (C, P), where C is the matrix of traffic cone positions and types and P is the matrix of path points.

Algorithm 1 Race Track Generation Algorithm

```
1: Generate a set of random points (blue dots in Figure 15).
2: Compute convex hull of the points (blue polygon in Figure 15).
3: Compute the midpoint of each pair of points in the convex hull and displace it
   by a random amount. (green polygon in Figure 15).
4: Given both a minimum angle and a minimum distance threshold, push apart
   the set of points resulting from steps 2 and 3 to try to guarantee that:
5: Obtain the final layout by interpolating all the points from step 4 with splines,
   passing through all the points (red curve in Figure 15).
```

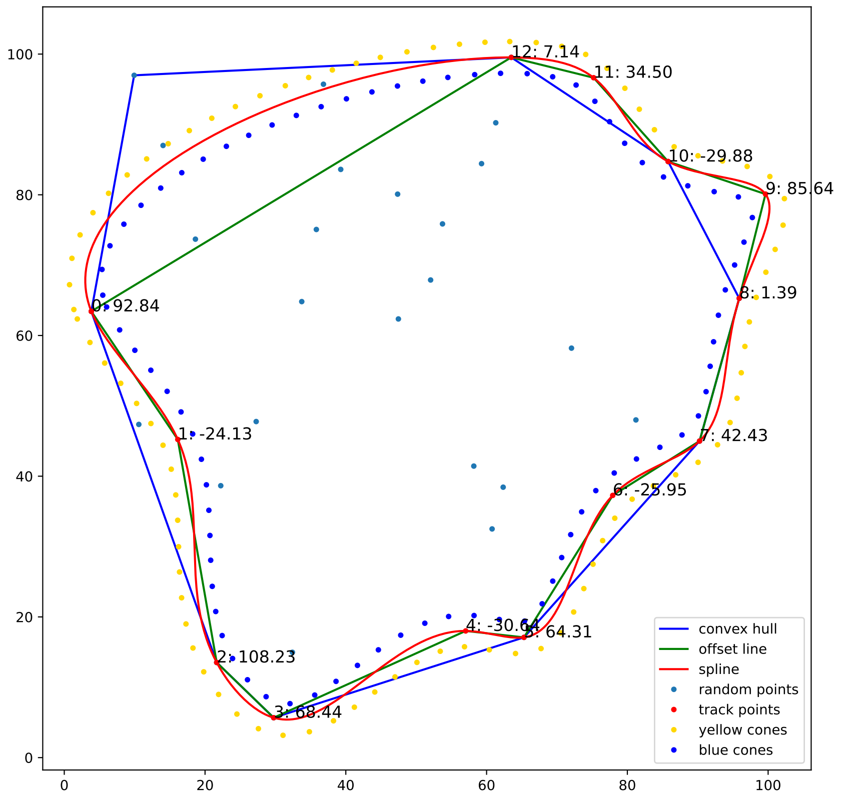


Figure 15: Generated circuit track with Algorithm 1, with each step of the generation process visualized.

3.1.1 Circuit tracks

The first type of track we generate is a randomly generated circuit track. It is generated by randomly sampling a set of points in the world frame, computing their convex hull and the shifting the convex hull points orthogonally by a random distance. These points are then interpolated with splines to obtain the final track layout, as is described in Algorithm 1.

The spline center line is then used to populate the track with traffic cones. We iterate over the center line and every d meters we place a blue and yellow traffic cone orthogonally to the left and right side of the center line respectively. The d parameter is a hyperparameter of the track population algorithm which determines the density of the cones on the track, we randomly sample it from a uniform distribution between 3 and 6 meters. There is also the track width parameter, which determines how far the cones are placed from the center line, we set this parameter to 4.5 meters.

3.1.2 Chicane turns

The second type of a track we generate is a chicane turn. The chicane turn is an S-shaped turn that is typically found on racing circuits. This type of turn is the first turn-specific track that we generate. We picked the chicane turn because the difference in performance between driving along the optimal path and the center line is quite significant. This leaves a lot of room for improvement in the car's lap time if the optimal path is taken through the turn. A path planner that can plan the optimal path through this turn will result in a significant improvement in the speed at which the car can drive through the turn resulting in reduction of the overall lap time.

The chicane turn consists of two turns in opposite directions (left/right or right/left). In order to minimize car's lateral force and maximize the speed through the turn, the optimal path a smooth narrow S-curve going through the middle of the turn.

The chicane turn is generated by first generating a center line that which consists of a initial straight line, followed by a sine curve and then the final straight line. Its defining parameters are the lengths of the initial and final straight lines and the amplitude and frequency of the sine curve. There are two types of chicane turns: left-right and right-left. For the left-right variant the sine curve is kept as it is, while for the right-left variant the sine curve is flipped along the x-axis.

The optimal path through the chicane turn is computed by taking the center line and applying curvature optimization algorithm to it, as described in subsection 2.6. We show examples, along with their optimal racing paths of chicane turns in Figure 16.

3.1.3 Right angle turns

Next we generate right angle turn tracks. The right angle turn is a second type of turn-specific track that we generate. Again, we picked the right angle turn because the it allows for a significant increase in speed if the optimal path is taken through the turn. The optimal path through the right angle turn is to hug the side of the turn and then make a sharp turn through the apex of the turn.

We generate the right angle turn by first generating a center line, consisting of a straight line followed by a sharp turn, generated as a quarter circle and then another straight line. The defining parameters of the right angle turn are the lengths of the initial and final straight lines and the radius of the quarter circle. Again, there are two types of right angle turns: left and right, depending on the orienttion of the quarter circle.

To compute the racing line through the right angle turn, we again apply the curvature optimization algorithm to the center line. However, to achieve a consistent pattern for driving through the right angle turn, we also add a constrain to the optimization algorithm, that only a portion of the center line is changed. Particularly, we only allow the 10 meters before and after the apex of the turn to be optimized, while the rest of the center line is kept as it is. This is done to ensure that the car is not expected to start setting up for the turn even before it sees it, which would be unrealistic and could lead to instability in training of the model.

Examples of left right-angle and right right-angle turns, along with their racing lines are shown in Figure 16.

3.1.4 Hairpin turns

Finally, we generate hairpin turn tracks. The hairpin turn is a third type of turn-specific track that we generate. It is the most challenging turn to drive through, as it requires the car to make a very sharp turn. There are several types of strategies to drive through a hairpin turn, but the most common one is to drive along the outside of the turn and then make a sharp turn through the apex of the turn.

We generate the hairpin turn by first generating a center line, consisting of a straight line followed by a half circle and then another straight line. The defining parameters of the hairpin turn are the lengths of the initial and final straight lines, the radius of the half circle and the orientation of the half circle (left or right).

For this type of turn, we compute the racing manually, as the curvature optimization algorithm does not work well on the hairpin turns, due to its circular nature. To compute the racing line, we increase the radius of the half circle to be a given distance away from the track boundary (e.g. enough for the car to fit next to the cones). We then shift the path downwards, towards the inside of the turn to take a more aggressive early turn through the hairpin.

Again, example of left and right hairpin turns, along with the racing lines are shown in Figure 16.

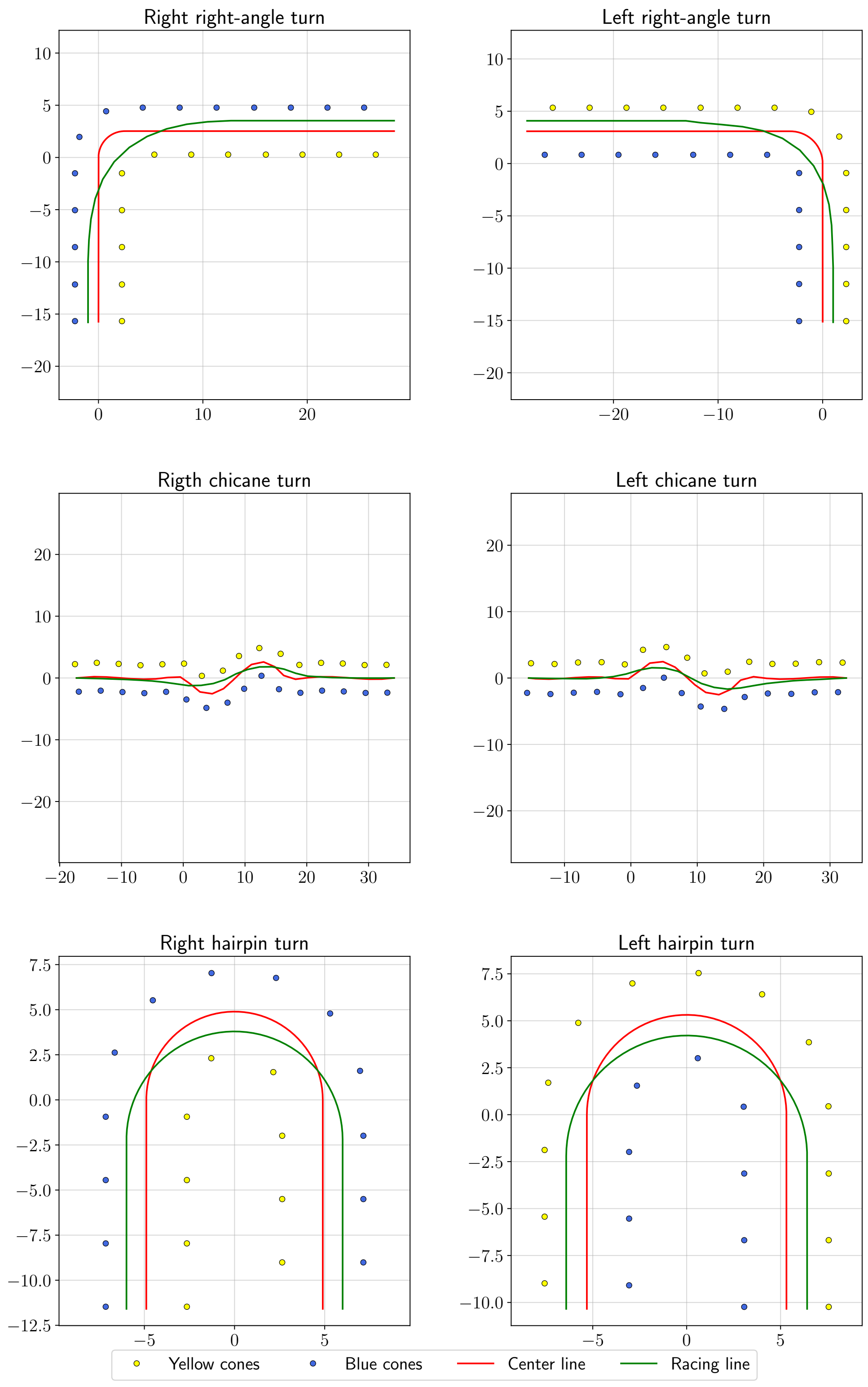


Figure 16: An example of a chicane, right angle and hairpin turn.

