Minimum Curvature Trajectory Planning via Probabilistic Inference for an Autonomous Racing Vehicle

Scientific Thesis for the procurance of the degree M.Sc.
from the Department of Electrical and Computer Engineering at the Technical University of Munich.

Supervised by	PD Dr.-Ing. habil. Dirk Wollherr M.Sc. Salman Bari (LSR) M.Sc. Andressa de Paula Suiti (ARRK Engineering GmbH) Chair of Automatic Control Engineering
Submitted by	B.Sc. Ahmad Schoha Haidari - - -
Submitted on	Munich, 18.03.2022

TECHNISCHE UNIVERSITÄT MÜNCHEN
LEHRSTUHL FÜR STEUERUNGS- UND REGELUNGSTECHNIK
PRIV.-DOZ. DR.-ING. HABIL. DIRK WOLLHERR

September 10, 2021

M A S T E R ' S T H E S I S
for
Ahmad Schoha Haidari
Student ID 03659167, Degree EI

Minimum Curvature Trajectory Planning via Probabilistic Inference for an Autonomous Racing Vehicle

Problem description:

Over the past few years trajectory planning for autonomous racing vehicle has been the focus of immense research. The goal of the trajectory planner is to optimize the race-line or global trajectory in order to drive around a given race track as fast as possible by generating minimum curvature path. Another objective of the trajectory planner is to produce a velocity profile that keeps check on the longitudinal and lateral acceleration limits of the vehicle. Optimization-based trajectory planning algorithms [1] have been already proposed but these approaches incur high computation cost. Computation time is very crucial aspect in competitive autonomous racing scenario. Probabilistic inference-based approaches [2, 3] can offer fast solution to the planning problem by exploiting the factored representation of trajectory. The main objective of this research work is to formulate the minimum curvature trajectory planning as graphical model inference problem and analyze the performance of the proposed approach in a simulated scenario for a benchmark race track.

Tasks:

- Literature research on trajectory planning for autonomous racing vehicle
- Formulation of theoretical framework for minimum curvature trajectory planning via probabilistic inference
- Detailed analysis, discussion and comparison of proposed framework with state of the art approaches
- Development of software stack for probabilistic inference based trajectory planning of autonomous racing vehicles

Bibliography:

[1] Alexander Heilmeier, Alexander Wischnewski, Leonhard Hermansdorfer, Johannes Betz, Markus Lienkamp, and Boris Lohmann. Minimum curvature trajectory planning and control for an autonomous race car. *Vehicle System Dynamics*, 58(10):1497–1527, 2020.

[2] Mustafa Mukadam, Jing Dong, Xinyan Yan, Frank Dellaert, and Byron Boots. Continuous-time gaussian process motion planning via probabilistic inference. *The International Journal of Robotics Research*, 37(11):1319–1340, 2018.

[3] M. Toussaint and C. Goerick. Probabilistic inference for structured planning in robotics. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3068–3073, 2007.

Supervisor:	M. Sc. Salman Bari (LSR) | M. Sc. Andressa de Paula Suiti (ARRK)
Start:	20.09.2021
Intermediate Report:	17.12.2021
Delivery:	18.03.2022

(D. Wollherr)
Priv.-Doz.

Abstract

Recently, trajectory optimization for autonomous racing has attracted great interest in both academic and industrial research. There, the goal is to develop an algorithm that will compute a feasible trajectory around a given racetrack such that the lap time is minimized. To achieve this, a trajectory optimization approach is needed that pushes the car to the handling limits and offers fast solutions. Raceline optimization, also referred to as global planning, generates such a trajectory by taking different performance criteria into consideration. Optimization-based approaches for raceline planning have already been proposed, but these approaches are computationally expensive. On the other hand, there have been proposals for using probabilistic inference based motion planning in the field of robotics offering fast solutions for planning problems by exploiting the factored representation of the trajectory. The objective of this research work is to explore the probabilistic inference approach in raceline optimization. Initial results of the proposed approach show similar results that are achieved with significantly less computation time compared to other state-of-the-art approaches.

Contents

Chapter 1

Introduction

For autonomous racing, a software stack is needed that handles the following three parts:

1. Building a map of the environment based on walls and free space
2. Generating a global trajectory given a racetrack
3. Feeding the global trajectory into the vehicle controller

In [HWH+20], this workflow is proposed and the corresponding software stack is developed. However, in this work, we will solely focus on the generation of the global trajectory.
According to [BCMS08], the optimal trajectory, i.e. the resulting trajectory of a minimum time optimization, would be a compromise between the shortest path trajectory and the minimum curvature trajectory. The weighting between the two is dependent on the vehicle dynamics, which includes factors such as the acceleration limits of the vehicle and tire performance. This approach would yield the best lap time possible, but at the cost of having a high computational complexity.
In [HWH+20] the trajectory is optimized towards a minimum curvature trajectory, which achieves a slightly worse lap time compared to a minimum time optimization, but with a much faster computation time (see [HWH+20, Secion 3.4]). The lower computational complexity can be explained by taking into account that the minimum curvature optimization is independent of the vehicle dynamics. For every racetrack, there exists only one minimum curvature trajectory, which is the same for every vehicle. The similar results in terms of lap time can be explained by the high performance vehicles used in racing. These vehicles can take full advantage of the high cornering speed in curves that can be achieved with a minimum curvature trajectory.
In general, there are three steps when generating a raceline using a conventional optimization-based approach:

1. Derive the precise mathematical model describing what should be optimized

2. Derive the objective function that should be optimized
3. Use a numerical algorithm to optimize the objective function

This is also the approach used in [HWH+20], where a mathematical model for the minimum curvature raceline is derived, and then an error is defined and minimized. One of the drawbacks of this approach is the need of a precise mathematical model of the raceline property that should be optimized. This makes it necessary to derive a new mathematical model whenever a new property is added.
This work proposes a different approach for global planning in racing using probabilistic inference. This approach promises to be computationally fast, while still maintaining and, in some cases, improving the quality of the final trajectory.
The concept of using probabilistic inference for trajectory optimization was already proposed before by [TG07]. There, the approach was used for structured planning in robotics with a humanoid upper body as an example. It also proposed to use factor graphs to efficiently solve the complex probabilistic inference problem.
Later, [MDY+18] used this approach for motion planning of arm robots. Here, the mathematical foundation is set and a concrete framework is developed for solving trajectory optimization problems for arm robots. The framework was built on top of the GTSAM library, in particular using its factor graph generation and optimization algorithms. These algorithms are prominently used in the SLAM community for efficiently solving large scale inference problems.
The framework developed in [MDY+18] works great for arm robots, however, after some testing, it proved to be ineffective for raceline optimization. The complexity of trajectory optimization for arm robots is significantly higher than the complexity needed for raceline optimization, which, in turn, makes their framework unneccesarily complex for our purposes. Instead, we use the GTSAM library directly and implement our own custom factors for raceline optimization.
This work aims to build a framework for using the probabilistic inference approach for raceline optimization. For this, the necessary knowledge about raceline optimization and factor graphs is explained, the factors used in the optimization of the raceline are derived, implemented, and finally the results are compared to [HWH+20].

Chapter 2

Raceline Optimization in Autonomous Racing

In this chapter, the most relevant works concerning both autonomous racing and probabilistic inference based motion planning are introduced and the most relevant aspects explained.

2.1 Planning in Autonomous Racing

The optimized trajectory for a racetrack heavely depends on the geometry of the given racetrack and the performance of the vehicle. As a result, it is common to split the task of trajectory optimization into two steps: path optimization and velocity profile generation. Additionally, the path optimization depends on the geometry of the given racetrack, which suggests to use a geometrical optimization for generating an optimized path. This is introduced in [BCMS08], where an optimized path is first found using a balancing of the shortest path and the minimum curvature path, and then the velocity profile of that path is generated. In [KSG19], they used a similar approach, but switched the steps around. There, they first generated a velocity profile, and then generated a path by minimizing the path curvature using convex optimization.
In [HWH+20], a software stack for an autonomous racing vehicle was developed with the main focus being on the design and implementation of the planning and control part for autonomous racing. The key contributions of the paper are the theoretical formulation of the minimum curvature path optimization, the introduction of curvature constraints originating from a real car's steering design, the iterative invocation of the quadratic problem solver to reduce errors in corners, and several extensions to the planning algorithm to guarantee robustness in real world scenarios.
The software stack is divided into three separate modules, each taking the output of the previous module as their input. The first module is the *perception module* that generates the environment — namely the centerline, racetrack boundaries and objects — using the vehicle sensors such as LiDAR. This environment data is fed into

the *planning module* that uses it to generate a global trajectory. This trajectory is then used to generate local trajectory snippets that are fed into the *control module*. Here, the vehicle is tracked and controlled such that it follows these local trajectory snippets.
During the actual racing session, two computation units are active. The *planning unit* monitors the system mission including the states of the vehicle, generates the local trajectory snippets from the previously computed global trajectory, and adjusts the step size of the trajectory based on the current velocity of the vehicle. The *real time control unit* tracks the lateral path and the velocity of the vehicle and converts the requested values from the local trajectory into steering angles and force requests.
The software stack was tested on the racetrack data from the Tempelhof Airport Street Circuit used in the Formula E Berlin ePrix. For this race, no obstacles and no opponents were present, which makes it possible to compute the global trajectory offline and omit the computation of a local trajectory. The trajectory was computed using two steps:

1. Find the minimum curvature path for the racetrack

2. Generate a suitable velocity and acceleration profile based on the previously computed path and the vehicle's handling limits

The general objective for raceline optimization is to achieve the lowest possible lap time. The algorithm for the trajectory optimization should also be robust and reliable, while still maintaining a low computation time. The idea behind the path generation approach of [HWH+20] is to vary the path of the initial reference line such that the globally summed quadratic curvature is minimized, thus generating a minimum curvature path. The variation of the reference line is done by moving the reference line points along their normals. The raceline itself is defined using third order spline interpolation. This definition gives them the first and second order derivatives explicitely, which are needed for their minimum curvature computation. To ensure smoothness, consecutive splines must also have the same first and second order derivatives.
The mathematical derivation of the minimum curvature path generation can be found in [HWH+20, Section 3.4]. In short, the optimization problem is formulated as a quadratic problem. Here, the deviations from the reference line points along the normals are given as a coefficient vector α. Then, the curvature equation is reformulated using the first and second order derivatives from the spline interpolation and the resulting equation is reformulated using matrices to entail the entire path. The resulting equation can be minimized to obtain the minimum curvature path.
This implementation of the minimum curvature path generation is quite sensitive to a noisy reference line making it necessary to pre-process the reference line. The preprocessing has to be done once for every racetrack incurring a higher computation time.

Finally, the trajectory is obtained by computing the velocity profile from the minimum curvature path using a forward-backward solver. This global trajectory can now be used to generate the local trajectory snippets for the control module.
The overall results from the approach designed and implemented in [HWH+20] shows an improved lap time and more robustness compared to other raceline optimization approached. The results in [HWH+20] were mainly compared to the results of [BCMS08]. Despite the overall improved results achieved in [HWH+20], the total computation time of 18s is still high. Most of this computation time was used for spline computation and interpolation.

2.2 Probabilistic Inference in Robotics

The idea of using probabilistic inference for motion planning problems is first proposed in [TG07]. There, the approach is examined for the usage in the field of robotics and the general methodology of using inference for planning is laid out. The main goal of [TG07] is to explore a motion planning approach that is scalable in high-dimensional systems. This scalability could be achieved by exploiting the structural knowledge about the system, which is done by using a factored representation of the state of the robot and the environment itself, and thus creating a factored model of the motion planning problem. Inference on these kind of factored models is already extensively addressed in the machine learning community and their knowledge could be leveraged for motion planning problems.
In the machine learning community, structured probabilistic models are used for generating a model for a given set of observed data. Here for the motion planning problem in robotics, these models are used for generating optimal control signals. The paper demonstrates the approach by using a simulation of a humanoid upper body and solving motion planning problems under task and collision constraints.
The approach proposed in [TG07] is split into 5 steps:

1. Formulate the problem using *Dynamic Bayesian Networks* (DBNs) and add variables, or states, depending on couplings in the system (e.g. joint and endeffector states)

2. Add constraints like smoothness, feasibility or consistency by designing a factor and adding it in the factor graph of the problem

3. Add target constraints like a target position using factors again

4. Use an inference algorithm on the factor graph to solve the problem

5. Extract the control parameters from the results of the inference algorithm

In [TG07], the generated factor graph is then solved using a message passing approach.

