Bachelor's thesis

Finding the Fastest Trajectory for Autonomous Student Formula

Michal Horáček

Supervisor: Ing. Jan Čech, Ph.D.

CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of eletrical engineering

Department of cybernetics

May 19, 2022

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Horáček Michal**	Personal ID number:	**492073**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Open Informatics**		
Specialisation:	**Artificial Intelligence and Computer Science**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Finding the Fastest Trajectory for Autonomous Student Formula

Bachelor’s thesis title in Czech:

Nalezení nejrychlejší trajektorie pro autonomní studentskou formuli

Guidelines:

Propose an algorithm for finding the fastest trajectory for the autonomous student formula. The input will be a list of points delineating the track. The output will be the trajectory as a list of waypoints comprising position and velocity. As a proxy problem, find (1) the shortest trajectory, and (2) the trajectory minimizing the average curvature. Formulate the problem mathematically, use a suitable representation and an optimization solver. The dynamic model of the formula can be very simplistic. Validate the algorithm on a synthetic experiment, or optionally on the real autonomous formula.

Bibliography / sources:

[1] Nitin R. Kapania. Trajectory planning and control of an autonomous race vehicle. PhD Thesis. Stanford University, 2016.
[2] Alexander Liniger. Path Planning and Control for Autonomous Racing. PhD Thesis. ETH Zurich, 2018.
[3] Adam Slomoi. Path Planning and Control in an Autonomous Formula Student Vehicle. Technical Report. Monash University, 2018.

Name and workplace of bachelor’s thesis supervisor:

Ing. Jan Čech, Ph.D. Visual Recognition Group FEE

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

Date of bachelor’s thesis assignment: **26.01.2022** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

Ing. Jan Čech, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	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 bachelor’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 bachelor’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, date ....

Signature

Acknowledgement

First and foremost, I would like to thank my thesis supervisor Jan Čech for giving me the opportunity to work on this thesis. His expert guidance was indispensable to ensure the steady progress of my work. Without his invaluable advice, this thesis could not be completed in its current form.

Next, I would like to thank Hynek Zamazal for implementing the Stanley control algorithm for our formula. Without him, my algorithm would not be as successful. In addition, I am very grateful for his help with the Formula Student Driverless Simulator and the integration of my algorithm with the rest of the formula's autonomous pipeline. I also wish to express my gratitude to Daniel Štorc for his excellent job in leading eForce Driverless and supporting me in my struggles with the Robot Operating System.

Furthermore, I would like to thank Tomáš Roun for developing SLAM for our formula and for his advice on writing a thesis in LaTeX. Without him, this thesis would not be possible. Moreover, I am very grateful to Matěj Zorek and Dima Khursenko for their work on the reactive path planning algorithm. Their efforts have been integral to the success of my algorithm. Finally, I would like to thank each member of eForce for maintaining the team spirit.

Abstract

This thesis proposes a trajectory planning algorithm for an autonomous formula built for the international Formula Student competition. In particular, the thesis focuses on the situation where the vehicle's environment is largely known and described by a SLAM–provided global map of cones delineating the track. We demonstrate a parametrization method based on a list of cones, that is robust to false detections. Next, we construct three optimization problems: the shortest path, the path minimizing the average curvature, and the fastest path that takes basic handling limits of the vehicle into account. The algorithm outputs both the path and its optimal speed profile. Our work is validated on multiple sources of data, including real–world SLAM maps and the Formula Student Driverless simulator, where we successfully decreased lap times by half compared to a simple baseline algorithm.

Keywords: Path planning, Trajectory planning, Autonomous racing, Vehicle dynamics, Optimization, Formula Student

Abstrakt

Tato práce navrhuje algoritmus plánování trajektorie pro autonomní formuli postavenou pro mezinárodní soutěž Formula Student. V této práci se zaměřujeme na situaci, ve které je okolí vozidla do značné míry známo a popsáno globální mapou kuželů poskytnutou algoritmem SLAM, která vytyčuje trať. Předvádíme parametrizační metodu založenou na listu kuželů, která je robustní vůči falešným detekcím. Dále konstruujeme tři optimalizační problémy: nejkratší cestu, cestu minimalizující průměrné zakřivení a nejrychlejší cestu, která bere v úvahu základní jízdní omezení vozidla. Náš algoritmus poskytuje jak cestu, tak i její optimální rychlostní profil. Naše práce je validována na několika zdrojích dat, včetně SLAM map sesbíraných v realném světě a v Formula Student Driverless Simulator, kde jsme úspěšně snížili čas projetí kola tratě na polovinu ve srovnání s jednoduchým základním algoritmem.

Klíčová slova: Plánování cesty, Plánování trajektorie, Autonomní závodění, Dynamika vozidla, Optimalizace, Formula Student

Table of Contents

Chapter 1
Introduction

This thesis proposes a path planning algorithm for an autonomous formula built for the international Formula Student competition. We focus specifically on finding the fastest trajectory for a known race track.

Path planning, the problem of finding an optimal path through an agent's environment, is an extensively studied problem in robotics. Its origins can be traced back to the invention of classical graph search algorithms such as Dijkstra's algorithm [1] or A* [2], which remain widely used to the present day. During the 1990's computers became more portable, powerful, and prevalent. New advances in vehicle dynamics [3] were used to augment cars with systems such as traction or launch control. In 1994 FIA banned all electronic driver aids from Formula One because they were believed to devalue the driver's skill. Meanwhile, tools that calculate the minimal time trajectory were developed to aid in the design process of Formula One teams [4, 5]. The new millennium saw the beginnings of autonomous racing as a standalone category of motor sport. The most famous early autonomous race was the DARPA Grand Challenge, especially its second iteration in 2005 [6]. Nowadays multiple autonomous racing series are being held annually, for example Roborace or Indy Autonomous Challenge. In the latter series, cars compete head–to–head, combining algorithms for autonomous driving with game theory [7].

The described trends were reflected in the Formula Student competition in 2016 when the driverless category was announced [8]. The first driverless racing season of the competition took place the following year. Since then, teams from all over the world have presented their innovations in path planning and other related fields [9, 10, 11, 12].

Section 1.1
Formula Student

Formula Student, then called Formula SAE, was founded in 1981 in the USA. 25 years later, the competition came to Europe in the form of Formula Student Germany racing event. Four years later, the original category of vehicles with internal combustion engines became accompanied by formulas powered by an electric powertrain. In 2016 a third category Formula Student Driverless was added thanks to pressure of sponsoring companies from German industry. Today, the most prestigious races are held in Germany, Hungary, Spain, and the United States.

Students from Czech Technical University have entered the Formula Student competition quite early. Czech Republic's first team, CTU CarTech, which competes in the combustion vehicle category, was established in 2006, the year the competition itself came to Europe. In 2010 the team has once again kept pace with developments in Formula Student and started CarTech Electric to compete in the new Formula Student Electric category. In its third racing season, CarTech Electric was rebranded as eForce FEE Prague Formula as it is known today. To date, eForce has constructed ten formulas, and the eleventh is being built for the current racing season.

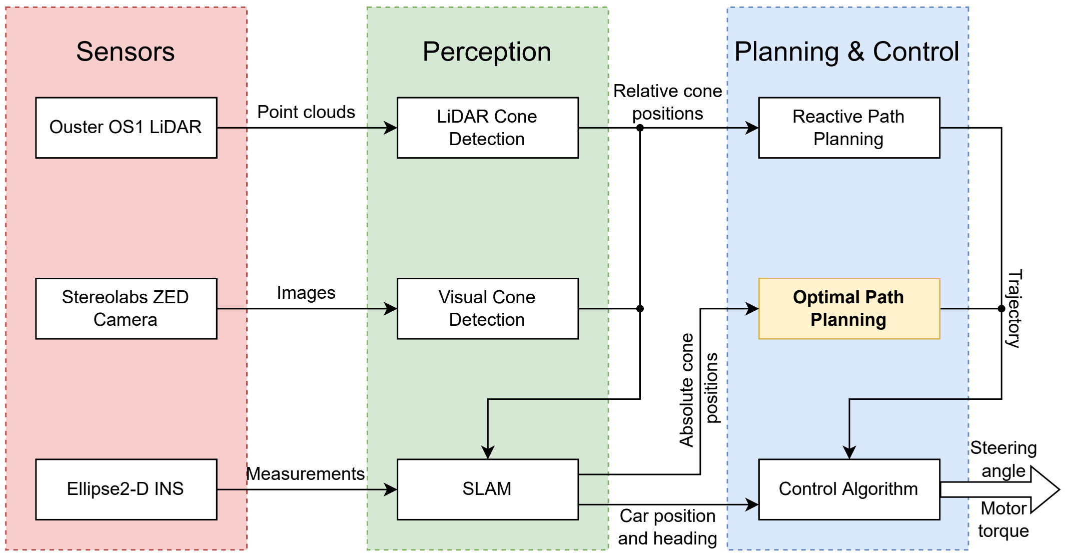


Figure 1.1: Diagram of the autonomous pipeline. This thesis focuses on the highlighted optimal path planning component of the pipeline.

From 2019 onward, eForce has been participating in the driverless category through the eForce Driverless team. The driverless team received the seventh generation of eForce formula to serve as the vehicle platform. Since then, eForce Driverless has been developing algorithms for autonomous driving, as well as augmenting the existing platform with the necessary hardware.

Section 1.2
Autonomous pipeline

Our car's autonomous pipeline runs on a Zotac Z-Box computer equipped with a standard Linux Ubuntu operating system. The autonomous system is made up of separate modules that communicate through the Robot Operating System (ROS) framework. A diagram of the modules and their connections is shown in Figure 1.1. There are additional nodes in the system that were omitted for clarity, for example, the CAN node that handles communication with the rest of the car or the lap counter node.

The pipeline begins with the data input provided by the car sensor array. Currently, we use an Ouster OS1 LiDAR, a Stereolabs ZED camera, and an SBG Systems Ellipse2–D inertial navigation system (INS). Their physical location in the car is shown in Figure 1.2.

In the perception step of the pipeline, sensory input is used to produce a representation of the environment. Camera images are processed by a convolutional neural network, based on the YOLOv3 object detection architecture, which detects cones in an image. The cone pixels are subsequently projected into the world coordinates via a homography mapping between the image and the world [13]. The cone positions can be obtained simultaneously from LiDAR point clouds. The LiDAR detector starts with removing the ground points that make up the majority of a given point cloud. Then the remaining points are clustered together, reconstructing the individual cones whose positions are returned [14]. Continuously generated

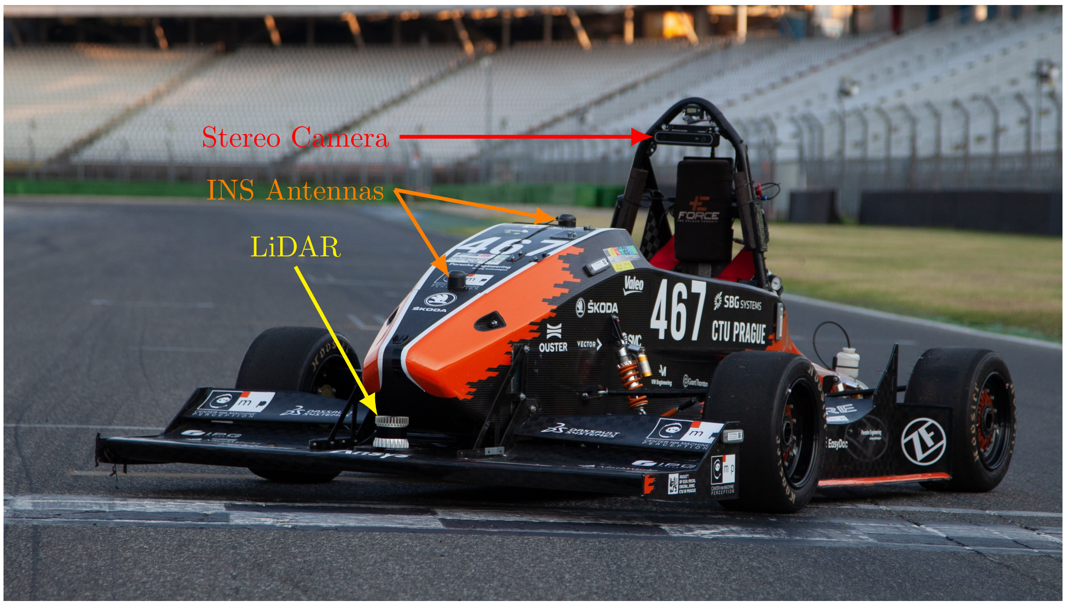


Figure 1.2: Photo of the autonomous formula with sensors marked. The INS itself is hidden inside the monocoque.

measurements of cone positions in vehicle coordinate system combined with INS readings are fed into a simultaneous location and mapping (SLAM) algorithm. In return an estimated map of cones and the car's position and heading on this map in the world coordinate system is produced [15].

In the first lap, vehicle coordinate system–based reactive path planning is used until the entire race track is explored and mapped by SLAM at its completion. This path planning algorithm provides a conservative path through the cones currently seen by the camera and LiDAR detectors. In the remaining nine laps, our algorithm computes the optimal trajectory, which is the subject of this thesis. Both path planning algorithms provide a trajectory to the Stanley control algorithm [6]. The control algorithm uses a nonlinear lateral regulator to transform the trajectory into a sequence of steering angles. At the same time, a longitudinal regulator calculates motor torque instructions to match the speed profile. Finally, the steering angle and motor torque commands are sent via controller area network (CAN) bus to the appropriate electronic control units that execute them.

Chapter 2

Proposed method

The algorithm outlined in this thesis consists of several components, which are all explained in this section.

In Section 2.1 we look at the input data. We discuss the dynamic disciplines of Formula Student and related competition rules. Then we explain the reactive path planning algorithm that navigates the car during the first lap. While the formula follows the path provided by the reactive path planning algorithm, the SLAM algorithm maps the cones delineating the race track and localizes the formula on its map. This information serves as a foundation for our work, so we explore its properties at the end of this section.

In Section 2.2, we develop a path representation on a given race track. In the process, we take steps to maximize the ratio of accuracy of representation to computation time required when working with the parametrization. We also decide on the intended behavior of our parametrization near the track boundaries.

In Section 2.3 we discuss the length and average curvature of a path. Afterwards we derive these measurements for the path parametrization method introduced in the second subsection and construct optimization problems which lead to shortest, respectively curvature–minimal paths. Furthermore, we examine properties of these paths in the context of searching for a fastest trajectory.

In Section 2.4 we consider the physical laws that constrain car performance. Subsequently, we derive an algorithm to calculate the optimal speed profile for a given path. With a speed profile, we can transform a path, which carries merely spatial data about its shape and orientation, into a trajectory. Finally, we combine partial results reached in the course of this thesis to obtain the desired time–optimal trajectory.

Section 2.1

Input data

The Formula Student competition consists of four dynamic disciplines. In the acceleration discipline, formulas are compared by their acceleration capabilities. The eight–shaped skidpad measures the car's ability to execute sharp turns. Both of these disciplines possess a race track predefined in the competition rules [16]. However, their simple layout is not interesting to us. The third discipline is autocross, which focuses on driving through an unknown environment. Because our algorithm relies on prior knowledge of the race track, this discipline is also unsuitable for our purposes. This leaves us with only the final discipline, trackdrive, which is illustrated in Figure 2.1.

In trackdrive, the teams aim to successfully complete 10 laps around the track, optimally in the fastest possible time. Teams start without precise knowledge of the environment and have to map it during the first lap. Therefore, we can consider the first lap to be a case of an autocross run and use the same approach to complete the lap. We briefly discuss how to complete the first lap in Section 2.1.1. Furthermore, autocross and track drive share the same set of guarantees on the properties of the race track, which is provided by the rules and the competition handbook [17]. The rules say that the track is at least 3 m wide and contains no turns of outside diameter

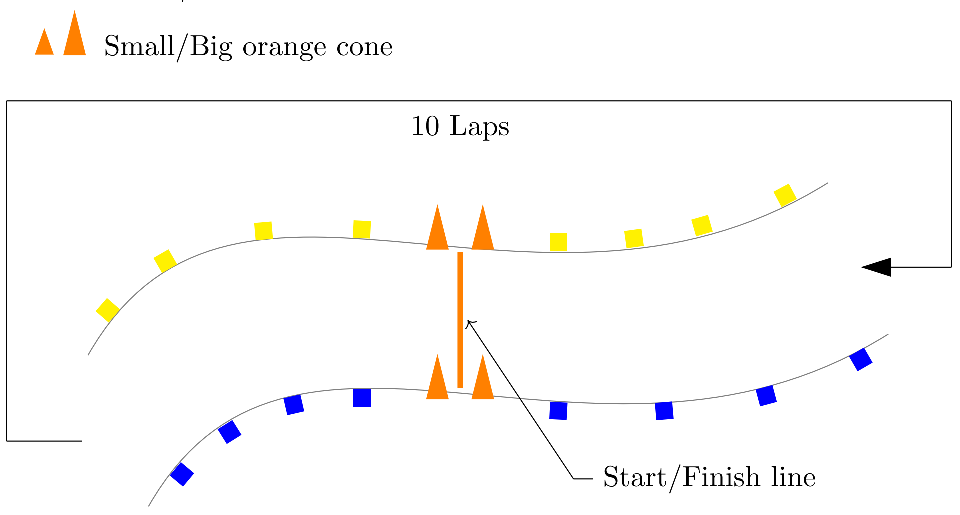


Figure 2.1: Trackdrive schematic taken from FSG rules [16]. The direction of travel is always oriented such that blue cones mark the left track boundary and yellow cones delineate the right track boundary.

smaller than 9 m. Furthermore, the distance between two successive cones of the same color is at most 5 m. Finally, the length of a single lap is known to be between 200 m and 500 m.

