

ForzaETH Race Stack - Scaled Autonomous Head-to-Head Racing on Fully Commercial off-the-Shelf Hardware

Nicolas Baumann^{*†‡}, Edoardo Ghignone^{†‡}, Jonas Kühne[†], Niklas Bastuck[‡], Jonathan Becker[‡], Nadine Imholz[‡], Tobias Kränzlin[‡], Tian Yi Lim[‡], Michael Lötscher[‡], Luca Schwarzenbach[‡], Luca Tognoni[‡], Christian Vogt[‡], Andrea Carron[§], and Michele Magno[‡]

ETH Zürich
Zurich, Switzerland

Abstract

Autonomous racing in robotics combines high-speed dynamics with the necessity for reliability and real-time decision-making. While such racing pushes software and hardware to their limits, many existing full-system solutions necessitate complex, custom hardware and software, and usually focus on Time-Trials rather than full unrestricted Head-to-Head racing, due to financial and safety constraints. This limits their reproducibility, making advancements and replication feasible mostly for well-resourced laboratories with comprehensive expertise in mechanical, electrical, and robotics fields. Researchers interested in the autonomy domain but with only partial experience in one of these fields, need to spend significant time with familiarization and integration. The ForzaETH Race Stack addresses this gap by providing an autonomous racing software platform designed for F1TENTH, a 1:10 scaled Head-to-Head autonomous racing competition, which simplifies replication by using commercial off-the-shelf hardware. This approach enhances the competitive aspect of autonomous racing and provides an accessible platform for research and development in the field. The ForzaETH Race Stack is designed with modularity and operational ease of use in mind, allowing customization and adaptability to various environmental conditions, such as track friction and layout. Capable of handling both Time-Trials and Head-to-Head racing, the stack has demonstrated its effectiveness, robustness, and adaptability in the field by winning the official F1TENTH international competition multiple times.

Keywords— Autonomous driving, Autonomous racing, Motion control, Robotic perception, Path planning, State estimation, Open source software

Supplementary Material

Open-source code of the proposed system: https://github.com/ForzaETH/race_stack

^{*}Corresponding Author:

Nicolas Baumann, ETH Zürich, ETF F110, Sternwartstrasse 7, 8092 Zürich, Email: nicolas.baumann@pbl.ee.ethz.ch

[†]Contributed Equally

[‡]Affiliated with the Center for Project-Based Learning (PBL), ETH Zürich, Zürich, Switzerland

[§]Affiliated with the Institute for Dynamic Systems and Control (IDSC), ETH Zürich, Zürich, Switzerland

1 Introduction

Motorsport has historically demonstrated its capability to be a catalyst for introducing cutting-edge technologies to the broader automotive industry [Jarvenpaa and Standaert, 2017, Finn, 2021]. Autonomous racing provides a valuable context to investigate some of the critical scenarios of general self-driving which necessitate operation at the friction limit, such as in high-speed or low-friction environments like icy or dusty roads. Autonomous racing inherently demands operation at these boundaries, compelling the vehicle to its physical, computational, and algorithmic limits, thus serving as a stress testing platform for self-driving [Kabzan et al., 2020, Betz et al., 2023, Betz et al., 2022, Wischnewski et al., 2022].

Drawing inspiration from human-driven motorsport, e.g. *Formula 1*, autonomous racing competitions are structured in a two-step process: first come the *Time-Trials*, a *Qualifying* phase where autonomous agents aim to clock the fastest lap times, and then the competition culminates in the *Grand Prix* where up to twenty cars ideally engage in *Head-to-Head* racing. Prominent autonomous racing leagues such as Formula Student Driverless (FSD) and Indy Autonomous Challenge (IAC) predominantly focus on the *Qualifying* aspect [Betz et al., 2023, Kabzan et al., 2020, Jung et al., 2023, Raji et al., 2023], as the challenges of full-scale autonomous racing, including high costs, safety considerations, and significant engineering overhead, often necessitate restrictions in the racing scenarios. Conversely, small-scale autonomous racing, exemplified by *F1TENTH*, offers an opportunity to fully embrace *Head-to-Head* racing dynamics in a more accessible environment [Li et al., 2023, O’Kelly et al., 2019]. *F1TENTH*’s unrestricted *Head-to-Head* racing and the therefore full robotic autonomy stack required to compete, present complex algorithmic challenges, especially when further considering that the miniaturization intensifies algorithmic design challenges due to limited hardware resources. Full-scale autonomous racing solutions typically require complex, custom, or proprietary hardware and software [Kabzan et al., 2020, Betz et al., 2023, Jung et al., 2023, Raji et al., 2023], limiting reproducibility only to well-resourced research facilities. In contrast, the *ForzaETH* team’s racecar, shown in Fig. 1, is fully built on Commercial off-the-Shelf (CotS) hardware and facilitates accessibility.

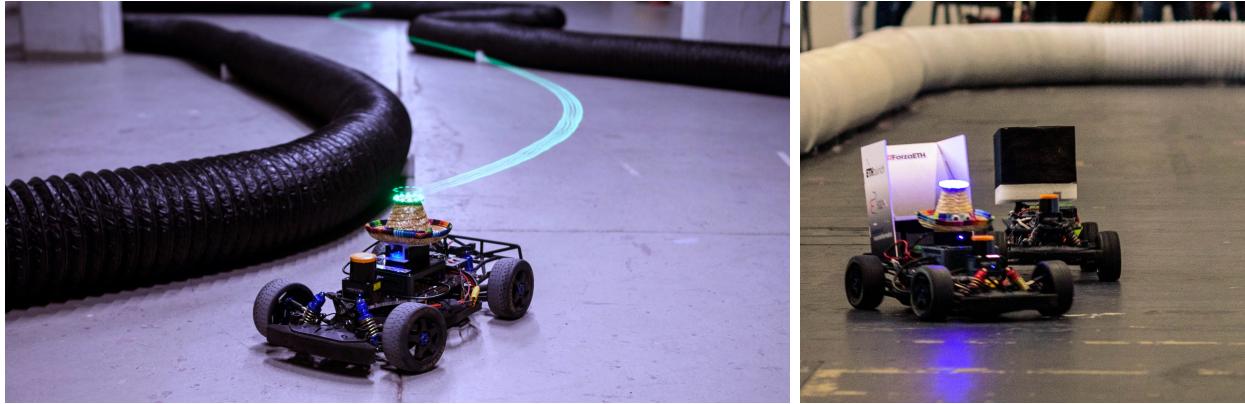


Figure 1: The physical *ForzaETH* autonomous racecar running the proposed *ForzaETH Race Stack*. On the right, an overtaking maneuver during the ICRA23 *F1TENTH* race.

F1TENTH racing competitions are typically held at robotics conferences, such as IROS and ICRA [Li et al., 2023, O’Kelly et al., 2020b, O’Kelly et al., 2019]. Predominantly, university teams from all over the world compete to develop the fastest and most intelligent racecar. The race is typically structured in the following phases:

I Free Practice: All teams can use the racetrack freely prior to the race. This time slot is typically used to map the track.

II Time-Trials: In this phase, each racecar navigates an empty track, aiming to complete as many uninterrupted laps as possible while minimizing lap time. Scores are calculated by considering both the

number of consecutive crash-free laps and the best lap time.

III Head-to-Head: In *F1TENTH*, the *Time-Trials* establish the seeding for *Head-to-Head* racing brackets. Thus, the first-place racer competes against the last, second against second last, and so forth, in a 1v1 knockout tournament. Each 1v1 battle consists of a best-of-three heat, with the victor being the first to complete 10 laps. Winners advance; losers are eliminated, continuing until a victor emerges.

F1TENTH imposes specific race constraints, including a 1:10 form factor, an electric drivetrain, infrastructure-less localization (no Global Positioning System (GPS) or motion capture system), and most importantly, fully onboard computing. From a robotic standpoint, the competition demands a full autonomy stack for a mobile robot that pushes the robot to its limit in terms of physical traction, computational efficiency, and sensor processing.

While the *F1TENTH* competition has paved the way for research in *Time-Trials* and *Head-to-Head* racing, its format focuses on a 1v1 *Head-to-Head* setting. The primary reason to limit this to a single opponent is the inherent complexity of the task. However, this challenge could be bridged and facilitated, by providing a fully open-sourced stack that allows for further development towards more complex challenges, like racing against multiple opponents. This work aims to do so, by offering a comprehensive and technical overview of the *ForzaETH F1TENTH* autonomous racing stack and open-sourcing the full race stack, that was used when winning both the *German Grand-Prix 2022* and the *ICRA Grand-Prix 2023* [Li et al., 2023]. With this, we aim to contribute to the autonomous racing and research community, sharing the intricate technical and algorithmic lessons learned and providing the first complete, embedded, fully onboard, and real-time *Head-to-Head* autonomous racing stack for CotS hardware. To summarise the contributions of this paper:

I Race Stack Architecture: We describe and detail the complete *Head-to-Head* capable race stack for scaled autonomous racing using CotS hardware, detailing technical and algorithmic intricacies of integrating and adapting the State-of-the-Art (SotA) algorithms in mobile robotics and embedded computing into a unified autonomous racing stack, adhering to the *See-Think-Act* cycle.