The usage of factor graphs in large-scale inference problems in robotics is explained in-depth in [DK17]. It shows the flexibility of using factor graphs in SLAM problems, which also translates into a more modular and flexible code base. Of particular interest for our purposes is the explanation of how nonlinear optimization techniques can be used to solve a maximum a posteriori (MAP) inference problem that is present in a factor graph. This technique is already used in the smoothing and mapping (SAM) algorithms that are commonly use in the SLAM community.
In general, MAP inference uses known data to find values for the unknowns that maximally agree to this known data. This MAP problem on the factor graph that tries to maximize the product of all factor graph potentials can be converted into a minimization of the sum of nonlinear least-squares. The converted nonlinear least-squares problem can now be solved by solving a succession of linear approximations of it using algorithms like Gauss-Newton or Levenberg-Marquardt.
In [MDY+18], a formulation of motion planning for time-continous trajectories as probabilistic inference is introduced. The paper contains the theory for solving motion planning problems for robot arms using gaussian processes and probabilsitic inference.
Here, the trajectory is defined as a function that maps time to a robot state. The optimization process then tries to find the optimal trajectory for the environment and the given constraints. The trajectory is represented with a small number of states using sparse Gaussian Process models. The lower number of states decreases the computation time of the optimization. If a higher resolution of the trajectory is needed, it can be efficiently interpolated using Gaussian Interpolation. This is the case for constraints like collision avoidance or before sending the resulting trajectory to the robot control module.
The inference problem is formulated as finding a trajectory given some desired events. In [MDY+18] the main focus is on the collision-free event. Now the posterior density (trajectory given events) can be maximized in order to get the optimal trajectory. This posterior density can be re-written using the prior on the trajectory, which encourages smoothness, and the probability of a desired event given the trajectory, which is a also called likelihood. This likelihood specifies that a trajectory given the events is more likely to be successful. This MAP inference problem can now be transformed into an optimization problem by taking the negative log of the posterior density. Now, the maximization of probabilities becomes a minimization of cost. This transformation makes it possible to use conventional nonlinear least squares optimization algorithms for this motion planning approach.
Additionally, the original GPMP2 algorithm is extended by an incremental version (iGPMP2) on the basis of the incremental Smoothing and Mapping (iSAM) algorithm used in the SLAM community. This incremental algorithm makes replanning problems very efficient, in case of dynamic changes in the environment. Finally, the results of GPMP2 and iGPMP2 are compared with other state of the art motion planning approaches including TrajOpt [SDH+14] and CHOMP [ZRD+13]. The results show a significantly lower computation time and higher success rate compared

to the other motion planning approaches.
The implementation of GPMP2 is solely build on top of the GTSAM library [BOR17] and uses the factor graph and solver implementations. For our purpose of raceline optimization, the GPMP2 framework is too complex and does not deliver the results we need. As a result, we will be building our own framework directly on top of GTSAM for raceline optimization.

Chapter 3

Planning as Probabilistic Inference

In this work, we will be exploring the probabilistic inference approach for raceline optimization. In essence, we are using tools usually used in probabilistic inference, in this case for SLAM problems, and are applying them to optimization problems, in our case raceline optimization. This chapter will be covering the theoretical knowledge related to probabilistic inference via a MAP estimator and how to convert this into a factor graph representation. Also, we will explain how this probabilistic approach can be interpreted from an optimization point of view. The theory concerning probabilistic inference and the use of factor graphs is mainly adopted from the excellent explanations from [DK17, Chapter 1].

3.1 Probabilistic Inference

The main objective – be it in a SLAM, robotics, or raceline optimization problem – is to find certain positions given some prior knowledge. In SLAM this encompasses finding poses and landmarks given some observed measurements; in robotics and for our raceline optimization this includes finding path points given some desired path properties. The poses, or path points in this case, are represented by the states

$$\mathbf{s} = [\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_N]^T, \text{ with } \mathbf{s}_i \in \mathbb{R}^d, \ \forall i = 1, ..., N, \tag{3.1}$$

where N is the number of states and d the dimensionality of the positions, e.g. 2 for a 2D position on a plane. The given information, be it observed measurements or desired trajectory properties, will be denoted by $\mathbf{z}$. Now, the inference of the unknown states $\mathbf{s}$ given some information $\mathbf{z}$ is called probabilistic inference and can also be expressed as the conditional probability density

$$p(\mathbf{s}|\mathbf{z}). \tag{3.2}$$

Here, the probability densities could be any kind of multivariate distribution, however, more commonly a multivariate Gaussian distribution is chosen

$$\mathcal{N}(s;\mu,\mathbf{\Sigma}) = \frac{1}{\sqrt{|2\pi\mathbf{\Sigma}|}}\, exp\,\{-\frac{1}{2}||s-\mu||^2_{\mathbf{\Sigma}}\}, \tag{3.3}$$

where $\mu \in \mathbb{R}^{\times}$ is the mean, $\mathbf{\Sigma} \in \mathbb{R}^{dxd}$ is the covariance matrix, and

$$||s-\mu||^2_{\mathbf{\Sigma}} = (s-\mu)^T\mathbf{\Sigma}(s-\mu) \tag{3.4}$$

is the squared Mahalanobis distance. For our purposes, we assume all priors and likelihoods to be Gaussian.

3.1.1 Maximum a Posteriori Estimation

In order to get an estimate of the unknown states $\mathbf{s}$ given the information $\mathbf{z}$, we can use a MAP estimator

$$\overset{*}{\mathbf{s}} = \arg\max_{\mathbf{s}} p(\mathbf{s}|\mathbf{z}). \tag{3.5}$$

Here, the optimal states $\overset{*}{\mathbf{s}}$ are searched that will maximize the posterior probability $p(\mathbf{s}|\mathbf{z})$. This equation can be re-written using Bayes' law as

$$\overset{*}{\mathbf{s}} = \arg\max_{\mathbf{s}} \frac{p(\mathbf{z}|\mathbf{s})p(\mathbf{s})}{p(\mathbf{z})}. \tag{3.6}$$

The normalization factor $p(\mathbf{z})$ can be eliminated here, because all the $\mathbf{z}$ values are given and thus it does not have any effect on the maximization. The probability $p(\mathbf{s})$ could be any prior knowlege we have about the states themselves. This includes known start or end positions, kinematic models that restrict the movement between two states to a certain area, etc. Lastly, the posterior probability $p(\mathbf{z}|\mathbf{s})$ can be re-written as

$$l(\mathbf{s};\mathbf{z}) \propto p(\mathbf{z}|\mathbf{s}) \tag{3.7}$$

with $l(\mathbf{s};\mathbf{z})$ being the likelihood of the states $\mathbf{s}$ given the information $\mathbf{z}$. Now, the MAP estimator from (3.6) can be expressed as

$$\overset{*}{\mathbf{s}} = \arg\max_{\mathbf{s}} p(\mathbf{s})\; l(\mathbf{s};\mathbf{z}). \tag{3.8}$$

3.1.2 Factor Graph for Probabilistic Inference

The MAP estimator from (3.8) could also equivalently be described using a factor graph. A factor graph is formally defined as a bipartite graph meaning that its nodes can be divided into two distinct groups. It consists of a set of *variable*, or *state*, nodes $\mathbf{s}_I \in \mathcal{S}$ and a set of *factor* nodes $f_i \in \mathcal{F}$ that are connected by *edges* $e_{ij} \in \mathcal{E}$. All together they form the factor graph $\mathcal{G} = (\mathcal{S}, \mathcal{F}, \mathcal{E})$. An example of

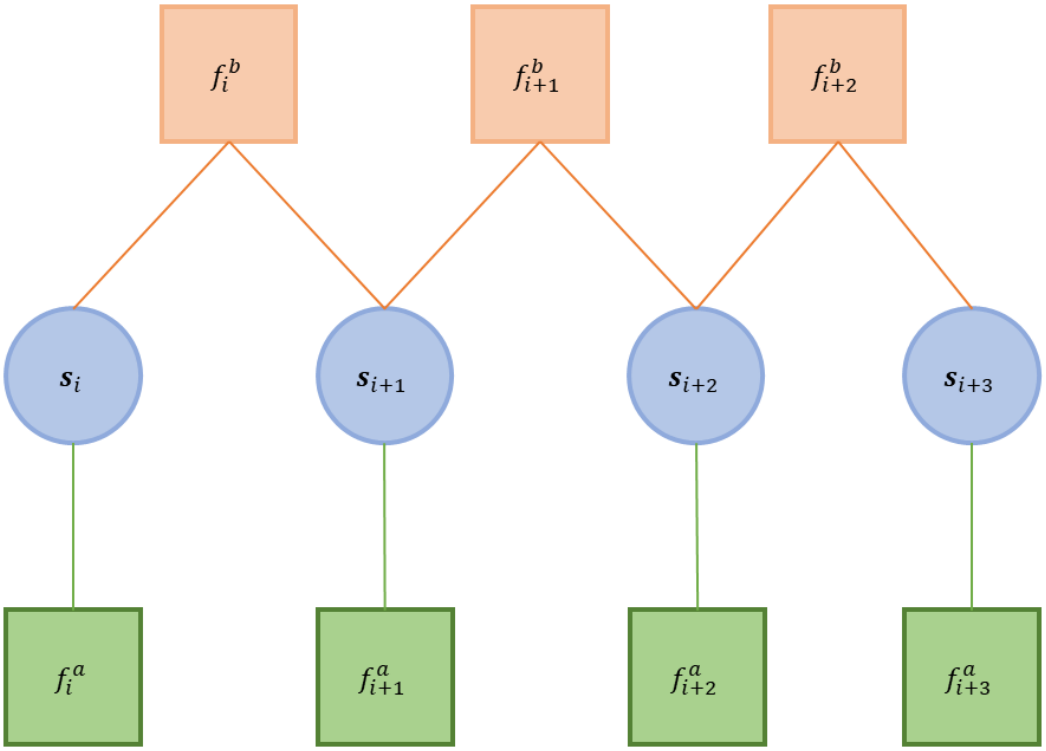

Figure 3.1: An example factor graph with states (variable nodes) represented with circles and factor nodes represented with squares.

such a factor graph can be seen in Figure 3.1 Each factor node can be connected to multiple state nodes, as well as, each state node can be connected to multiple factor nodes. Therefore, we define $\mathcal{S}_i$ as the set of states that the factor node f_i is a function of and we get

$$f_i(\mathcal{S}_i), \text{ with } \mathcal{S}_i \subseteq \mathcal{S}. \tag{3.9}$$

Now, the conditional PDF $p(\mathbf{s}|\mathbf{z})$ can be represented with a factor graph as

$$p(\mathbf{s}|\mathbf{z}) \propto \prod_{m=1}^{M} f_m(\mathcal{S}_m). \tag{3.10}$$

This representation of the probabilistic inference approach has some advantages. It lets us easily evaluate the factor graph for any set of states by computing each of the factor nodes separately and then multiplying them together. The easy evaluation opens the door for using different optimization methods in order to solve the factor graph and find an optimal solution. With the conditional PDF from (3.10) the MAP estimation can be written as

$$\overset{*}{\mathbf{s}} = \arg\max_{\mathbf{s}} \prod_{m=1}^{M} f_m(\mathcal{S}_m). \tag{3.11}$$

3.2 From Inference to Optimization

The MAP estimation on a factor graph from (3.11) can easily be converted into an optimization problem by taking the negative log of it

$$\overset{*}{\mathbf{s}} = -log\ \{\arg\max_{\mathbf{s}} \prod_{m=1}^{M} f_m(\mathcal{S}_m)\} \tag{3.12}$$

$$= \arg\min_{\mathbf{s}} \sum_{m=1}^{M} \{-log\ f_m(\mathcal{S}_m)\} \tag{3.13}$$

With our previous assumption that all priors and likelihoods are following a Gaussian distribution, the factor function must also follow a Gaussian distribution. Thus we define our factor function to be

$$f_i(\mathcal{S}_i) = exp\ \{-\frac{1}{2}||\mathbf{e}_i(\mathcal{S}_i)||^2_{\mathbf{\Sigma}}\} \tag{3.14}$$

with $\mathbf{e}_i(\mathcal{S}_i)$ being the error function of the $i-th$ factor. Then finally the optimization from (3.12) becomes

$$\overset{*}{\mathbf{s}} = \arg\min_{\mathbf{s}} \sum_{m=1}^{M} \{\frac{1}{2}||\mathbf{e}_i(\mathcal{S}_i)||^2_{\mathbf{\Sigma}}\}, \tag{3.15}$$

which is also the objective function of our optimization.
Viewing this problem from an optimization perspective, we can see that the factor functions consist of an error compared to the ideal scenario and a covariance matrix $\mathbf{\Sigma}$, which can be viewed as an inverse cost weight. The higher the cost $\mathbf{\Sigma}$ is set, the lower is the weight of the error of that given factor. This behavior can be seen in (3.4), where the error from the mean value is weighted against the inverse of $\mathbf{\Sigma}$. (3.4) also shows that we are using a squared error, where the error function $\mathbf{e}(\mathcal{S})$ is, in general, nonlinear. As a result, we can use nonlinear least squares algorithms like Gauss-Newton or Levenberg-Marquardt in order to optimize our factor graph.
Furthermore, instead of using observed measurements for our raceline optimization, the given information will be desired properties we want the path to have. In this work, we will be looking at designing a prior factor that keeps the path in the bounds of the racetrack and another factor promoting a minimum curvature path. In general, the design of a factor consists of two steps: defining an error function $\mathbf{e}(\mathcal{S})$ and finding a suitable covariance or cost matrix $\mathbf{\Sigma}$. This process and the actual

design of the prior and the minimum curvature factor will be discussed in Chapter 4.

Chapter 4

Raceline Optimization using Probabilistic Inference

This chapter describes what global planning is, how the probabilistic inference based motion planning approach can be used for raceline optimization and what considerations have to be made, when using this approach. Afterwards, the factors used for the raceline optimization are developed.

4.1 Global Planning

Trajectory optimization can be divided into two parts: global planning and local planning. Generally, in global planning the optimal path is searched on a macrolevel. This means that based on a given map a trajectory through this map to the goal is searched. Here, often the robot or vehicle dynamics are not taken into account or are simplified as the trajectory is not applied directly anyway. The local planner on the other hand will apply this global trajectory on a much smaller scale. The scale at which the local planner works is highly dependent on the specific application. The local planner will then also take into account the dynamics of the vehicle and dynamic obstacles.
In the context of autonomous racing, global planning is also called raceline optimization. Here, the global planner plans the general trajectory for the entire racetrack and the local planner will then apply this trajectory for the next few meters, potentially maneuvering around other vehicles.