3.2 Clip generation

After generating the tracks, we generate clips of the car driving on the track. We do this by simulating the car driving on the track and recording the traffic cone detections and the optimal path each timestep. The car is placed at the start of the track path and is set in motion at a constant speed. The path is planned as if the car is driving autonomously and the optimal path is generated by the path planner. The car then drives along the path using a simple carrot on a stick controller, which steers the car towards the next point on the path.

At each time step, the detections of the traffic cones are simulated by taking a cone shaped field of view crop of the track traffic cones in front of the car's position. This way we simulate the car's vision input as if it would be detecting the cones in real life. Same is done for the path, we take a crop of the path points in front of the car's position.

For each clip, we also record the turn type of the track, which is then used to train the model to predict the optimal path conditioned on the turn type. Therefore, the clip is a tuple (C, P, T), where C is an $N \times 3$ matrix representing the traffic cone detections, P is an $M \times 2$ matrix representing the path points and T is the turn type token.

For a portion of the clips, we add a cumulative gaussian noise to the car's steering angle commands. This is done to make the model more robust to going off the optimal path and to be able to recover from such situations. The motivation for this is described in subsection 2.2.

3.2.1 Data augmentation

In order to make the model robust to different types of noise that might occur, while driving on a real track, we augment the traffic cone detections in the clips in several ways. Firstly, we simulate the error in the traffic cone detections by adding random noise to the cone positions. In particular, we add a noise translation vector with y-component sampled from a normal distribution with mean 0 and standard deviation of 0.1 and x-component sampled from a skewed normal distribution with mean 0, standard deviation of 0.1 and the shape parameter of 1.0. This models the fact that the error of the cone detection is typically larger in the x-direction (forward and backward) than in the y-direction (left and right).

Secondly, we simulate the possibility for cone class misclassification by randomly changing the class of a detected cone with a small probability. This probability is conditioned on the true class of the cone, as the misclassification is more likely to happen between cones of similar colors, such as orange and big orange cones.

Another type of noise is a false negative detection, where a cone is not detected by the car's vision system. With a small probability, we remove a cone from the vector of detections. Similarly, we simulate the false positive detections by adding a cone to the detection at a random position with a small probability.

Lastly, we simulate the error in the building of the track itself by adding a random gaussian noise to each traffic cone position in the track before beginning of each drive.

3.3 Autocross clips

As mentioned before, a clip is a sequence of triplets (C, P, T), where C is the matrix of traffic cone detections, P is the matrix of path points and T is the turn type token. The turn type token is used to condition the model to predict the optimal path for the turn coming up ahead.

For the autocross discipline, we distinguish between seven turn conditioning tokens:

Turn type	**Token ID**
CENTER_LINE	0
LEFT_CHICANE	1
RIGHT_CHICANE	2
RIGHT_RIGHT–ANGLE	3
LEFT_RIGHT–ANGLE	4
RIGHT_HAIRPIN	5
LEFT_HAIRPIN	6

Table 1: Enumeration of the conditioning tokens for path planning model driving in the Autocross discipline.

These tokens are used to condition the model to change its driving behavior to suit the upcoming turn. For example, when the model is conditioned with the `LEFT_RIGHT-ANGLE` token, it should predict the path for the car to hug the right side of the track and then make a sharp left turn at the right moment.

3.4 Skidpad clips

In addition to learning to plan the optimal path for several types of turns and for general tracks, we also generate clips for learning to perform a skidpad run. As mentioned in the /refsec:skidpad, the skidpad is an eight-shaped track that is used to test the car's lateral acceleration.

From the perspective of planning a path for the car at each time step, the skidpad is uniquely challenging because depending on the lap counter, the car needs to drive take a different path through the main intersection. Due to this needed conditioning and the complexity of the traffic cone scene in the middle of the track, the simple geometry based path planning algorithms do not work on the skidpad as they would on the trackdrive tracks. The typical solution to this problem is to use a localization algorithm to determine the car's position on the track and then just follow the precomputed path for that position.This is possible because the skidpad layout is known beforehand and is always the same. The typically used algorithms include Extended Kalman Filter and Monte Carlo Localization.

We propose a different approach to driving the skidpad. We generate clips of the car driving on the skidpad in different context and record the traffic cone detections and the optimal path for each time step. The context, determined by the lap counter, is gives the car its turn direction at the main intersection. There are three possible turn directions: left, right and straight, as shown in Figure 17. During the first two passes, the car drives around the right circle, then it drives twice around the left circle and finally it drives straight through the middle intersection and into the braking zone.

The skidpad clip is also just a tuple (C, P, T) as the trackdrive clips just with the T token representing a skidpad turn direction instead of a track turn type. For each skidpad direction token, we generate the clips by simulating the car driving from a particular starting position and then taking the correct turn direction at the main intersection with accordance with the current direction token. The all possible situations of driving through the skidpad center can be divided into 9 situations, which are defined by the starting position of the car: left circle, starting zone, right circle and the turn directions: turning left, driving straight and turning right. Of these 9 unique combinations only 5 occur during a skidpad run. Namely, the car starts in the starting zone and drives right, then it finds itself in the right circle and the turns right again, then it finds itself in the right circle, but takes the left turn, then it starts in the left circle and takes the left turn and finally it finds itself in the left circle and drives straight into the braking zone. The division of these situations between the three skidpad tokens is shown in Figure 17.

These 5 situations can be further divided into 3 behavioural patters, which are based on the way the car drives through the main intersection. It either turns right, turns left or drives straight through the finish line into the braking zone. This leaves us with the following skidpad turn type conditioning tokens:

Turn type	**Token ID**
SKIDPAD_RIGHT	0
SKIDPAD_LEFT	1
SKIDPAD_END	2

Table 2: Enumeration of the conditioning tokens for path planning model driving in the Skidpad discipline.

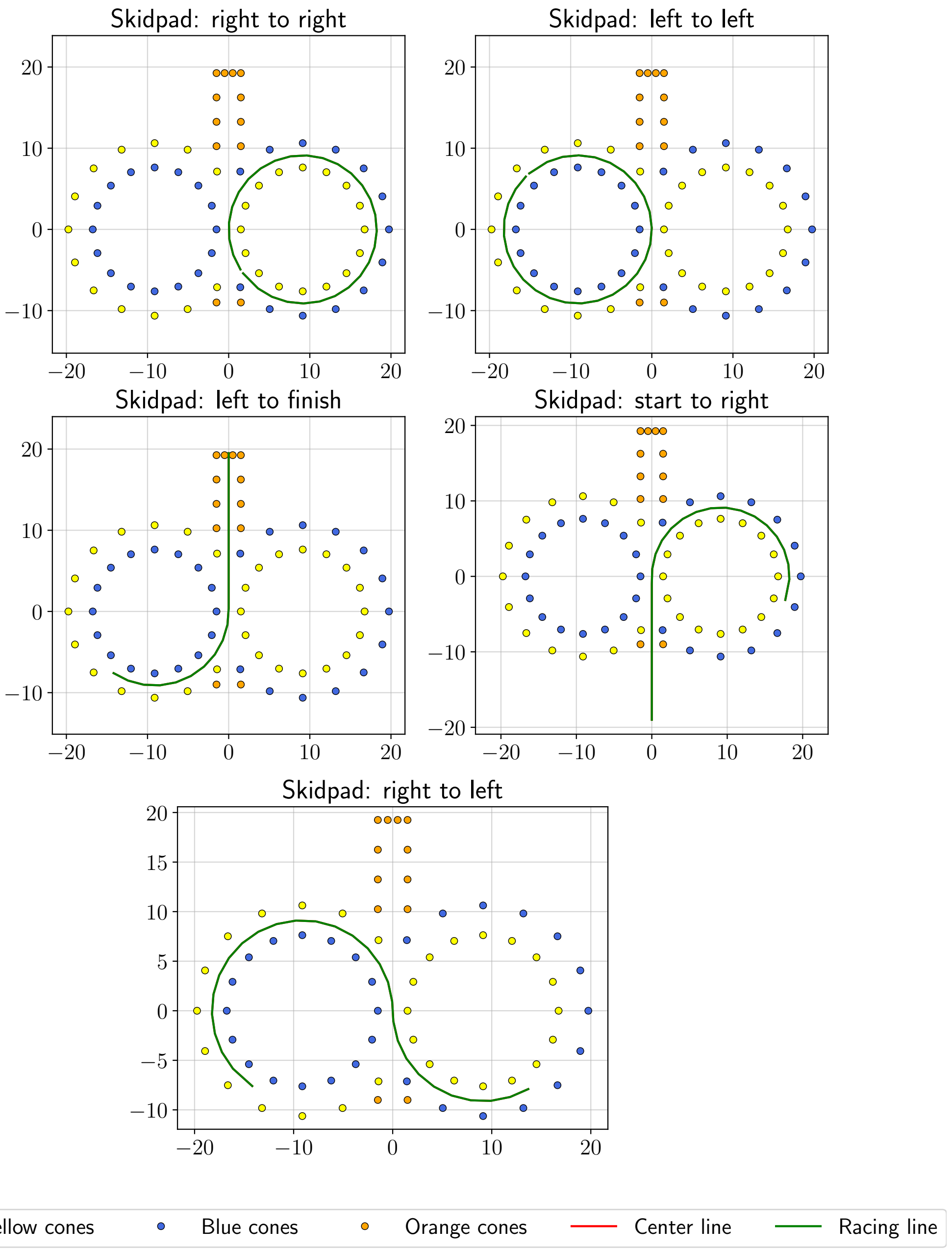


Figure 17: Examples of 5 situations of skidpad turns. Each situation is defined by where the car starts and which turn it takes at the center intersection.