II Performance: We provide comprehensive quantitative, and qualitative performance evaluations for each of the introduced subsystems and overall performance of the *ForzaETH* race stack. This allows us to assess and compare robotic SotA algorithms in the context of autonomous racing and embedded computing.

III Open-Source: We provide the complete and race-winning robotic autonomy stack of *ForzaETH* on CotS hardware, with reproducibility in mind. Hence, we enable the broader autonomous racing research community to build upon this stack, lowering the barrier of entry even more and allowing for further research in the challenging *Head-to-Head* domain.

2 Related Works

Autonomous driving competitions go back to the 2005-2009 period, when the *DARPA Grand Challenges* [Thrun et al., 2007] and *URBAN Challenge* [Urmson et al., 2009] were held. The task for autonomous vehicles in such settings was to complete a predefined path on their own, either in a desert environment or in an urban one. More recently, the *SAE autodrive challenge* [Burnett et al., 2021] defines a competition where level 4 autonomous driving was the final target. While these competitions are of great importance for their fundamental work in sparking interest in autonomous vehicles, they are in rather different set-ups when compared to more recent autonomous racing competitions, such as FSD, IAC, *Roborace* and *F1TENTH*.

Starting in 2017, one of the first examples of autonomous racing competition started to occur in the shape of the FSD challenge [Kabzan et al., 2020], in which the *Formula Student* racing cars, previously only human-driven, needed to be adapted to be able to drive in different types of challenges. The competition sparked

a parallel research interest, that was then manifested in multiple works detailing the high-performance algorithms, not only as a part of the system [Strobel et al., 2020, Srinivasan et al., 2020, Vázquez et al., 2020a, Costa et al., 2023], but also in a broader view with the description of the full system, similarly to this work. The work of [Kabzan et al., 2020], for example, details both the hardware and software architecture that brought the Academic Motorsports Club Zurich (AMZ) team to multiple victories in their respective challenges. Comparably, the *Roborace* [Roborace, 2023] (2017-2021) and the IAC [IAC, 2023] (2019-now) associations organized competitions that attracted research interest both on the specific algorithms [Heilmeier et al., 2020, Fabian Christ and Lohmann, 2021, Wischnewski et al., 2021] [Karle et al., 2023, Stahl et al., 2019b, Seong et al., 2023], and on full racing stack integrations [Betz et al., 2019] [Jung et al., 2023, Betz et al., 2023, Raji et al., 2022].

Compared to the full-size autonomous systems discussed, the specific platform utilized in this paper is much more accessible, as it is made up of CoTS hardware, comes smaller at a 1:10 scale, and requires generally lower monetary and infrastructural investments (as can be seen in the comparison in Table 1). The forthcoming sections delve into the tangible benefits of this accessibility, as the platform turned out to be a validation platform for different research works, inspiring parallel 1:10 autonomous driving projects and attracting interest through the recurrent championships organized throughout the years.

F1TENTH competitions have been organized since 2016 [F1TENTH, 2023], and since 2018 races have been held at least twice a year in conjunction with scientific conferences, therefore appealing to a broader and worldwide audience. Furthermore, many other research platforms were developed simultaneously, highlighting the high flexibility and adaptability of research projects on downscaled platforms. Such projects came out both on a similar 1:10 scale, such as the Berkeley Autonomous Racecar [BARC, 2023], the MIT RACE-CAR [Karaman et al., 2017], the RoSCAR [Hart et al., 2014], and the MuSHR racecar [Srinivasa et al., 2019], and at different smaller scales, such as Chronos [Carron et al., 2023], IDS3C [Chalaki et al., 2022], Robotarium [Wilson et al., 2020], and the Cambridge Minicar [Hyldmar et al., 2019]. However, while there are papers detailing the specifications that define an *F1TENTH* car [O’Kelly et al., 2020b, O’Kelly et al., 2019] and papers that describe high-level optimization toolchains for the platform [O’Kelly et al., 2020a], there is, to the best of the authors’ knowledge, no work that, similarly to [Betz et al., 2023, Kabzan et al., 2020], describes a full algorithmic architecture. This work aims to fill this gap by presenting the *ForzaETH Race Stack*, a reproducible, autonomous robotic system deployed on CoTS hardware during official competitions that achieved competitive results.

In the next paragraphs, we will detail SotA works related to the specific subsystems of the stack that were previously deployed on either the *F1TENTH* platform or similar autonomous racing machines, therefore being subject to the sensor and computational constraints considered in this work. An overarching comparison is then presented in Table 2.

Localization and State Estimation: Low-latency localization and state estimation are needed to ensure correct knowledge of the robot’s position, velocity, yaw rate, and acceleration. The first step needed in an autonomous racing stack is obtaining the position of the ego robot in the space using localization algorithms, and, in the case of *F1TENTH* autonomous racing, this typically happens in a predetermined environment. Localization is usually done with pose-graph based Simultaneous Localization And Mapping (SLAM) techniques, such as *Cartographer* [Hess et al., 2016] and *Slam Toolbox* [Macenski and Jambrecic, 2021], or Monte Carlo Localization (MCL) based techniques [Gerkey, 2023], [Walsh and Karaman, 2018], [Lim et al., 2024], [Stahl et al., 2019b]. To then obtain a filtered state estimate of the car, different filtering techniques are used, such as an Extended Kalman Filter (EKF) or a Unscented Kalman Filter (UKF) [Wan and Van Der Merwe, 2000], [Moore and Stouch, 2014], [Buckman et al., 2022]. In addition, filtering techniques have also been previously described in the literature in the context of a full-system description similar to this work. A first example is the one proposed in [Wischnewski et al., 2019] where a Kalman Filter (KF), that employs a kinematic model, is used to fuse GPS and Light Detection and Ranging (LiDAR) localization based on [Gerkey, 2023]. Another example is presented in [Kabzan et al., 2020], where a highly specialized, vision-based SLAM algorithm fuses velocity estimation with cone detection to obtain precise localization estimates. While these two latter systems were deployed on more computationally powerful computers, to

	F1TENTH	FSD	IAC	Roborace
Scale	1:10	1:1.5	1:1	1:1
Maximum Speed	$\sim 15 \text{ m s}^{-1}$	$> 28 \text{ m s}^{-1}$	75 m s^{-1}	$\sim 84 \text{ m s}^{-1}$
Computation Units	Intel NUC or NVIDIA Jetson Xavier NX	PIP39 + NVIDIA Jetson TX 2 and RTX 1050Ti	ADLink AVA-3501	NVIDIA Drive PX2 + Speedgoat Mobile Target Machine
Hardware Availability	Off-the-shelf components	Fully custom platform	Platform provided by the competition	Platform provided by the competition
Competition Format	<i>Time-Trials</i> <i>Head-to-Head</i>	<i>Time-Trials</i> † Efficiency, Business, Cost, Design	<i>Time-Trials</i> Overtaking Competition	<i>Time-Trials</i>
Cost	$\sim 5000 \text{ USD}$	$> 100\,000 \text{ USD}$	350 000 USD	1 000 000 USD ‡

Table 1: Comparison of sizes, maximum speed, used computation platforms, and hardware availability of various autonomous racing platforms. The data for FSD, IAC and *Roborace* are taken, respectively, from [Kabzan et al., 2020], [Betz et al., 2023], [Betz et al., 2019]. †: the FSD competition is composed of multiple different disciplines where a single car is tested. These disciplines are here grouped under the *Time-Trials* name. ‡: from <https://thearsenale.com/products/robocar>

the best of the authors’ knowledge, no complete state estimation pipeline was deployed on CotS hardware without GPS before this paper, and only partial pieces of a system were studied, e.g. cone-based localization, which was also deployed on 1:10 scaled cars in [Brunnbauer and Bader, 2019] but in a very restricted space (maximum 2 m by 2 m) and at presumably low velocities (below 5 m s^{-1}). Whereas this work has been tested at speeds up to 11 m s^{-1} [Becker et al., 2023].