4.2 Raceline Optimization

There are three things that have to be considered for optimizing the raceline of a racecar:

1. The racecar cannot go out of the racetrack

2. The shorter the lap time, the better

3. The shorter the computation time, the better

The first consideration can be solved by using a bounding factor that will put a cost on going out of the bounds of the racetrack. This can be achieved by using priors that put a cost on the trajectory depending on the deviation from a pre-defined area.
The second consideration can be solved by minimizing the curvature of the trajectory, which would result in a maximized cornering speed and therefore a low lap time. This approach is also used in [HWH+20] to achieve a low lap time with significantly less computational cost in comparison with minimum time optimization, which would minimize the lap time.
The third consideration can be optimized in a number of ways starting with the approach that is used for raceline optimization, and then striking a balance between quality of the resulting trajectory and the computation time. Here, one example would be the resolution of the racetrack and the trajectory: the lower the resolution is set, the lower the computational time will be. However, one has to consider that a lower resolution will also lead to a loss of detail of the map and the trajectory, which in turn might lead to smaller obstacles being ignored or very curvy racetrack parts being simplified to straighter parts.
An example of how raceline optimization looks like can be seen in Figure 4.1.
In our approach, the first two considerations are done by adding corresponding factors to the factor graph of the probabilistic inference based approach and solving it. The racetrack bounding factor is developed in Section 4.3.1 and the minimum steering factor in Section 4.3.2. Lastly, the computation time is significantly reduced by using the proposed approach of probabilistic inference for motion planning.

4.3 Factor Development

In general, a factor in a factor graph can be connected to any arbitrary collection of states. The factors could also just be attached to individual parts of the raceline. This could be useful if only some parts of the trajectory needs optimization or if different optimizations are necessary for different parts of the trajectory (e.g. straight parts or curves). Each factor must define an error function $\mathbf{e}$ given a set of states $\mathcal{S}$ that are connected to it. These error values are summed up over the entire factor graph and then minimized in order to find the optimal trajectory. In our implementation, the error is always split up into its x and y components corresponding to the error in x-direction and the error in y-direction. The error functions of the factors are designed in such a way that the error could be split into its individual components. Also, a cost matrix $\mathbf{\Sigma}$ must be defined for each factor that weights the individual factor errors against each other. It should be noted here that the values of the cost matrix $\mathbf{\Sigma}$ are inverted, and thus a lower value means a higher

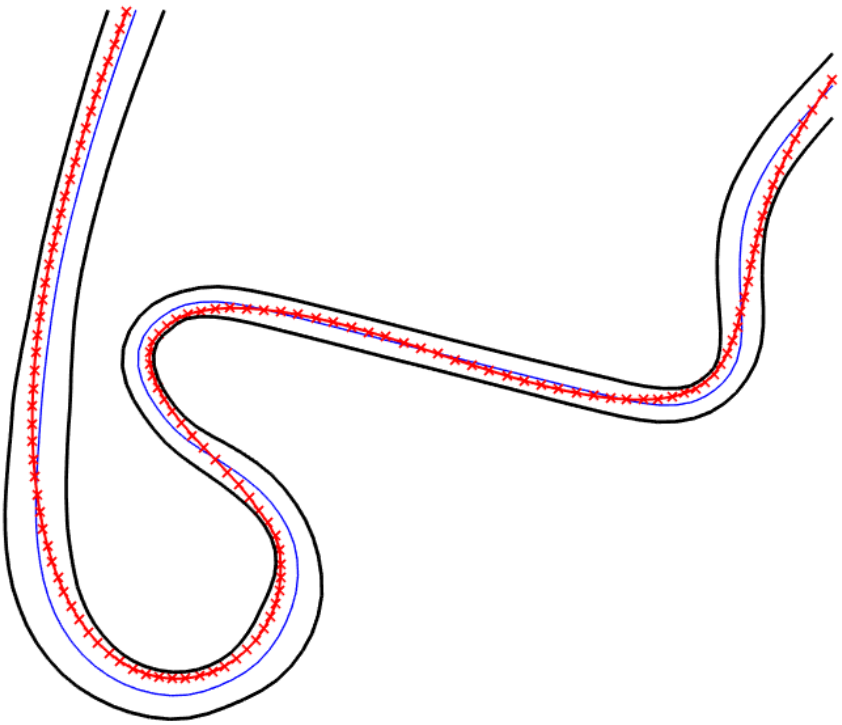

Figure 4.1: An example for raceline optimization. The blue line is the reference or centerline and the red line is the optimized raceline.

weight. The cost matrix will not be part of the design of the factor, as it is highly dependent on the actual scenario, i.e. racetrack, that it is used on. As a result, the cost matrix $\mathbf{\Sigma}$ will be set in the evaluation in Chapter 5. Lastly, the factors must compute a Jacobian matrix for each attached state in the set $\mathcal{S}$ containing the gradient of each component of the error function derived by the components of the state. This Jacobian is used later on in the GTSAM factor graph optimization to find a possible solution. If both the state $\mathbf{s}$ and the error $\mathbf{e}$ are in $\mathbb{R}^2$ then the Jacobian can be computed as

$$\mathbf{H} = \begin{bmatrix} \frac{\partial e_x}{\partial s_x} & \frac{\partial e_x}{\partial s_y} \\ \frac{\partial e_y}{\partial s_x} & \frac{\partial e_y}{\partial s_y} \end{bmatrix}. \tag{4.1}$$

4.3.1 Racetrack Bounding Factor

Initially, the raceline is bound by creating prior factors from the centerline points. This implementation would incur a high cost, if the state would go way out of the bounds of the racetrack. However, the issue with this bounding method is that it penalizes any deviation from the centerline. In addition to restricting the raceline from going out of bounds, it makes it also stick to the centerline too much. In racing, the vehicles are usually not driving along the centerline and are, more often than not, along either side of the track near the track boundaries.
In order to allow the optimization to search for a valid raceline along the entire track width, a custom racetrack bounding factor was implemented. This factor is attached to each state and bounds this state to the line between the left and right track boundary, rather than bounding it to a single point. As a result, the entire width of the racetrack can be searched for an optimal raceline without incurring a higher cost.

Racetrack Bounding Factor Development

The error that is evaluated in this factor is equal to the distance of the state to the nearest point on the line between the left and right track boundary. This distance is computed by first projecting the state onto the infinite line that goes through the left and right track boundary points and then clamping this value to the left or the right track boundary, if it lies outside of the racetrack boundaries. A visual representation of this procedure without the clamping can be seen in Figure 4.2 and with clamping in Figure 4.3.
For this, we first compute the heading vector pointing from the left to the right boundary, its length using the 2-norm, and then we normalize the heading vector.

$$\begin{aligned} \mathbf{v}_h &= \mathbf{b}_r - \mathbf{b}_l \\ d_h &= ||\mathbf{v}_h||_2 \\ \mathbf{v}_{h,n} &= \frac{\mathbf{v}_h}{d_h}. \end{aligned} \tag{4.2}$$

It should be noted here that the choice of defining the heading vector from the left to the right track boundary is arbitrary. The same computations could be done by going from the right to the left boundary. Now, to get any point on the infinite line that goes through the left and right track boundary, we would have to multiply the normalized heading vector $\mathbf{v}_{h,n}$ with the distance of that point and the left track boundary

$$\mathbf{s}_t = \mathbf{b}_l + d_t \mathbf{v}_{h,n}. \tag{4.3}$$

So in order to get the nearest point on the line from a given state, we need to know the distance from this point to the left track boundary. This distance can

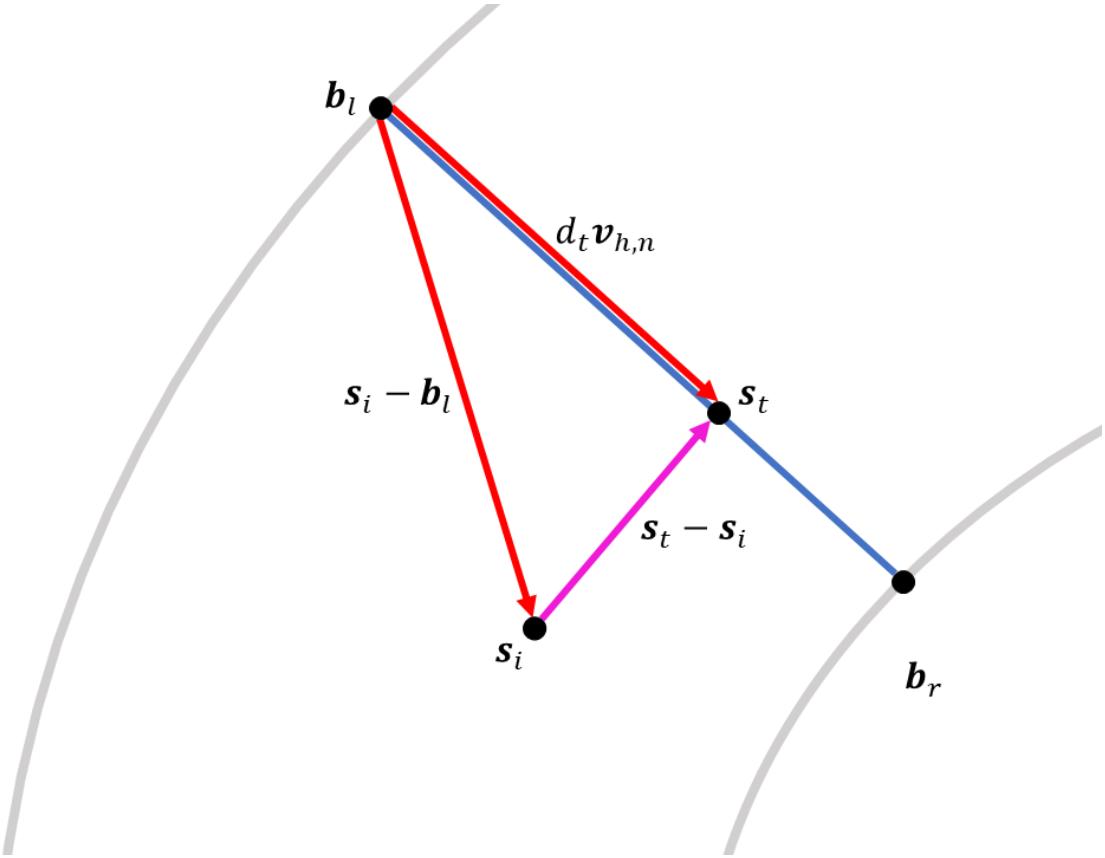

Figure 4.2: The projected state lies on the line between the left and right track boundary, thus no clamping is done.

be computed by using the scalar product of the normalized heading vector and the vector pointing from the left track boundary to the state

$$d_t = (\mathbf{s}_i - \mathbf{b}_l) \cdot \mathbf{v}_{h,n}. \tag{4.4}$$

If we multiply the distance d_t with the normalized heading vector $\mathbf{v}_{h,n}$, we get the projection of the given state onto the infinite line that goes through the track boundaries. However, we want the projection onto the finite line between the left and right track boundary. In order to do this, we need to clamp the value of d_t to a minimum and maximum value. Using (4.3), we can see that the minimum value is 0, which would result in $\mathbf{s}_t = \mathbf{b}_l$ and the maximum value is d_h as that would result in $\mathbf{s}_t = \mathbf{b}_r$. Now the distance d_t can be clamped using a clamping function

$$d_{t,c} = clamp(d_t, 0, d_h) = max(0, min(d_t, d_h)). \tag{4.5}$$

Using the clamped distance $d_{t,c}$ and (4.3), the projection onto the finite line between the left and right track boundary can be computed as

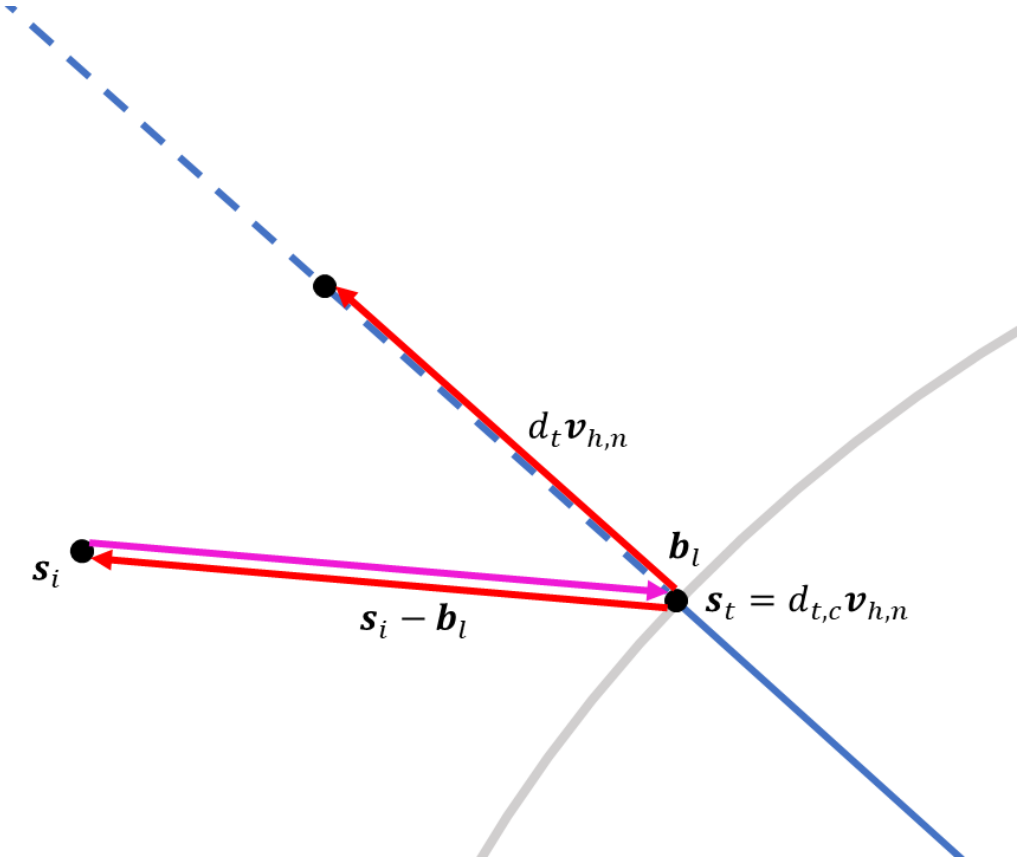

Figure 4.3: The projected state is out of bounds, thus the projection is clamped to the left track boundary here.