3.5 Model architecture

The model architecture is a sequence-to-sequence model using a GRU (Gated Recurrent Unit) encoder-decoder architecture. This architecture is used for tasks where the input and output sequences have different lengths such as machine translation. In our case, the input sequence is the traffic cone detections and the output sequence are the path points. At each time step the car can see a different number of cones and it may need to output a different number of path points, making this architecture suitable for our task.

The full model architecture is shown in Figure 18. The model consists of a GRU encoder, a GRU decoder and a fully connected layer in between, that also incorporates the turn type token into the model. The model first encodes the variable length input sequence into a fixed length context vector. Following this, we append the one-hot encoded turn type token to the context vector and pass it through a fully connected layer to obtain a new context vector, which contains the information about the current turn type. Finally, the decoder takes this context vector and decodes it into the output sequence of path points.

During each output path point prediction, the GRU decoder also outputs a context vector. We take the final context vector of the current time step and pass it as the initial context vector the next time step's GRU encoder. This allows the model to take into account information from previous time steps. We believe this is a crucial attribute for the model to be able to perform complex maneuvers such as setting up for a turn.

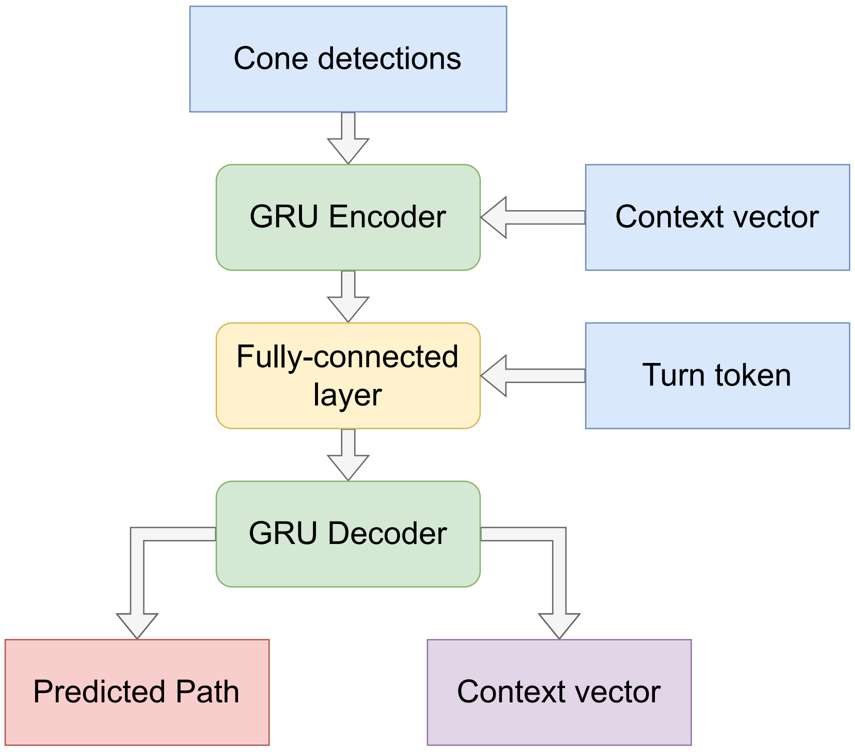


Figure 18: The path planning model architecture. Cone detections, turn type token and the previous tokens are passed through the encoder and the decoder to predict a path.

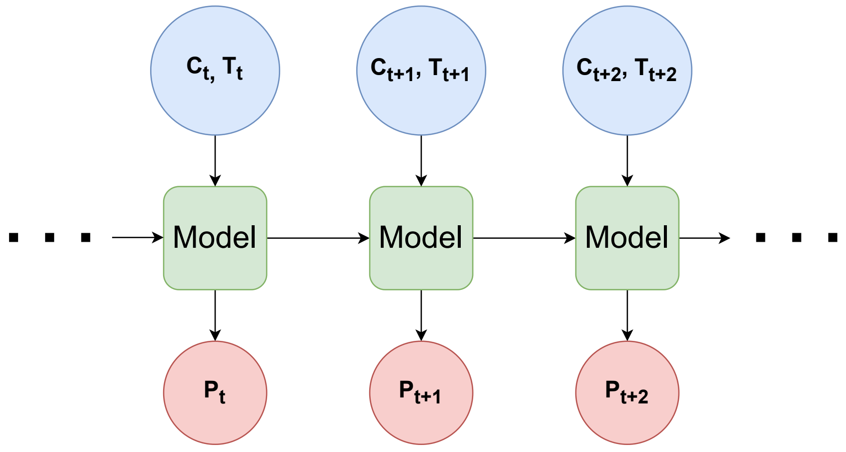


Figure 19: The processing of a temporal sequence of cone detections and turn type tokens to predict the path in each time step.

3.6 Training

We divide the training process into two stages: pre-training and training on clips.

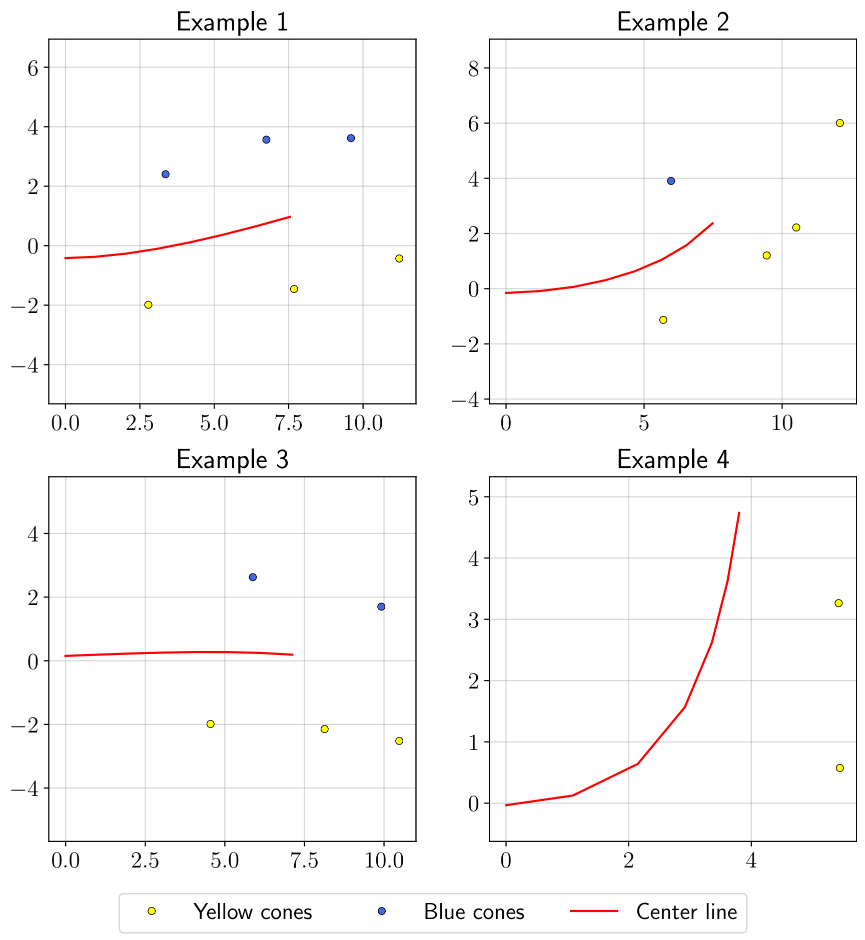


Figure 20: Examples of sequences from the dataset used for pre-training. Single frames with no temporal information or turn type context.

3.6.1 Pre-training

In the pre-training stage, we effectively train the model on clips of length 1, containing only a single sequence of traffic cone detections and corresponding path points. This is done to make the model learn the basic mapping between the traffic cone detections and the center line. It could be seen as first learning to imitate the center line path planner algorithms, which do not take into account any temporal or turn type information.

The data for this is generated only on the generic circuit tracks. We place the car at a random position on the track, along the center line and take a single snapshot of the traffic cones the car would see and the appropriate center line, the path planner should predict. In order to give the model more robustness, we also offset the car's position from the center line by random translation vector and randomly rotate it. A sample of the pre-training sequences is shown in Figure 20.

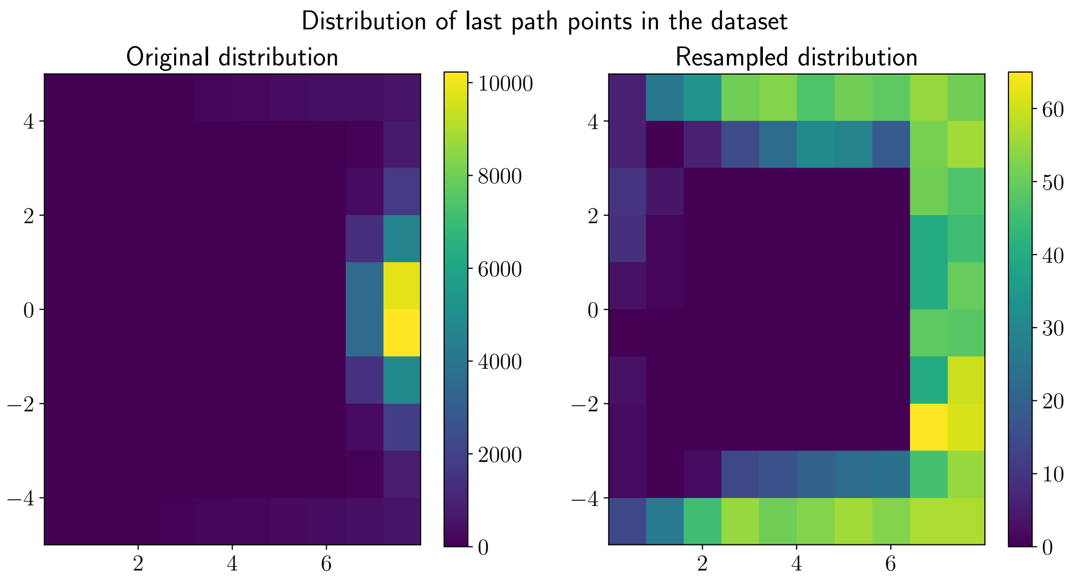


Figure 21: Distribution of the sequences dataset.