Detection and Tracking: While the SotA in 3D detection and tracking is clearly achieved with Machine Learning (ML) techniques both for camera-based [Zong et al., 2023] and LiDAR-based [Lu et al., 2023] settings, the platform considered in this paper has a limited set of sensors at its disposal (e.g. no 3D LiDAR) and an even more limited computational capacity, that does not allow for the deployment of the SotA ML models with a sufficient latency, requiring research on either smaller models or classical algorithms.

For camera-based ML systems, FSD was a driving force of research, as the detection of cones is crucial for this task. The most commonly employed architecture is the one named You Only Look Once (YOLO), which was deployed on the *Formula Student* platform in its v2 [Dhall et al., 2019], v3 [Strobel et al., 2020] and v5 versions [Benjumea et al., 2021]. When considering non-ML Computer Vision (CV) techniques, there are fewer works in general, as the standard sensor setup of the *F1TENTH* car uses the less common 2D LiDAR. A work that explains how such a setup can be incorporated in the *F1TENTH* platform is [Konstantinidis, 2020], showing how basic rectangle fitting techniques and global nearest neighbour can be exploited to obtain detections that can then be incorporated in a UKF. We incorporate similar techniques and the *Adaptive Breakpoint* method from [Amin et al., 2022] into the detection and tracking system presented in this work.

Planning: When considering global planning, since this task is usually performed offline, the *F1TENTH* platform does not specify any limitations, therefore any global planner may be used to compute a racing line. An example of such a planner is the one presented in [Heilmeier et al., 2020], which computes a minimum curvature path iteratively solving a Quadratic Programming (QP) problem and then calculates a velocity profile for such a trajectory using the longitudinal and lateral limits of the car. This work was deployed on the *Roborace*, similarly to [Fabian Christ and Lohmann, 2021], and instead proposes an optimization program that minimizes lap time and approximates the vehicles’ behaviour with a single track model. For ease of tuning and the higher versatility given by the smoother minimum curvature racing line, in this work, we deployed and tested the method of [Heilmeier et al., 2020], nevertheless having the minimum time

implementation also integrated.

On top of a global planner, a local planner can then be employed to incorporate obstacle-avoidance capabilities. Due to the online nature of such algorithms, the constraints of the platform need to be considered, making SotA algorithms harder to deploy directly, such as the graph-based planner presented in [Stahl et al., 2019a], that describes how a lattice can be used to search for different behavioural strategies (overtake left-/right, follow, go straight) in the context of *Roborace*, or the Model Predictive Control (MPC) presented in [Wischniewski et al., 2023], that incorporates obstacle avoidance by modifying the reference racing line and boundaries used by the controller, deployed in the context of the IAC. In the context of *F1TENTH*, MPC solutions for obstacle avoidance were considered, for example in [Zhu et al., 2022], where Gaussian Processss (GPs) are used to predict the opponent trajectory and incorporated into the MPC formulation, or also in [Bulsara et al., 2020], where an MPC is used to follow a trajectory generated around an obstacle by an external planner. However, both methods differ greatly from the setup of this work, as both use very low velocities or testing spaces ([Zhu et al., 2022] uses a max velocity of 2.8 m s^{-1} and [Bulsara et al., 2020] uses a 4.85 m by 3.5 m space). Furthermore, the obstacle was either perceived through a motion caption system in the former case or static as in the latter, and computational requirements were evaluated offline. To ease the computation of MPC controllers, also [Heetmeyer et al., 2023] proposes a small-batch parallel gradient descent optimization strategy, to handle the non-linear model and non-convex constraints of an *F1TENTH* setting, which is still quite different in the final velocity and setting tested on the real platform (around 2.4 m s^{-1}). Planning for obstacle avoidance on *F1TENTH* platforms can be implemented through more standard techniques, such as with *Frenet* planners, as in [Raji et al., 2022] in the context of the IAC. An accelerated version for NVIDIA CUDA platforms was described in [Muzzini et al., 2023] and this version can be deployed on an *F1TENTH* platform, e.g. when the main computation board used is the NVIDIA Jetson Xavier NX. This work, however, demonstrates fully real-time collision avoidance and overtaking trajectory generation using online detections and estimations of the opponent through onboard computing and sensing.

Control: Similarly to the considerations done for the local planners, the best performing MPC algorithms used in high-performance autonomous racing (e.g. [Wischniewski et al., 2021, Vázquez et al., 2020b]) yield lower performances when constrained by the computation limits of *F1TENTH* platforms. For example, the work of [Jain and Morari, 2020] proposes to use an MPC with car dynamics learned with the use of GPs. However, the test is carried out only in the context of the *F1TENTH* simulator, and the MPC solve time yielded a $\sim 4 \text{ Hz}$ frequency. Another MPC strategy is presented in [Wang et al., 2021], where the authors propose to learn the model dynamics via data-driven Deep Koopman Representations for Control. The resulting maximum speed is however not exceeding 3.5 m s^{-1} while the computational platform to carry out such an algorithm is unclear from the source. The work in [Alcalá et al., 2020] deploys a Linear Parameter Varying (LPV) MPC on a 1:10 scaled platform, but the computation is carried out on a remote computer and the localization is done by means of an indoor positioning system. To avoid the complex requirements in parameter identification and computation complexity of MPC, researchers have often resorted to ML techniques, often in the context of Reinforcement Learning (RL) [Ghignone et al., 2023, Brunnbauer et al., 2022, Evans et al., 2023]. These methods however suffer from the sim-to-real gap, and the performance they achieve in simulation is either not tested on hardware (as in [Evans et al., 2023]) or tested at final speeds lower than those of simulation (as in [Brunnbauer et al., 2022, Ghignone et al., 2023]).

The most performant techniques that have been deployed on *F1TENTH* vehicles are closer to classical techniques than to MPC or RL. A first example is the one in [O'Kelly et al., 2020a], where a *Pure Pursuit* algorithm is deployed on tracks that reach up to 7 m s^{-1} and achieves lap times up to 21% faster when compared to expert solutions on real-world race tracks. The controller that is tested on the highest velocities is the one presented in [Becker et al., 2023], where a dynamic model with *Pacejka* tire formulas is used to extend the only geometric properties of *Pure Pursuit*, reaching significantly faster laptimes at tested speeds up to 11 m s^{-1} . This last controller is also the one used in this work, with only minor modifications. A further controller used predominantly in *F1TENTH* competitions is the Follow The Gap (FTG) controller [Özdemir and Sezer, 2017, Sezer and Gokasan, 2012], a reactive method that processes directly the 2D-LiDAR scans and steers the car towards the largest available cone of free space, without relying on neither *State Estimation* nor *Planning*. This type of controller, while avoiding obstacles directly as a result of its reactivity, is unfit

for higher speeds, and can only drive in a circuit if a single closed trajectory is present.

Source	Modules (Details)	Onboard Localization?	Onboard Computation?	Testing Constraints
[Brunnbauer and Bader, 2019]	Localization, (Camera)	Yes	Yes	max size: 2 m × 2 m
[Konstantinidis, 2020]	Detection, Tracking	Yes	Yes	max size: 4 m × 4 m, max velocity: <3 m s ⁻¹
[Bulsara et al., 2020]	Planning, Control (MPC)	No	No	max size: 5 m × 3.5 m, max velocity: N/D, constant
[Zhu et al., 2022]	Local Planning, Control (MPC)	No	No	max velocity: <3 m s ⁻¹
[Alcalá et al., 2020]	Control (MPC)	No	Yes†	max size: 7 m × 7 m, max velocity: <2.8 m s ⁻¹
[Brunnbauer et al., 2022]	Control (RL)	Yes	Yes	max velocity: <5 m s ⁻¹
[Ghignone et al., 2023]	Control (RL)	Yes	Yes	max velocity: <3 m s ⁻¹
[O’Kelly et al., 2020a]	Planning, Control (<i>Pure Pursuit</i>)	Yes	Yes	max velocity: <7 m s ⁻¹
[Kabzan et al., 2020]	Full Stack	Yes‡	Yes‡	N/A
[Betz et al., 2023]	Full Stack	Yes‡	Yes‡	N/A
Ours	Full Stack	Yes	Yes	max size: 30 m × 10 m, max velocity: <11 m s ⁻¹

Table 2: Comparison of previous works in the context of *F1TENTH*. Maximum size under the testing constraints is indicated to quantitatively assess the difference between the previous works test setups with the one of an official *F1TENTH* competition (ca. 30 m × 10 m), as detailed for example at this link, available at the time of writing, 31st January 2024: https://icra2023-race.f1tenth.org/orientation_2.html. N/D: No Data. †: Computation in [Alcalá et al., 2020] is not carried out onboard, but the computing platform, using an Intel Core i7-8850U and no Graphics Processing Unit (GPU), is comparable in power to the *F1TENTH* setup. ‡: [Kabzan et al., 2020], [Betz et al., 2023] are not deployed in the context of *F1TENTH*, but are included to highlight the previous works in the context of autonomous racing that propose full software stacks. N/A: Not Applicable.