$$\mathbf{s}_t = \mathbf{b}_l + d_{t,c}\mathbf{v}_{h,n}. \tag{4.6}$$

Finally, the error of the racetrack bounding factor is computed as

$$\mathbf{e}_{bound} = \mathbf{s}_t - \mathbf{s}_i. \tag{4.7}$$

The factor function for the racetrack bounding factor is then computed as

$$f^{bound} = exp\ \{-\frac{1}{2}||\mathbf{s}_t - \mathbf{s}_i||^2_{\mathbf{\Sigma}_{bound}}\}. \tag{4.8}$$

The Jacobian for this factor has to be split into two different cases. The first case is when the projection is clamped to the left or right boundary. Here, the target point $\mathbf{s}_t$ is either set to $\mathbf{b}_l$ or $\mathbf{b}_r$, which can be written as some constant c and (4.7) becomes

$$\mathbf{e}_{bound} = \mathbf{c} - \mathbf{s}_i = \begin{bmatrix} c_x - s_{i,x} \\ c_y - s_{i,y} \end{bmatrix}, \tag{4.9}$$

which can be derived with respect to the components of $\mathbf{s}_i$ to get the Jacobian

$$\mathbf{H}_i = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{4.10}$$

The second case is when the projected point lies on the line between the left and right track boundary. Then the clamping from (4.5) results in no change, i.e. $d_{t,c} = d_t$, and using (4.3), (4.4), and (4.7), we get

$$\begin{aligned} \mathbf{e}_{bound} &= \mathbf{b}_l + (\mathbf{s}_i - \mathbf{b}_l) \cdot \mathbf{v}_{h,n}\mathbf{v}_{h,n} - \mathbf{s}_i \\ &= \mathbf{b}_l + (s_{i,x}v_{h,n,x} - b_{l,x}\ v_{h,n,x} + s_{i,y}\ v_{h,n,y} - b_{l,y}\ v_{h,n,y})\mathbf{v}_{h,n} - \mathbf{s}_i. \end{aligned} \tag{4.11}$$

and finally the Jacobian for this can be computed as

$$\mathbf{H}_i = \begin{bmatrix} v_{h,n,x}^2 - 1 & v_{h,n,x}\ v_{h,n,y} \\ v_{h,n,x}\ v_{h,n,y} & v_{h,n,y}^2 - 1 \end{bmatrix}. \tag{4.12}$$

This Jacobian also shows that the racetrack bounding factor is nonlinear, thus making it necessary to use nonlinear optimization in order to find a solution for the factor graph.

Individual Cost Boundaries

One issue that arises, when using the actual racetrack boundary values for $\mathbf{b}_l$ and $\mathbf{b}_r$, is that the resulting path might go out of bounds. Small deviations from the given prior line are normal, and when these deviations happen near the outer ends of the prior line, the path might go out of bounds. To counteract this issue, cost boundaries are used instead of the actual racetrack boundaries. The cost boundaries lie on the line between the actual racetrack boundaries, but are moved towards each other creating a kind of soft safety margin to the actual boundaries. Using a cost boundary margin d_{cost} describing the distance in meters to the actual boundary, and the unit heading vector $\mathbf{v}_{h,n}$ pointing from the left to the right track boundary, the cost boundaries $\mathbf{b}_{cost,l}$ and $\mathbf{b}_{cost,r}$ can be computed as

$$\begin{aligned} \mathbf{b}_{cost,l} &= \mathbf{b}_l + \mathbf{v}_{h,n}d_{cost} \\ \mathbf{b}_{cost,r} &= \mathbf{b}_r - \mathbf{v}_{h,n}d_{cost}. \end{aligned} \tag{4.13}$$

Now, an appropriate value for the cost parameter d_{cost} has to be defined. The simplest way to do this, is to use a fixed cost parameter for the entire racetrack. This works well, when the curvature through the entire racetrack is approximately equal, e.g. on a circular racetrack, or a relatively straight racetrack. However, if the curvature differs significantly between different racetrack segments, a fixed cost boundary will also lead to suboptimal, or even unfeasible, results. This is a result of the interplay between the racetrack bounding factor and the minimum steering factor (See Section 4.3.2 for more in-depth explanation of how the minimum steering factor works).

In short, the minimum steering factor tries to straighten out the curvy parts of the path, which results in a very high cost in sharp curves. Due to the high cost, the path is pushed outside the racetrack in order to get it down. Here, the cost of the racetrack bounding factor is not high enough compared to the minimum steering cost.
This issue can be solved by simply increasing the cost parameter d_{cost}. At some point, the cost will be high enough that even in sharp curves the path stays inbound. However, this would also lead to a narrower search area for feasible paths on straight track segments, instead of using the entire width of the racetrack here.
In order to solve the issue of getting out of bounds in sharp curves, while still keeping the entire track width as a valid search area in straighter track segments, individual cost boundaries are introduced. Instead of using one fixed cost parameter d_{cost} for the entire racetrack, individual cost parameters are computed for each racetrack bounding factor depending on the angle at that specific part of the racetrack. The higher the angle is, the higher the cost parameter for that factor will be set.
For the computation of the individual cost boundaries, we first need to define a fixed minimum distance of the cost boundary to the actual track boundaries in meters. This parameter is denoted by $d_{cost,min}$. The maximum value is always set to half the track width at that specific state corresponding to the distance from the centerline to the track boundary. It is computed as

$$\mathbf{d}_{cost,max} = \frac{1}{2}(||\mathbf{b}_r - \mathbf{b}_l||_2). \tag{4.14}$$

Now, we need to compute the angle at each state. For this we use three adjacent states $\mathbf{s}_i$, $\mathbf{s}_{i+1}$ and $\mathbf{s}_{i+2}$, then we compute the two vectors $\mathbf{v}_a$ and $\mathbf{v}_b$ that are formed by the three states, and finally we compute the angle ϕ_i between the two vectors:

$$\begin{aligned} \mathbf{v}_a &= \mathbf{s}_{i+1} - \mathbf{s}_i \\ \mathbf{v}_b &= \mathbf{s}_{i+2} - \mathbf{s}_{i+1} \\ \phi_i &= arc\,cos\frac{\langle \mathbf{v}_a, \mathbf{v}_b \rangle}{\sqrt{||\mathbf{v}_a||_2||\mathbf{v}_a||_2}}. \end{aligned} \tag{4.15}$$

Then, the angles are rescaled as

$$\boldsymbol{\phi}_{rescaled} = rescale(\boldsymbol{\phi}, 0, 1) = \frac{\boldsymbol{\phi} - min(\boldsymbol{\phi})}{max(\boldsymbol{\phi} - min(\boldsymbol{\phi}))}, \quad \boldsymbol{\phi}_{rescaled,i} \in [0:1] \tag{4.16}$$

with $min(x)$ and $max(x)$ being the function that evaluates to the minimum and maximum value of the vector respectively.
Next, the cost parameter vector $\mathbf{d}_{cost}$ is computed using the rescaled angles vector $\boldsymbol{\phi}_{rescaled}$ such that they lie between the minimum value $d_{cost,min}$ and the corresponding maximum value from $\mathbf{d}_{cost,max}$ defined in (4.14):

$$\mathbf{d}_{cost} = max(\boldsymbol{\phi}_{rescaled}\, \mathbf{d}^{T}_{cost,max},\ d_{cost,min}),\ d_{cost,i} \in [d_{cost,min} : d_{cost,max,i}]. \tag{4.17}$$

We decided to clamp low values to $d_{cost,min}$ instead of scaling it to the interval $[d_{cost,min} : d_{cost,max,i}]$ in the first place, because it yielded better results.
Finally, the cost boundaries can be computed using the (4.13) and (4.17) as

$$\begin{aligned} \mathbf{b}_{cost,l,i} &= \mathbf{b}_l + \mathbf{v}_{h,n}\mathbf{d}_{cost} \\ \mathbf{b}_{cost,r,i} &= \mathbf{b}_r - \mathbf{v}_{h,n}\mathbf{d}_{cost}. \end{aligned} \tag{4.18}$$

The computation of the individual cost boundaries is done before the factors are created and are part of the pre-processing step of our approach. An illustration of how the cost boundary looks like in a curvy part of the racetrack can be seen in Figure 4.4.

4.3.2 Minimum Steering Factor

As stated in Section 4.2, a minimum curvature path is used here in order to get a near-optimal lap time with a reasonably low computation time. In the following, a methodology for minimizing the curvature is introduced and then the minimum steering factor is developed.

Minimum Curvature

Given the curvature κ for the entire racetrack, our objective is to find states that minimize the total summed curvatures such that

$$\overset{*}{\mathbf{s}}_{mincurv} = \arg\min_{\mathbf{s}} \sum_{i=1}^{N} \kappa(\mathcal{S}_i). \tag{4.19}$$

The general definition of the curvature κ for a circle is given as the reciprocal of the radius R of the circle:

$$\kappa = \frac{1}{R}. \tag{4.20}$$

In order to use this equation for our purposes, we will transform it using the arc length d_{arc} of a circle, which is defined as

$$\begin{aligned} d_{arc} &= \frac{\phi_{arc}}{2\pi} U \\ &= \frac{\phi_{arc}}{2\pi} 2\pi R \\ &= \frac{\phi_{arc}}{R} \end{aligned} \tag{4.21}$$

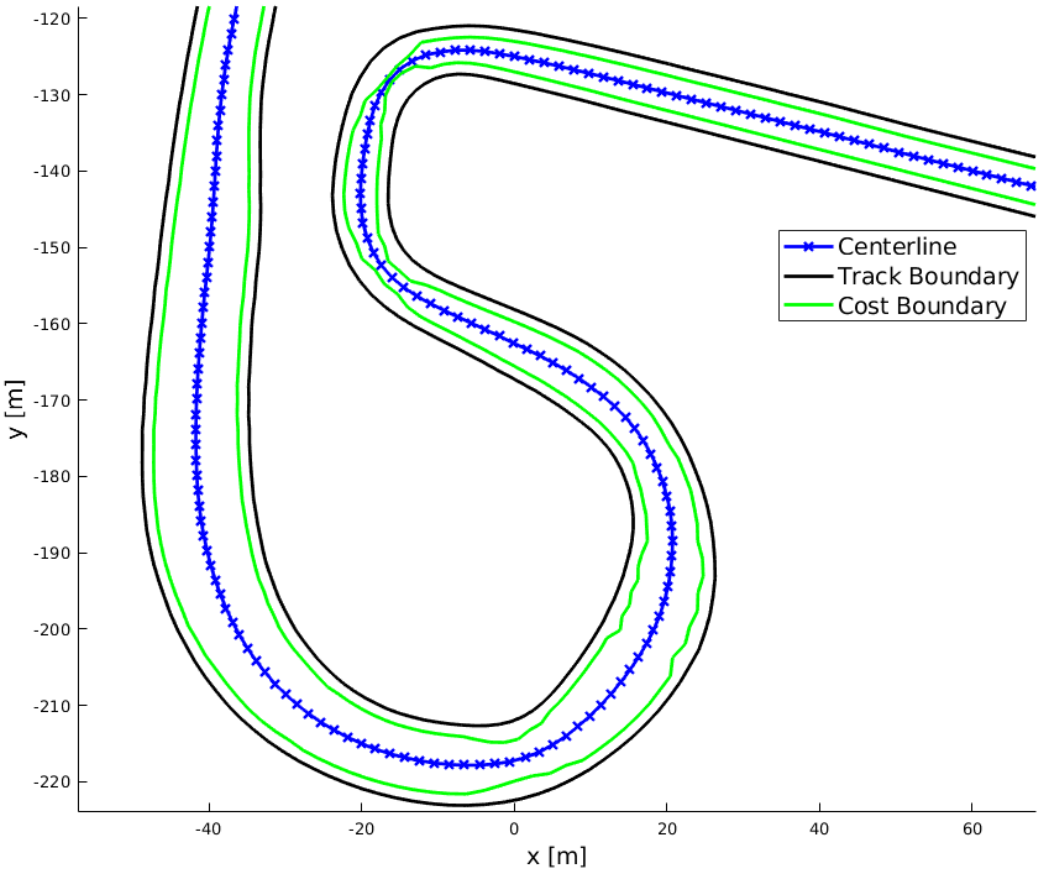

Figure 4.4: Individual cost boundaries that are used for the racetrack bounding factor instead of the actual track boundaries.