For the pre-training dataset, we first generate 30,000 sequences. Due to the nature of the track generation algorithm, most of the sequences are of the car driving straight. In order to balance the dataset, to contain more sequence of the car driving through turns, we resample the dataset. We divide the dataset into a 2D–histogram over the positions in the car's local coordinate frame and then, where each data point is a represented by the last path point of the sequence. We then resample the dataset, so that sequences with the last path point of the sequences is uniformly distributed over the edges of the histogram. The distribution of the dataset before and after resampling is shown in Figure 21.

3.6.2 Training on clips

In the second stage of training, we train the model on the clips of the car driving on the different types of tracks. The clips are generated as described in the previous section and are of varying lengths. As we can see, the clips are of varying lengths, with minimum length set at 3 seconds and maximum set at 7 seconds.

Just as in the pre-training stage, we use the Adam optimizer and batch size of 32. We first train the model for 100 epochs with a learning rate of 10^{-3} and after that we reduce the learning rate to 10^{-4} and train for another 100 epochs. We use the mean squared error loss function to train the model.

We train two separate models, one for the general circuit tracks, which appear in the autocross discipline and the other to drive the skidpad track. The sizes of the datasets for the autocross and skidpad models are shown in the following tables.

Turn type	**Number of clips**
CENTER_LINE	232
LEFT_CHICANE	126
RIGHT_CHICANE	143
RIGHT_RIGHT–ANGLE	119
LEFT_RIGHT–ANGLE	131
RIGHT_HAIRPIN	66
LEFT_HAIRPIN	88

Table 3: Clip count for each turn type token used for training of the Autocross path planning model.

Turn type	**Number of clips**
SKIDPAD_RIGHT	122
SKIDPAD_LEFT	116
SKIDPAD_END	113

Table 4: Clip count for each turn type token used for training for the Skidpad path planning model.

3.7 Deployment

When running in the car, the model is ran in real-time on the car's onboard computer. In each iteration of the main processing node, it receives the traffic cone detections from the vision system. It then outputs a path, which is then used by a path tracking controller to steer the car according to the path.

So far, we have only described how to train the model to drive on certain track segments, while providing the model with the conditioning token for the situation it is in. This is simple in simulations, but when the model is deployed to drive a full lap on a track, we need to provide the model with the correct turn type token at each time step. We now describe how we handle this situation for the autocross and skidpad disciplines respectively.

3.7.1 Autocross

In the autocross discipline, the car drives on a track with an unknown layout. In the competition rule book, it is stated, that the members of the team can walk the track at a designated time before the autocross event.[7] They cannot use any electronic devices to record or map the track layout.

We plan to utilize this rule to our advantage. During the track walk, we will count the number of cones on each side of the track, which typically have a constant density (e.g. 1 pair of cones every 5 meters). This will then give us the approximate distance to each turn from the start of the track. Using this information, we can then determine the turn type token for each segment of the track, based on the driven distance by the car.

For the second attempt at the autocross, we can then use the collected information from the first drive to make the estimation of turn type segments more accurate. This way, we can provide the model with the correct turn type token at each time step, which will allow it to plan the optimal path through the track.

3.7.2 Skidpad

Deploying the model in the skidpad discipline is simpler. The conditioning token provides the model with the about information about how to turn at the main intersection. Due to the nature of the skidpad track, this can be estimated by counting the number of times the car has driven through the start–finish line. If it has been 0–1 times, the car should turn right, it it has been 2–3 after the last left turn, the car should drive straight into the braking zone.

4 Experiments

In this section we evaluate the performance of the path planner neural network, as described in subsection 3.6. First, we evaluate the pre-training model on the dataset of single-frame clips it was trained on. We show that the model is able to predict the centerline path of the track given only a single frame of traffic cone detections. However, we show examples where the model struggles to match the ground truth path due to scene ambiguity.

Secondly, we present the training evaluation, where we evaluate the model on the dataset of clips of expert trajectories it was trained on. Thirdly, we evaluate the model in a virtual environment on a testing set of tracks similar to the ones the training clips were generated from. This way we can evaluate how well the model generalizes to working in dynamic environments, where its perceptions are influenced by the actions it takes. We perform this evaluation for both the autocross and skidpad models.

Next, we evaluate the model on a full virtual circuit track, which contains some of the specific turns it was trained on, as well as some more ambiguous turns, where the model should follow the centerline. We compare the model to the classical geometric path planner, which is used as a baseline for the experiments.

For the skidpad model, we also evaluate it by making the model drive a full skidpad run. We compare the model to a classical solution that uses Monte Carlo Localization to estimate the correct path to take. We show that the model is able to compete with the classical solution.

Finally, we evaluate the autocross model on a physical car on a real-world track. We analyze its driving performance and compare it to the results from the virtual evaluation, as well as to the performance of the classical geometric path planner in the same conditions.

4.1 Single frame model

As described in subsection 3.6, before training the model on clip sequences, which contain several seconds of temporal context, we first pre-train the model on a dataset of single-frame clips. This is done to first teach the model the basic task of predicting the centerline path of the track given only the current traffic cone detections. It is the exact same task as the one performed by the classical path planner.

In figure Figure 22, we show the progression of training and validation loss of the model during training. We can see that the validation loss reaches root mean squared error of just above 0.5 meters after 80 epochs of training. The error is calculated across all individual points of the predicted path, which gives it a good degree of interpretability.

We now evaluate the model on a test set of single-frame clips, which were not seen during training and present some of the best, worst and random sequences of the model's predictions.

First, we show the sequences with the lowest error. In Figure 24, we can see that the model is able to predict the centerline path of the track with high accuracy

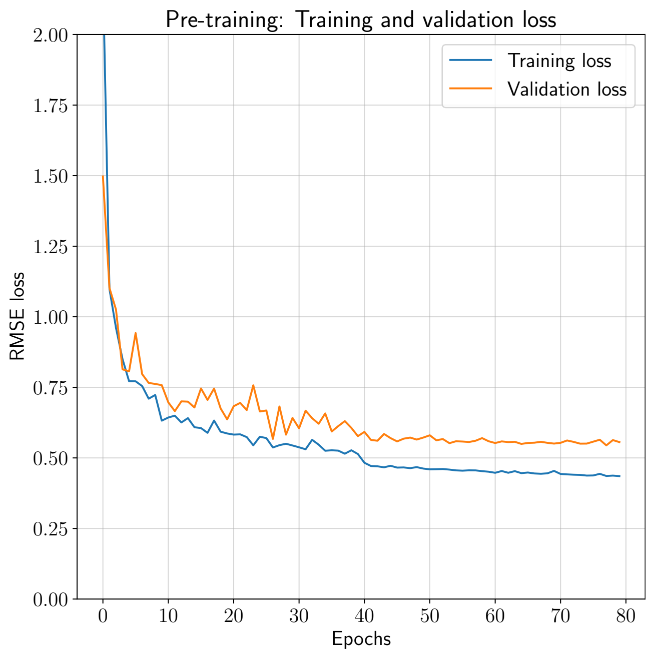


Figure 22: Training and validation loss progression for the pre-training of the model on single-frame clips.

even on examples with non-trivial turns and traffic cone placements. Also, notice how the model is able to predict its deviation from the centerline path, by correctly predicting the position the first path point away from the car, but on the centerline.

Next, we look at the sequences with the highest error. In Figure 24, we can see that the model struggles with predicting the path in cases where the correct centerline is ambiguous, even for a human observer without additional context. However, we notice that the predicted paths are still rather reasonable, given the context of the scene. The model learns to respect the track rules, such as blue and yellow cones marking the track boundaries on the left and right side, respectively. Even though the loss is high, the model predicts a reasonable path for each example and it could be argued that in a real driving situation, the model would eventually see more cones and correct its path.

Similarly, the randomly chosen sequences in Figure 25 show a similar pattern of the model predicting reasonable paths, even when they do not match the ground truth path.

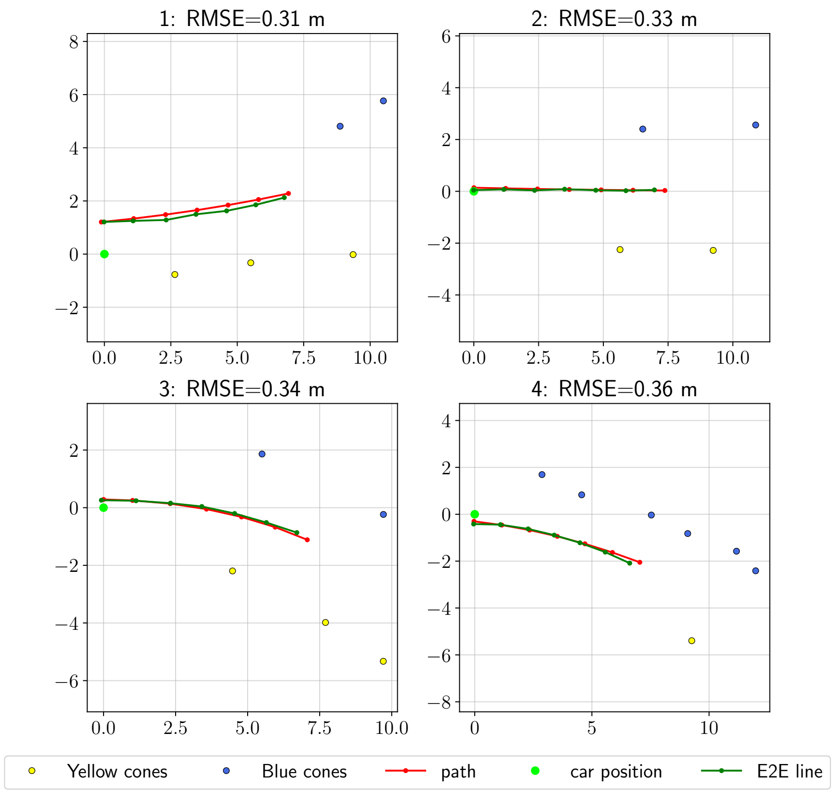


Figure 23: The selection of lowest error sequences from the test set of single-frame clips.