Subsection 2.1.1
Reactive path planning

During the first lap of trackdrive, the car relies on a reactive path planning algorithm. This algorithm is not the author's work. It was developed by Matěj Zorek, one of the team's founding members. Recently, a fellow eForce Driverless member Dima Khursenko has improved upon Matěj Zorek's original design. In principle, the algorithm identifies a centerline through the cones currently seen by our neural network. The neural network detects up to 20 m distant cones [13], which translates to about five cones on both sides of the track. In our experience, at least three cones of both colors (corresponding to the next 10 m of the track) must always be localized to successfully complete the first lap.

However, during sharp turns, the cones on the track's inner edge may fall out of camera view. In this case, no cones of that color are detected. Therefore, the reactive path planning algorithm employs several heuristics that help solve common situations such as this. For example, it can fill in the missing cones on the inner turn radius using the visible cones on the outer edge and assuming that the track is 3 m wide. A more detailed explanation of this algorithm can be found in Boháč [9]. The reactive path planning algorithm has been extensively tested and it has performed reasonably well in the last year's competitions. In conclusion, we expect the car to finish the first lap, at which time our algorithm assumes control over path planning.

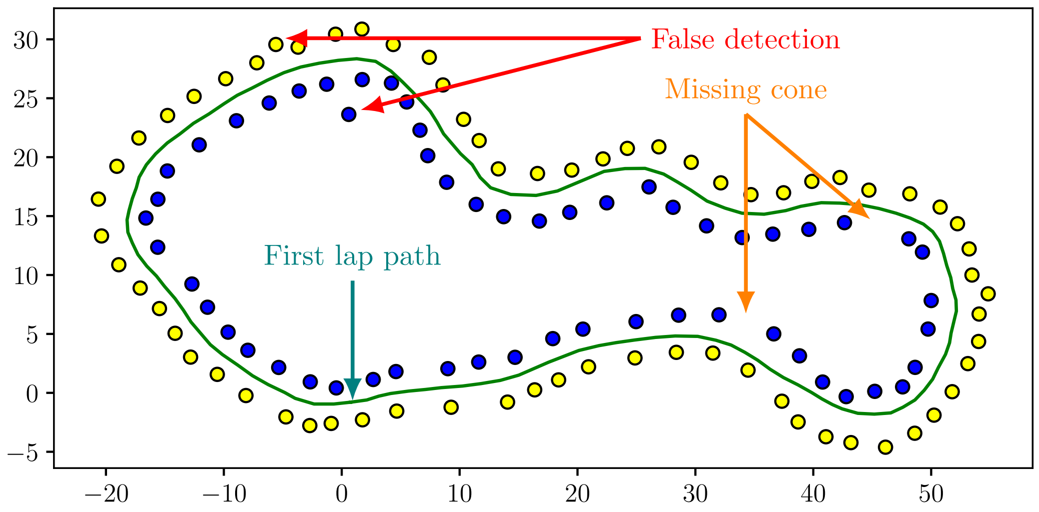


Figure 2.2: A SLAM cone map built during the first lap. This map has been created from real data. We highlight examples of false positive and false negative errors typically encountered on SLAM maps.

Subsection 2.1.2

SLAM map

As the formula travels through the race track, its detectors generate measurements of the cone positions from different places. In addition to reactive path planning, which we have already discussed, these data are sent to the SLAM algorithm [15]. SLAM continuously integrates cone measurements taken from numerous angles and positions into a global cone map in the world coordinate system. By the time the first lap is finished, SLAM will have explored the entire race track and built a global map of the cones that delineate it.

The second function of the SLAM algorithm is localization. SLAM continuously provides the formula's position on the global cone map. Our algorithm saves the positions during the first lap and subsequently uses them during the parametrization procedure, as we describe in Section 2.2.1.

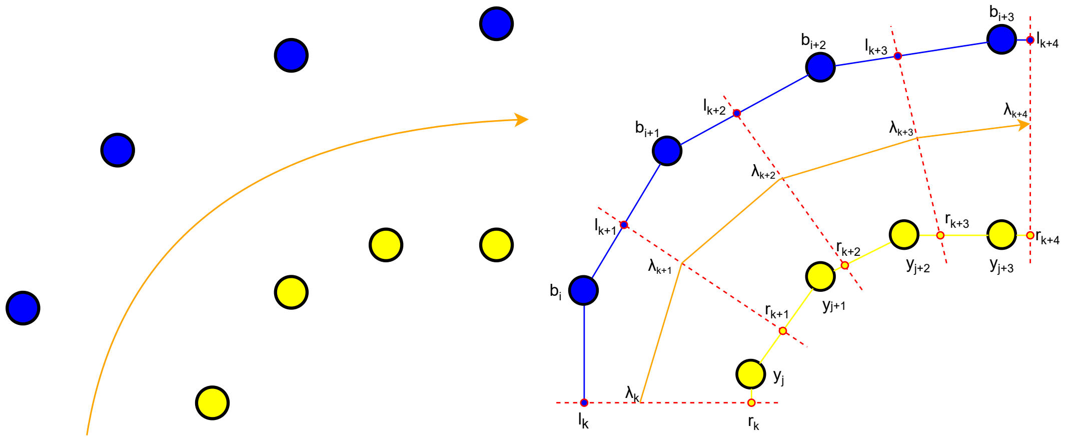
Figure 2.2 demonstrates how a typical SLAM cone map constructed from real data looks. Although the outline of the track is clearly visible, a few false negatives and false positives are also easily recognizable. Imprecision in the reported cone positions is caused by inaccuracies in sensory measurements and failures of the cone detectors. The topic of dealing with these errors is addressed in the next section.

Section 2.2

Track parametrization

Several parametrization schemes can be found in literature. For example, Kapania [18] stores information about the track and the path using curve functions. Slomoi [12] discretizes the track through fixed transverse lines and then keeps a single point on every such line. Together, the points provide a piecewise approximation of a path. For the purposes of this project, we decided to use the latter approach for its simplicity. Its principle is illustrated in Figure 2.3.

The entire parametrization process is depicted in Figure 2.4. In Section 2.2.1 we demonstrate how to estimate the track boundaries shown in Figure 2.4b. In Section 2.2.2 we show how to gain representation accuracy in sharp turns while saving computation time on straight segments of the track, which we depict in Figure 2.4c. We complete the parametrization process in Section 2.2.3 to obtain the final result in Figure 2.4d.



(a) A continuous path defined by a curve.

(b) Representation of the same path produced by our parametrization method.

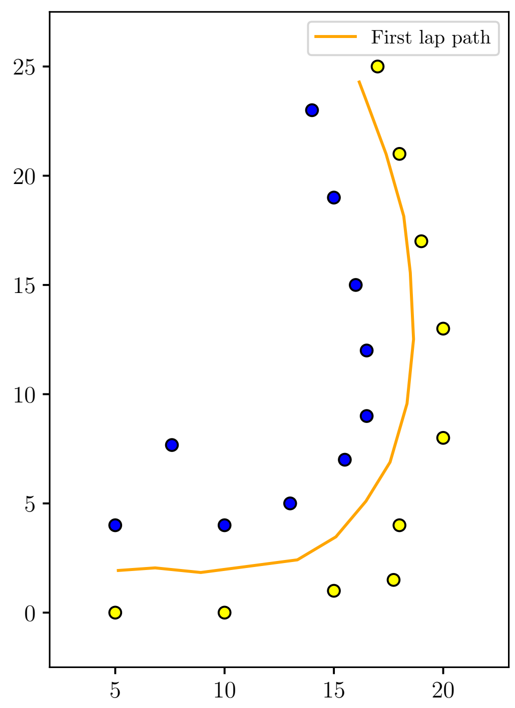
Figure 2.3: Original continuous path and its discretized approximation based on transverse parametrization lines. We mark the relevant points of interest with the appropriate symbols used in this section.

Subsection 2.2.1

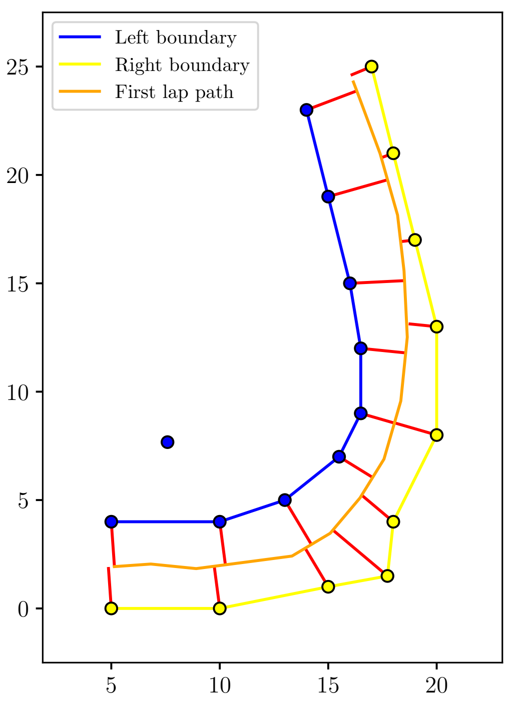
Estimating track boundaries

When our algorithm receives a SLAM map at the completion of the first lap, several preprocessing steps are performed. The cones that form the map are internally stored as a two–dimensional array whose rows consist of arbitrarily ordered triplets $(x, y, color)$ representing individual cones. Color is an integer label that specifies blue, yellow, small orange, or big orange cones. We ignore the small and big orange cones, since the former are completely absent in trackdrive and the latter only mark the finish line (see Figure 2.1).

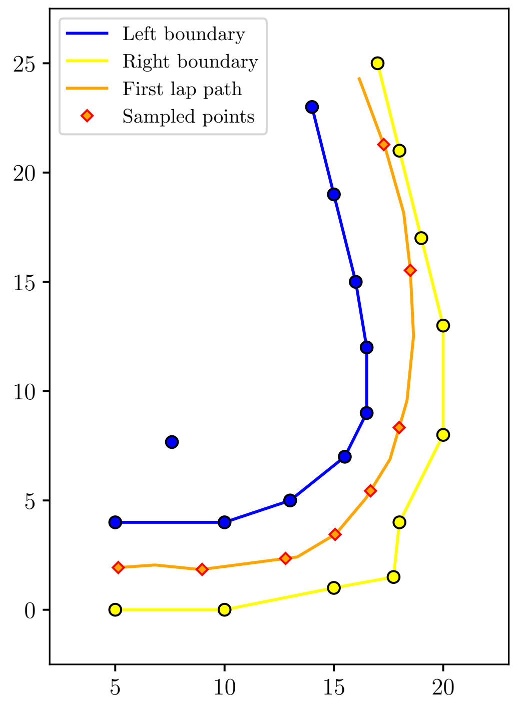
The remaining blue and yellow cones are separated into an array of blue cones and an array of yellow cones. In our notation summarized in Table 1, we refer to the positions of the blue cones as the sequence $(\boldsymbol{b_i})_{i=1}^{B}$ and the positions of the yellow



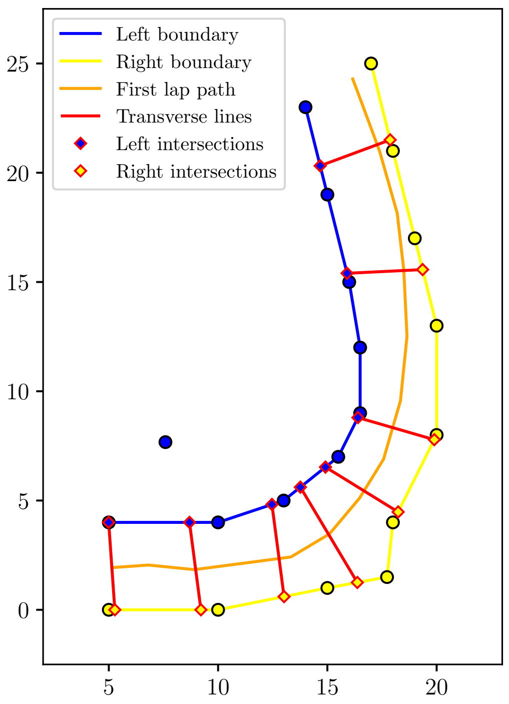
(a) Cones forming a track segment, including two false detections. Orange line is the path taken by the formula during the first lap.



(b) Track boundary estimation using cone projections on the first lap path. Outlying blue cone was filtered out.



(c) Sampling the first lap path according to local curvature. Selected points $(\boldsymbol{d_i})_{i=1}^{D}$ are marked with orange diamonds, $D = 8$



(d) Final transverse parametrization lines. Left intersections $(\boldsymbol{l_i})_{i=1}^{D}$, and right intersections $(\boldsymbol{r_i})_{i=1}^{D}$ delineate the track.

Figure 2.4: Illustration of the parametrization process on a track segment.

cones as $(\boldsymbol{y_i})_{i=1}^{Y}$, where B is the total number of blue cones and Y the number of yellow cones.

Next, we determine the left and right track boundaries from the array of blue and yellow cones, respectively. We decompose this problem into two parts. First, sorting the cones into the order in which they are encountered while driving along the race track. Second, interpolating between neighboring cones using a method of our choice. Our task is made significantly easier because SLAM provides the current position and heading of the formula with respect to its map. Our algorithm collects these data during the first lap and uses them to reconstruct the path taken by the car. We represent the path taken in the first lap as the sequence $(\boldsymbol{t_i})_{i=1}^{T}$. Crucially, this sequence is already sorted in driving order. Although many interpolation techniques are possible, we choose the simplest in the first version of our algorithm. We linearly interpolate between every pair of neighboring points $\boldsymbol{t_i}$ and $\boldsymbol{t_{i+1}}$ and produce a piecewise linear approximation of the first lap path.

We use the sorted property of $(\boldsymbol{t_i})_{i=1}^{T}$ to sort the cones. For every cone, we find its orthogonal projection on the nearest line segment created from the sequence $(\boldsymbol{t_i})_{i=1}^{T}$. We illustrate this process in Figure 2.4b. Afterwards, we permute the cones into the order in which they are projected onto the path from the first lap.

In addition to sorting the cones, this process filters out serious false detections. Although the competition track is unknown prior to the race for the formula, teams may check and measure it using analog devices during a track walk organized before the race. Therefore, we can always obtain an upper bound on the cone distance to any valid path through the race track. We delete all cones further to the nearest point on the path from the first lap because they are false detections. In Figure 2.2 we eliminate the lower false detection in this way, and our system is robust enough to handle the upper one.

Finally, the sorted blue cones form the left track boundary, while the sorted yellow cones create the right track boundary. Geometrically, both track boundaries are piecewise linear curves. We explored other options in addition to linear interpolation, for example cubic splines [19]. Ultimately, we decided to prefer simplicity in the first version of this algorithm over other potential benefits such as differentiability of the interpolation at each point. An alternative robust solution to determining the track boundaries is demonstrated by the AMZ Racing team [11].

Subsection 2.2.2

Curvature–based sampling

The track boundaries trivially lead to a parametrization using Equation (2), which connects opposing points on the track boundaries. This is a valid parametrization which is capable of representing paths. However, it faces two serious issues. First, it places the lines f_i with the same density in both hairpin turns and straight segments of the track. This is not wrong, but a smarter scheme allocates the parametrization lines f_i more densely in regions with high curvature where they are needed to maintain accuracy and places them further apart in straight segments where few are sufficient. Obviously, fewer parameterization lines lead to a less computationally demanding parametrization on which optimization problems converge to local minima faster. Second, the connecting line f_i often crosses the track boundaries at a sharp angle, severely affecting the quality of the solution to any optimization problem; see Figure 4.3.