with ϕ_{arc} being the angle of the arc, R being the radius, and U being the circumference of the entire circle. This equation is now reformulated to express the radius R as

$$R = \frac{d_{arc}}{\phi_{arc}}. \tag{4.22}$$

Plugging this equation into the general curvature definition from (4.20), we get

$$\kappa_{arc} = \frac{\phi_{arc}}{d_{arc}} \tag{4.23}$$

describing the curvature of an arc. It should be noted that this equation only holds if a moving origin coordinate system is considered. Now, if we regard the path of the raceline as a combination of arc segments, we can see that in order to minimize

the cuvature of the raceline, we need to reduce the angles of the arc segments (i.e. amount of steering), increase the arc lengths (i.e. drive a longer distance), or do a combination of both.
As the goal of raceline optimization is a lower laptime, it would be counterintuitive to increase the travelled distance. Instead, for our approach we are going to focus on reducing the steering angle needed to get through the racetrack by reducing the angles formed by consecutive states. As a result, the objective function from (4.19) is reformulated as

$$\overset{*}{\mathbf{s}}_{mincurv} = \arg\min_{\mathbf{s}} \sum_{i=1}^{N} \phi(\mathcal{S}_i). \tag{4.24}$$

Minimum Steering Factor Development

The minimum steering factor is always attached to three adjacent states $\mathcal{S}_i = \{\mathbf{s}_i, \mathbf{s}_{i+1}, \mathbf{s}_{i+2}\}$. From these three states, two vectors are computed that go from the middle state $\mathbf{s}_{i+1}$ to each of the outer states $\mathbf{s}_i$ and $\mathbf{s}_{i+2}$

$$\begin{aligned} \mathbf{v}_a &= \mathbf{s}_i - \mathbf{s}_{i+1} \\ \mathbf{v}_b &= \mathbf{s}_{i+2} - \mathbf{s}_{i+1}. \end{aligned} \tag{4.25}$$

This arrangement can be seen in Figure 4.5. In order to minimize the steering angle of the vehicle, the two vectors should ideally be parallel to each other. This results in a target angle of $\phi_t = 180°$ between the two vectors, which is done here using basic vector geometry.
Initially, the angle was directly optimized by first converting the cartesian vectors $\mathbf{v}_a$ and $\mathbf{v}_b$ into their polar coordinate equivalent, now consisting of a radii r and an angle ϕ. Then, the difference between the two angles ϕ_d could be optimized, ideally being 180° and thus the two vectors being parallel to each other. However, after implementing this approach in GTSAM, it did not work. It seems like the issue was that the error is a scalar, while the states are 2D and consist of an x and y component. As a result, we tried to split the difference in angle between the two vectors into their x and y components. This, however, proved to be quite difficult to do mathematically. In addition to that, the computation of this error was very expensive, even without splitting the angle difference up (using square roots, sines, cosines, etc.). As a result, this approach was abandoned and a computationally more efficient solution is used.
In order to get a 180° angle between the two vectors formed by the three states, these have to lie on a straight line. To achieve this, one could move one of the outer states, for example $\mathbf{s}_i$, such that its vector, here $\mathbf{v}_a$, is antiparallel to the other vector. The basic concept can be seen in Figure 4.6.
Here, the displacement vector $\mathbf{v}_d$ moves the outer state, here $\mathbf{s}_i$, to its target position $\mathbf{s}_{i,t}$ to form a straight line with the other two states. Incidently, the same

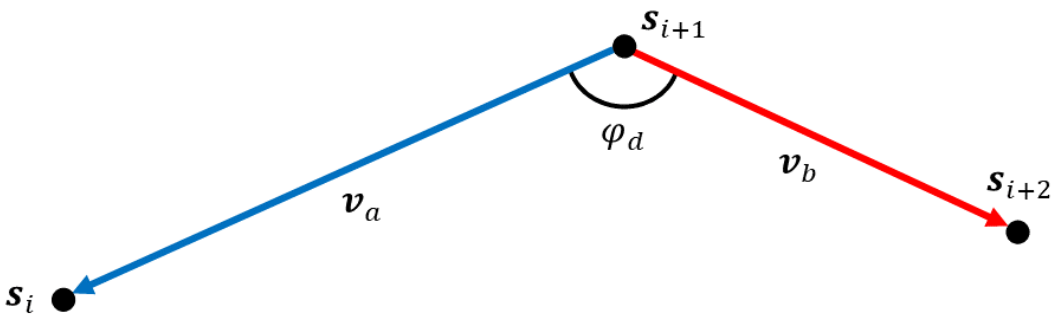

Figure 4.5: The initial setup for the minimum steering angle factor.

displacement vector can be used to move the other outer state, here $\mathbf{s}_{i+2}$, to its target position, if the other two states are fixed. The displacement vector $\mathbf{v}_d$ is equal to the negative sum of the two initial vectors $\mathbf{v}_a$ and $\mathbf{v}_b$

$$\mathbf{v}_d = -(\mathbf{v}_a + \mathbf{v}_b). \tag{4.26}$$

Finally, the error for the minimum steering angle factor is set equal to the displacement vector $\mathbf{v}_d$

$$\begin{aligned} \mathbf{e}_{steer} &= -(\mathbf{v}_a + \mathbf{v}_b) \\ &= 2 * \mathbf{s}_{i+1} - \mathbf{s}_i - \mathbf{s}_{i+2} \\ &= - \begin{bmatrix} 2 * s_{i+1,x} - s_{i,x} - s_{i+2,x} \\ 2 * s_{i+1,y} - s_{i,y} - s_{i+2,y} \end{bmatrix} \end{aligned} \tag{4.27}$$

and then the factor function computes as

$$f^{steer} = exp\ \{-\frac{1}{2}||\mathbf{v}_d||^2_{\mathbf{\Sigma}_{steer}}\}. \tag{4.28}$$

Lastly, the Jacobians for each state are computed as

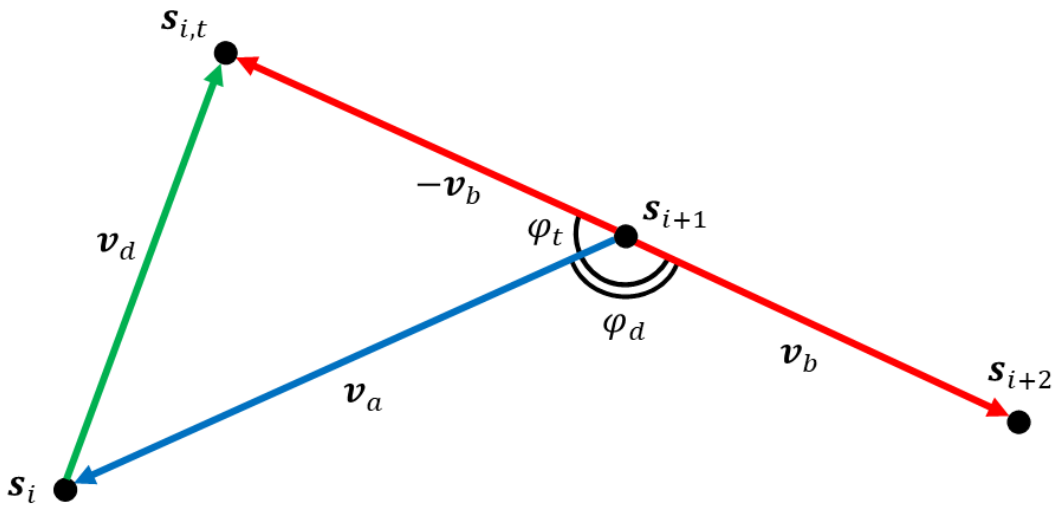

Figure 4.6: The basic concept behind the minimum steering angle factor.

$$\begin{aligned} \mathbf{H}_i = \mathbf{H}_{i+2} &= \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \\ \mathbf{H}_{i+1} &= \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}. \end{aligned} \tag{4.29}$$

4.3.3 Minimum Distance Factor

In order to show the advantages of using factor graphs for solving motion planning problems, we additionally implemented a minimum distance factor that, in combination with the racetrack bounding factor from Section 4.3.1, computes the *shortest path* through the racetrack.

Minimum Distance Factor Development

The minimum distance factor is relatively simple in comparion with the other two implemented factors. In general, it tries to minimize the distance between two adjacent states $\mathcal{S}_i = \{\mathbf{s}_i, \mathbf{s}_{i+1}\}$

The error of this factor is equal to the distance between the two states. Thus, if the two states are on top of each other, the error is minimized, and the distance is 0:

$$\mathbf{e}_{dist} = \mathbf{s}_{i+1} - \mathbf{s}_i = \begin{bmatrix} s_{i+1,x} - s_{i,x} \\ s_{i+1,y} - s_{i,y} \end{bmatrix}. \tag{4.30}$$

The factor function then computes as

$$f^{dist} = exp\ \{-\frac{1}{2}||\mathbf{s}_{i+1} - \mathbf{s}_i||^2_{\mathbf{\Sigma}_{dist}}\}. \tag{4.31}$$

Then, the Jacobian for this factor is computed as

$$\begin{aligned} \mathbf{H}_i &= \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \\ \mathbf{H}_{i+1} &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \end{aligned} \tag{4.32}$$

4.4 Factor Graph for Raceline Optimization

The general structure of the final factor graph with both the racetrack bounding factor and the minimum steering factor can be see in Figure 4.7. This factor graph will be used in our approach to optimize a given centerline to a minimum curvature path. The factor graph for the shortest path optimization can be seen in Figure 4.8.

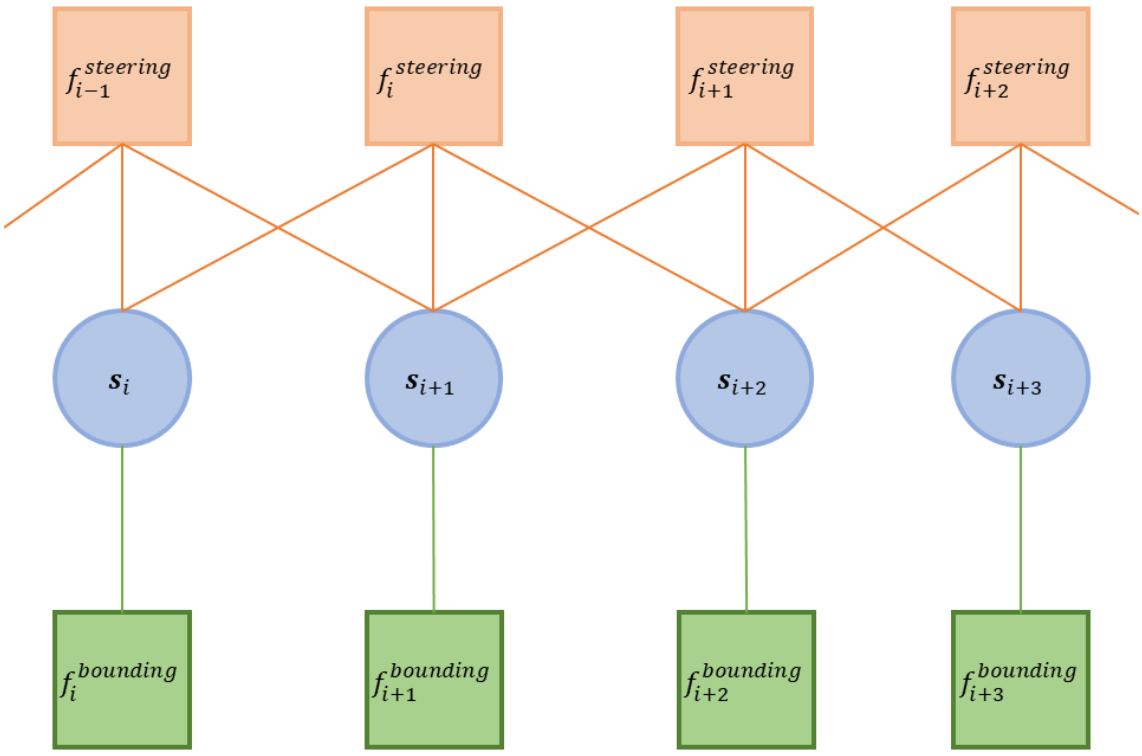

Figure 4.7: The complete factor graph for a minimum curvature raceline optimization.

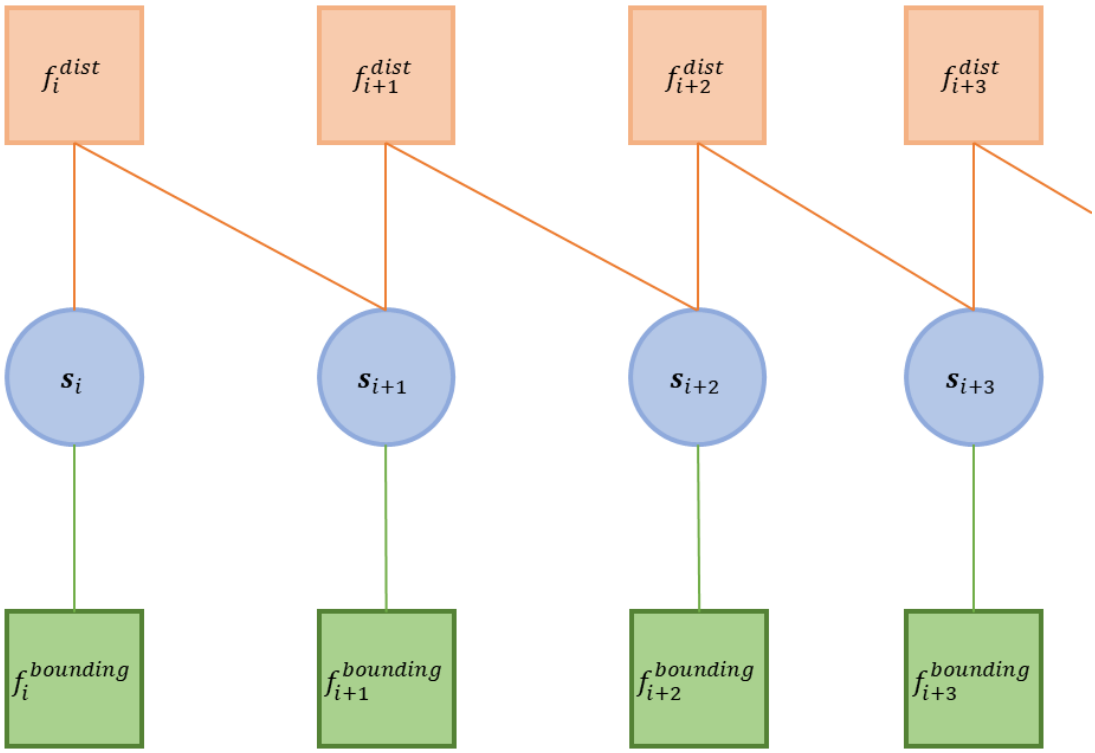

Figure 4.8: The general structure of the factor graph for computing the shortest path through the racetrack.