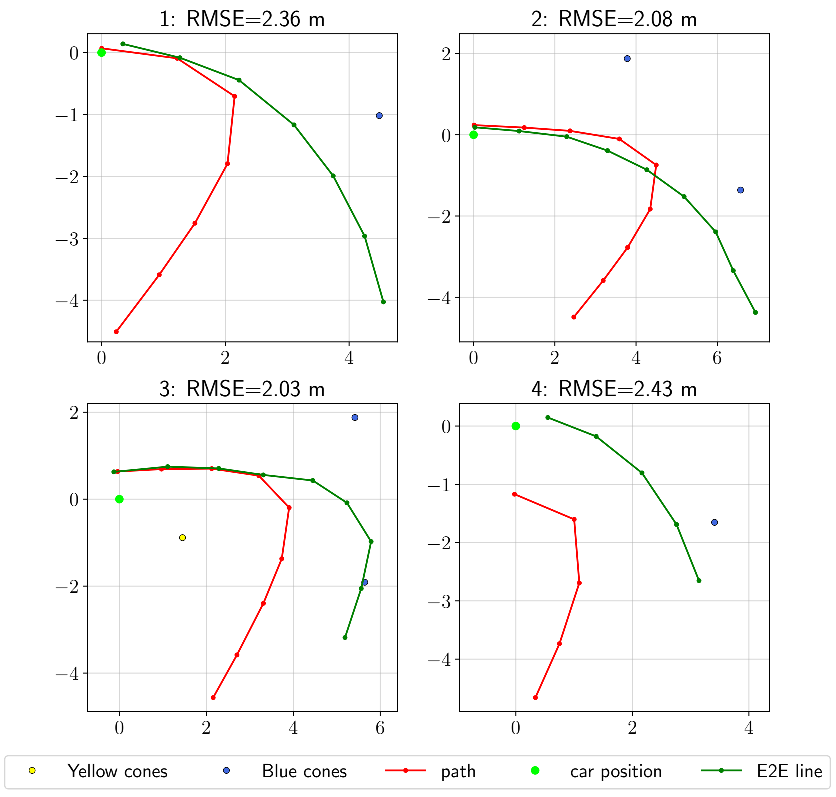


Figure 24: The selection of highest error sequences from the test set of single-frame clips.

4.2 Autocross evaluation

In this section, we evaluate the autocross path planner model, which was trained to plan a path based on the temporal context of the scene and high-level turn type conditioning. We first present the training progression and results on the training set of clip sequences. Subsequently, we evaluate the model on a hold-out set of turn tracks to test the model in a virtual environment. Finally, we evaluate the model on a full virtual circuit track.

4.2.1 Training evaluation

In Figure 26, we show the progression of training and validation loss of the model during training. We also show the validation RMSE error's of the model's predictions divided into the different turn types.

We notice that the validation loss reaches lower RMSE than the pre-trained model, which is expected, as the model has access to temporal context and high-level turn type vector, which provides it with more information about the scene. The model performs best on the centerline clips, which are just general driving clips without any specific turns. The model performs worse, in terms of RMSE, on the all the turn-specific clips. One possible explanation for this is that the center line

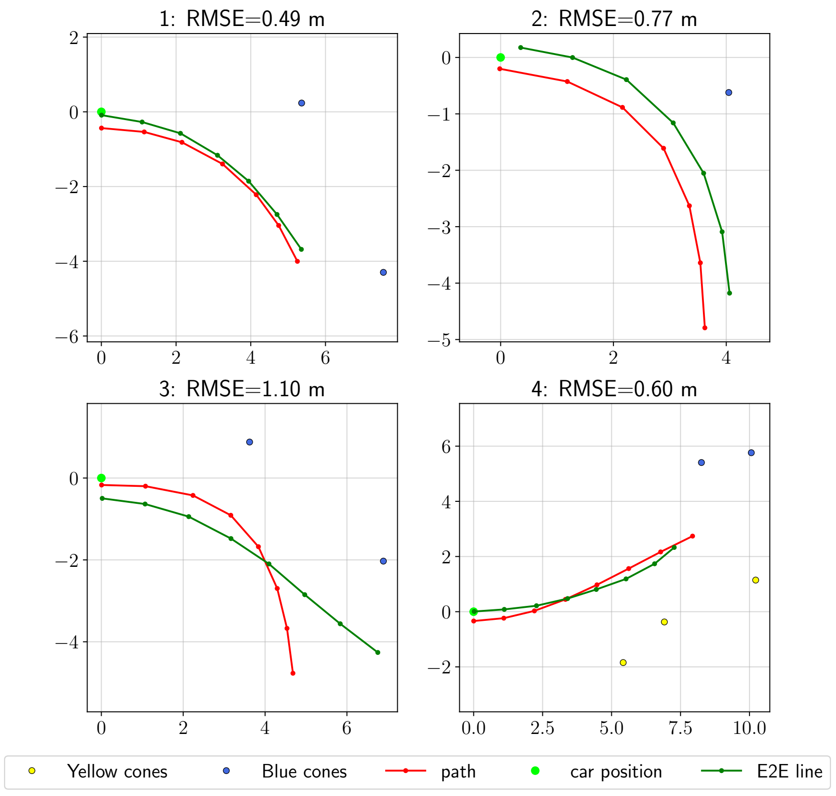


Figure 25: Random selection of sequences from the test set of single-frame clips.

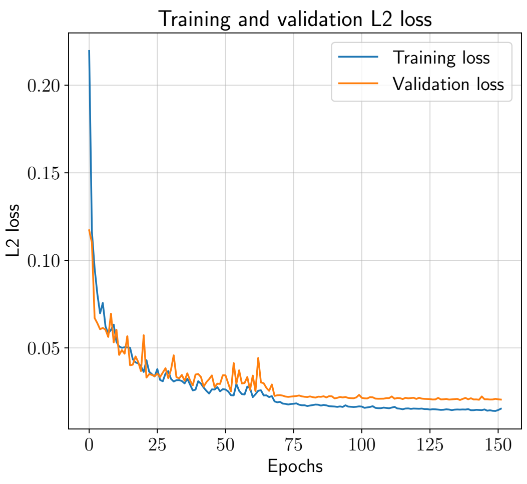
clips contain a lot of straight line driving, which is very easy to score a low loss on, while the turn-specific clips always contain a challenging turn, making them more ”action-packed” and difficult to predict.

4.2.2 Hold-out track set evaluation

Next, we evaluate the model on a hold-out set of turn tracks, which were not used for clip generation during training, however, they were generated using the same process. Therefore they are similar to the tracks depicted in Figure 16. We evaluate how the model performs in same conditions as during training clip generation with only difference being that the the path tracking controller receives the model's predicted path as the input, instead of the ground truth path. That is, we let the car drive on the virtual track for 7 seconds per track at a constant speed of 5 m/s. Subsequently, we perform the same evaluation with the classical geometric path planning algorithm. We then compare the results of the two methods.

In Table 5, we show the results of the evaluation on the hold-out containing 16 tracks for each different turn type. We evaluate the planners in terms of four metrics: mean deviation from the ”optimal” racing line, cumulative number of cones hit, mean maximum steering angle and mean steering angle.

The motivation for the deviation metric is to measure how well the end-to-end model is able to make the car follow the optimal line compared to the center line planner. The cones hit metric is obvious, as the car should avoid hitting cones. The steering angle metrics are interesting, as they show how much did the steering



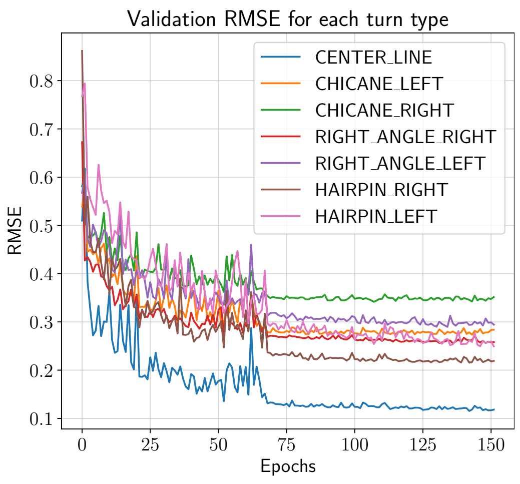


Figure 26: Progression of individual turn type losses during training of the autocross model.

Planner-Track	**Deviation** (m)	**Cones Hit**	**Max Steer** (°)	**Steer** (°)
classical–center_line	0.17	0	47.94	8.35
e2e–center_line	0.08	0	34.26	8.33
classical–right_angle	0.67	1	79.7	17.73
e2e–right_angle	0.13	0	54.39	13.99
classical–chicanes	0.27	0	71.63	15.76
e2e–chicanes	0.16	0	35.53	11.68
classical–hairpin	0.9	2	79.7	35.29
e2e–hairpin	0.23	1	73.51	27.97

Table 5: Evaluation results of the path planner on a test of virtual turn specific tracks. Results are averaged over 16 tracks per each type.

controller have to drive the given trajectory. As previously discussed, we consider trajectories with high steering angles to be suboptimal, as they limit the maximal speed the car can achieve without losing traction. For this, the mean maximum steering angle is the most important metric, since even if the mean steering angle is low, if the car has to steer aggressively at some point, it will affect the entire run.