3 System Overview

This section introduces the hardware components of the racecar in Section 3.1. Subsequently, the design philosophy behind the proposed *ForzaETH Race Stack* is presented in Section 3.2, emphasizing the interaction among individual autonomy modules and their collective contribution to the overall racecar architecture. Finally, the robotic conventions utilized within this work are detailed and defined in Section 3.4.

3.1 F1TENTH Hardware Architecture

The foundation of the proposed robotic platform is the *Traxxas TRAX68086-4FX* Radio-Controlled (RC) racecar, as suggested by the official *F1TENTH* bill of material. It is reduced to its core parts, namely the frame, the axles with tires and suspension, the actuator as well as front and rear bumpers. Over this foundation, an acrylic plate is mounted and acts as an even level for the mounting of the autonomy parts of the platform. Onto this platform, a LiDAR, the On Board Computer (OBC), the Vedder Electronic Speed Controller (VESC), acting as the motor controller, with its built-in Inertial Measurement Unit (IMU) and

the power distribution board are mounted. This setup can be seen in Fig. 2 while the hardware components are listed in Table 3.

Component	Manufacturer	Model
Tires	ARRMA	Dboots Hoons 42/100 2.9 Belted Tires Gold
Suspension	Traxxas	Rustler VXL Aluminium Shocks
LiDAR	Hokuyo	UST-10LX
VESC	Trampa Boards Ltd.	VESC 6 MkIV
IMU	Bosch	BMI160
Actuator	Velineon	3500 Brushless Motor
OBC Device	Intel	Next Unit of Computing (NUC) 10
OBC CPU	Intel	Core i5-10210U
OBC RAM	Corsair	Vengeance 32 GB 3200 MHz
Power Board	Murata Power Solutions	UWE-12/10-Q12N-C
LiPo Battery	Traxxas	2827X
Microcontroller Unit (MCU)	Arduino	Micro

Table 3: Detailed list of the key hardware components utilized in the *F1TENTH* autonomous racecar. This table enumerates the type of the components, their respective manufacturers as well as the specific model of the component.

To power the racecar, a Lithium Polymer (LiPo) battery with a battery capacity of 5000 mAh is utilized and feeds power into the VESC as well as the power board. The board regulates the battery voltage to a stable 12 V and delivers power for the Intel NUC OBC and the LiDAR sensor.

Range sensing is achieved by a *Hokuyo UST-10LX* LiDAR system, which employs Time-of-Flight (ToF) technology to measure the interval between the emission of a laser beam and the reception of the reflected signal. LiDAR is an active exteroceptive ranging modality, sweeping a laser beam 270° around the scene while ranging up to 10m at 40 Hz in case of the used model. The capabilities of the utilized LiDAR, as well as comparisons with other range sensors in the context of autonomous racing are further explored in the work of [Loetscher et al., 2023].

The racecar’s orientation and motion are tracked by the IMU, analyzing rotational and linear movements using an accelerometer and a gyroscope to determine the 3D-orientation with 6 Degrees of freedom (DoF) at 50 Hz, as this is the same frequency at which the Electric Revolutions Per Minute (ERPM) information is sampled from the VESC motor controller.

The propulsion of the racecar is powered by a *Traxxas Velineon 3500* brushless motor. This motor is characterized by a Kv rating of 3500 RPM/V and achieves maximum power at 300 W. The golden *DBoots Hoons Belted Tires* from *ARRMA* are used. The challenge of slippage on various racetrack surfaces necessitates prioritizing tire traction rather than speed maximization. The golden version of the tire model is chosen for its pronounced siped tread pattern, promising the best traction among the models. The suspension is composed of the *Traxxas Rustler VXL Aluminium Shocks* in the low Center of Gravity (CG) configuration. These shock absorbers are stiff, leading to enhanced stability of the racecar and consequently more predictable handling on smooth surfaces, which is the typical condition in *F1TENTH* races. The stability of the racecar also simplifies the acquisition of sensor readings from the LiDAR and IMU as no explicit roll and pitch compensation is needed.

To visualize the state or other telemetry data, a system using an *Arduino* MCU indicates the data on an Light Emitting Diode (LED) ring, mounted on a sombrero hat. A magnetically detachable socket situated atop the OBC ensures secure placement of the components.

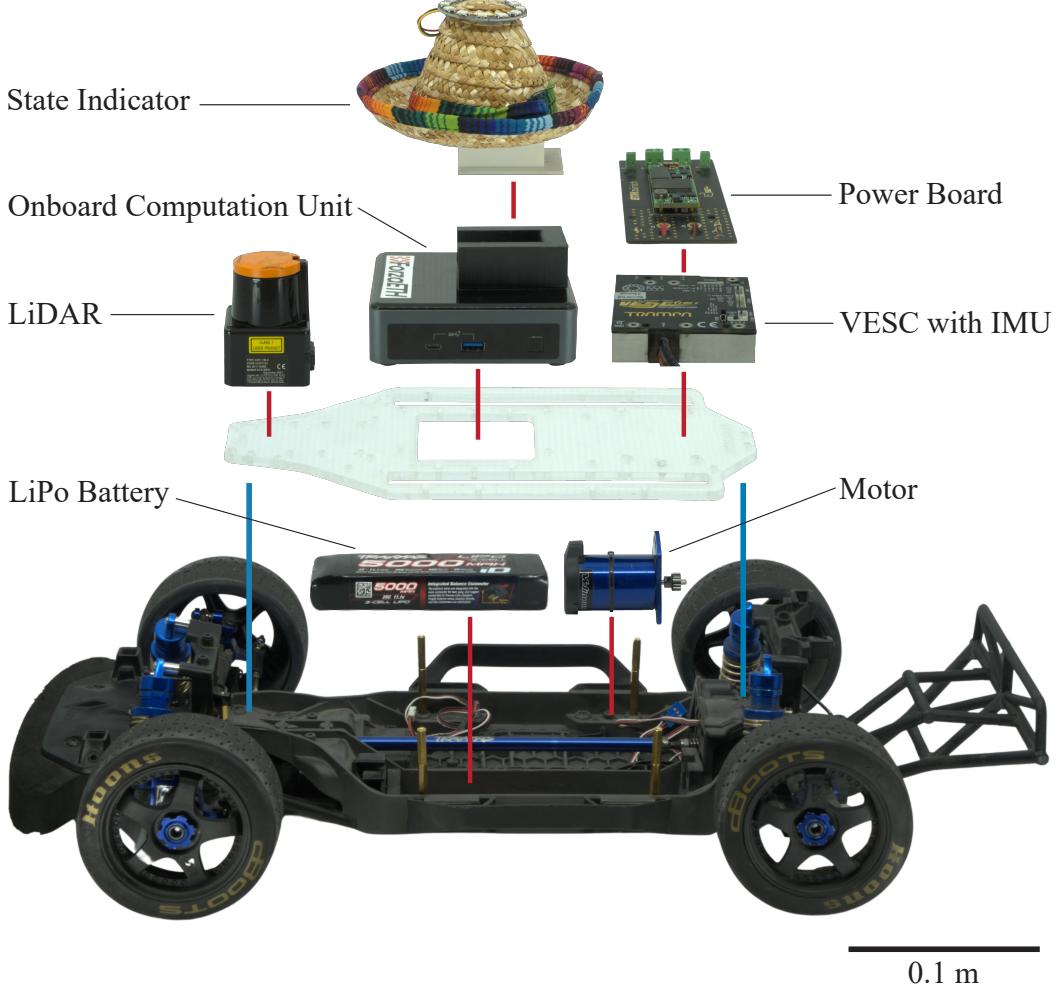


Figure 2: Comprehensive overview of the autonomous racing platform’s hardware architecture. This figure presents an exploded view of the racecar, highlighting its key components as well as their integration.

3.2 ForzaETH Race Stack Architecture

Fig. 3 illustrates the *ForzaETH Race Stack*, a software architecture developed to implement the *See-Think-Act* paradigm [Siegwart et al., 2011], ensuring a structured and coherent approach to autonomous racing. The architecture emphasizes the interaction and connectivity among various autonomy modules and hardware components, aiming to ensure efficient vehicle operation in a *Head-to-Head* racing environment.

The architecture not only describes the functionalities of each module but also illustrates how upstream tasks can have cascading effects on subsequent autonomy modules, which is vital in scenarios demanding real-time decision-making and adaptability. For instance, even with an optimal controller, the robot will not operate effectively if its upstream state estimation task performs poorly. Thus, each part of the pipeline is meticulously adapted and configured to work holistically within the *ForzaETH Race Stack*.