Chapter 5

Evaluation

In this chapter, the optimization using the factor graph desgined in Chapter 4 is evaluated and compared to the results achieved in [HWH+20]. The implementation of our factor graph approach is published at `https://github.com/Ashn-1/pi-racing`.

5.1 Utility Functions

For the evaluation of the results from our approach, we need to first define some utility functions that are used for computing some of the performance criteria. This includes the generation of the velocity and acceleration profiles, and the computation of the lap time, total curvature and travelled distance.

5.1.1 Velocity and Acceleration Profile Generation

The velocity profile is generated using a Forward/Backward solver (see [HWH+20, Section 3.5]; see [TUM19b] for the implementation). The input to the solver will be the optimized path from our minimum curvature path optimization. The solver computes the velocity profile in three steps:

1. Initialization step
2. Forward step - generating the Acceleration profile
3. Backward step - generating the deceleration/braking profile

Initialization Step

First, the velocity for all trajectory points is initialized to the maximum possible vehicle velocity as

$$v_i = v_{max}, \quad \forall i = 1, ..., N. \tag{5.1}$$

Then, compute the maximum possible longitudinal velocity given the curvature of the raceline. The higher the curvature of the raceline is at any given state, the lower the longitudinal velocity must be in order to sufficiently provide enough lateral velocity to drive the curve. The maximum possible longitudinal velocity given the curvature κ and the maximum lateral acceleration of the vehicle $a_{y,max}$ can be computed as

$$v_{max,long,i} = \sqrt{\frac{1}{|\kappa_i|}\ a_{y,max}}. \tag{5.2}$$

If the computed maximum possible longitudinal velocity is lower than the maximum vehicle velocity, then the velocity value is updated:

$$v_i = \min(v_i, v_{max,long,i}). \tag{5.3}$$

If friction losses are taken into account, then the maximum lateral acceleration used in (5.2) must be adjusted using the friction coefficient μ before computing the maximum possbile longitudinal velocity:

$$a_{y,max,f} = \mu\ a_{y,max}. \tag{5.4}$$

It should be noted here that the friction coefficient μ might consist of an array with coefficients for each trajectory point. Also, the lateral acceleration might be velocity dependent. In that case, the velocities computed in (5.2) are used as estimates to get the velocity dependent maximum lateral acceleration, which are then used to compute the actual maximum possible longitudinal velocity.

Forward Solver

In the forward part of the velocity generation, the acceleration of the vehicle is taken into account to compute the possible velocities for the path. Here, we compute the maximum possible velocity for the next path point v_{i+1} given the velocity of the current path point v_i, the spacial distance d_i between them, and the maximum possible longitudinal acceleration $a_{x,i}$ for the current velocity. The resulting velocity values can be computed as

$$v_{forward,i+1} = \sqrt{v_i^2 + 2a_{x,i}\ d_i}. \tag{5.5}$$

Here again, the velocity is updated, if the forward velocity from (5.5) is lower than the currently saved velocity in the velocity profile:

$$v_{i+1} = \min(v_{i+1},\ v_{forward,i+1}). \tag{5.6}$$

The maximum possible longitudinal acceleration $a_{x,i}$ is dependent on the tire potentials, machine limitations, and drag. The computations of these go beyond the scope of this work and are thus left out.

Backward Solver

In the backward solver part, the deceleration, or braking, of the vehicle is taken into account to compute the possible velocity values. In general, this part works exactly the same as the forward solver, but only in reverse direction, i.e. the parameters are all flipped upside down. Here, the computation of the possible velocity for the next path point is looped once, because the computation goes in the backward direction and the parameters for computing the possible acceleration (e.g. start velocity, radius, and friction) might not apply there.
The two velocities are computed as

$$\begin{aligned} v_{i+1,1} &= \sqrt{v_i^2 + 2a_{x,i}\, d_i} \\ v_{i+1,2} &= \sqrt{v_i^2 + 2a_{x,i+1}\, d_i}, \end{aligned} \tag{5.7}$$

and the lower of the two values is then taken as the final velocity from the backward solver:

$$v_{backward,i+1} = \min(v_{i+1,1-step},\ v_{i+1,2-step}). \tag{5.8}$$

Finally, the value from the velocity profile is updated, if the velocity from the backward sovler is lower than the current value:

$$v_{i+1} = \min(v_{i+1},\ v_{backward,i+1}). \tag{5.9}$$

This is also the resulting velocity profile that can be driven by the vehicle, given its handling limits and the optimized path. The acceleration profile is then simply generated by taking the difference between the velocity of two adjacent points. This can be computed as

$$a_i = v_{i+1} - v_i. \tag{5.10}$$

Implementation

The forward-backward solver is already implemented in the Trajectory Planning Helpers repository at [TUM19b] that is created by the same team as the reference paper [HWH+20].

5.1.2 Lap Time Computation

Using the velocity and acceleration profile generated in Section 5.1.1 and the distance between adjacent states, the computation of the lap time is trivial. The time needed to travel between two adjacent states is computed using the quadratic formula (acceleration between two states) or directly using the velocity (no acceleration between two states). The lap time then computes as

$$
\begin{aligned}
t_i &= \begin{cases} \frac{-v_i + \sqrt{v_i^2 + 2a_i\ d_i}}{a_i} & \text{, if } a_i \neq 0 \\ \frac{d_i}{v_i} & \text{, else} \end{cases} \\
t_{laptime} &= \sum_{i=1}^{N} t_i.
\end{aligned}
\tag{5.11}
$$

5.1.3 Total Curvature Computation

The curvature κ_i of a racetrack segment can be computed using (4.23). Here, the angular difference $\Delta\phi_i$ and the spacial distance Δd_i between two adjacent states are used in order to compute the curvature:

$$
\kappa_i = \frac{\Delta\phi_i}{\Delta d_i}, \quad \forall i = 1, ..., N. \tag{5.12}
$$

In [HWH+20], the total curvature is computed as the sum of all individual curvature values as

$$
\kappa_{total} = \sum_{i=1}^{N} \kappa_i. \tag{5.13}
$$

However, this approach can lead to an issue when comparing the total curvatures of different approaches. The curvature values can be positive and negative, thus a zig-zaggy path might still have a low curvature, because the curvature values cancel each other out. As an alternative, we are computing the total curvature using the sum of absolute curvature values as

$$
\kappa_{total} = \sum_{i=1}^{N} |\kappa_i|. \tag{5.14}
$$

5.1.4 Total Distance Computation

The length of the optimized path can be computed using the positions of the states as

$$
d_{path} = \sum_{i=1}^{N} ||\mathbf{s}_{i+1} - \mathbf{s}_i||_2, \tag{5.15}
$$

where the state $\mathbf{s}_{N+1} = \mathbf{s}_1$, i.e. the path loops back around to the start position.

5.2 Racetracks

We tested our approach on two different racetracks: the Tempelhof Airport Street Circuit used in the Formula E Berlin ePrix (Berlin 2018) and the Autodromo di Modena circuit (Modena 2019). The racetrack data is taken from [TUM19a], which is the implementation of the approach from [HWH+20].

The racetrack data consists of a reference line given in x and y coordinates and the width of the track to the left w_l and right w_r from each reference point. Here, left and right are with respect to the direction of the reference line, so in order to build the racetrack the normal vector at each reference point has to be computed. The normal vector at each reference point can be computed as

$$\begin{aligned} \Delta x &= x_i - x_{i+1},\ i = 1, ..., N-1, 1 \\ \Delta y &= y_i - y_{i+1},\ i = 1, ..., N-1, 1 \\ \mathbf{n}_i &= \begin{bmatrix} \Delta y \\ -\Delta x \end{bmatrix}. \end{aligned} \tag{5.16}$$

This normal vector points to the left of the reference line. Next, the racetrack boundaries to the left and right, $\mathbf{b}_l$ and $\mathbf{p}_r$ respectively, compute as

$$\begin{aligned} \mathbf{b}_{l,i} &= \begin{bmatrix} x_i \\ y_i \end{bmatrix} + \mathbf{n}_i\ w_{l,i}^T \\ \mathbf{b}_{r,i} &= \begin{bmatrix} x_i \\ y_i \end{bmatrix} - \mathbf{n}_i\ w_{r,i}^T. \end{aligned} \tag{5.17}$$

Finally, we can compute the centerline of the track, which is later used as the initial path for the path optimization. Here, the centerline is just the middle point between two track boundaries and thus computes as

$$\mathbf{s}_{initial,i} = \mathbf{b}_{r,i} + \frac{1}{2}(\mathbf{b}_{l,i} - \mathbf{b}_{r,i}). \tag{5.18}$$

The centerline points are also downsampled in order to achieve a better performance. This can be done when using probabilistic inference without affecting the resulting path, because the path can be upsampled to perform computations that need a higher resolution. In all other cases, the downsampled path can be used to save computational cost. Depending on how much the centerline is downsampled, the time for the path optimization can be drastically reduced. In our implementation, we downsampled the centerline by a factor of 2, in order to approximately match the number of states that are output from the approach in [HWH+20]. The resulting racetrack with centerline for the Berlin 2018 circuit can be seen in Figure 5.1.

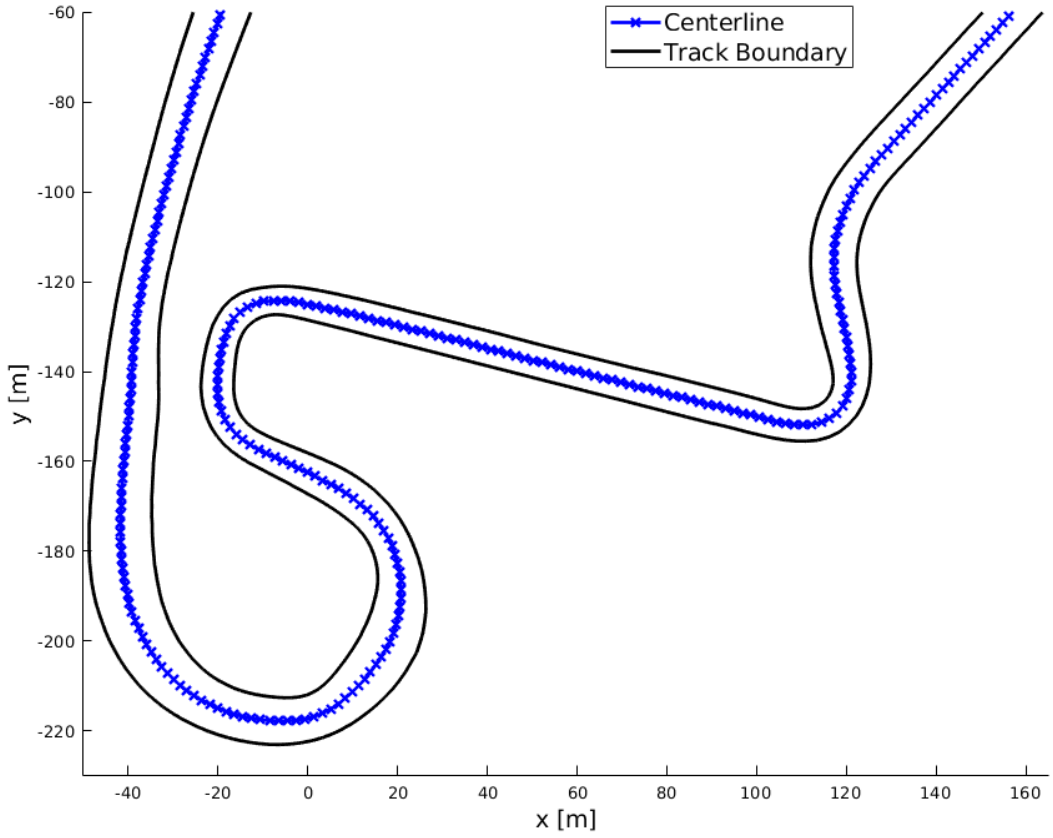

Figure 5.1: The initial racetrack computed from the dataset for the Berlin 2018 circuit.