To solve the first issue, we begin with our initial first lap path $(\boldsymbol{t_i})_{i=1}^{T}$, on which we will allocate new curvature adjusted points. First, we use the inscribed circle function ϱ from Algorithm 1 to calculate the curvature at every point $\boldsymbol{t_i}$. The measured curvatures form an empirical distribution. Using the quantile function, we separate the track into Q distinct regions of similar curvature. We associate the weight w_q, which is drawn from a weight sequence $(w_q)_{q=1}^{Q}$, with the q–th quantile. Each point $\boldsymbol{t_i}$ is then assigned a weight according to its quantile. We equidistantly interpolate D points on a weighted version of the first lap path, where the segment from $\boldsymbol{t_i}$ to $\boldsymbol{t_{i+1}}$ is given the length

$$\delta(\boldsymbol{t_i}, \boldsymbol{t_{i+1}}) = \frac{w_i + w_{i+1}}{2} ||\boldsymbol{t_{i+1}} - \boldsymbol{t_i}||. \tag{1}$$

Transforming the newly assigned points back to fit onto the original path $(\boldsymbol{t_i})_{i=1}^{T}$, we obtain D curvature–adjusted sample points $(\boldsymbol{d_i})_{i=1}^{D}$. These are depicted in Figure 2.4c. In summary, D controls the smoothness of discretization and the weights $(w_q)_{q=1}^{Q}$ influence how aggressively the placement of $(\boldsymbol{d_i})_{i=1}^{D}$ discriminates according to the local curvature.

Subsection 2.2.3

Constructing transverse lines

We solved the first problem by allocating points $(\boldsymbol{d_i})_{i=1}^{D}$ according to the local curvature. We solve the second outlined issue in a natural way. At every point $\boldsymbol{d_i}$ we find the normal vector $\boldsymbol{n_i}$ to the segment $\boldsymbol{t_j}$ to $\boldsymbol{t_{j+1}}$ on which $\boldsymbol{d_i}$ is placed. Next, we draw a line through the point $\boldsymbol{d_i}$ in the direction of the normal vector $\boldsymbol{n_i}$. We call this line f_i and it intersects with the track boundaries gained in Section 2.2.1 at some point $\boldsymbol{l_i}$, respectively $\boldsymbol{r_i}$. Line f_i can thus be expressed as an affine combination of $\boldsymbol{l_i}$ and $\boldsymbol{r_i}$ using the parameter λ_i

$$f_i(\lambda_i) = \lambda_i \boldsymbol{l_i} + (1 - \lambda_i)\boldsymbol{r_i}. \tag{2}$$

Now we are able to represent a path on the race track as a sequence $(\lambda_i)_{i=1}^{D}$. However, because λ_i is currently an unconstrained parameter, many invalid paths are allowed. In the next section, we discuss how to select an appropriate interval $[a_i, b_i]$ for λ_i and how to enforce it during optimization. A summary of the symbols used in this section and their meaning is presented in Table 1.

Subsection 2.2.4

Enforcing track boundaries

In the context of Equation (2), it is clear that ensuring that the path remains within the track bounds is equivalent to condition $\lambda_i \in [0, 1]\ \forall i \in 1, \ldots, D$. Furthermore, we need to consider the physical width of the car (1.6 m) so we avoid hitting the cones. Consequently, we need to restrict every λ_i to some interval $[a_i, 1 - a_i]$ that creates a minimal margin 0.8 m wide around the track boundaries. In practice, we allocate larger margins to account for inaccuracies; see Figure 4.11a.

From a simple reasoning derived from Figure 2.6 we obtain the equation

$$a_i = \frac{M}{||\boldsymbol{l_i} - \boldsymbol{r_i}||} \tag{3}$$

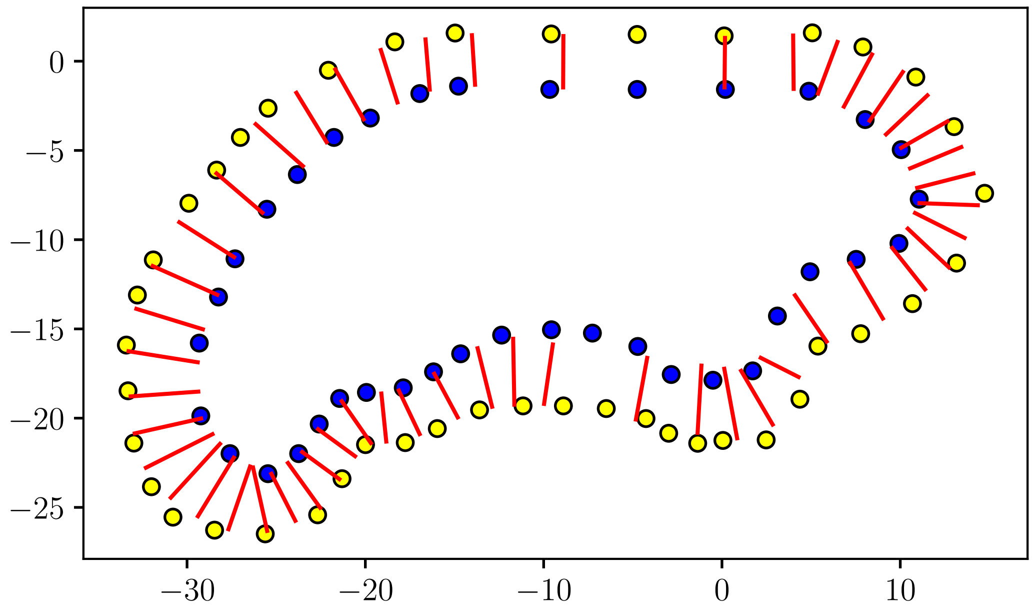


Figure 2.5: Parametrization of an example race track using $D = 50$ transverse lines.

Name	Symbol	Domain	Type
Number of blue cones	B	$\mathbb{N}$	scalar
Number of yellow cones	Y	$\mathbb{N}$	scalar
Number of points in the first lap path	T	$\mathbb{N}$	scalar
Number of quantiles considered	Q	$\mathbb{N}$	scalar
Discretization smoothness	D	$\mathbb{N}$	scalar
Minimal distance to track boundaries	M	$\mathbb{R}_+$	scalar
Weight associated with q–th quantile	w_q	$\mathbb{R}_+$	scalar
Normal vector to i–th path segment	$\boldsymbol{n_i}$	$\mathbb{R}^2$	2D vector
Intersection of f_j and left boundary	$\boldsymbol{l_j}$	$\mathbb{R}^2$	2D point
Intersection of f_j and right boundary	$\boldsymbol{r_j}$	$\mathbb{R}^2$	2D point
Valid region on line f_j	$[a_i, 1 - a_i]$	$a_i \in [0, 0.5)$	interval
Blue cone positions	$(\boldsymbol{b_i})_{i=1}^{B}$	$\boldsymbol{b_i} \in \mathbb{R}^2$	2D points
Yellow cone positions	$(\boldsymbol{y_i})_{i=1}^{Y}$	$\boldsymbol{y_i} \in \mathbb{R}^2$	2D points
Points representing the first lap path	$(\boldsymbol{t_i})_{i=1}^{T}$	$\boldsymbol{c_i} \in \mathbb{R}^2$	2D points
Points resampled based on curvature	$(\boldsymbol{d_i})_{i=1}^{D}$	$\boldsymbol{d_i} \in \mathbb{R}^2$	2D points
Track path representation	$(\lambda_j)_{j=1}^{D}$	$\lambda_j \in [a_i, 1 - a_i]$	scalars
Transverse parametrization lines	$(f_j)_{j=1}^{D}$	$f_j : \mathbb{R} \to \mathbb{R}^2$	functions
Weighted path distance function	δ	$\mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}_+$	function
Off–track distance function	θ	$\mathbb{R} \to \mathbb{R}_+$	function
Off–track penalization function	η	$\mathbb{R}_+ \to \mathbb{R}_+$	function

Table 1: Summary of used mathematical symbols and notation

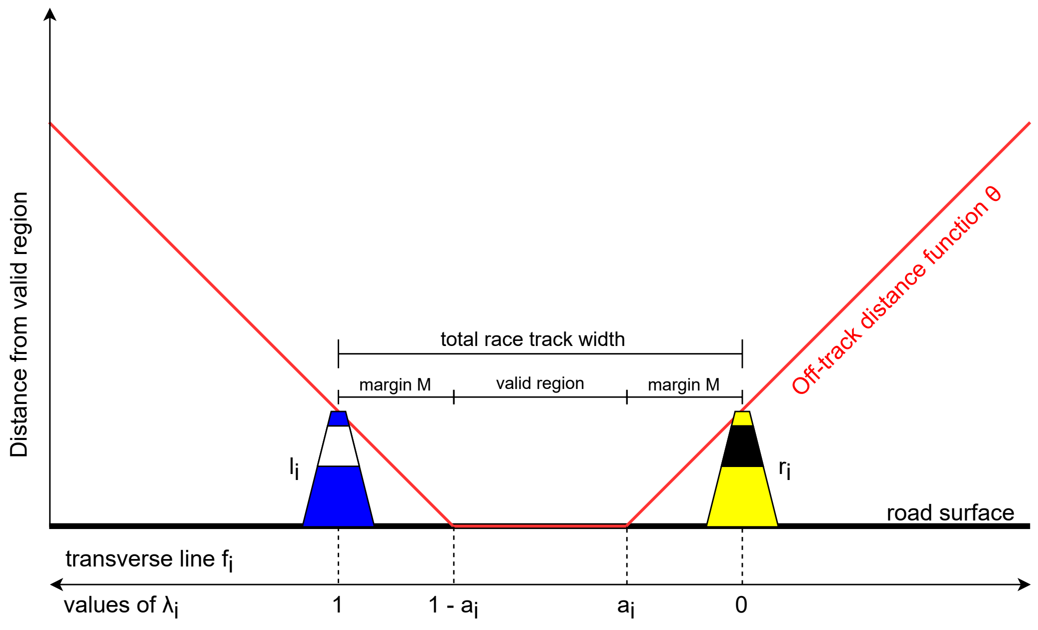


Figure 2.6: Illustration of the off–track function θ which calculates distance of λ_i from its valid interval $[a_i, 1 - a_i]$. The valid region corresponds to values of λ_i, for which the resulting point from Equation (2) lies further than M meters right to $\boldsymbol{l_i}$ and M meters left to $\boldsymbol{r_i}$ along the line f_i.

where M is the desired margin width. Next, we construct an off–track distance function θ that returns the distance outside the valid region along the transverse line f_i

$$\theta(\lambda_i) = \max\{a_i - \lambda_i, 0, \lambda_i - (1 - a_i)\}. \tag{4}$$

Finally, we select a new function $\eta : \mathbb{R}^+ \to \mathbb{R}^+$ that assigns a penalty based on the distance from the valid region. In Section 3, we choose to solve optimization problems with iterative optimizers and automatic gradient computation. Therefore, we require η to be differentiable.

The selection of a concrete function η is further determined by the rules that define two types of penalties for diverging from the marked race track. First, hitting a cone incurs 2 s penalty to the total track time. Second, an off-course event, which occurs when the vehicle has all four wheels outside the track boundaries, is penalized with 10 s penalty. Therefore, executing a faster trajectory by driving at or even beyond the track boundaries is never worth it under current rules. Consequently, we want the penalization function η to be hard enough to prevent these events.

In practice, a simple linear function $\eta(x) = \alpha x$ performs flawlessly. We choose α proportionately to typical values of the problem's objective function. For curvature optimization, whose objective function returns values less than one, $\alpha = 1$ provides sufficient penalization. In contrast, when optimizing the length of a given path, we expect values somewhere in the hundreds; thus, we choose $\alpha = 100$.

The track boundary enforcement feature appears as a term in every optimization problem defined in the following sections as

$$P_{offtrack}(\lambda_1, \dots, \lambda_D) = \sum_{i=1}^{D} (\theta \circ \eta)(\lambda_i) = \sum_{i=1}^{D} \eta(\theta(\lambda_i)) \tag{5}$$

Section 2.3
Proxy tasks

Once a valid path through the race track can be formed, the natural next step is to ask about its quality. In this thesis, we ultimately aim to minimize the time required to execute the path. In general, time is expressed as distance over speed, $\frac{d}{s}$. To start, we formulate minimization of path length as an optimization problem and introduce a way to efficiently compute an initial guess close to the optimal solution. Afterwards we show that maximizing speed is related to minimizing the curvature of a path; then we express it as an optimization problem as well.

Both properties are based on continuous paths specified by differentiable curves. However, they can naturally be extended to operate on discretized tracks, as we discussed them in the preceding section.

Subsection 2.3.1
Length

The importance of length in the context of finding the fastest trajectory is apparent. A smaller distance to cover intuitively leads to faster lap times. The length of a path defined by a continuous curve $\phi : [a, b) \to \mathbb{R}^n$ is expressed by the integral

$$L(\phi) = \int_a^b ||\phi'(s)||ds \tag{6}$$

In the previous sections, we used a piecewise linear approximation of a continuous path ϕ. Because its linear segments have a constant first derivative, the integral can be substituted for a simple summation

$$L(\lambda_1, ..., \lambda_D) = \sum_i ||f_{i+1}(\lambda_{i+1}) - f_i(\lambda_i)|| \tag{7}$$

Note that for a fixed path ϕ the length of its approximation is always smaller than the length of ϕ itself. It can be proven that as the discretization parameter D approaches infinity, the length of the increasingly smooth approximation converges to the original length of ϕ. Combining the track length function with the soft constraint term $P_{offtrack}$ to enforce track boundaries, we get the optimization problem

$$\underset{\lambda_1, ..., \lambda_D}{\text{arg min}} \quad L(\lambda_1, ..., \lambda_D) + P_{offtrack}(\lambda_1, \ldots, \lambda_D) \tag{8a}$$

$$\text{subject to} \quad \lambda_1 = \lambda_D. \tag{8b}$$

The optimization constraint in Equation (8b) forces the path represented by $(\lambda_i)_{i=1}^D$ to be a closed loop. Minimizing this optimization problem results in the path depicted in Figure 2.7.

Moreover, we are able to quickly obtain an excellent initial guess $(\lambda_i)_{i=1}^D$ of the shortest path, which is shown in Figure 2.8a, through dynamic programming. We discretize each λ_i to G values, which correspond to G 2D points placed on the appropriate line f_i. Next, we connect every point from the G points on line f_i with the G from the previous line f_{i-1} and the next line f_{i+1}. Every edge between two

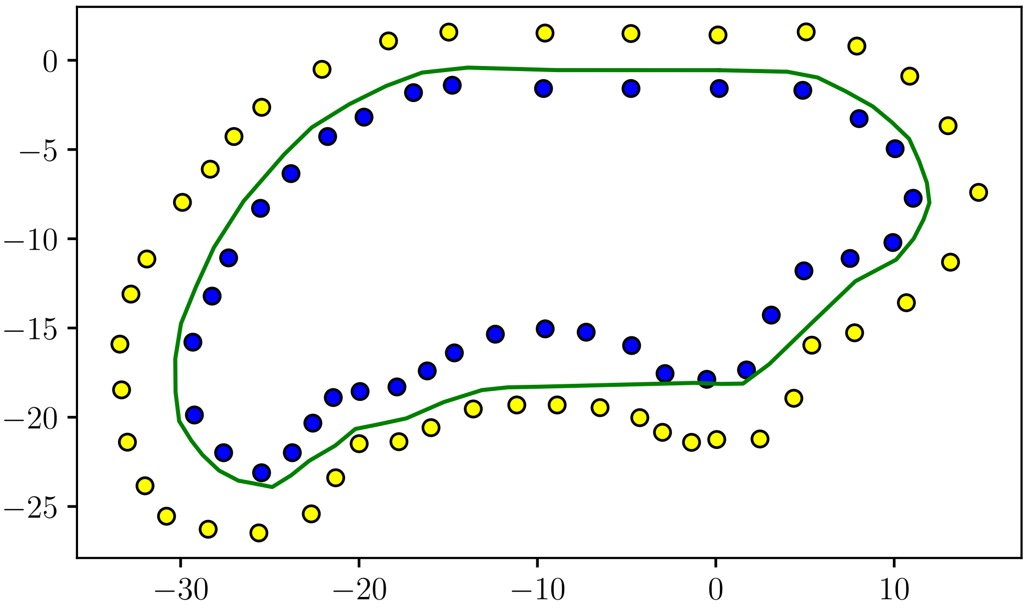
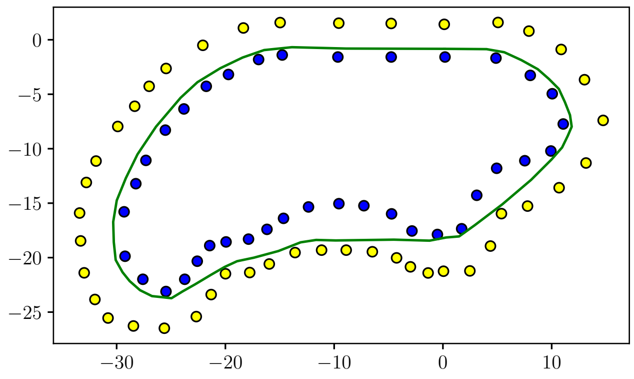
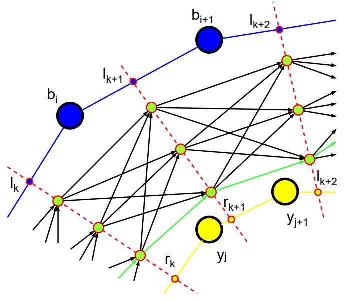


Figure 2.7: The shortest path through the track.

points is weighted by its length. Thus, we obtain a directed acyclic graph (DAG) with nodes and edges illustrated in Figure 2.8b. Finally, we add an additional layer of G points from the first line f_1 to the end of the DAG. The duplicated layer of points in the first and the last layer of DAG simulates the closed nature of the trackdrive race track. We calculate the initial guess using a standard DAG search and then transform the nodes that form the best path through the DAG back to the appropriate $(\lambda_i)_{i=1}^{D}$ values.