While the architecture of Fig. 3 draws inspiration from the work presented in [Betz et al., 2022, Siegwart et al., 2011], it emphasizes the criticality of state estimation as a standalone module, underscoring its significant influence on all downstream autonomy modules.

The depicted autonomy modules of the race stack have been fully implemented in Robot Operating System

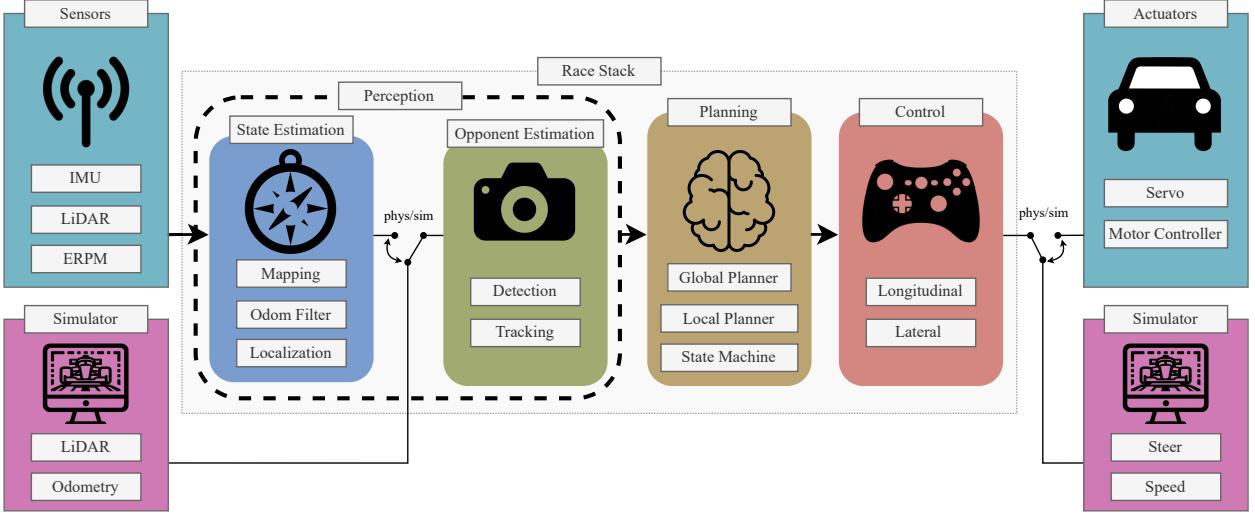


Figure 3: Architecture overview of the proposed *ForzaETH Race Stack* following the *See-Think-Act* paradigm. It highlights the interplay and interconnectivity of autonomy modules and hardware and illustrates how upstream tasks have knock-on effects that influence the subsequent autonomy modules. This depiction is inspired by [Betz et al., 2022], but emphasizes the importance of *State-Estimation* for autonomous racing, as a standalone autonomy module within *Perception*. Further, it is depicted how the *ForzaETH Race Stack* can switch seamlessly between the physical robot and the simulation environment.

(ROS)1 *Noetic* using **Python** and **C++**. At the time of writing an additional ROS2 *Humble* version is being implemented for future-proving and open-sourcing. From a design philosophy point of view, the following is adhered to: **C++** where necessary, **Python** where possible. Latency critical nodes necessitate the speed and performance of **C++**, while the development simplicity and efficiency of **Python** can be leveraged otherwise. Using the ROS ecosystem, many open-source robotics tools, algorithms, and sensor-drivers can be purposed for the race stack.

3.3 Simulation Environment

A minimalistic and lightweight ROS simulation environment is utilized within the development of the proposed race stack. The simulator is slightly modified from the original *F1TENTH* simulation environment [O’Kelly et al., 2019]. The interfacing between the simulator and the physical system has been designed to be identical, such that seamless switching between the simulation and the physical system is enabled. The simulation model corresponds to a dynamic bicycle model, which can be selected to use either linear or *Pacejka* tire dynamics, as in [Althoff et al., 2017]. When using the race stack in the simulation, the architectural overview of Fig. 3 is nearly identical, with the exception that *State-Estimation* is replaced with the ground-truth state forwarding from the simulator, as well as the sensors and actuators being provided by the simulator as well. This simulation environment allows for testing, verifying logic, and executability throughout the development process. Yet, due to the simulator’s lightweight and simplicity, a considerable *Sim-to-Real* gap exists, hence extrapolation of racing performance to the physical domain is not advisable, and the simulator is mostly advised for debugging purposes [Zhang et al., 2024].

3.4 Robotic Conventions

Within this work, the ROS right-hand-rule coordinate convention is utilized, as in ROS Enhancement Proposal (REP)-103, following convention and units as in [Foote and Purvis, 2010]. The coordinate frames, as depicted in Fig. 4a, are `map`, `base_link`, `imu`, and `laser`. The body frame attached to the car is the

`base_link` frame, situated in the middle of the car's rear axle. Two sensor frames are then also rigidly attached to the car, the `laser` frame, located at the laser sensor, and the `imu` frame, located at the VESC's position, where the IMU is located. Rigid transformations link these frames to the body frame, and, whenever necessary, the sensor data is transformed to the `base_link` frame before being used. The `map` frame is the inertial frame of reference and, in this frame, (x, y) represent the positional and *Cartesian* coordinates of the car, and (s, d) represent the *Frenet* coordinates, as explained in Section 3.4.1.

The terms v_x and v_y denote the longitudinal and lateral velocities, respectively, in the `base_link` frame, whereas v_s and v_d , respectively denote the components of the velocity tangential and perpendicular to our reference trajectory, i.e. the velocity in *Frenet* coordinates. To distinguish between the ego vehicle and an opponent vehicle, the subscripts *ego* and *opp* are used respectively. For example, the tangential velocity of the ego and the opponent vehicle are $v_{s, ego}$ and $v_{s, opp}$. When this subscript is omitted, *ego* is assumed.

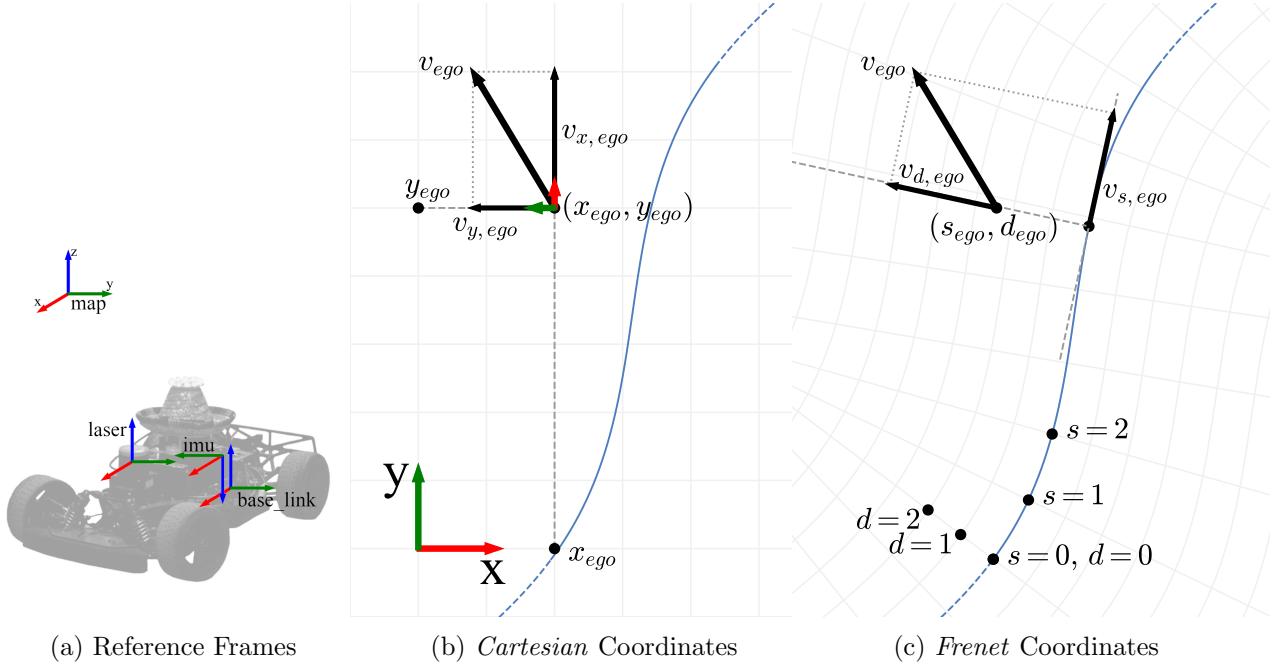


Figure 4: In Fig. 4a, the frames of reference used for the *ForzaETH Race Stack* are shown. The inertial `map` frame is the reference frame for the global *Cartesian* coordinates used across this work. The other frames are rigidly attached to the car, with the body frame `base_link` at the center of the rear axle, the two sensor frames `laser` and `imu` at the center of the respective sensors. A representation of the two used coordinate systems is further shown in Fig. 4b and Fig. 4c, with a reference trajectory in blue. In Fig. 4b, the origin is represented by the red and green arrows, as it corresponds to the inertial `map` frame. The body frame `base_link` is further represented with a pair of red-green arrows, centered in the position of the car. The car's velocity is further represented in both the body frame, in Fig. 4b, and in *Frenet* coordinates in Fig. 4c. Fig. 4c further shows the *Frenet* coordinate system's axis of the *Cartesian* system, with the origin and a few example points on the reference axes shown in black to facilitate the reader.