In Table 5, we can see that the end-to-end model outperform the classical planner in all metrics. The end-to-end model has lower mean deviation, which doesn't necessarily mean its better than the classical planner, but it shows that the model correctly predicts the racing line. Most importantly, the end-to-end generated paths lead to a smoother drive, as both the mean maximum steering angle and the mean steering angle metrics are lower than for the classical planner. In practice, this should result in the car being able to drive faster and perhaps safer than with the classical path planning algorithm.

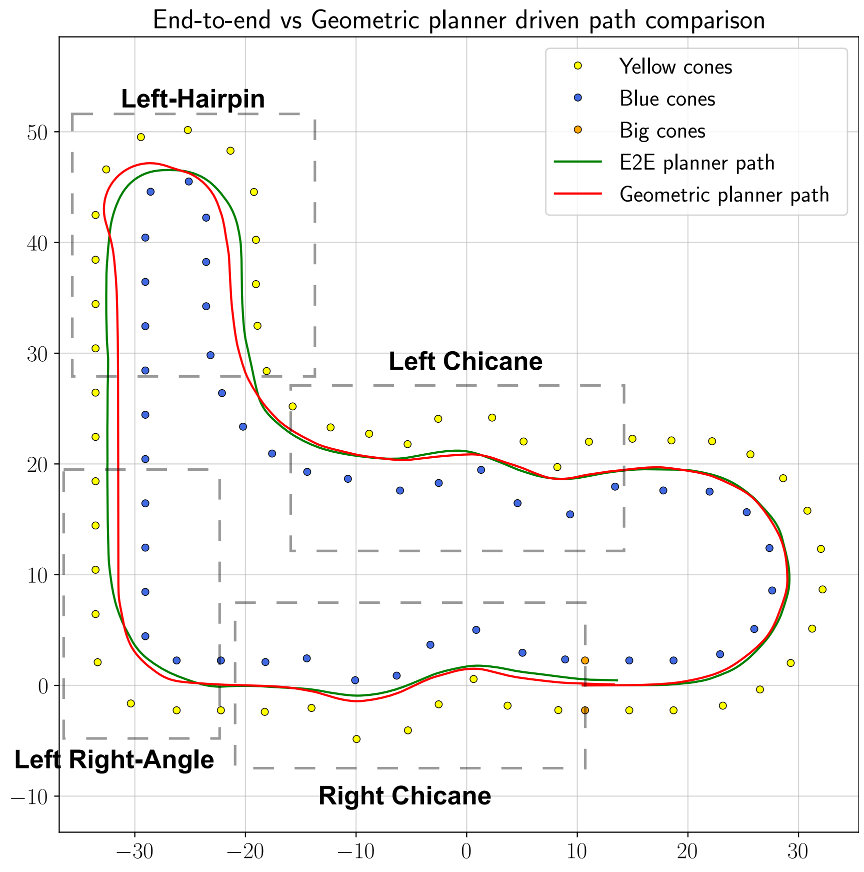


Figure 27: Virtual track used for comparison between the end-to-end neural network path planner and the classical planner.

4.2.3 Virtual circuit evaluation

We now evaluate the model on a full virtual circuit track, which is depicted in Figure 27. The track contains several specific turns, such as two left-turning right-angle turns, two chicanes and a hairpin. It also contains some more ambiguous turns, such as the high radius turn at the start and the right turn after the first chicane, where the model has to drive the centerline.

As discussed in the previous section, in deployment, we have to be able to first define the high-level turn type sequence of the track, which is then used to condition the model while driving. We do this by constructing a table of turn types for each track segment, which is defined by integrating the driven distance since the start of the run. (i.e., the 0–50 meters is a `CENTER_LINE` segment, then the 50–100 meters are a `RIGHT_ANGLE` segment, etc.) In this case, we construct this sequence of turn types manually, as we know the track layout. The division of the track into segments based on the driven distance is depicted in Table 6.

Distance (m)	**Turn Type**
0–32	`CENTER_LINE`
32–65	`RIGHT CHICANE`
65–78	`CENTER_LINE`
78–120	`LEFT HAIRPIN`
120–150	`LEFT RIGHT_ANGLE`
150–∞	`LEFT CHICANE`

Table 6: The assignment table for the turn type based on the distance from the start of the track for the virtual autocross track.

We evaluate the model on the track by performing several full runs, for both the end-to-end model and the classical geometric path planner. Each run is defined by the constant speed setpoint, we start at 5 m/s and increase it by 1 m/s for each run.

We present the results in Table 7. The table contains only runs starting with 8 m/s, as both the planners performed very similarly in terms of laptime and cones hit for the lower speeds. This is expected, as the advantage of a optimal racing line in terms curvature and smoothness is only pronounced at higher speeds. Low curvature racing line can be slower at lower speeds, as it might happen that the minimal curvature path is longer by distance than the center line path.

In the results, we observe that for the classical planner, the mean steering angle increases with the speed, as the car is forced to steer more aggressively to able follow the path. This is not the case for the end-to-end model, which holds a much lower mean steering angle up to the speed of 12 m/s. Starting at speed of 10 m/s, the car with the classical planner starts to fail, by spinning out of the track in hairpin turn, while the end-to-end model is able to drive the track without any issues up until the speed of 12.5 m/s. These results show that the end-to-end model has learned to predict the path that is in some sense superior to the classical planner, that only plans the centerline path.

Planner	**Speed** (m/s)	**Laptime** (s)	**Steer**(°)	**Cones hit**
e2e-planning	8	24.86±0.13	21.16±0.25	0±0.0
classic-planning	8	24.81±0.16	23.55±0.26	1±0.0
e2e-planning	9	22.21±0.13	21.41±0.14	0.0±0.0
classic-planning	9	22.11±0.17	24.43±0.41	0.0±0.0
e2e-planning	10	20.62±0.08	21.70±0.26	0.0±0.0
classic-planning	10	20.37±0.09	24.87±0.30	0.0±0.0
e2e-planning	11	19.23±0.12	22.24±0.31	0.0±0.0
classic-planning	11	19.19±0.05	28.60±0.68	0.2±0.4
e2e-planning	12	18.11±0.09	24.25±0.57	0.0±0.0
classic-planning	12	FAILED	29.73±0.86	0.2±0.4
e2e-planning	13	FAILED	30.89±1.44	1.4±0.8
classic-planning	13	FAILED	28.62±2.39	2.0±0.0

Table 7: Autocross evaluation results, of driving through the virtual track with different speed setpoints. The results are averaged over 5 runs, with standard deviation included.

4.2.4 Real-world evaluation

Finally, we evaluate the model using the physical car on a real-world track. To set up the experiment, we have built a small 100x50 meters track in an airfield, which the team uses for testing. We built the track according to the same principles as were used for building the virtual one.

We made the track 4.5 meters wide and inserted some of the specific turns discussed in this thesis. The track contains two right-angle turns, a chicane and an hairpin turn. It also contains several more ambiguous turns, where the car has to drive the centerline.

We first had to generate the turn type sequence, which divides the track into several segments based on the driven distance, where each segment is assigned a turn type. In order to do so, we have measured the approximate distances between individual turns using a measuring tape. Since the track is rather narrow and the turn type sequence is not very sensitive to precision, we then manually assigned the turn types to each segment based on the driven distance in the previous run. In Figure 28, we show the track layout together with the driven path of the car with turn type segments visualized.

We performed two full runs on the track with the end-to-end model and the classical path planner each. We started with a speed setpoint of 5 m/s and subsequently increased it by 1 m/s to 6 m/s. We show the results of the runs in Table 8.

When driving 5 m/s, the car was able to complete the track without hitting any cones using both planners. Visually, the end-to-end model was able to plan a path that was closer to the optimal racing line. During the lap, it setup for the right angle turns, cut through the chicanes and drove through the outside of the hairpin turn. As discussed previously, driving the racing line at lower speeds does not result in a faster lap time, so both the end-to-end model and the classical planner performed similarly.

When driving with speed a setpoint of 6 m/s, the car was able to the track with

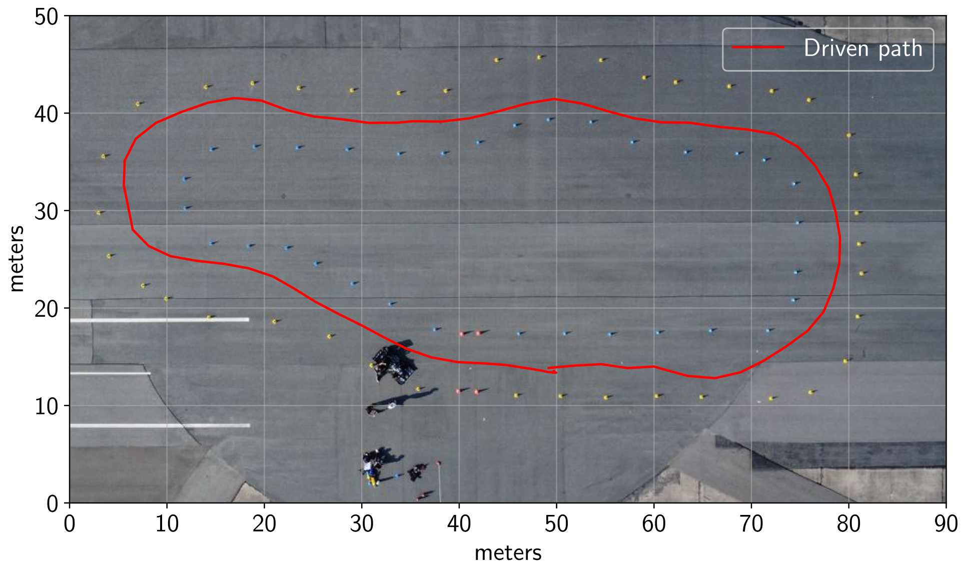


Figure 28: Drone image of the real-world track used for the experiments with the physical car. The driven path of the car at 5 m/s is visualized as a red line.