5.3 Setup

For testing our approach on raceline optimization, we implemented it using the GTSAM library [BOR17]. The factors are implemented in C++ and made available in MATLAB using the MATLAB wrapper functionality of GTSAM. The code for testing the approach is then written in MATLAB. The script is structured as follows:

1. Initialization
2. Data handling
3. Pre-processing
4. Factor graph building

5. Optimization

6. Post-processing

7. Check optimized path

8. Plot and output results

First, in the initialization step, the GTSAM and our custom library are loaded. Additionally, all settings that are adjustable by the user can be found here including the racetrack that is used, whether to use a flying lap, or what should be plotted. Next, in the data handling step, the racetrack dataset is loaded and the racetrack boundaries and the centerline are computed (see Section 5.2). Then, in the pre-processing step, the individual cost boundaries are computed based on the angle of the centerline at each point (see Section 4.3.1). Afterwards, in the factor graph building step, the factors are created and the factor graph is built. Next, in the optimization step, the factor graph is optimized using a nonlinear least squares optimization algorithm. In our current implementation, there is no post-processing done. However, it is still listed here, because the approach from [TUM19a] uses post-processing and it has to be considered for the timing of the computation. Our next step is the check of the optimized path, where we check the path for any out of bounds states. This check also considers the safety margin, i.e. even if a state is on the racetrack, if it does not keep a certain safety distance to the track boudaries, it is considered out of bounds. Lastly, we plot and output our results.
For timing purposes, only the steps 3 - 6 (pre-processing, factor graph building, optimization, and post-processing) are considered. The same is done for the timing of the reference results from [HWH$^+$20]. The script uses a flying start for the optimization. This means that the path assumes that the vehicle already finished one lap and drives through the start, instead of starting from standstill at the start line. The script includes a flag in order to toggle the usage of a flying start. The computations are performed on a laptop system running Ubuntu 18.04.6 LTS with an i5-7200 CPU @ 2.5 GHz x4 and 8 GB of RAM. Version R2021b of MATLAB is used. The approach from [HWH$^+$20] was run on the same device in order to ensure better comparability.

5.3.1 Hyperparamters

In our implementation, there are four hyperparameters that are needed to be set: the cost matrix of the racetrack bounding factor, the cost matrix of the minimum steering factor, the minimum cost boundary for the individual cost boundaries, and the safety distance to compute out of bounds states.

Cost Matrices

The values of the cost matrices are relative to each other, thus changing the magnitude of the cost matrices of all factors by the same amount will result in the same optimized path. The only difference will be higher error values as the cost matrices only scale the errors. As a result, we can fix one of the cost matrices to the identity matrix and only change the other one to get good results. The decision on which of the matrices to fix is arbitrary; we decided to fix the cost of the racetrack bounding factor:

$$\mathbf{\Sigma}_{bound} = \mathbf{I}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{5.19}$$

The cost of the minimum steering factor was then decreased (corresponding to a higher weight on it) until the threshold for a feasible path was found. Decreasing the cost further would have lead to an unfeasible path, i.e. some out of bounds states. This cost is highly dependent on the racetrack that it is used on. Here, the cost matrices for the Berlin 2018 and the Modena 2019 circuit were computed as

$$\begin{aligned} \mathbf{\Sigma}_{steer,Berlin} &= \begin{bmatrix} 6 \times 10^{-}3 & 0 \\ 0 & 6 \times 10^{-}3 \end{bmatrix} \\ \mathbf{\Sigma}_{steer,Modena} &= \begin{bmatrix} 2 \times 10^{-}3 & 0 \\ 0 & 2 \times 10^{-}3 \end{bmatrix}. \end{aligned} \tag{5.20}$$

The same approach for optimizing the cost was used for the cost matrix of the minimum distance factor. For the Berlin 2018 circuit, it computes as

$$\mathbf{\Sigma}_{dist,Berlin} = \begin{bmatrix} 5 \times \times 10^{-}1 & 0 \\ 0 & 5 \times \times 10^{-}1 \end{bmatrix}. \tag{5.21}$$

Individual Cost boundaries

The individual cost boundaries needs one hyperparameter assigned. Here it is the minimum cost boundary $d_{cost,min}$ that determines the minimum distance from the actual track boundaries at which the prior line ends. We did multiple tests with different values and achieved the best results in terms of lap time by setting it to 1.5m.

Safety Distance

The safety distance ϵ_{safety} determines the minimum acceptable distance between the actual racetrack boundaries and the vehicle center. Assuming that the vehicle width will most likely not exceed 2m, the safety distance was set to 1m. Technically, this hyperparameter is not used for the optimization itself and only for the checking of the path after the optimization is finished. Nonetheless, it was still included as an hyperparameter here, because it affects the cost values. The lower the safety

distance is set, the lower the cost of the minimum steering angle can be set (i.e. more weight on the factor).

5.4 Results

The optimized paths for the Berlin 2018 circuit can be seen in Figure 5.2 and for the Modena 2019 circuit in Figure 5.3. Both of the figures also include the path optimized by the approach from [HWH+20] using the QP approach and also the improved iterative QP approach. Lastly, the results of the shortest path optimization for the Berlin 2018 circuit can be seen in Figure 5.4. In Table 5.1, the comparison of the performance criteria between the proposed approach and the benchmark framework [HWH+20] can be seen.

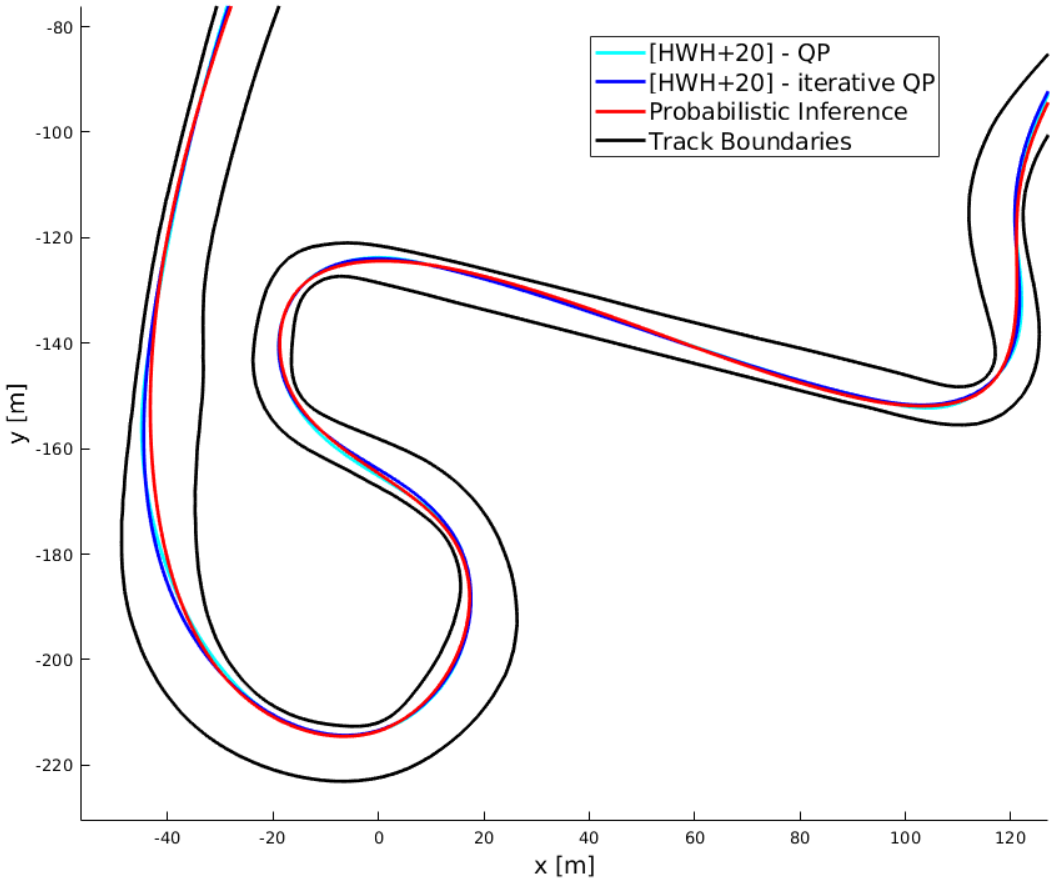

Figure 5.2: The optimized paths for the Berlin 2018 circuit using the probabilistic inference approach.

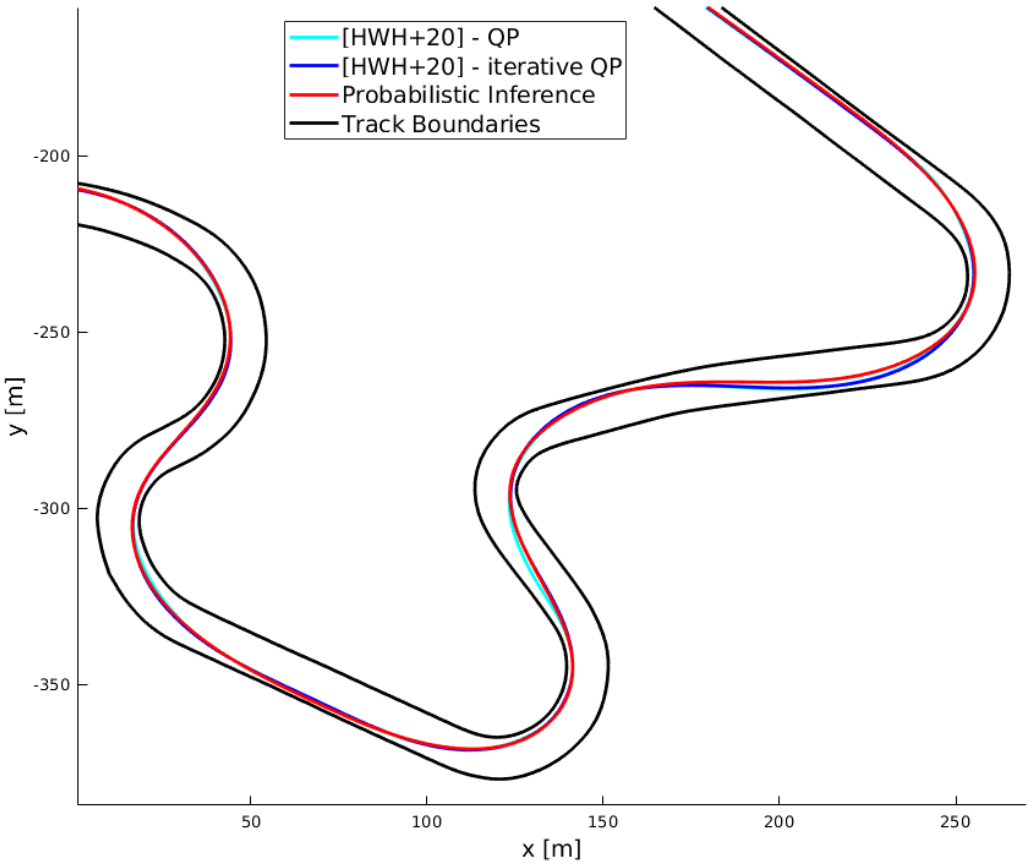

Figure 5.3: The optimized paths for the Modena 2019 circuit using the probabilistic inference approach.

The results from both the proposed approach and [HWH+20] are very similar. The most notable difference is the early entering of curves in our approach. While this leads to a lower overall distance travelled, it also leads to a lower velocity in the curves, which cancels each other out. The quality of the resulting paths with respect to the performance criteria seem to be approximately equal. However, our approach computed these results in a fraction of the time that the two approaches from [HWH+20] needed.

Besides the lower optimization time that our approach needs due to the use of factor graphs, one of the biggest aspects for the low computation time is the pre-processing of the racetrack. For our approach, only the individual cost boundaries are computed in the pre-processing, which only took a few milliseconds to compute. The approach from [HWH+20] had to smoothen out the input racetrack using splines before feeding it to the optimization. The optimization approach is very sensitive

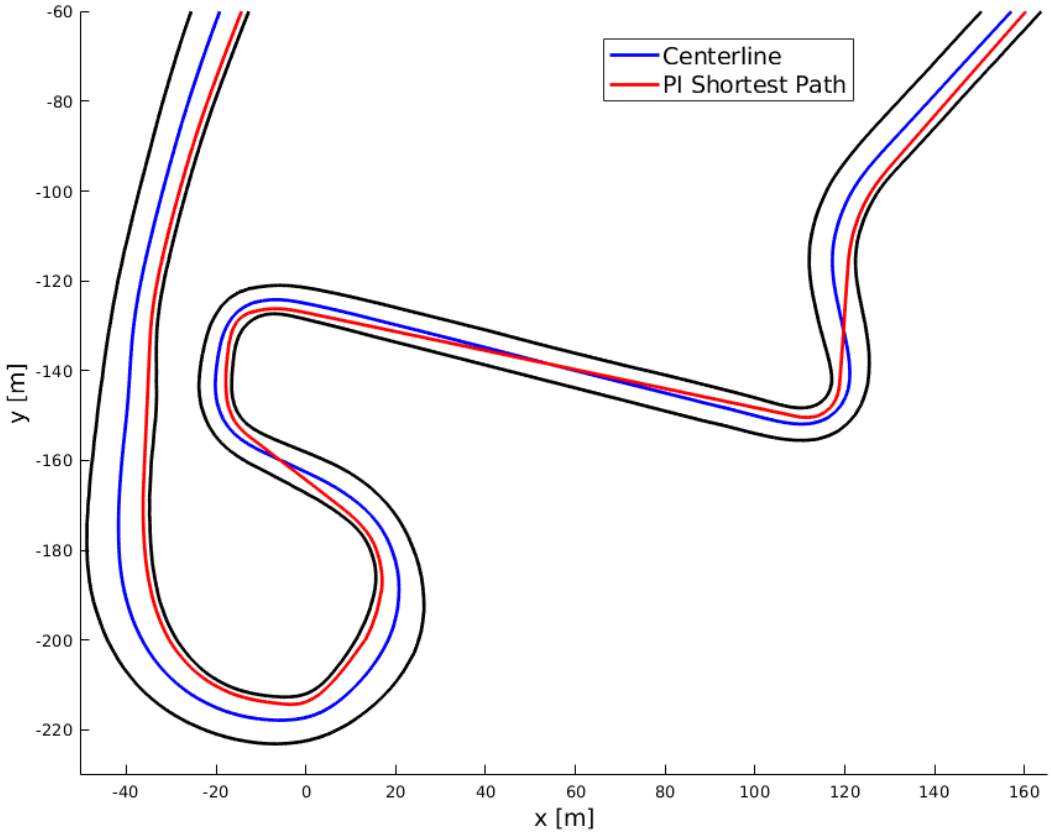

Figure 5.4: The shortest path optimization for the Berlin 2018 circuit using the probabilistic inference approach.

to large changes, thus this step was necessary in order to get good results. However, this also leads to a lot of splines being computed, which is computationally very expensive and clocked in at around 5s on our system.
Also, the shortest path computation using the probabilistic inference approach show the versatility of our approach. By only switching out the minimum steering factor with the minimum distance factor, we could easily change the desired output path. The velocity profile of the optimized path using the probabilistic inference approach for the Berlin 2018 circuit can be seen in Figure 5.5 and for the Modena 2019 circuit in Figure 5.6.