Within this work robotic naming conventions such as `scan`, `pose`, and `odom` are used. These conventions adhere to ROS standards and are defined as in Table 4.

3.4.1 Frenet Frame Adoption

A central feature of the proposed race stack is its extensive utilization of the curvilinear *Frenet-Serret* frame, as detailed in [Werling et al., 2010, Vázquez et al., 2020b]. The *Frenet* frame establishes a coordinate system relative to a designated reference path. In the context of our system, this reference path corresponds to

the global racing line, further discussed in Section 6.2. Consequently, *Cartesian* coordinates (x, y) can be mapped to *Frenet* coordinates (s, d), where s denotes progression along the racing line, and d signifies the orthogonal distance from the racing line. It is noteworthy that the s coordinate is cyclical, wrapping upon completing a lap, necessitating careful management of the path wrapping. The d coordinate is defined such that values to the right of the racing line are negative, while those to the left are positive. A representation of the car’s position in both *Cartesian* and *Frenet* coordinates is available in Fig. 4.

The adoption of the *Frenet* frame offers several advantages in specific contexts. Tasks such as determining a point’s position relative to the racetrack, describing motion models in relation to the reference racing line, or generating potential evasion waypoints that align with the racing line are considerably simplified within the *Frenet* frame compared to the *Cartesian* system. Succinctly, any computation involving spatial coordinates relative to the racing line benefits from the *Frenet* frame transformation. As such, the *ForzaETH Race Stack* places significant emphasis on simple and efficient transformations between *Cartesian* and curvilinear coordinates, ensuring seamless and simplified operation.

Symbols	Description	Contained Symbols	Corresponding ROS message
<code>scan</code>	Array of 2D range data from the LiDAR in the <code>laser</code> frame.	<code>ranges</code> []	<code>LaserScan.msg</code>
<code>imu</code>	Orientation, angular velocities ω_a , and linear acceleration a_l in <code>imu</code> frame.	$q_x, q_y, q_z, q_w,$ $v_{ax}, v_{ay}, v_{az}, a_{lx},$ a_{ly}, a_{lz}	<code>Imu.msg</code>
<code>pose</code>	Position and orientation <code>map</code> frame, using quaternion notation.	$x, y, z, q_x, q_y,$ q_z, q_w	<code>Pose.msg</code>
<code>odom</code>	Position, orientation in <code>map</code> frame, using quaternion notation and velocities in <code>base_link</code> frame.	$x, y, z, q_x, q_y,$ q_z, q_w, v_x, v_y, v_z	<code>Odometry.msg</code>

Table 4: Robotic naming conventions used within this work. Positions are indicated with x, y, z and quaternions are indicated with qx, qy, qz, qw .

4 State Estimation

Accurate and robust state estimation is essential in autonomous mobile robotics, particularly in high-performance scenarios such as autonomous racing. Within the *See-Think-Act* cycle illustrated in Fig. 3, the *State Estimation* module is the foundational element that precedes and influences crucial downstream tasks including *Opponent Estimation*, *Planning*, and *Control*. The quality of state estimation data directly impacts the extent to which a racecar’s performance can be optimized, pushing it to its physical limits. Therefore, precision in state estimation is not just a technical requirement; it is a critical factor that determines the racecar’s overall performance and safety [Betz et al., 2022, Lim et al., 2024].

In this chapter, we detail the methodologies and algorithms employed in the *ForzaETH Race Stack* to extract accurate pose (position and orientation) and longitudinal velocity from the raw sensor readings. This involves sensor fusion of data from LiDAR, IMU, and wheel-odometry (obtained through ERPM data) to achieve a reliable and high-fidelity representation of the vehicle’s state, which is indispensable for executing complex racing maneuvers at the car’s limit of friction.

4.1 Architecture

The task of state estimation is divided into localization and velocity estimation. The pipeline is depicted in Fig. 5. The sensor inputs used are linear accelerations and angular velocities from the IMU, a 2D laser scan from the LiDAR, and ERPM obtained wheel-odometry from the VESC motor controller. The ERPM

odometry is computed through the measured current and voltage within the VESC motor controller and it is combined with the commanded steering angle to estimate the wheel odometry using the implementation of the *F1TENTH* platform presented in [O’Kelly et al., 2020b], which can be significantly affected by tire-slip.

Within the *State Estimation* module, this odometry signal is then fused with IMU data in the *Odom Filter* module (more details in Section 4.2). This first filtering step is crucial, as not only are sufficiently accurate velocity estimates required for the control algorithm but [Lim et al., 2024] has also shown that later localization modules are significantly sensitive to the accuracy of the odometry prior. The filtered odometry signal is then fed to the localization algorithm, which is either the SLAM-based *Cartographer* [Hess et al., 2016] or the MCL-based *SynPF* [Lim et al., 2024]. The localization algorithms are mutually exclusive and both of them are described in Section 4.4 and evaluated later in Section 4.5. The reason for having two mutually exclusive methods for localization is that both techniques have fundamentally different operation characteristics. The *SynPF* MCL-based localization tends to be more robust towards wheel-slippage than the *Cartographer* SLAM-based approach, yet *Cartographer* performs smoother and more accurately under nominal conditions [Lim et al., 2024]. Having both methods available underscores the strategic advantage of the *ForzaETH Race Stack*, enabling adaptability and optimization of race strategy based on specific track conditions and requirements. Lastly, the final car state is aggregated merging the localization pose from *Localization* and the velocity signal from *Odom Filter*. Localization is carried out in a pre-mapped racetrack, and the mapping procedure is further described in Section 4.3.

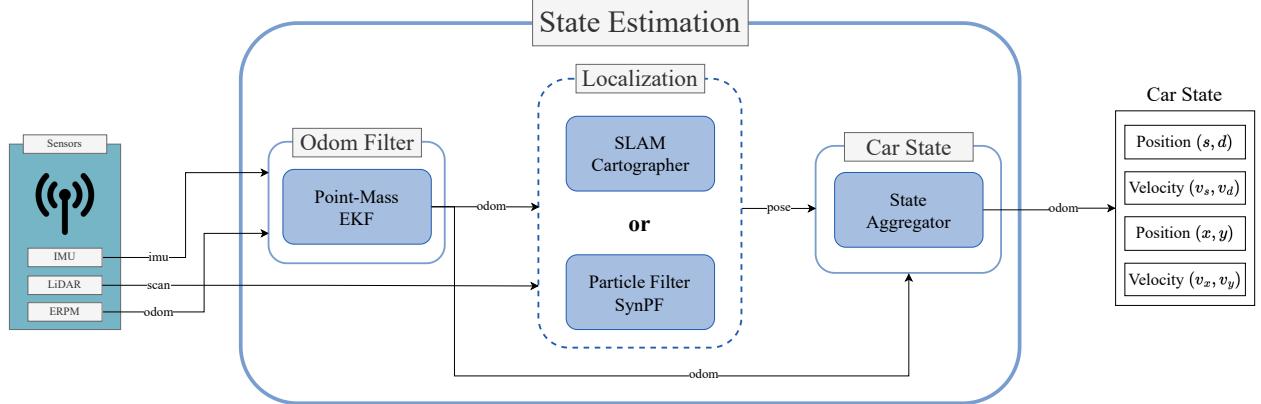


Figure 5: An overview of the proposed state estimation system architecture. The state estimation module incorporates velocity estimation and localization with respect to the pre-mapped racetrack and aggregates this information in a final car state odometry output both in Cartesian and in *Frenet* coordinates.