Planner	**Speed** (m/s)	**Laptime** (s)	**Steer**(°)	**Cones hit**
e2e-planning	5.0	26.31	26.45	0
classic-planning	5.0	26.02	25.15	0
e2e-planning	6.0	22.30	29.30	3
classic-planning	6.0	21.62	24.89	0

Table 8: Comparison of the autocross path planner model and the classical path planner on a real-world autocross track wtih a physical car.

both planners, however, when using the end-to-end model, the car hit three cones in the hairpin turn. After closer inspection, we found that the path tracking controller was not able to follow the path in as stable and responsive way as in the virtual environment. This problem was present in the classical planner run as well, however, as the car drove in the middle of the track, it was able to avoid hitting any cones.

We have also noticed that the car didn't drive through the inside of the hairpin turn in the same was as it was supposed to. (as defined in Figure 16) This could be due to the combination of the car's steering controller being too slow to react and perhaps also some additional simulation to reality discrepancy. This discrepancy could be caused by potentially unaccounted for sources of error, such as the cone detections being less accurate in the real world due to car's roll and pitch variability while driving.

Due to time constraints, we were not able to perform more runs with the physical car to further evaluate these issues. However, with the obtained results, we believe that the end-to-end model has shown potential to match and the performance of the classical path planner in real-world conditions. The simulation to reality gap has to be further investigated and overcome for the model to be able to used in a real-world competition setting.

4.3 Skidpad evaluation

In this section, we evaluate the model's performance on the skidpad track. First, similarly to the autocross model, we present the training progression and individual losses for each turn type vector. Next, we evaluate the model on a hold-out set of skidpad clip tracks. Finally, we test the model's performance at higher speeds on a full skidpad run. We compare the model's performance to the Monte Carlo Localization solution developed and currently used by the team for the skidpad runs.

4.3.1 Training evaluation

In Figure 29, we show the progression of training and validation loss of the model during training. The plot also shows the validation RMSE error's of the model's prediction split into the different skidpad turn maneuvers.

We note that the model reaches reasonably low RMSE errors on the validation set. Compared to the pre-trained model, the skidpad model reaches twice as low RMSE error. This can be attributed to the fact, that the skidpad track is simpler in the way, that the turns are very consistent and the track is symmetric, so in theory the model doesn't have to "guess" as much as in the autocross tracks.

We notice, that the highest RMSE error is for the `SKIDPAD_LEFT` turn type. This turn type contains the maneuver, where the car has to transition from driving around the right circle to driving around the left circle. This is a challenging maneuver for the model to learn, as it has to predict a different path, that is a left turn, in the exact position where it was predicting a right turn before.

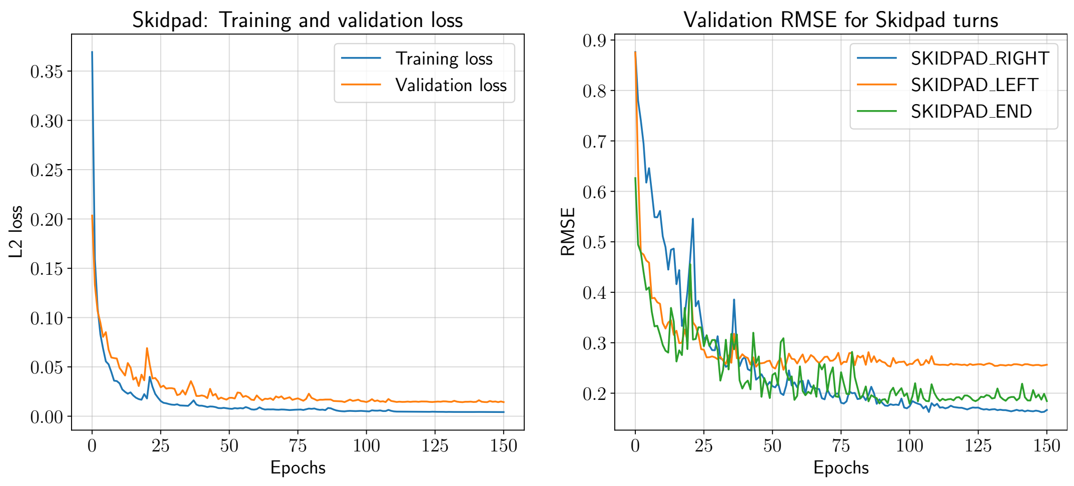


Figure 29: The training and validation loss progression for the skidpad mode. The plot also shows the validation errors for each skidpad turn type.

4.3.2 Hold-out track set evaluation

Next, we evaluate the model on a hold-out set of skidpad tracks, similar to the ones used to generate the training clips, as shown in Figure 17. We evaluate the model in a similar way to the autocross model. We let the car drive on a virtual track for 7 seconds at a constant speed of 5 m/s. For each drive, we check if the car is able to follow the ground truth path.

For skidpad tracks, the max and mean deviation metrics are not important only because they show the predicted path is close to the ground truth path. But, if the max deviation is low, it means the car not only followed the path with precision, but also that the model was able to take the right turn at the skidpad middle intersection. This is the most challenging part of the skidpad track, as the car has to take a different turn at the same position in the track, depending on the turn type conditioning.

Planner-Track	**Max Deviation** (m)	**Mean Deviation** (m)	**Cones Hit**
e2e–skidpad_right	0.45	0.31	0
e2e–skidpad_left	0.49	0.24	0
e2e–skidpad_end	0.64	0.14	0

Table 9: Evaluation results of the skidpad path planner on virtual tracks used for training. The tracks are of three different types: right, left and end based on the type of turn the model is supposed to take in the middle intersection.

In Table 9, we show the results of the evaluation on the hold-out set of skidpad tracks. We notice that the model is able to follow the path with precision similar and in some cases better than in the autocross evaluation. The maximal deviation doesn't go over 0.5 meters for the `SKIDPAD_RIGHT` and `SKIDPAD_LEFT` and 0.64 for the `SKIDPAD_END` turn type. This means the model always took the right path in the middle intersection, which shows that the model did learn to interpret the turn type context conditioning correctly.

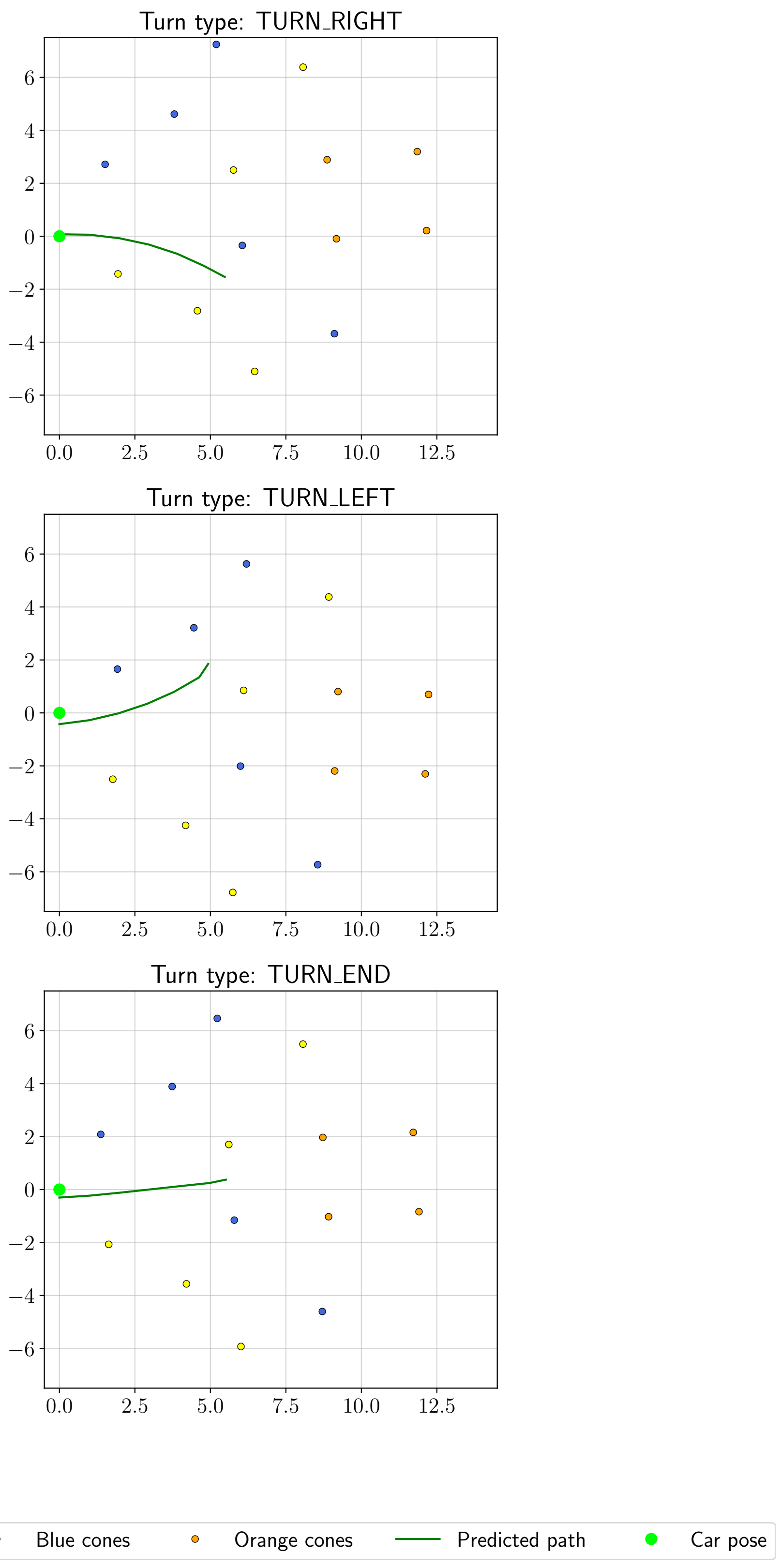


Figure 30: The skidpad path planner model plannning different paths, when the car is in the same position on the track, but the turn type conditioning token changes.