(a) An initial guess of the shortest trajectory obtained from DAG graph search.



(b) Graph edges between layers in DAG. Edges possess a cost equal to euclidean distance between their end nodes.

Figure 2.8: Using DAG to find an initial guess of the shortest trajectory.

Subsection 2.3.2

Curvature

The importance of the curvature–minimal path is derived from the friction ellipse which is discussed in Section 2.4. The curvature–minimal path can be traveled using the least lateral force F_y. Consequently, it maximizes the available longitudinal force F_x which influences the speed of the vehicle.

The curvature κ at the point $\phi(s)$ of a continuous differentiable curve defined by the function $\phi : [a, b) \to \mathbb{R}^n$ is commonly defined as

$$\kappa(s) = \frac{T'(s)}{N(s)} \tag{9}$$

$T(s)$ stands for the tangent to the curve at point $\phi(s)$ and $N(s)$ represents the normal vector to ϕ at $\phi(s)$. The total curvature of ϕ that we wish to minimize is then

$$K(\phi) = \int_a^b |\kappa(s)| ds = \int_a^b k(s) ds \tag{10}$$

where k is the magnitude of the signed curvature κ, $k = |\kappa|$. We aim to minimize the curvature of a path regardless of its orientation, therefore we consider k instead of κ. In addition, if ϕ is a closed curve as in our case (i.e. $\phi(a) = \phi(b)$), $\int_a^b \kappa(s)ds = 0$.

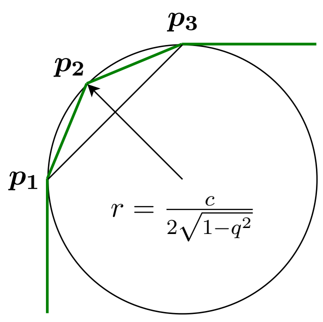
However, the piecewise linear approximation we chose in Section 2.2 to represent a path is not differentiable at its every point and, therefore, lacks a defined normal and tangent vector at those points. We overcome these limitations by using an equivalent, historically used, definition of curvature. The alternative definition is based on the osculating circle, which is the best approximating circle to a given curve ϕ at the point $\phi(s)$. Mathematically,

$$k(s) = \lim_{\boldsymbol{p_1}, \boldsymbol{p_2}, \boldsymbol{p_3} \to \phi(s)} \varrho(\boldsymbol{p_1}, \boldsymbol{p_2}, \boldsymbol{p_3}) \tag{11}$$

where $\boldsymbol{p_1}, \boldsymbol{p_2}, \boldsymbol{p_3} \in \phi$ and $\varrho : \mathbb{R}^3 \to \mathbb{R}$ is a function that returns the curvature of a circle on whose circumference lie points $\boldsymbol{p_1}$, $\boldsymbol{p_2}$ and $\boldsymbol{p_3}$. The function ϱ is explained by Algorithm 1 and is illustrated in the added sketch. We use the property of a circle that its curvature is equal to the inverse value of its radius $\frac{1}{r}$ at each point.

Algorithm 1 Curvature of a circle inscribed to three points

```
Input:  Points $\boldsymbol{p_1}, \boldsymbol{p_2}, \boldsymbol{p_3}$
Output: Curvature $k$ of the inscribed circle
1: $a = ||\boldsymbol{p_2} - \boldsymbol{p_1}||$
2: $b = ||\boldsymbol{p_3} - \boldsymbol{p_2}||$
3: $c = ||\boldsymbol{p_1} - \boldsymbol{p_3}||$
4: $q = \frac{a^2+b^2-c^2}{2ab}$
5: $k = \frac{2\sqrt{1-q^2}}{c}$
6: return $k$
```



Returning to our approximation of a path through a piecewise linear function, the curvature is zero when we pick three points lying on the same segment $f_i(\lambda_i)$ to $f_{i+1}(\lambda_{i+1})$. We consider meaningful only the curvatures derived from the osculating

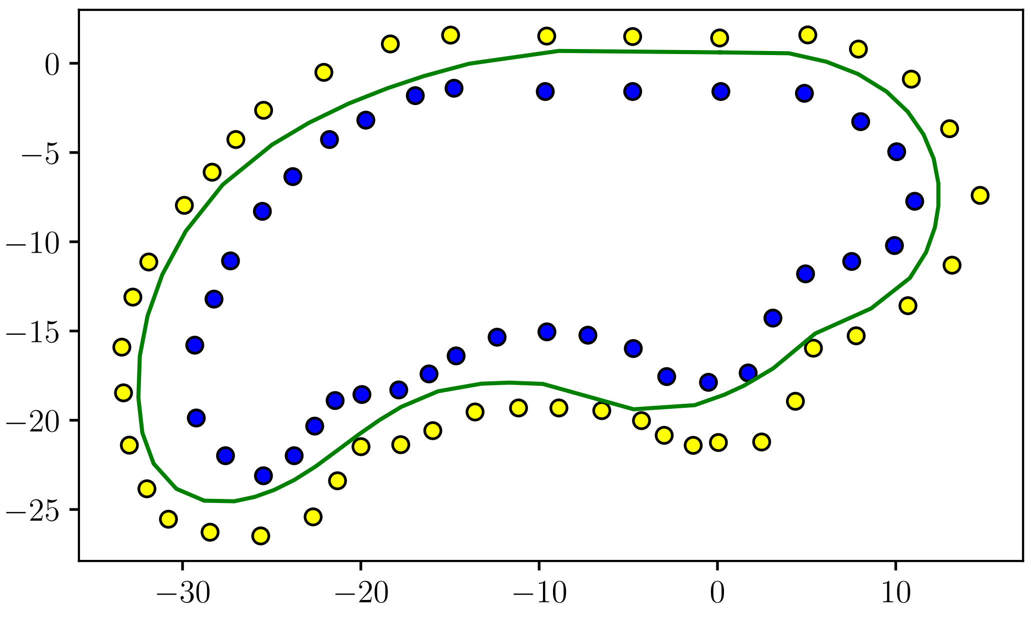


Figure 2.9: Path minimising average curvature.

circles inscribed to three successive points $f_{i-1}(\lambda_{i-1}), f_i(\lambda_i), f_{i+1}(\lambda_{i+1})$ of the path. If we sum these terms together, we obtain the expression

$$K(\lambda_1, \ldots, \lambda_D) = \sum_i \varrho(f_{i-1}(\lambda_{i-1}), f_i(\lambda_i), f_{i+1}(\lambda_{i+1})) \tag{12}$$

Increasing the value of the discretization smoothness parameter D corresponds to the limit in Equation (11). From this we can prove that Equation (12) is an approximation that converges to the true total curvature of the continuous curve ϕ defined in Equation (10) as D increases.

Similarly to searching for a path with the shortest length, we can use Equation (12) to search for a path with the smallest curvature. We formulate the appropriate optimization problem as follows:

$$\begin{aligned} \underset{\lambda_1, ..., \lambda_D}{\text{arg min}} \quad & K(\lambda_1, ..., \lambda_D) + P_{offtrack}(\lambda_1, \ldots, \lambda_D) && \text{(13a)} \\ \text{subject to} \quad & \lambda_1 = \lambda_{D-1}, && \text{(13b)} \\ & \lambda_2 = \lambda_D && \text{(13c)} \end{aligned}$$

which results in the path shown in Figure 2.9. Once again the constraints ensure that the obtained path is a closed loop.

Unlike in the case of a shortest path, dynamic programming cannot provide a good initial guess of the desired path. DAG is based on the fact that when we begin exploring the neighbors of a node, we have already reached the optimal value in that node. Because ϱ from Algorithm 1 requires points from three successive layers, we cannot ensure this requirement, and so we cannot formulate curvature minimization as a DAG search problem.

In this section, we discussed the beneficial qualities of shortest distance paths as well as paths with smallest curvature. In the next section, we will explore how to combine their positive properties into a time–optimal trajectory while acknowledging physical limits of the car's abilities to execute a given trajectory in reality.

Section 2.4

Time optimization

So far, we only discussed trajectories in terms of geometry, regardless of the physical requirements they impose on a vehicle attempting to travel them. However, if we wish to provide a good racing line, we need to make sure it is feasible, otherwise the entire effort expended towards its calculation is worthless.

Movement is in reality governed by highly complex and decidedly nonlinear physical laws, which are hard to model while maintaining an acceptable computation speed. Therefore, we need to decide which laws we will keep and which we neglect.

We model the race car as a single tire with the combined force transfer and acceleration abilities of the four real tires and the mass of the entire car. We neglect any tire modeling, for example the slip ratio. The only enforced physical constraint is the friction ellipse, which is also called the circle of forces

$$F_x^2 + F_y^2 \leq (\mu F_z)^2 \tag{14}$$

This inequality describes the driving limits of a car. A vehicle is pressed against the road surface with normal force F_z. There is a certain maximum force μF_z that tires can transfer to the road surface. This force is a function of the normal force F_z and the friction coefficient μ which describes the tire–road interaction. The right–hand side of Equation (14) can be distributed among longitudinal force F_x, which accelerates/decelerates the car, and lateral force F_y, which changes the car's heading. In total, the longitudinal and lateral forces must remain within the handling limits defined by μF_z.

The grip coefficient μ is affected by many things, from road composition, temperature, or wetness, to tire material, size, inflation pressure, or temperature. Because accurate knowledge of μ is integral to safety–critical applications such as anti–lock braking systems or adaptive cruise control, it has been a focus of numerous research projects. The approaches to learning μ range from optical [20] to slip–based [21] and a comprehensive overview can be found in [22]. However, for our purposes, it is common to assume a constant value for μ corresponding to driving on dry asphalt [23]. According to literature [24], $\mu = 0.75$ is a reasonable choice.

The available normal force F_z consists largely of the gravitational term mg, which remains constant. But when the car is moving, it increases due to the aerodynamic downforce

$$F_{aero} = \rho A_{ref} C_L v^2 \tag{15}$$

where ρ is the air density, acting on the vehicle. Equipping the formula with an aerodynamic package decreases the aerodynamic lift coefficient C_L, which in turn generates a higher downforce. Our formula has an aerodynamic package designed to produce 977 N of downforce at its design target speed of $16\,\mathrm{m\,s^{-1}}$ [25]. The relevant parameters of our aerodynamic package are included in Table 2. However, the design speed of $16\,\mathrm{m\,s^{-1}}$ is not yet always reachable by our autonomous formula. To evaluate the importance of F_{aero}, we consider the aerodynamic downforce with our formula's parameters from Table 2 and set $v = 7.5\,\mathrm{m\,s^{-1}}$. This is approximately the average speed we estimate our formula can consistently maintain; see Figure 4.5c. In this situation, $F_{aero} \approx 313\,\mathrm{N}$. That is about a third of the downforce that the aerodynamic package was originally designed to provide and one seventh of the gravitational term.

Parameter	Symbol	Value	Unit
Friction coefficient	μ	0.75	—
Formula mass	m	212	kg
Coefficient of lift	C_L	-3.82	—
Coefficient of drag	C_D	1.49	—
Reference aerodynamic area	A_{ref}	1.19	m^2
Maximum acceleration	a_{max}	2	$\mathrm{m/s}^2$
Maximum deceleration	a_{min}	-4	$\mathrm{m/s}^2$

Table 2: Physical parameters of our car relevant to the speed profile algorithm. Properties of the aerodynamic package come from an unpublished engineering design document [25].

However, the aerodynamic package also introduces aerodynamic drag to the system, which slows down the car, weakening the benefits of higher downforce. We have decided to neglect aerodynamic effects in the first version of our formula's path planning algorithm. Instead, we count any unmodeled aerodynamic improvements of our car's driving abilities towards a safety margin for its operation.

Finally, we estimate the maximum acceleration a_{max} from the performance of our formula in the acceleration discipline. Last summer it has typically completed the 75 m long track in 9 s. Under our maintained simplification of acceleration's independence on speed, it amounts to acceleration of 2 m/s^2.

Subsection 2.4.1

Speed profile

Given a path, we wish to calculate the maximum speed along the path that the car is capable of executing under the physical laws considered. In our case, we use the friction ellipse to derive the forces encountered along the path into their lateral and longitudinal components. We begin by calculating the maximum speed at which the formula remains within the handling limits. In other words, we find the speed at which the centrifugal force is equal to the normal force at each point. We initialize our speed profile at the s–th point of the path $(\boldsymbol{p_i})_{i=0}^{P}$ with the following equation:

$$U_x(s) = \sqrt{\frac{\mu g}{k(s)}} \tag{16}$$

where $k(s)$ is the average path curvature at the point $\boldsymbol{p_s}$ calculated by Algorithm 1. The initial speed profile for the trajectory in Figure 2.11 is drawn in Figure 2.10b.

In the next step, we calculate the maximal speed at the point $\boldsymbol{p_s}$ that can be reached from a parameterizable initial speed at the beginning of the path. At each point, we therefore take the smaller of the maximum speed according to Equation (16) and the speed reached with the maximum acceleration from the previous point on the path. Algorithm 2 describes this process, whose result is shown in Figure 2.10c. When the first pass of the algorithm finishes, we have a speed profile that is viable in terms of lateral force and accelerates feasibly.

In Algorithm 3 we use a similar process to ensure the feasibility of braking maneuvers. Although the update step is the same, we start from the end of the path and progress back to its beginning.

Algorithm 2 First pass of the speed profile algorithm (acceleration)

```
Input:  Speed profile $U_x$, path $(\boldsymbol{p_i})_{i=0}^{P}$
Output: Speed profile $U'_x$ adjusted for acceleration capabilities
1: for $s = 1$ to $P$ do
2:     $c = 2a_{max}||\boldsymbol{p_s} - \boldsymbol{p_{s-1}}||$
3:     $U'_x[s] = \min\left(U_x[s], \sqrt{U_x[s-1]^2 + c}\right)$
4: end for
```

Algorithm 3 Second pass of the speed profile algorithm (braking)

```
Input:  Speed profile $U_x$, path $(\boldsymbol{p_i})_{i=0}^{P}$
Output: Speed profile $U'_x$ adjusted for braking capabilities
1: for $s = P$ to $1$ do
2:     $c = 2a_{min}||\boldsymbol{p_s} - \boldsymbol{p_{s-1}}||$
3:     $U'_x[s-1] = \min\left(U_x[s-1], \sqrt{U_x[s]^2 + c}\right)$
4: end for
```

The resulting speed profile identifies the optimal transition points between maximum acceleration and maximum braking. This behavior is a logical consequence of neglecting the time required to change the input commands and the delay before the car acts on them. Due to these deviations from reality, it is sensible to set the parameters in Table 2 conservatively. Overestimating the braking ability a_{min} and the grip μ of the car can easily lead to dangerous situations. In contrast, more adventurous settings naturally lead to better performance, but given that this is the first version of path planning for our formula, we opt for the conservative approach.

The derivation of equations used in the speed profile algorithm from basic physical laws is demonstrated in Filip [19]. The implementation is based on the algorithm's description in Kapania [18]. Once the speed profile has been computed using the algorithm described above, we can focus on our ultimate goal of minimizing lap time.