4.2 Odometry Filter - Extended Kalman Filter

Velocity state estimates are generated using an approach based on the EKF [McElhoe, 1966, Moore and Stouch, 2014]. It fuses the wheel odometry and the data provided by the IMU sensor. This aids the longitudinal and lateral velocity estimation for the robot in situations where tire slip occurs, as the accelerometer data from the IMU can be leveraged to compensate for this within the EKF. The EKF is integrated through the `robot_localization` package [Moore and Stouch, 2014]. The estimates are generated at a rate of 50 Hz, corresponding to the IMU and wheel-odometry update frequency.

The EKF model consists of an omnidirectional, three-dimensional, point-mass motion model. The state X and the discrete-time transfer function $f(X)$ used in the library are defined in the Appendix A.2. To account for the fact that the *F1TENTH* setup can be considered two-dimensional, the `two_d_mode` parameter is then set to `true`, which effectively enforces the measurements of the states $z, \phi, \theta, \dot{z}, \dot{\phi}, \dot{\theta}, \ddot{z}$ to zero, the respective covariances to 10^{-6} and the states to 1 (which have then to be neglected).

The remaining states that are then used by this approach are the longitudinal velocity, lateral velocity, and

the yaw rate. However, if no lateral acceleration data is supplied to the motion model, the vehicle’s lateral velocity is assumed to be 0 m s^{-1} , which is equivalent to assuming that the vehicle experiences no lateral slip. While more vehicle-specific motion models have been investigated, this approach was observed to be more robust to significantly varying track conditions than the implemented vehicle models and performed sufficiently well under realistic driving conditions.

As outlined in the documentation of the `robot_localization` package [Moore and Stouch, 2014], the a posteriori update step of the EKF can be configured by selecting the specific measurement sources from each sensor or input to be considered. The fusion configuration is determined experimentally using ground-truth motion capture data and qualitative observations, as described in Section 4.5. In the selected configuration, the EKF fuses the IMU measurements of angular velocity, heading, and the full twist stemming from the wheel odometry. Note that linear accelerations observed by the IMU sensor are not considered (linear acceleration of the ERPM-based wheel-odometry is however used), as they were observed to experience significant noise due to shaking and may be sensitive to the exact positioning and orientation of the IMU sensor on the car. Nevertheless, the configuration of EKF inputs can be changed quickly and easily to maximize the estimation accuracy given the external conditions.

The covariances associated with the individual measurements are determined at their respective sources and are used in forming the measurement update of the EKF. In this implementation, the covariances of the IMU data and the control odometry are predetermined and set to be static. Numerical values can be found in the Appendix A.1.1.

4.3 Mapping

Mapping is a fundamental part of autonomous racing, serving as the foundation for both localization and trajectory planning [Betz et al., 2022]. The proposed *ForzaETH Race Stack* uses a pose-graph optimization SLAM method, *Cartographer* [Hess et al., 2016] to create an initial map of the racetrack.

In a race setting, mapping is conducted during the free practice session. The occupancy grid produced is subsequently used for the computation of a global trajectory, detailed in section Section 6.2. The most relevant tuning parameters for *Cartographer* are listed in Appendix A.1.1.

4.4 Localization

Accurate localization is crucial to the trickle-down effects on the rest of the *ForzaETH Race Stack*. Therefore, the proposed race stack supports two mutually exclusive options for localization to suit varying conditions:

- I **Cartographer:** A pose-graph optimization SLAM method [Hess et al., 2016]. This localization method yields the most accurate and smoothest pose estimate, given high-quality odometry input data. This localization technique yields high accuracy localization up to Root Mean Squared Error (RMSE) of 0.0535 m, as long as the racecar experiences relatively low levels of tire slip, such that the wheel odometry signal is sufficiently accurate, as further demonstrated in [Lim et al., 2024].
- II **SynPF:** A high-performance MCL-based Particle Filter (PF) localization technique optimized for racing, specifically developed for the *ForzaETH Race Stack* [Lim et al., 2024]. As opposed to *Cartographer*, this method yields slightly less accurate pose estimates with higher jitter, as can be seen in Fig. 10. However, this method is highly robust against low-quality odometry input. In a racing environment with high levels of wheel slip, *SynPF* is a highly effective localization alternative. Furthermore, *SynPF* is computationally lighter, resulting in a Central Processing Unit (CPU) utilization from 30% to 50% lower as compared to the *Cartographer* counterpart, as from Section 8.1.1, Section 8.2.1.

Given the contrasting strengths of each localization method (pure performance versus robustness), human-

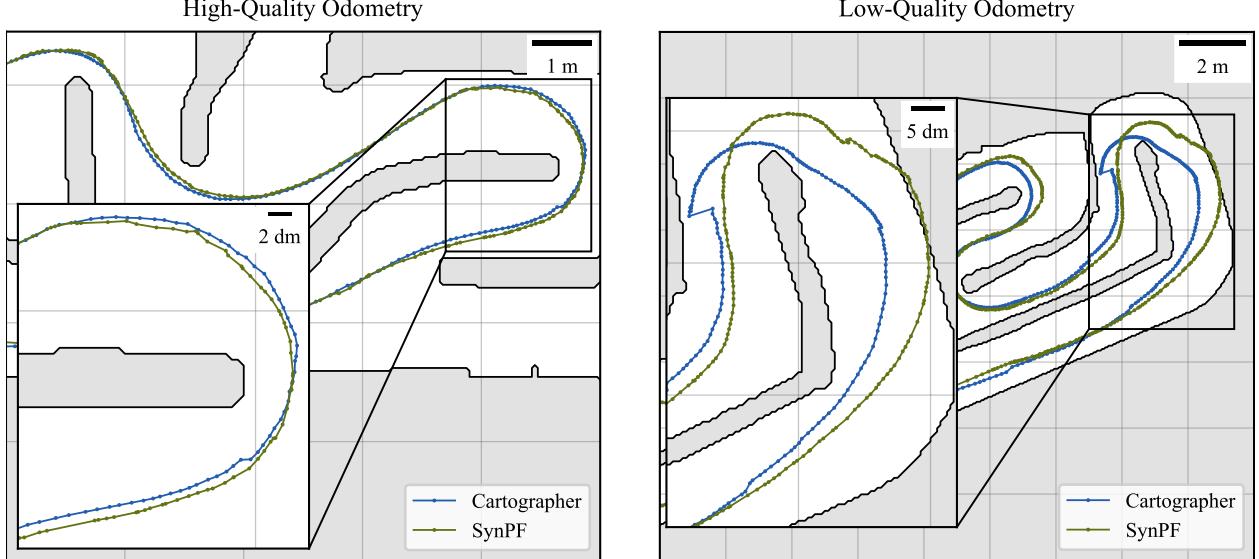


Figure 6: A comparison of the supported localization methods, *Cartographer* and *SynPF*. **Left:** Nominal conditions showing high-quality odometry. *Cartographer* is the favorite method in this case, given the smoother pose and the higher operational frequency (200 Hz versus 50 Hz for *SynPF*). **Right:** Low-grip conditions with wheel spin. *Cartographer* is unable to provide an accurate pose estimate, estimating that the racecar is in the middle of the track, while in reality, it has crashed into the boundaries, as evident from the LiDAR scans in Fig. 7. However, *SynPF* operates nominally, correctly estimating the racecar’s actual position. This comparison was made possible by retrospectively applying the *SynPF* algorithm to telemetry data recorded during the car’s operation with *Cartographer*.

operators can decide which one to use given the conditions at each race event. A qualitative example to assess the different performance characteristics between *Cartographer* and *SynPF* given high- and low-quality odometry, i.e. wheel-slip, input is illustrated in Fig. 6 and Fig. 7. Specifically, Fig. 7 demonstrates the catastrophic failure case of *Cartographer* SLAM, which could not handle the compromised odometry signal, leading to a collision against the track’s right barrier, as evidenced by the LiDAR scans. In contrast, *SynPF* was able to accurately localize using the same dataset. As in a racing scenario, one can not rely on a motion-capture system to be available, lap time is used as a proxy measurement for localization accuracy, as from a holistic viewpoint of the race stack, the improvement of localization yields overall better performance. Hence both *Cartographer* and *SynPF* algorithms were tuned to minimize the lap time in a *Time-Trials* scenario over multiple maps. The utilized parameters are listed in Appendix A.1.1.

4.5 State Estimation Results

This section evaluates the accuracy of the entire state estimation framework against ground truth data, primarily using a motion-capture system for validation. The test vehicle was equipped with reflective markers and navigated autonomously on the track depicted in Fig. 8. Positional data was recorded from a motion-capture system composed of 6 *Vicon Vero v2.2* cameras, recording position in a 4 m by 4 m space. Furthermore, we also recorded the vehicle’s sensor and control data, synchronizing measurements via the ROS timestamp. This comprehensive dataset enables offline analysis of different state estimation frameworks, including the two localization algorithms previously discussed. Fig. 8 illustrates the experimental setup. The nominal parameters described in Appendix A.1.1 were used, and data was recorded over one minute of uninterrupted driving.