4.3.3 Virtual track evaluation

Finally, we evaluate the model on a full skidpad run. As described in the previous section, in deployment on the skidpad track, we set the turn type conditioning vector based on the lap count. The lap count is incremented every time the car passes through start line, realized by the four big orange cones in the middle intersection, as shown in Figure 30.

Similarly to the autocross evaluation, we perform several full runs with increasingly higher speed setpoints. For each speed setpoint, we perform a test run with both our end-to-end model and the Monte Carlo Localization solution. We compare the results for both methods in terms of the final time and the number of cones hit.

We present the results in Table 10. We notice that the for most of the different speeds, the both the end-to-end model and the Monte Carlo Localization solution perform identically. The skidpad track is very simple and symmetric and both algorithms use the same path tracking controller and drive the same constant speed. Therefore, given that both models predict the correct path, they should achieve the same result.

Planner	**Speed** (m/s)	**Time** (s)	**Cones hit**
E2E-skidpad	8	6.90	0
MCL-skidpad	8	6.90	0
E2E-skidpad	9	6.20	0
MCL-skidpad	9	6.20	0
E2E-skidpad	10	5.65	0
MCL-skidpad	10	5.65	0
E2E-skidpad	11	5.10	0
MCL-skidpad	11	5.10	0
E2E-skidpad	12	4.70	0
MCL-skidpad	12	FAIL	4
E2E-skidpad	12.5	FAIL	8
MCL-skidpad	12.5	FAIL	0

Table 10: Evaluation of the skidpad path planner model in comparison with the Monte Carlo Localization solution on the virtual skidpad track.

We notice that the Monte Carlo Localization solution first fails at the speed of 12 m/s, while the end-to-end model is able to drive through the track, at the same speed. After closer inspection, we found that the Monte Carlo Localization solution doesn't fail at 12 m/s due to localization error, but due to losing traction, while following what looks as the correct path. Since the end-to-end solution runs into the same issue, at speed of 12.5 m/s, which is only 0.5 m/s higher, we conclude that the methods perform similarly in virtual testing environment. For accurate evaluation at higher speeds, we would have to test the solutions on a real-world track to avoid the simulation inaccuracies.

5 Conclusion

5.1 Summary

In this thesis, we have set the goal of designing a path planning algorithm for a driverless formula car, competing in the Formula Student Driverless competition. The goal was to design such an algorithm, which would allow the car to drive as fast as possible through the track. In particular, to be able to drive a racing line through an unmapped track, utilizing only the car's camera perception system and knowledge about the types of turns that appear on the track. Similarly to a human driver, the system should be able to adjust its high-level driving strategy based on its perception and knowing what type of turn is coming up next. For example, setting up for a right angle turn to the left, by driving on the right side of the track and then sharply turning to cut the corner to maximize the speed through the turn.

To achieve this goal, we have decomposed potential track layouts into a set of typical turns, namely: right angle, chicane and hairpin turns. We have then designed a data generation pipeline, which allows us to synthetically generate expert trajectories of the car optimally driving through these turns. We have achieved this by first generating virtual tracks, which contain these typical turns. Subsequently, we have used a racing line optimization algorithm to find the optimal path to drive through the particular track. Lastly, using a simulator, we have driven the car through the track, following the optimal path, to record expert trajectories.

We have then designed a neural network model, which is able to learn to imitate these expert trajectories, mapping the car's perception inputs to the correct path to follow. This type of model is called an end-to-end model, as it learns to map the perception inputs directly to the driving action, without any hand-crafted rules.

We also provide the model with a conditioning token, which gives it information about the type of turn that is coming up next. The conditioning vector tokens include different types of turns and driving behaviours, such as: center–line, right–angle turn, turn–left, turn–right, etc. This allows the model to adapt its driving behaviour to be able to drive through the upcoming turn as fast as possible.

We have designed the path planning model as a sequence to sequence neural network, taking sequence of traffic cone detections as the input and outputting a sequence of path points. The model uses the Gated Recurrent Unit (GRU) for implementing the encoder and decoder. The GRU is a type of Recurrent Neural Network (RNN), which is able to learn long term dependencies over inputs it has seen in the past. Giving the model temporal memory in the form of a context vector, which is passed along from one frame to the next. This allows the model to learn the necessary temporal dependencies to be able to plan and execute complex driving manoeuvres. (ie. setting up for a turn to be able to cut a corner)

After training the model on the expert trajectories, we have evaluated the model on a set of virtual tracks. We have compared the model to a classical path planning algorithm, which plans the centerline of the track using basic geometry. The results show that the model is able to drive the centerline of the track as well as the classical path planning algorithm. For tracks containing some of the typical turns, the model is able to plan such a path, that allows the car to reach higher speeds, leading to lower lap times.

Lastly, we have also shown this path planning model to be able to learn to drive the 8–shaped skidpad track, which is one of the four tracks in the Formula Student Driverless competition. This track contains a middle intersection, in which the car has to decide whether to turn left or right, depending on the lap. We show that the model is able to learn to drive the track, utilizing the conditioning vector to decide which way to turn at the intersection. We compared the model to a Monte Carlo Localization (MCL) solution and shown that the model is able to match the performance of the MCL solution in virtual settings.

5.2 Challenges and limitations

Learning to plan a path through a track, by learning from synthetically generated expert trajectories has several advantages. It allows us to express complex behavioural patterns in data, utilizing the full control over the environment. The model is then able to to imitate these expert trajectories and learn complex driving behaviours, such as setting up for a turn to cut a corner. While this approach has these advantages, it also has several limitations and challenges that need to be addressed.

One of the big challenges is the potential discrepancy between the virtual and real world. Since the model is trained on virtual tracks, it is hard to estimate how well the model will generalize to real world tracks. This discrepancy can be cause by many different factors. The virtual track modeling could contain patterns that are not present in the real world, leading to the model learning to exploit these pattern and then failing to generalize. On the other hand, the real world could contain sources of noise, which are not properly modeled in the simulation environment. Physical factors such as varying pitch and roll of the car during braking and cornering, could lead to loss of accuracy of the cone detections. This could lead to the model failing due to being exposed out–of–distribution data.

Due to these challenges, extensive real world testing and validation is needed to ensure that the simulation environment is a good approximation of the real world. This can be challenging from a resource perspective, requiring a lot of time and effort, which might not be available. In such case, choosing an off-the-shelf classical algorithm might be a better choice.

Lastly, as the model is realized as a neural network, it might be challenging to debug and interpret the model's decisions. This is a common challenge when working with neural networks, which are often called black or grey boxes. This difficulty in interpreting the model's decisions makes the previously mentioned real world validation and debugging even more difficult and time consuming.

5 References

[1] Dean A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann, 1988.

[2] Y. Lecun, E. Cosatto, J. Ben, U. Muller, and B. Flepp. Dave: Autonomous off-road vehicle control using end-to-end learning. Technical Report DARPA-IPTO Final Report, Courant Institute/CBLL, http://www.cs.nyu.edu/˜yann/research/dave/index.html, 2004.

[3] Andrew Bagnell and March. An invitation to imitation. 2015.

[4] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars, 2016.

[5] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning, 2018.

[6] Encoder-decoder diagram.

[7] FSG: fs-germany.org [online]. *FSG: Formula Student Rules.* 2024. `https://www.formulastudent.de/fileadmin/user_upload/all/2024/rules/FS-Rules_2024_v1.1.pdf`.

[8] Michal Horáček. Finding the Fastest Trajectory for Autonomous Student Formula. Bachelor's thesis, Czech Technical University, 2022.

[9] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 2, pages 1322–1328 vol.2, 1999.

[10] Juraj Kabzan, Miguel de la Iglesia Valls, Victor Reijgwart, Hubertus Franciscus Cornelis Hendrikx, Claas Ehmke, Manish Prajapat, Andreas Bühler, Nikhil Gosala, Mehak Gupta, Ramya Sivanesan, Ankit Dhall, Eugenio Chisari, Napat Karnchanachari, Sonja Brits, Manuel Dangel, Inkyu Sa, Renaud Dubé, Abel Gawel, Mark Pfeiffer, Alexander Liniger, John Lygeros, and Roland Siegwart. Amz driverless: The full autonomous racing system, 2019.

[11] eForce FEE Prague Formula. `https://eforce.cvut.cz`.

[12] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[13] Roman Šíp. Visual Detection of Traffic Cones for Autonomous Student Formula. Bachelor's thesis, Czech Technical University, 2024.

[14] Ondřej Kuban. Autonomous vehicle traction system development. Bachelor's thesis, Czech Technical University, 2024.

[15] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Ng. An application of reinforcement learning to aerobatic helicopter flight., 01 2006.

[16] Y. Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. volume 3, pages 932–938, 01 2000.

[17] Pieter Abbeel and Andrew Ng. Apprenticeship learning via inverse reinforcement learning. *Proceedings, Twenty-First International Conference on Machine Learning, ICML 2004*, 09 2004.

[18] Abdoulaye O. Ly and Moulay Akhloufi. Learning to drive by imitation: An overview of deep behavior cloning methods. *IEEE Transactions on Intelligent Vehicles*, 6(2):195–209, 2021.

[19] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert L. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[21] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 05 2015.

[22] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[23] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[24] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[25] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

[26] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

[27] E. Velenis and Panagiotis Tsiotras. Optimal velocity profile generation for given acceleration limits: Theoretical analysis. pages 1478 – 1483 vol. 2, 07 2005.

[28] Nitin R. Kapania, John Subosits, and J. Christian Gerdes. A sequential two-step algorithm for fast generation of vehicle racing trajectories. *Journal of Dynamic Systems, Measurement, and Control*, 138(9), June 2016.

[29] Peter Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, Leilani Gilpin, Piyush Khandelwal, Varun Kompella, HaoChih Lin, Patrick MacAlpine, Declan Oller, Takuma Seno, Craig Sherstan, Michael Thomure, and Hiroaki Kitano. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602:223–228, 02 2022.