Subsection 2.4.2

Racing line

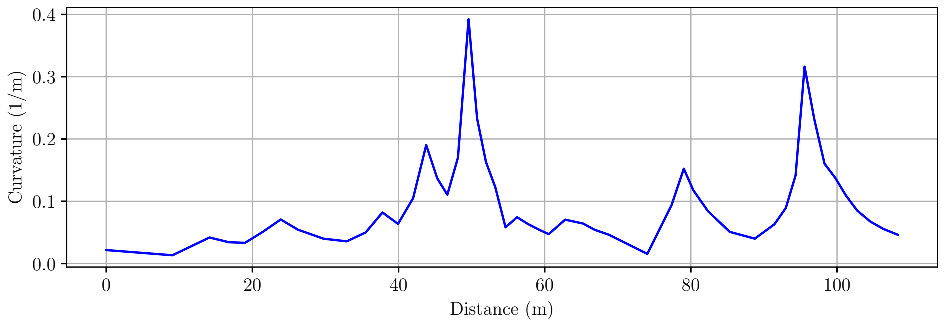
In the previous sections, we calculated the length of a given path and its optimal speed profile under simplified physical constraints. Now, we combine this information to express the time required to execute the provided path ϕ as

$$T(\phi) = \int_0^{L(\phi)} \frac{1}{v(s)} ds \tag{17}$$

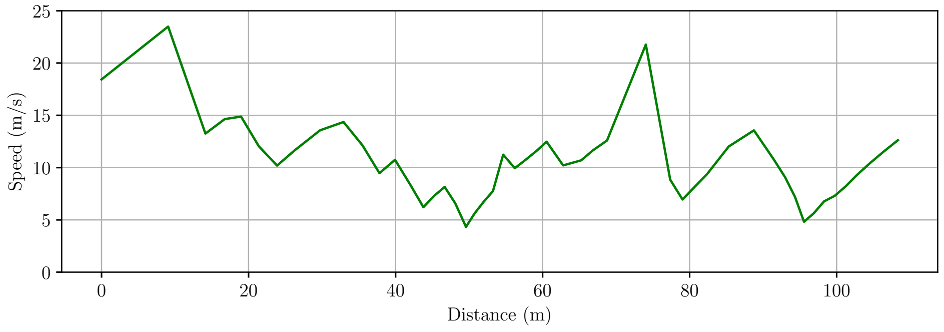
where $L(\phi)$ is the length of ϕ and $v(s)$ the instantaneous speed at distance s along the trajectory. Taking advantage of the piecewise linear nature of our approximations, we can rewrite Equation (17) to

$$T(\lambda_1, \dots, \lambda_D) = \sum_i \frac{||f_{i+1}(\lambda_{i+1}) - f_i(\lambda_i)||}{U_x(i)} \tag{18}$$

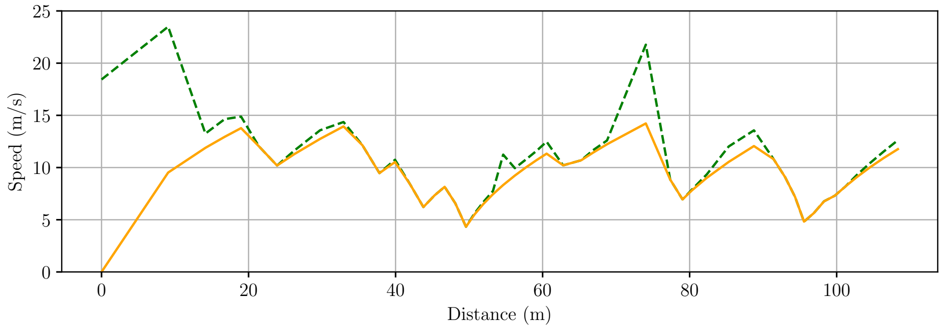
where U_x is the speed profile which is the piecewise linear approximation of momentary speeds $v(s)$. Adding the soft constraint term $P_{offtrack}$ we formulate the



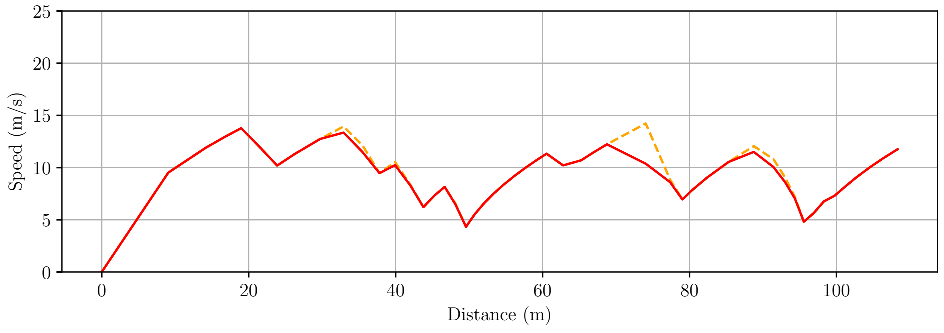
(a) Curvature profile k of path from Figure 2.11.



(b) Initial speed profile resulting from Equation (16). It accounts only for centrifugal force at each point separately.



(c) Speed profile adjusted for the formula's acceleration a_{max} using Algorithm 2.



(d) Final speed profile adjusted for the formula's braking ability a_{min} using Algorithm 3.

Figure 2.10: Speed profile algorithm illustrated on path from Figure 2.11 using parameters from Table 2.

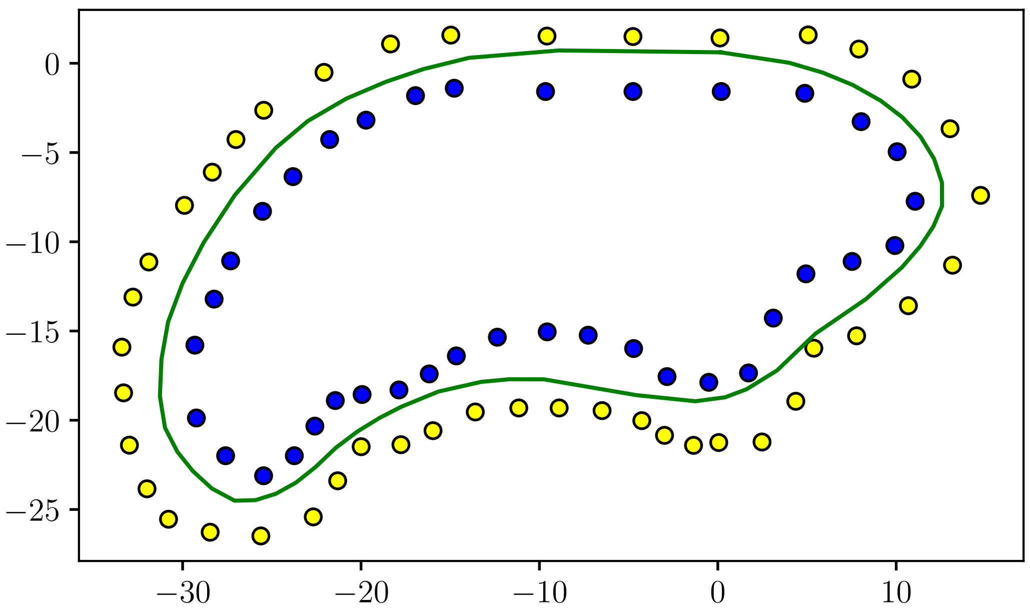


Figure 2.11: Time–minimal trajectory

optimization problem

$$\underset{\lambda_1,\dots,\lambda_D}{\arg\min} \quad T(\lambda_1,\dots,\lambda_D) + P_{offtrack}(\lambda_1,\dots,\lambda_D) \tag{19a}$$

$$\text{subject to} \quad \lambda_1 = \lambda_{D-1}, \tag{19b}$$

$$\lambda_2 = \lambda_D \tag{19c}$$

The optimization constraints defined by Equation (19b) and Equation (19c) ensure the closed nature of the resulting trajectory. We require two constraints to take into account every set of three consecutive points in Equation (12), which is involved in the computation of the speed profile.

When we use the appropriate solving tools, we obtain the trajectory shown in 2.11. These tools are discussed in the next section.

Chapter 3

Implementation

In the previous section, we have defined several optimization problems that represent their respective objectives. This section focuses on efficiently computing a solution to these problems in the context of our application. We begin by discussing how to choose the right software library. Next, we describe the code that implements the algorithm, as well as other supporting programs such as visualization code or the race track modification utility tool in Section 3.1. Finally, we present the communication of the algorithm with the rest of the autonomous pipeline in greater detail in Section 3.2.

The optimization problems defined in Section 2 are too complex to be solved analytically. Instead, we implement iterative optimization methods to solve them numerically. This approach requires an automatic differentiation library.

Furthermore, we are limited in our choice of automatic differentiation library by ROS, which is currently implemented in C++, Python, and LISP [26]. In eForce Driverless, we do not use LISP at all, and we follow a simple rule of thumb regarding C++ and Python: avoid C++ unless it is necessary. In practice, we write everything in Python, and only when the component is proven to be a performance bottleneck in the pipeline, we rewrite it in C++. Therefore, we implemented the path planning algorithm in Python.

Fortunately, thanks to the boom of deep neural network learning, Python offers a diverse ecosystem of automatic differentiation libraries. The leading positions among them are occupied by PyTorch and TensorFlow. Both have their own advantages. We decided for the former because we consider PyTorch to be cleaner and more intuitive. Additionally, the convolutional neural network which detects cones on a camera image is built on PyTorch, while TensorFlow is not currently used anywhere in eForce Driverless.

Section 3.1

System architecture

The core of the algorithm is formed by a model–solver pair. The former defines the optimization problem, while the latter determines how the given problem will be solved. These pairs are stored in a container class which simplifies manipulation with them and implements a load/save mechanic.

The model consists of a class hierarchy at the top of which sits the *PathModel* class. *PathModel* class extends the PyTorch base class *torch.nn.Module* and defines the shared aspects of each model. These include model initialization, track boundary enforcement, and individual measurements like path length and curvature. The sequence $(\lambda_i)_{i=1}^{D}$ specifying a path is wrapped in *torch.nn.Parameter*. The particular model sub–classes such as *LengthModel*, *CurvatureModel* or *TimeModel* inherit from *PathModel* and implement their own version of the *forward* method. The *forward* method specifies the forward pass of the model, in which the objective function of the appropriate optimization problem is evaluated using the current values of $(\lambda_i)_{i=1}^{D}$. The backward pass that updates $(\lambda_i)_{i=1}^{D}$ is performed automatically by PyTorch's automatic differentiation package *torch.autograd* and the optimizer used.

Compared to the model, the solver class is simple. It stores the name of the selected optimizer, a dictionary of keyword arguments to be passed to the optimizer later, and a number of iterations for which the optimizer is to be run. The optimizer itself is only created when the optimization begins since the construction of an optimizer requires the model parameters. Currently, we only support optimizers from *torch.optim*, which, however, offers a wide range of ready to use optimizers. The selected optimizers are shown in Figure 4.7. Moreover, the PyTorch optimizer base class *torch.optim.Optimizer* allows us to easily define new special optimizers which are compatible with predefined optimizers in PyTorch and our algorithm.

But before we start optimizing, we need to parametrize the track and initialize the model. The first is completed in the preprocess module, which follows the procedure described in Section 2.2. The DAG solver module computes the initialization for the length optimization problem through dynamic programming techniques as outlined in Section 2.3.1. Generally, we initialize $(\lambda_i)_{i=1}^{D}$ to "centerline", which means $\lambda_i = 0.5$ for $i = 1, \dots, D$.

Finally, our algorithm is supported by a visualization and race track editing module which aids in the testing process. Both modules are based on Python's matplotlib library. Visualization is composed of multiple plots, for example, the race track plot on which the cones, paths, or transverse parametrization lines are drawn. To maximize clarity, we only render plots that have received data and dynamically position them to make use of the space freed by the unused plots. The edit module provides us with a way to insert or remove cones from a race track. It is controlled by mouse clicks and key bindings that switch between operation modes. Finally, we connected it to the reactive path planning algorithm to aid in its development.

Section 3.2

Pipeline integration

Our algorithm is not meant to exist on its own. As a node in the ROS framework, it consumes and provides data to other ROS nodes. This communication was concisely described in Figure 1.1 in the Introduction section, but given its importance, it is beneficial to expand this explanation.

The handling of incoming data is relatively simple. In reality, SLAM offers both the cone map and the current car position, which is aggregated by our algorithm into a path driven in the first lap. In FSDS (see Section 4.3) the cone positions and the location of the car are provided as ground truth data by the simulator. Because the simulator is based on ROS1, integrating SLAM would require rewriting it in Python 2.7, which is a lot of effort for little gain.

Sending outgoing data requires more thought. First, as Section 4.2 shows, the full optimization can take an entire minute, which is potentially enough time to finish a single lap. Fortunately, the iterative nature of our algorithm allows for the publication of intermediate results. However, because Table 4 confirms that the main benefit of this algorithm in the context of a typical Formula Student race track lies in the speed profile, we are already able to gain a substantial advantage in the first iteration. Taking into account the time required to parameterize the track and perform other preprocessing steps, the first results become available in a span of several seconds after receiving the required input.

Before the given trajectory can be sent to the control algorithm, it must be transformed into the vehicle coordinate system. This step begins with the localization of the car on the track based on its position and heading on the SLAM map. Position and heading are then used to translate and rotate the trajectory to align it with the current vehicle coordinate system.

Like reactive path planning, the control algorithm is also not the author's work. It implements the well–known Stanley control algorithm [6] and has been written by a fellow member of eForce Driverless, Hynek Zamazal. The control algorithm has already been tested and calibrated in both FSDS and real life.

Chapter 4

Validation

The algorithm proposed in Section 2 is designed to be modular. We rely on a bottom–up testing approach where constituent components are first tested individually, then we assess the entire path planning algorithm, and finally we verify its integration with the car's autonomous pipeline that was outlined in Section 1.2.

During the design process, we relied on matplotlib visualizations to prove validity and correctness on module–wide and algorithm–wide scope. Our input data come from sources described in Section 4.1. We present our component testing procedures in Section 4.2. Once we judged that the entire algorithm was ready to use, we validated its integration with the autonomous pipeline in a vehicle simulator. We demonstrate our simulation results in Section 4.3.

Section 4.1

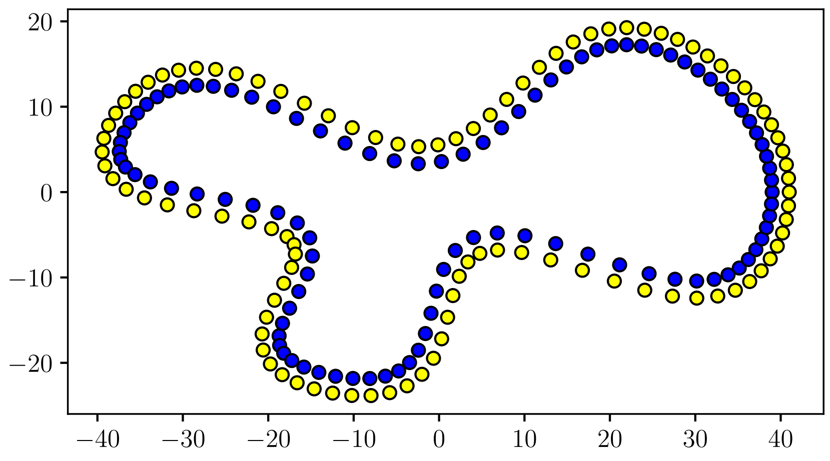
Data sources

The testing of the constituent components is based on several sources of validation data sets. Initially we used an inherited race track generator which has been developed very early in the history of eForce Driverless. Later, our internal track generator was complemented by a second track generator created by a foreign Formula Student team [27]. When our team overcame a few technical obstacles relating to INS and CAN handling, we were able to procure SLAM maps created from real–world data. Finally, we programed a graphical utility tool, which allowed us to manually modify existing cone maps or create completely new ones from scratch; see Section 3.

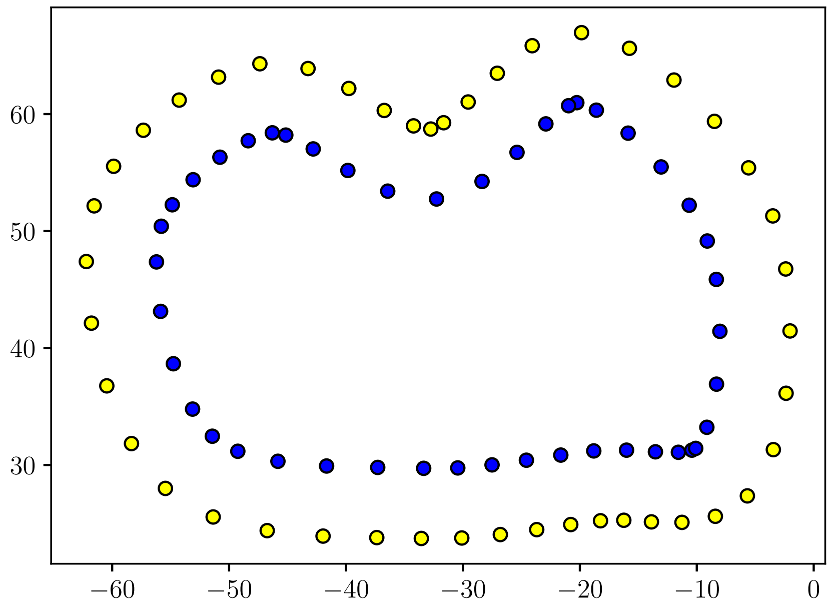
We start by noting that the eForce track generator is not the author's work. It was originally written by fellow eForce Driverless member Tomáš Roun to provide testing data for the reactive path planning algorithm. Its principle is simple. Initially, it allocates a preset number of points along a circle with parametrizable radius and then distorts their location through a few Gaussians. This elementary approach produces clean tracks, such as the one depicted in Figure 4.1a. The obvious drawback is the low variability of possible outputs, because it is impossible to significantly change the shape of the resulting race track. On the other hand, because the generated race tracks are so close to being ideal, we consider this generator to be suitable for testing at the start of development.