Table 5.1: Benchmark results for probabilistic inference framework.

Approach	Computation Time [s]	Lap Time [s]	Curvature [1/m]	Path Length [m]
Berlin 2018				
PI Minimum Curvature	**1.4**	81.6	12.1	2326.1
[HWH+20] QP	11.9	81.8	11.0	2326.7
[HWH+20] iterative QP	29.1	80.9	10.9	2324.0
PI Shortest Path	1.5	88.55	11.839	2292.6
Modena 2019				
PI Minimum Curvature	**0.9**	78.8	13.0	1995.6
[HWH+20] QP	9.3	79.7	13.2	2000.1
[HWH+20] iterative QP	19.7	79.0	13.1	2001.9
PI Shortest Path	0.9	89.52	12.4	1971.3

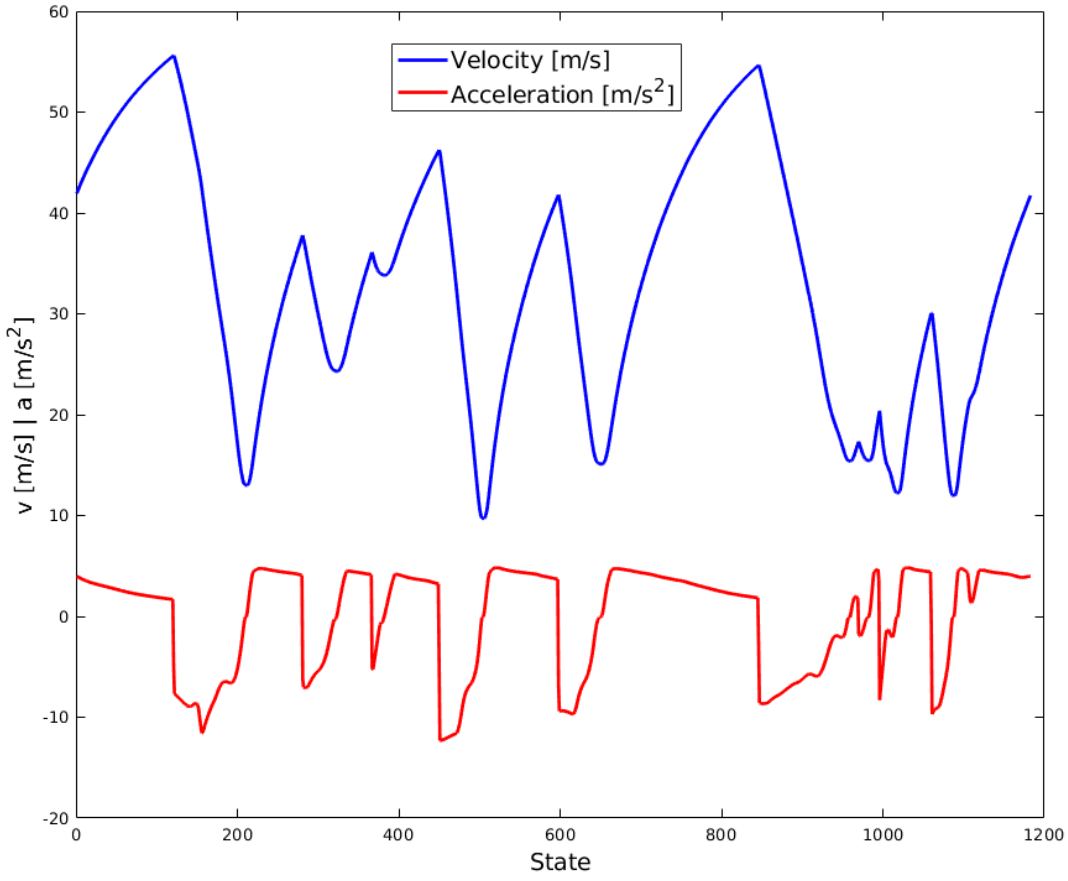

Figure 5.5: The velocity and acceleration profile for the optimized path on the Berlin 2018 circuit using the probabilistic inference approach. The corresponding path can be seen in Figure 5.2.

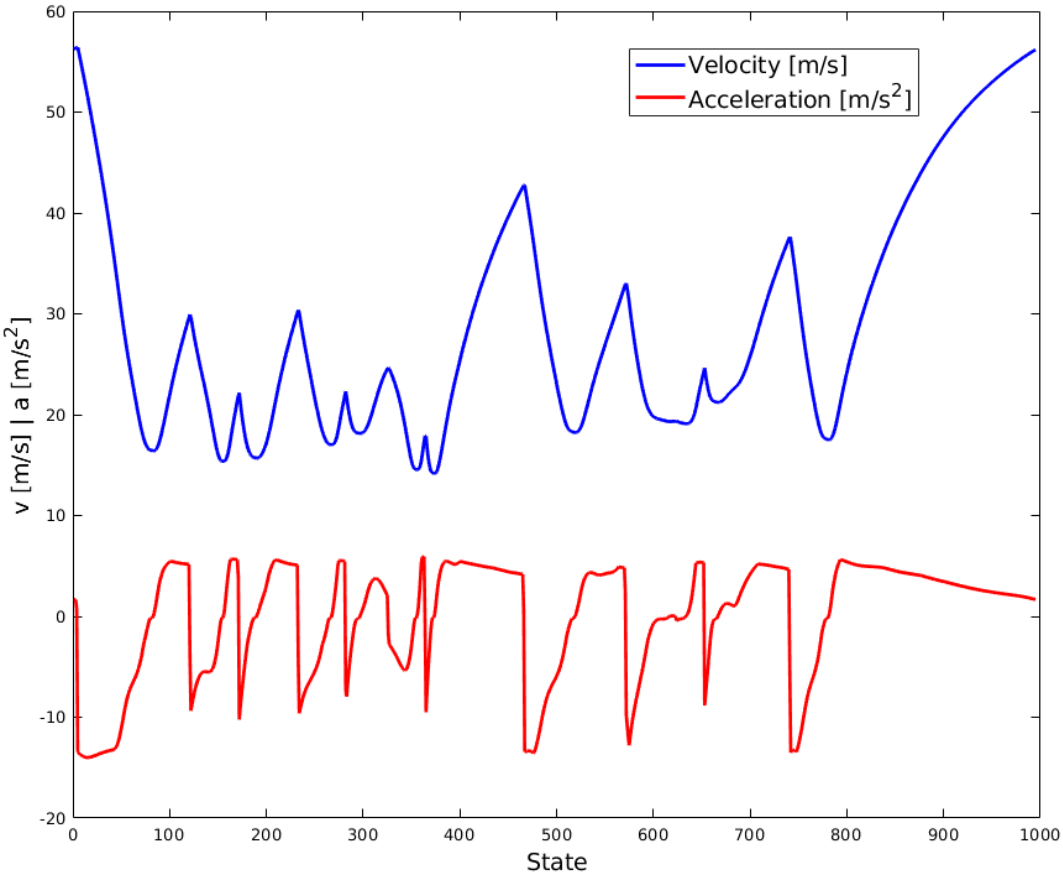

Figure 5.6: The velocity and acceleration profile for the optimized path on the Modena 2019 circuit using the probabilistic inference approach. The corresponding path can be seen in Figure 5.3.

Chapter 6

Conclusion

In this work, we explored the probabilistic inference approach for motion planning. Specifically, our goal was to use this approach to generate a minimum curvature path for a given racetrack, to build a framework for raceline optimization using probabilistic inference, and to compare the achieved results with other state-of-the-art raceline optimization algorithms. In order to do this, we first laid out the general functionality of probabilistic inference using a maximum a posteriori estimator. This problem is then represented using factor graphs, in order to decouple the individual dependencies within the inference problem. Then, the duality between optimization and inference is outlined and the probabilistic inference problem is converted into an optimization problem that is solved using factor graphs.

Next, we developed the necessary factors for our desired minimum curvature path, namely the racetrack bounding factor and the minimum steering factor. Also, we have to include some additional measures in form of the individual cost boundaries, in order to ensure that the optimization computes a path that is both feasible and has a low lap time. Finally, we ran our implementation of the probabilistic inference approach for raceline optimization on two racetracks and compared our results with those achieved by the approach from [HWH$^+$20]. To show the flexibility of this approach, we also implemented a factor for a shortest path optimization and also included the results in our evaluation section.

The results of the probabilistic inference approach are qualitatively on par with the results from the state-of-the-art approach used in [HWH$^+$20]. Both, the resulting paths and the lap times, are comparable and only show minor differences. However, our approach computes these results in significantly less time compared to [HWH$^+$20].

In summary, the probabilistic inference approach shows promising results in terms of future applicability. The low computation time makes it a viable candidate for real-time racing, where the trajectory is not computed offline before the race, but during the actual racing event. Also, the quality of the results and the flexibility of using factor graphs for the optimization shows that this approach could replace other state-of-the-artplanning approaches. However, to achieve this, the probabilis-

tic inference approach has to be further developed and other aspects of racing have to be considered.
One of the major drawbacks of the probabilistic inference approach is that it is an unconstrained optimization. This means that the results of the approach might not be feasible, i.e. parts of it might be out of bounds. However, the optimization of the cost values can be easily automated to compute multiple iterations with different costs, until none of the states are out of bounds anymore. Other than the cost values, none of the other parameters usually need to be changed between different racetracks.
Moreover, in this work, only the global planning is taken care of by the probabilistic inference approach. However, the low computation time makes it a viable candidate for local optimization as well. This, however, has to be tested in-depth in future work to evaluate its viability there. Lastly, the current implemention of the racetrack bounding factor is nonlinear and thus makes it necessary to use a nonlinear factor graph. In future, a new racetrack bounding factor could be developed that is linear. This would make it possible to use a linear factor graph, which would significantly reduce the computation time compared to the current approach.

List of Figures

List of Tables

Bibliography

[BCMS08] F. Braghin, F. Cheli, S. Melzi, and E. Sabbioni. Race driver model. *Computers & Structures*, 86(13):1503–1516, 2008. Structural Optimization.

[BOR17] BORG Lab at Georgia Institute of Technology. Georgia Tech Smoothing and Mapping Library. https://github.com/borglab/gtsam, 2017. Accessed: 2021-05-31.

[DK17] Frank Dellaert and Michael Kaess. Factor graphs for robot perception. *Found. Trends Robotics*, 6(1-2):1–139, 2017.

[HWH+20] Alexander Heilmeier, Alexander Wischnewski, Leonhard Hermansdorfer, Johannes Betz, Markus Lienkamp, and Boris Lohmann. Minimum curvature trajectory planning and control for an autonomous race car. *Vehicle System Dynamics*, 58(10):1497–1527, 2020.

[KSG19] Nitin R. Kapania, John K. Subosits, and J. Christian Gerdes. A sequential two-step algorithm for fast generation of vehicle racing trajectories. *CoRR*, abs/1902.00606, 2019. URL: `http://arxiv.org/abs/1902.00606`, `arXiv:1902.00606`.

[MDY+18] Mustafa Mukadam, Jing Dong, Xinyan Yan, Frank Dellaert, and Byron Boots. Continuous-time gaussian process motion planning via probabilistic inference. *Int. J. Robotics Res.*, 37(11), 2018.

[SDH+14] John Schulman, Yan Duan, Jonathan Ho, Alex X. Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *Int. J. Robotics Res.*, 33(9):1251–1270, 2014.

[TG07] Marc Toussaint and Christian Goerick. Probabilistic inference for structured planning in robotics. pages 3068–3073, 2007.

[TUM19a] TUMFTM. Global racetrajectory optimization. `https://github.com/TUMFTM/global_racetrajectory_optimization`, 2019. Accessed: 2021-06-10.

[TUM19b] TUMFTM. Trajectory planning helpers. `https://github.com/TUMFTM/trajectory_planning_helpers`, 2019. Accessed: 2022-03-06.

[ZRD+13] Matthew Zucker, Nathan D. Ratliff, Anca D. Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M. Dellin, J. Andrew Bagnell, and Siddhartha S. Srinivasa. CHOMP: covariant hamiltonian optimization for motion planning. *Int. J. Robotics Res.*, 32(9-10):1164–1193, 2013.

License