To account for the low diversity of the output of the eForce generator, we introduced a second race track generator [27]. It has been developed by the Formula Student team eRacing of the Brazilian University in Campinas. It offers more varied race tracks, such as the one depicted in Figure 4.1b. Nevertheless, it still does not provide race tracks sufficiently general to test our algorithm in its entirety. The problem lies in the absence of noise in the cone positions.

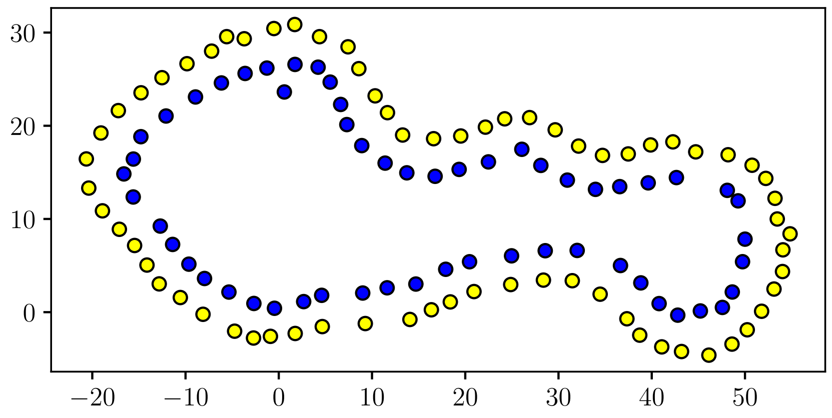
This problem can easily be solved by adding noise to an existing race track. But we did not need to do this, because when our team integrated SLAM into our ROS system in the autumn of 2021, we gained access to SLAM maps based on formula perception abilities. Therefore, we obtained the most accurate and representative data possible; see Figure 4.1c.



(a) eForce generator



(b) eRacing generator



(c) Real cone map from SLAM algorithm [15].

Figure 4.1: Examples of race tracks created by introduced data sources.

The most severe downside of SLAM maps is that collecting real–world sensory input is time consuming. Due to organizational constraints, it is not feasible to organize more than a dozen testing events per racing season. Although SLAM has worked since October 2021, we have collected only three SLAM maps to this day. Usually, we would be content with a more lightweight solution than organizing a new testing event, which requires the participation of at least 6 team members. Typically, we wish to slightly modify an existing race track and observe the impact of the change on the algorithm. For such reasons, we have developed a graphical editor of race tracks that allows us to add or remove cones with a click of a mouse. The editor has also been integrated with the reactive path planning algorithm and also with the parametrization process developed in this thesis, allowing for swift workflow.

Section 4.2
Component testing

This subsection roughly follows the structure of Section 2. We begin by choosing whether to sample the curvature adjusted points $(\boldsymbol{d_i})_{i=1}^{D}$ on the "centerline", which is the sequence of middle points between the left and right track boundaries, or on the original first–lap trajectory. Then we investigate the influence of the discretization parameter D on the produced time–minimal trajectories. Next, we compare the trajectories resulting from the optimization problems defined in Section 2 and explore their relationships with each other in the context of the car parameters from Table 2. Finally, we analyze the convergence rate of our optimization problems and compare the performance of selected optimizers. All experiments in this section are based on the track from Figure 4.1c because it offers the most realistic conditions that we have available.

When the algorithm receives a new SLAM map, its first step is to estimate the track boundaries, as described in Section 2.2. In Figure 4.2, the closest point to each cone is found on the first lap path. Subsequently, cone projections are sorted on individual path segments, and the sorted segments are connected, naturally sorting the cones in the formula's direction of travel. The ordered cones together form a piecewise linear approximation of the track boundaries according to our wishes.

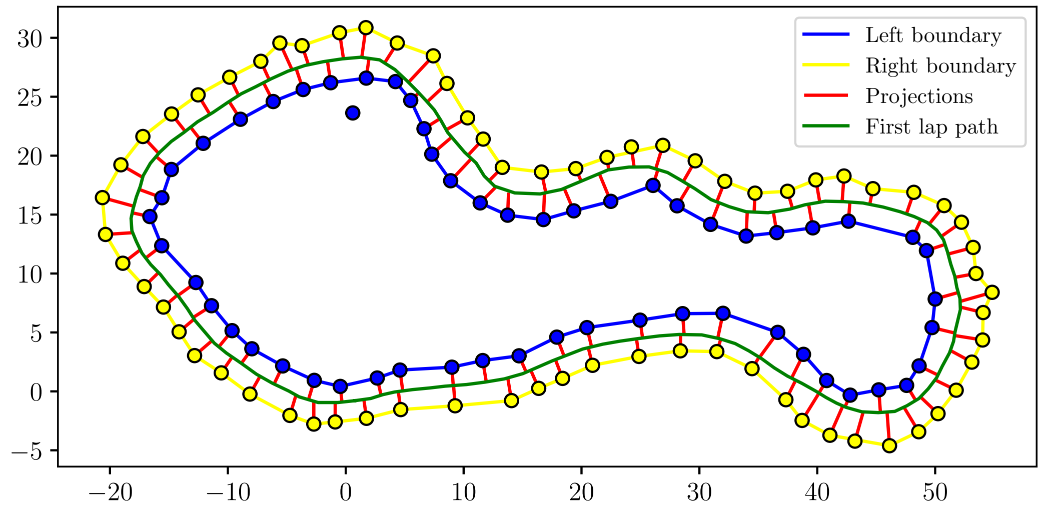


Figure 4.2: Projection of cones onto the first lap path and the track boundaries.

Once we have obtained the track boundaries in the previous step, we move on to calculating a parametrization of the race track. Here, we face a new decision. We either follow Section 2.2.2 and use the first lap path or we estimate a "centerline" path from the track boundaries. This alternative path is defined as the arithmetic mean of the left and right track boundaries. In Figure 4.3, sampling from the first lap path leads to the parametrization consisting of the red transverse lines, while sampling from the alternative path results in the orange lines. In general, both approaches lead to similar results; however, the former is more robust in parts of the track with high curvature. This is apparent in Figure 4.3 in the leftmost turn, where the red lines, unlike the orange ones, do not cross each other inside the track boundaries.

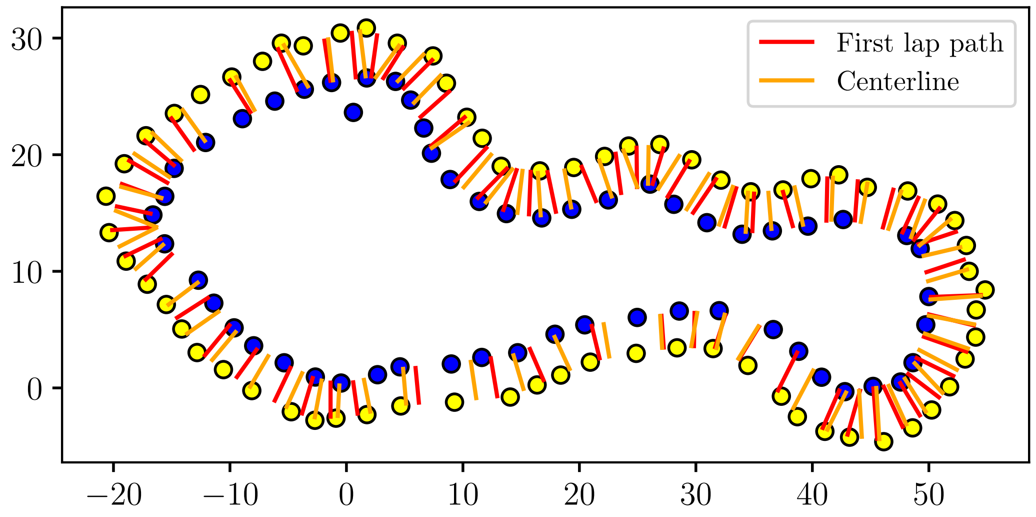


Figure 4.3: Two parametrizations of a SLAM map. Orange lines are built on an estimation of centerline. Red lines are created from sampling from the first lap path. Discretization parameter D is 64 in both cases.

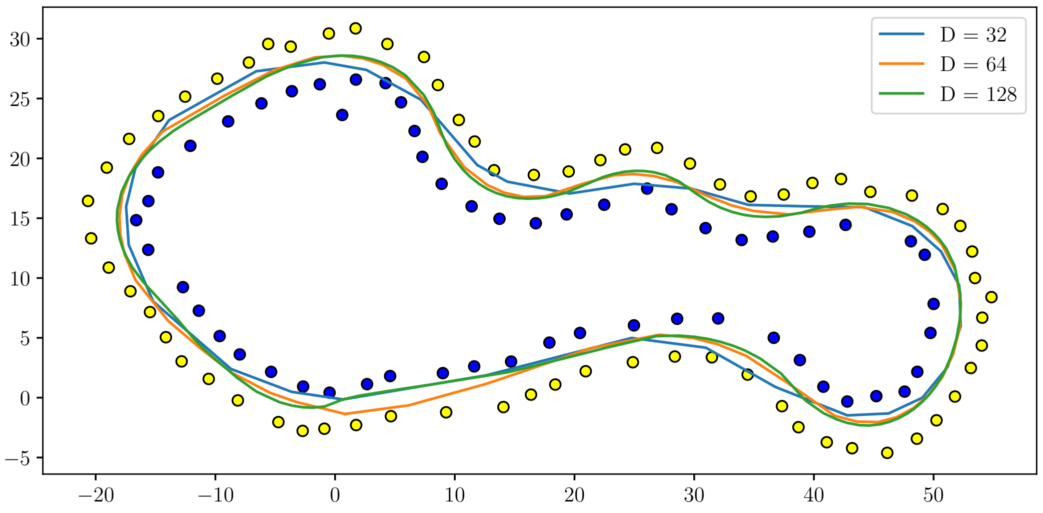


Figure 4.4: Trajectories produced from parametrizations using different values of parameter D.

As Figure 4.4 proves, the most important parameter to set correctly in this step is the discretization parameter D. Intuitively, lower values of D lead to a shorter computation time but less accurate trajectories, while higher values result in the opposite result. We believe that a simple rule of thumb exists. The obvious approach is to set D to the number of cones present. The cones are, by definition, placed in a way that clearly delineates the track without being placed unnecessarily densely. Figure 4.4 consists of 128 cones of blue or yellow color. At the same time, the orange parametrization with $D = 64$ performs the best from our selection of parameters D. Unfortunately, we were unable to prove conclusively the validity of this rule.

Figure 4.4 suggests the viability of periodically increasing the discretization parameter D during an optimization. Such a scheme would be analogous in principle to learning rate schedulers, which are widely used in the training of neural networks. Once again, we lacked time to pursue this idea further.

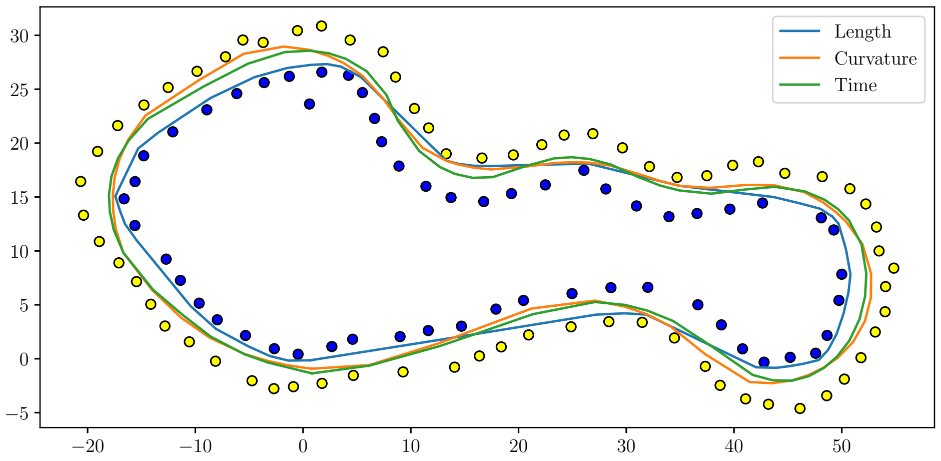
Next, we compare the trajectories resulting from the three optimization problems defined in the course of Section 2. In the Formula Student competition, the track boundaries are usually placed only the minimal distance of 3 m apart. Taking into account the width of the car of 1.6 m, we receive a tight maneuvering space. Therefore, we do not expect significant differences between the individual trajectories. Our expectation is confirmed by Figure 4.5a, which shows that the optimal trajectories for the curvature and time optimization problems remain closely together.

However, Figure 4.5 does not provide a complete picture. In Section 2.3 we mentioned a view of the time–minimal trajectory as a certain combination of properties of the length–minimal and curvature–minimal trajectory. The precise ratio is informed by the ability of the car to execute the trajectory. If we consider a car equipped with ideal tires that are capable of transferring any force, the fastest trajectory will always be the shortest. On the other hand, as the term μF_z in Equation (14) decreases, the fastest trajectory approaches the trajectory with the lowest average curvature.

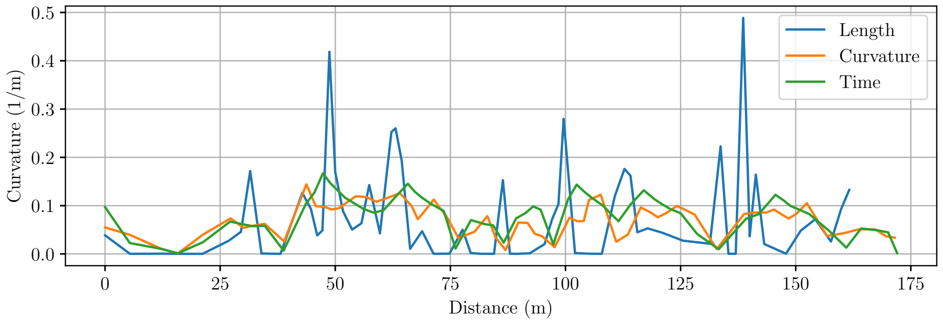
Acceleration abilities that govern the optimal speed profile follow the same pattern. Higher acceleration a_{max} and braking capabilities a_{min} allow a more aggressive driving style. In turn, the ability to quickly change speed promotes driving nearer to the shortest trajectory: as Figure 4.5a shows, the length–minimal trajectory combines long straight segments with sharp turns. In contrast, the curvature–minimal trajectory possesses a more balanced speed profile. Given our car's low acceleration, its fastest trajectory is significantly closer to the curvature–minimal trajectory than to the length–minimal.

The other major source of influence are the different convergence rates of optimization problems for curvature and time, as illustrated in Figure 4.6. Because the objective function in Equation (18) is more complicated than the objective function in curvature optimization defined in Equation (12), it takes longer for the optimizer to find a local minimum. For example, in Figure 4.6 we see that the curvature reaches its local minimum at approximately the 1000th iteration, while time continues to decrease even at iteration 2000. In comparison, we regularly schedule approximately 500 iterations per optimization problem.

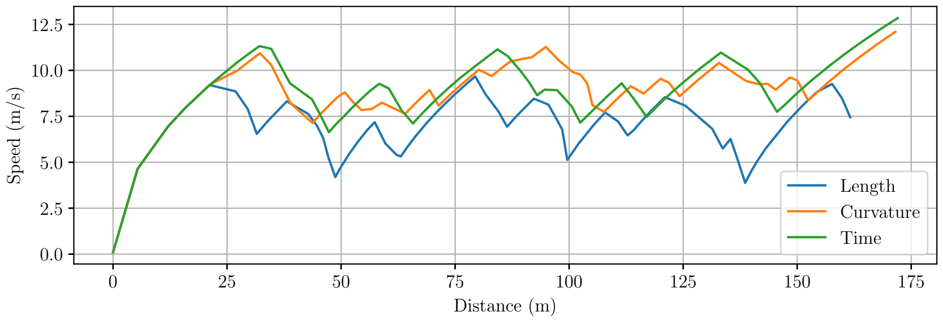
We also influence the convergence rate by selecting the appropriate optimizer. We solve optimization problems using the optimizers from package *torch.optim* (see Section 3.1). In Figure 4.7 we compare the speed with which they approach a local minimum of the time optimization problem. Subsequently, we plot the average time required for a single iteration of this problem in Figure 4.8. All optimizers were constructed with a learning rate of 0.001 and no other parameter. We conclude that most optimizers are largely equivalent both in terms of convergence and computational cost. We choose the Adam optimizer because it is a currently widely–used optimizer designed for optimizing non–convex functions, especially neural networks. LBFGS is an outlier in both graphs because it is the sole second–order method, which evaluates the function gradient multiple times in a single iteration. Its complex nature seems unsuitable for optimizing a complicated function, such as our time–minimization problem.



(a) Trajectories corresponding to optimization problems defined in Section 2. The starting line is at the beginning of the coordinate system.



(b) Curvature profiles



(c) Speed profiles

Figure 4.5: Comparison of trajectories produced by different optimization problems and their associated curvature profiles and optimal speed profiles. The speed profile algorithm is set to parameters from Table 2.

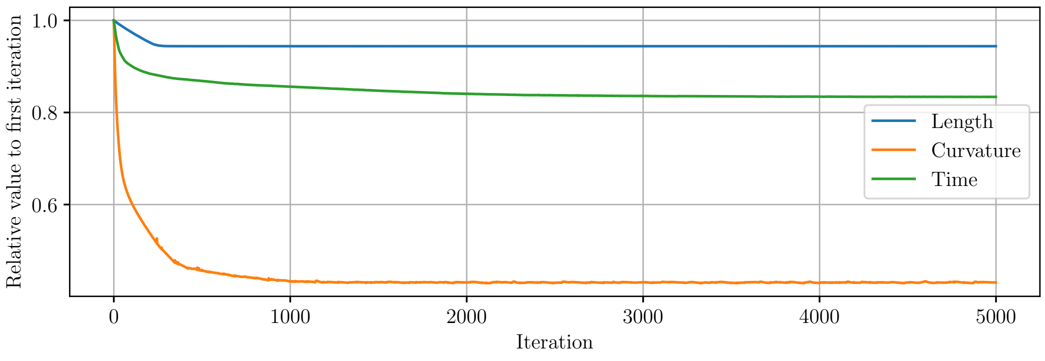


Figure 4.6: Relative improvement of different optimization problems. We plot the ratio of value in the i–th iteration to value obtained in the first iteration. We use the Adam optimizer with a learning rate of 0.001.

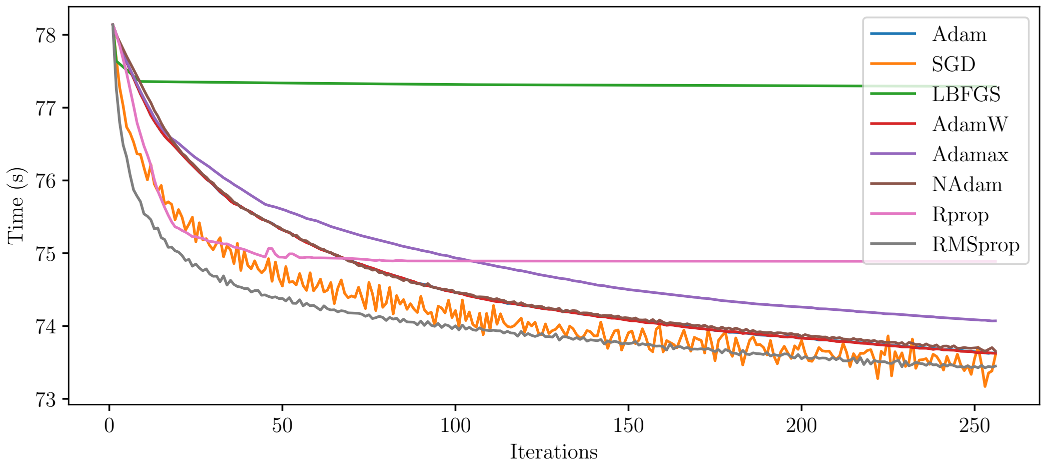


Figure 4.7: Comparison of convergence rates of selected optimizers from *torch.optim* on the time optimization problem from Equation (19a). Adam and AdamW optimizers coincide in the figure. Curiously, SGD would benefit from a lower learning rate than the value of 0.001 commonly used in this thesis.

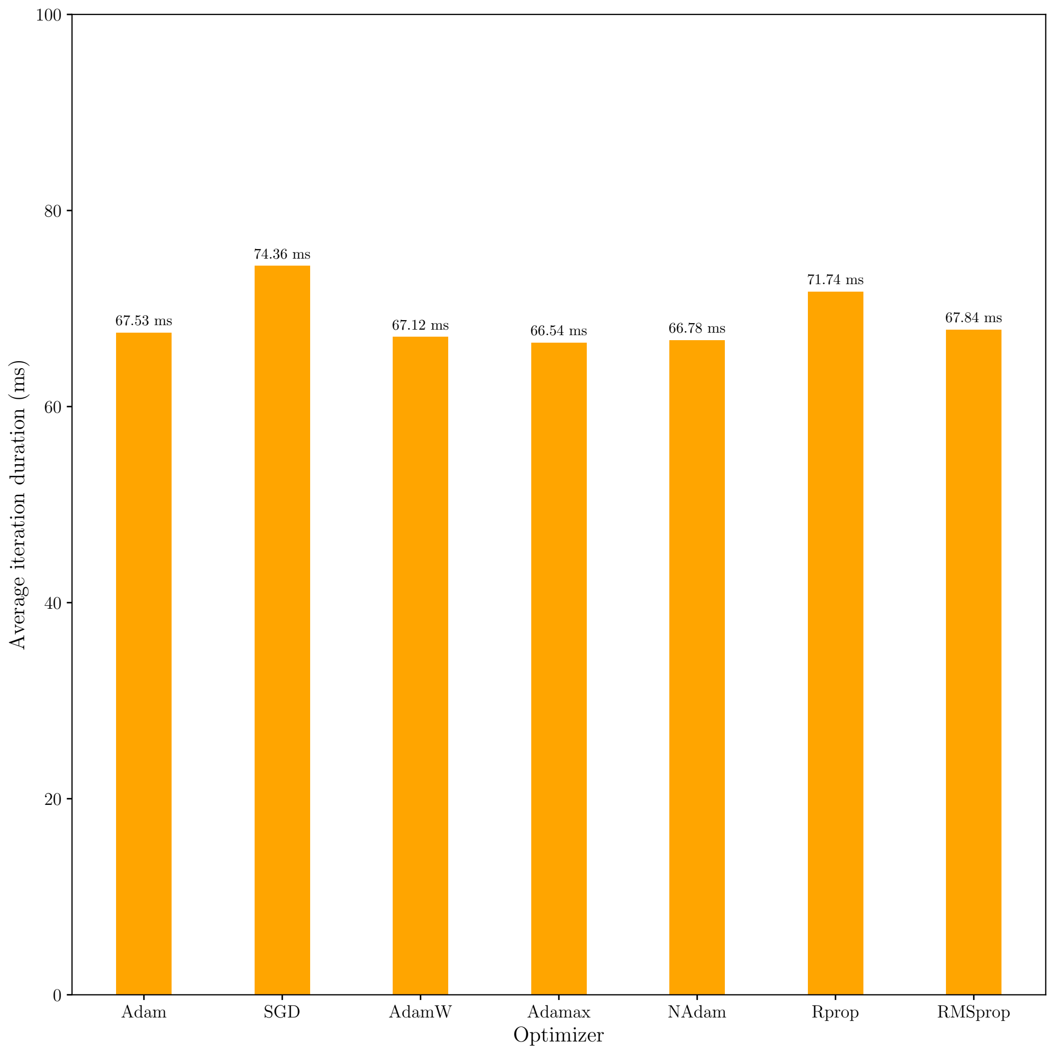


Figure 4.8: Time required by selected optimizers for a single iteration of the time optimization problem from Equation (19a) for SLAM map from Figure 4.1c with 64 transverse lines. In this experiment, a single iteration of LBFGS took over 1.5 s and therefore was not included in this graph due to the difference in scale.

Section 4.3

Formula Student Driverless Simulator

The entire 2020 racing season had to be canceled due to COVID-19 restrictions in place at the time. Instead, an online event called Formula Student Online was held as a replacement and it was the only competition of the 2020 season. This prompted the development of the Formula Student Driverless Simulator (FSDS), in which teams would compete against each other in the online competition. FSDS is built on Microsoft AirSim [28], which is, in turn, based on Unreal Engine 4 and offers a ROS interface for communication with the connected autonomous system. Because the organizers of Formula Student Online have envisioned FSDS to serve the teams beyond the online competition from the beginning, it is supported and continuously updated to the present day. Our team eForce Driverless was one of the four driverless teams which competed in the driverless category of Formula Student Online. Therefore, we have an autonomous system that is extensively tested in the context of FSDS. Recently, we explored the possibility of using other simulators such as Carla or Carmaker. However, FSDS currently stays as our main simulator.

In 2020, eForce Driverless has competed in Formula Student Online with a simple system of reactive path planning described in Section 2.1.1 and an implementation of the pure pursuit control algorithm. Since then, the reactive path planning algorithm has been improved, and pure pursuit has been replaced with Hynek Zamazal's implementation of the Stanley control algorithm [6]. We use the current version of these algorithms to obtain a baseline against which we measure the performance of our algorithm.

We test our algorithm on the competition track from the first race day of the Formula Student Online competition. Therefore, this is in fact the first track that eForce Driverless has ever competed on. The video footage can be found on YouTube [29] and contrasted with the results presented in this section. For reference, in 2020 we completed this track in 85 s.

The physical model of FSDS assumes different parameters for the car compared to Table 2. The parameters used for FSDS can be found in Table 3. As the true parameters used in the internal physics engine of FSDS remain unknown to us, we estimate them from experiments. The same settings of the speed profile algorithm are used in all the experiments performed in this section.

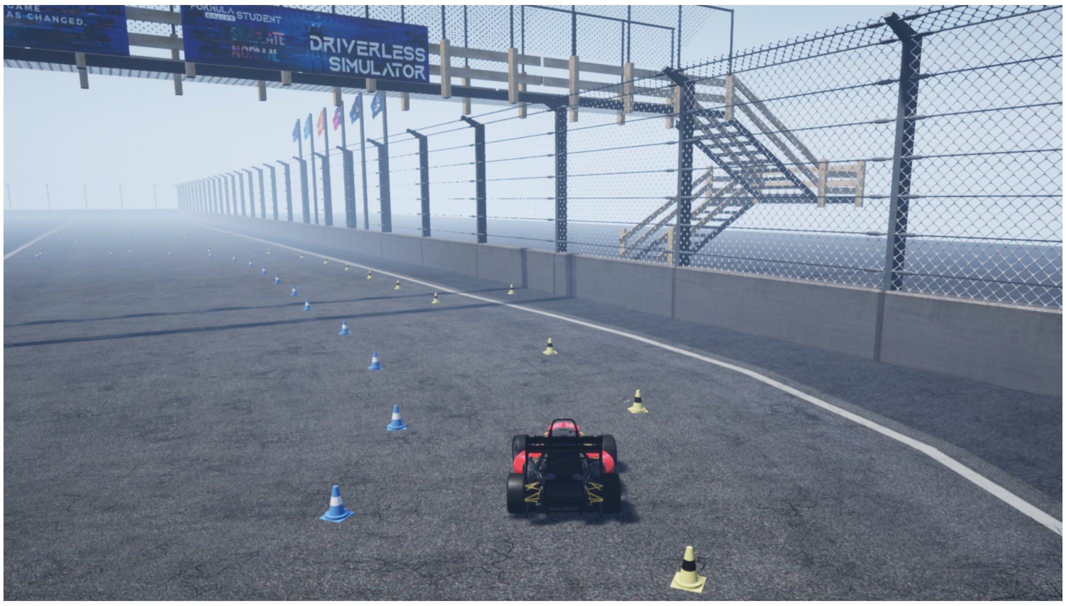
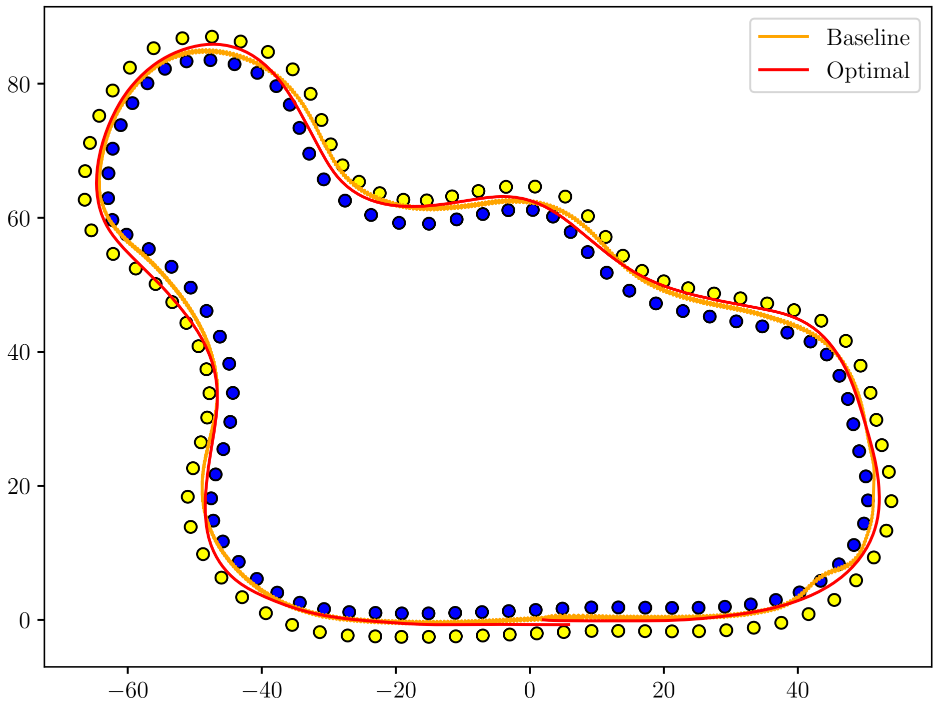


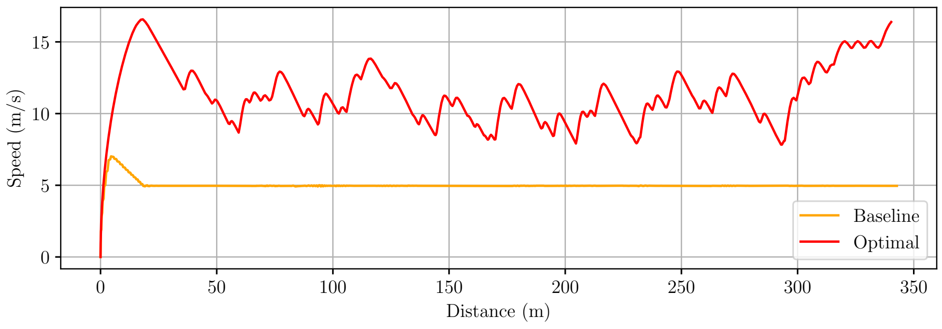
Figure 4.9: Screenshot of Formula Student Driverless Simulator

Parameter	Symbol	Value	Unit
Friction coefficient	μ	0.5	—
Formula mass	m	250	kg
Coefficient of lift	C_L	-3.5	—
Coefficient of drag	C_D	0.3	—
Reference aerodynamic area	A_{ref}	1.14	m^2
Maximum acceleration	a_{max}	4	$\mathrm{m/s}^2$
Maximum deceleration	a_{min}	-2	$\mathrm{m/s}^2$

Table 3: Parameters from Table 2 updated for a simulated formula in FSDS [30]. Parameters which cannot be found in the documentation (μ, a_{max}, a_{min}, C_L, and A_{ref}) are set either experimentally or taken from experiments previously performed by Marek Boháč [9]. The most unusual feature of FSDS physics engine is that vehicles accelerate better than they brake.



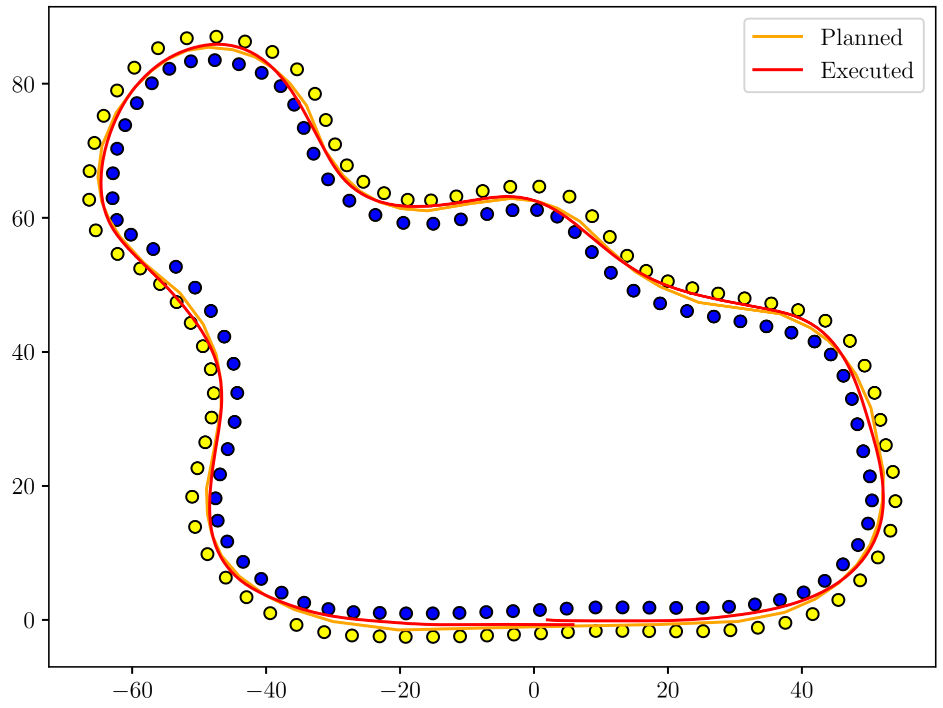
(a) Trajectories of baseline and our algorithm compared on a map.



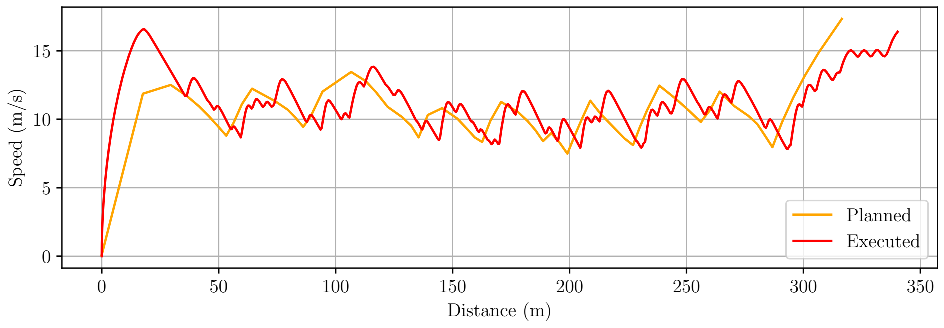
(b) The reactive path planning supplies a constant velocity of $5\,\mathrm{m\,s^{-1}}$, but the regulator initially overshoots the target, before it settles down at the 20th meter of the trajectory. The optimal speed profile steadily maintains approximately double the speed.

Figure 4.10: Trajectory realised by the reactive path planning algorithm is plotted in orange, while the optimal trajectory computed by our algorithm is drawn in red.

In Figure 4.10 we compare the trajectories performed by the Stanley control algorithm from different input trajectories. The baseline trajectory was produced by the reactive path planning algorithm from Section 2.1.1, whereas the time–optimal trajectory was planned by our algorithm. The baseline trajectory is accompanied by the currently used constant speed profile of 5 m, our optimal trajectory uses the speed profile algorithm presented in Section 2.4.1. We see that the optimal trajectory deviates from the baseline trajectory quite significantly, considering the limitations discussed above regarding the track width. Intuitively, we expect these changes to have a positive impact on lap time. Our expectations are confirmed by the results, which are presented in Table 4.



(a) Trajectory planned by our algorithm versus the same trajectory as it was travelled by the formula in FSDS.



(b) Planned speed profile compared with its realization in FSDS.

Figure 4.11: Comparison between the planned fastest trajectory in orange and the corresponding trajectory realised by the control algorithm in FSDS.

In Figure 4.10a the formula managed to stay on track, but it hit multiple cones. Figure 4.11a investigates what causes this behavior. It shows that the traveled path in red always lies closer to the inner edge of a turn than the planned trajectory, which is drawn in orange. Therefore, the responsibility for this phenomenon lies with the control algorithm. However, we can mitigate this problem in our algorithm by setting the margin M from Section 2.2.4 higher.

In Figure 4.10b we contrast the planned speed profile with the speed profile realized by the control algorithm. Although both profiles are quite similar, there are regions where they deviate. We identify two types of error. First, the experimentally estimated parameters from Table 3 we used are not completely equal to to the parameters used by the simulator. Second, the regulator cannot flawlessly execute the planned speed profile. Although some difference is always present, we suspect that the simple linear interpolation scheme exacerbates the issue.

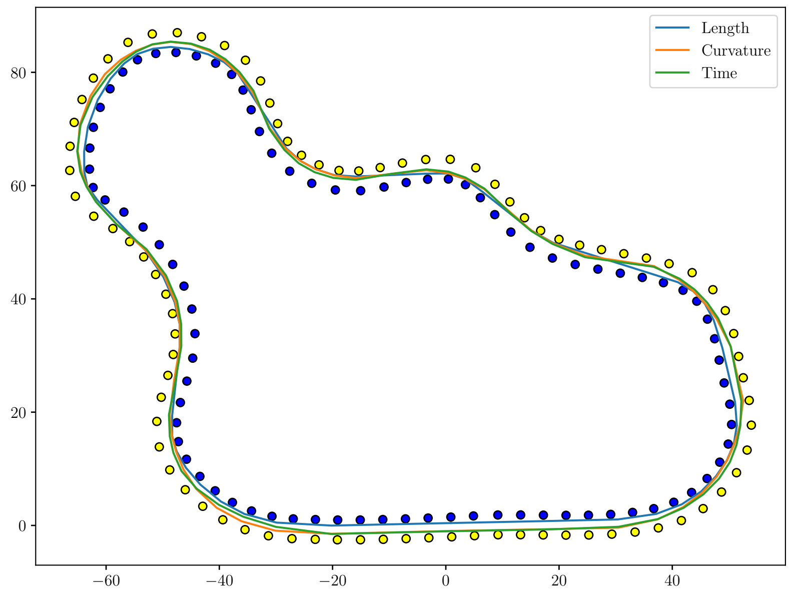
Planning objective	Speed profile	Lap time	Relative improvement
Baseline	Constant $5\,\mathrm{m\,s^{-1}}$	66.84 s	1
Time	Constant $5\,\mathrm{m\,s^{-1}}$	66.65 s	1
Length	Yes	33.46 s	2
Curvature	Yes	30.76 s	2.17
Time	Yes	30.48 s	2.19

Table 4: Overview of lap times resulting from different configurations of our formula's path planning algorithm in FSDS.

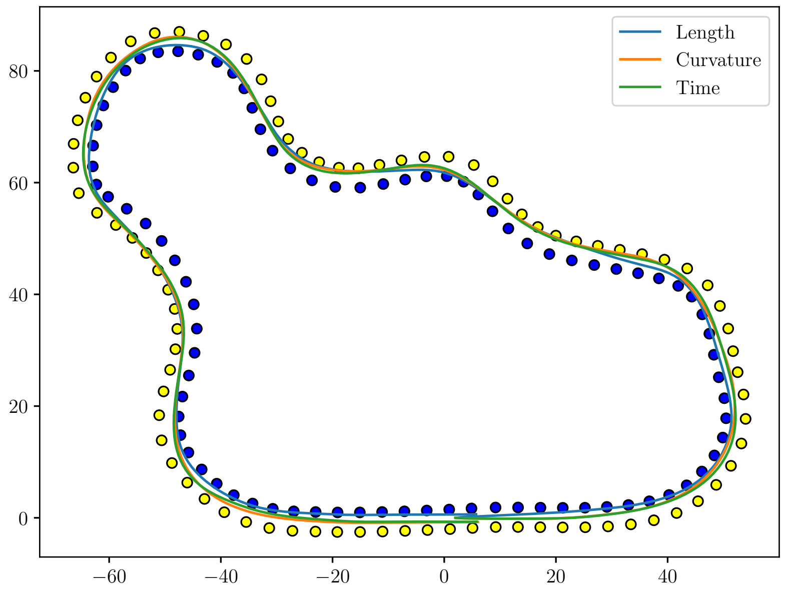
Given the parameters from Table 3, we expect the time–minimal trajectory to be close to the curvature–minimal trajectory. Figure 4.12 largely confirms our hypothesis, as the trajectory minimizing average curvature and the fastest trajectory barely differ at any point. The high degree of similarity of both trajectories is reflected in their speed profiles: in Figure 4.12c, the green profile of the fastest trajectory is mirrored by curvature's orange profile only a fraction of a unit lower on the vertical axis.

Arguably, the small width of the typical Formula Student race track prohibits any substantial spatial differentiation between "reasonable" trajectories. This is true, but because we focus specifically on the Formula Student competition, any other unorthodox tracks are not of immediate practical interest to us.

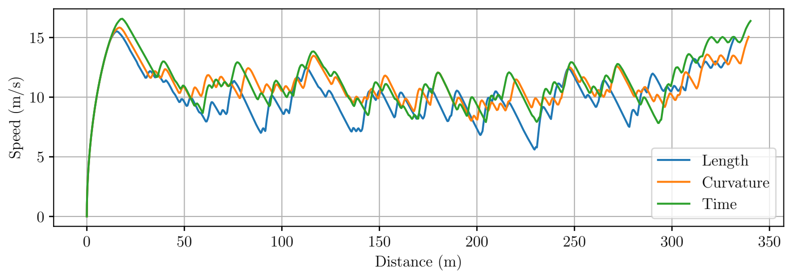
The lap times resulting from different optimizations are summarized in Table 4. Data in this table match the trajectories shown in other figures in this section. Trajectories from bottom three rows are shown in Figure 4.12c.



(a) Length, Curvature and Time minimal trajectories planned for FSDS by our algorithm.



(b) Trajectories corresponding to Figure 4.12a as they were realised by the control algorithm.



(c) Speed profiles corresponding to Figure 4.12a as they were executed by the control algorithm.

Figure 4.12: Comparison of realized trajectories for different optimization problems.

Chapter 5
Conclusion

This thesis proposed a trajectory planning algorithm for an autonomous formula built for the international Formula Student competition. We have decomposed trajectory planning into several constituent components.

In Sections 2.1 and 2.2, we prepared a parametrization scheme that produces a suitable problem representation from a SLAM cone map. Our approach filters out major errors in the input data and is sufficiently robust to handle the uncaught minor errors. We designed a flexible parametrization method, which can be successfully applied to any race tracks valid in the context of Formula Student rules. Moreover, our representation saves computational resources by adapting the parametrization density to the local path curvature.

In the future, we plan to interpolate between cones that form the track boundaries or points that make up the trajectory with cubic splines. In contrast to the currently used linear interpolation, cubic splines provide a better interpolation curve. Moreover, we can ensure the smoothness of the first derivative at each point of the interpolation, including the knot points. Because cubic splines are polynomials of degree 3, we can reformulate Equation (12) to a continuous rational function derived from Equation (9). Then, we use it to replace the mechanism of osculating circles [19]. We expect this change to reduce the non–convexity of the resulting optimization problem and improve its convergence speed.

In Sections 2.3 and 2.4, we derived length and curvature measurements for a path parameterized by our method. We formulated optimization problems which minimize these objectives. Then, we discussed the physical laws that affect the formula on the race track and implemented a speed profile algorithm that determines the optimal speed of the formula along the supplied path. Finally, we combined everything together to obtain the fastest trajectory.

At the present time, we use a naive and simple physical model; for example, we neglect tire modeling. Therefore, extending the physical model will increase the accuracy and reliability of our algorithm. Currently we also have the data needed to dynamically estimate μ instead of relying on a constant value. Implementing an estimation method will also make our model more realistic.

In Sections 3 and 4, we implemented our method in Python and connected it with the rest of the autonomous pipeline of eForce Driverless. We verified our work on testing data from multiple sources, including a SLAM map assembled from real sensory input. Moreover, we proved that our algorithm works properly in a closed loop with the control algorithm in the Formula Student Driverless Simulator. Finally, we demonstrated a measurable improvement in total lap time.

The most imminent task that we face today is to implement our algorithm directly in the formula. Although this will involve a new set of challenges, it also offers several new opportunities. For example, we plan a feedback mechanism to visually verify that the formula stays within track boundaries during all 9 trackdrive laps that are navigated by our algorithm. In this regard, the most important test to come will be the upcoming 2022 competition season. We will see how will the new improvements to the autonomous pipeline of eForce Driverless, including our trajectory planning algorithm, perform in Formula Student races in the Czech Republic, Italy and Hungary.

Chapter A

References

[1] Edgar Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[2] Peter Hart, Nils Nilsson, Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[3] Egbert Bakker, Hans B. Pacejka, Lars Lidner. A new tyre model with an application in vehicle dynamics studies. *SAE Transactions*, 98:101–113, 1989.

[4] Marco Gadola, David Vetturi, Danilo Cambiaghi, and Luca Manzo. A tool for lap time simulation. Technical report, SAE Technical Paper 962529, 1996.

[5] T J Gordon, M C Best, and P J Dixon. An automated driver based on convergent vector fields. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 216(4):329–347, 2002.

[6] Sebastian Thrun, Michael Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, and Pamela Mahoney. Stanley: The robot that won the darpa grand challenge. *Journal of Field Robotics*, 23:661–692, 1 2006.

[7] Alexander Liniger. *Path Planning and Control for Autonomous Racing*. PhD thesis, ETH Zurich, 2018.

[8] Formula Student Germany. Autonomous driving at formula student germany 2017. `https://www.formulastudent.de/pr/news/details/article/autonomous-driving-at-formula-student-germany-2017/`, 2016.

[9] Marek Boháč. Design of control system for an autonomous racecar. Bachelor's thesis, Czech Technical University, 2020.

[10] Juraj Kabzan, Miguel de la Iglesia Valls, Victor Reijgwart, Hubertus Franciscus Cornelis Hendrikx, Claas Ehmke, Manish Prajapat, Andreas Bühler, Nikhil Bharadwaj 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. *CoRR*, abs/1905.05150, 2019.

[11] Leiv Andresen, Adrian Brandemuehl, Alex Honger, Benson Kuan, Niclas Vödisch, Hermann Blum, Victor Reijgwart, Lukas Bernreiter, Lukas Schaupp, Jen Jen Chung, Mathias Burki, Martin R. Oswald, Roland Siegwart, and Abel Gawel. Accurate mapping and planning for autonomous racing. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4743–4749, 2020.

[12] Adam Slomoi. Path planning and control in an autonomous formula student vehicle. Technical report, Monash University, 2018.

[13] Roman Šíp. Visual Detection of Traffic Cones for Autonomous Student Formula. Bachelor's thesis, Czech Technical University, 2022.

[14] Daniel Štorc. Detection of Traffic Cones from LiDAR Point Clouds. Bachelor's thesis, Czech Technical University, 2022.

[15] Tomáš Roun. Navigation system for autonomous student formula. Master's thesis, Czech Technical University, 2021.

[16] Formula Student Germany. Formula student rules 2022. `https://www.formulastudent.de/fileadmin/user_upload/all/2022/rules/FS-Rules_2022_v1.0.pdf`, 2022.

[17] Formula Student Germany. FSG handbook 2022. `https://www.formulastudent.de/fileadmin/user_upload/all/2022/rules/FSG22_Competition_Handbook_v1.1.pdf`, 2022.

[18] Nitin R. Kapania. *Trajectory planning and control of an autonomous race vehicle*. PhD thesis, Stanford University, 2016.

[19] Jan Filip. Trajectory tracking for autonomous vehicles. Master's thesis, Czech Technical University, 2018.

[20] David Vosahlik, Jan Cech, Tomas Hanis, Adam Konopisky, Tomas Rurtle, Jan Svancar, and Tomas Twardzik. Self-supervised learning of camera-based drivable surface friction. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 2773–2780, 2021.

[21] Kanwar Bharat Singh and Saied Taheri. Estimation of tire–road friction coefficient and its application in chassis control systems. *Systems Science & Control Engineering*, 3(1):39–61, 2015.

[22] Changsun Ahn, Huei Peng, and H. Eric Tseng. Robust estimation of road friction coefficient. In *Proceedings of the 2011 American Control Conference*, pages 3948–3953, 2011.

[23] Michal Bahnik, Dominik Filyo, David Pekarek, Martin Vlasimsky, Jan Cech, Tomas Hanis, and Martin Hromcik. Visually assisted anti-lock braking system. In *2020 IEEE Intelligent Vehicles Symposium (IV)*, pages 1219–1225, 2020.

[24] Bo Persson, U. Tartaglino, O. Albohr, and Erio Tosatti. Rubber friction on wet and dry road surfaces: The sealing effect. *Physical Review B*, 71, 03 2005.

[25] eForce. Engineering design report: Aerodynamic devices. Presentation of aerodynamics included in the Engineering Design discipline of Formula Student competition, 2018.

[26] Open Robotics. ROS introduction. `http://wiki.ros.org/ROS/Introduction`. Accessed: 07/05/2022.

[27] Lucas Barretto. Path following demonstration. `https://github.com/lucasbarretto/path_follower`, 2020.

[28] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. *CoRR*, abs/1705.05065, 2017.

[29] Formula Student Online. Formula Student Online - Day 1 Driverless Simulator Event Competition. `https://youtu.be/TCgKwuLo3Eo?t=1426`. Accessed: 18/05/2022.

[30] FSDS development team. Formula Student Driverless Simulator documentation. `https://fs-driverless.github.io/Formula-Student-Driverless-Simulator/v1.4.0/vehicle_model/#more-properties-of-competition-vehicles`. Accessed: 18/05/2022.