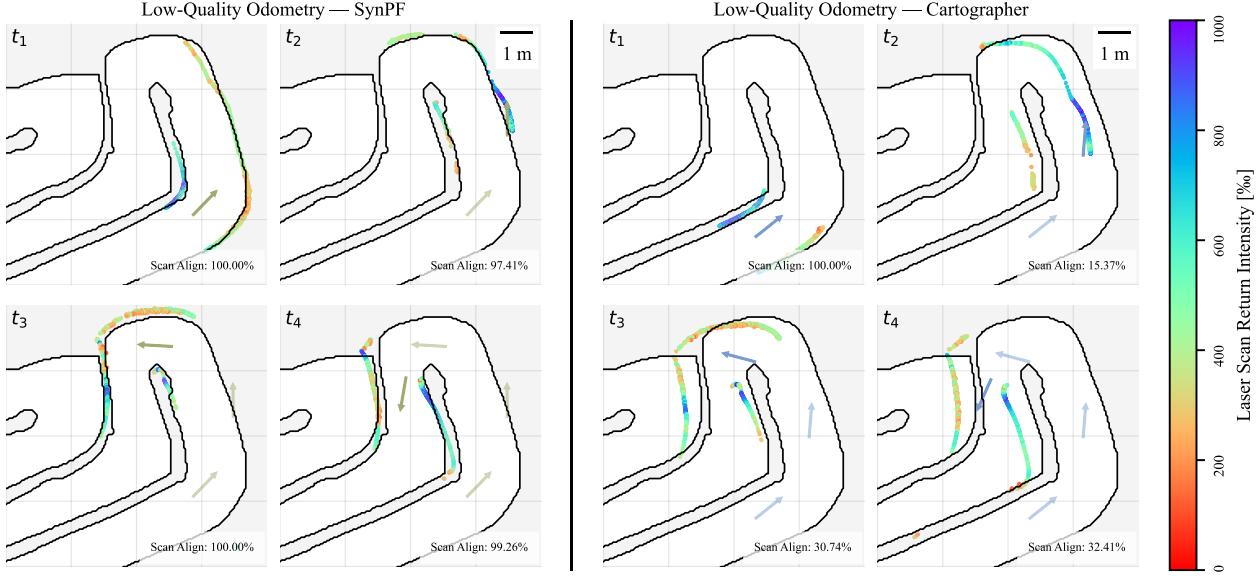


Figure 7: Low-quality odometry conditions corresponding to the right-hand side of Fig. 6 showcasing both *Cartographer* and *SynPF* under worst-case conditions in four sequential time steps, indicated with t_i . At t_2 , *Cartographer* is unable to align the LiDAR scans with the track boundaries, while *SynPF* maintains a visibly steady track alignment.

4.5.1 Localization Accuracy

Fig. 9 quantifiably shows that *Cartographer* is capable of delivering accurate pose estimates, especially with the high-quality wheel odometry inputs in the given scenario, with an average positional RMSE of 0.0535 m. However, the limited coverage area of the motion capture system necessitated a small racetrack setup. *SynPF* struggles to give accurate localization in this scenario, with a higher RMSE of 0.1998 m.

However, it should be stressed that the racetrack setup with the motion capture system is significantly smaller compared to a competition scenario. Furthermore, to assess the localization performance with respect to the quality of the wheel odometry signal, larger racetracks are needed. This is particularly necessary to capture wheel-slip effects at the edge of traction [Lim et al., 2024].

4.5.2 Velocity Estimation Accuracy

The longitudinal velocity state estimates produced by the two systems in question are depicted in Fig. 10. The algorithm was executed for approximately one minute in order to calculate performance metrics, while the trajectories were plotted for a reduced time corresponding to two laps of the test track. The RMSE over the entire testing time was then computed for both localization frameworks, as summarized in Table 5. The localization performance yields a low positional RMSE of ~ 0.08 m, on the other hand, the velocity estimation RMSE yields a rather high $\sim 0.25 \text{ m s}^{-1}$ and still leaves room for improvement. A possible reason for the worse performance of the velocity estimation is, that the velocity directly originates from the *Odometry Filter* as in Section 4.2 which fuses the information from ERPM wheel-odometry and IMU data, where the ERPM data is highly influenced by the friction of the track and can easily result in a low-quality odometry signal due to wheel-slip.

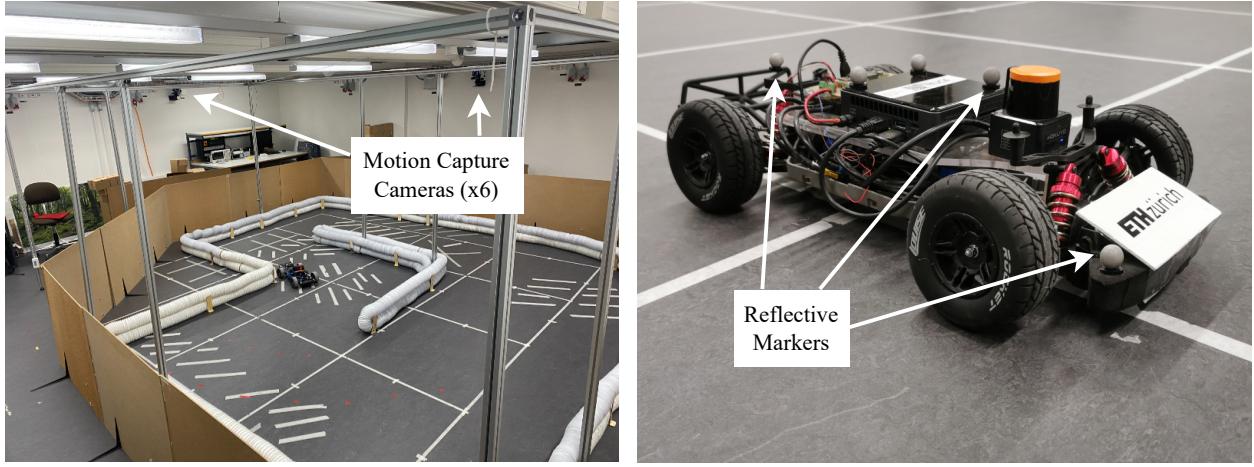


Figure 8: Experimental setup with the motion capture system and mounted reflective markers.

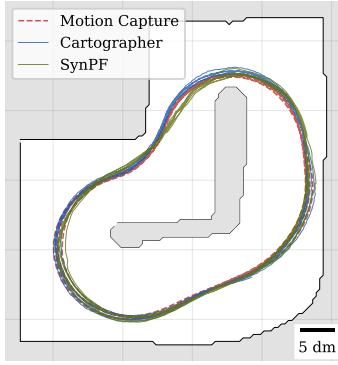


Figure 9: On the small race-track set up with the motion capture system, *Cartographer* is able to closely match the ground-truth trajectory, while *SynPF* performs slightly worse.

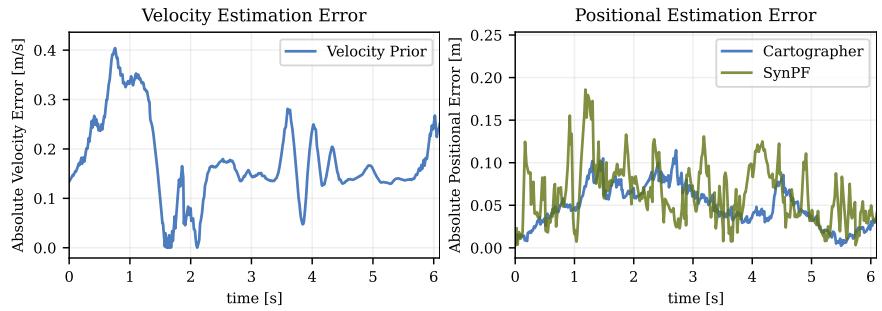


Figure 10: State estimation error curves for cartesian coordinates and longitudinal velocity corresponding to one lap around the test track. Where relevant, the two different localization techniques available are compared against ground-truth data, and the velocity and positional error are depicted for the duration of a single lap.

<i>Configuration</i>	v_{lon} RMSE \downarrow [$m s^{-1}$]	s RMSE \downarrow [m]	d RMSE \downarrow [m]	ψ RMSE \downarrow [rad]
<i>Odom Filter + Cartographer</i>	0.1970	0.0729	0.0408	0.0284
<i>Odom Filter + SynPF</i>	0.1893	0.0795	0.0725	0.0654

Table 5: State estimation accuracies of various odometry states while driving several laps around the test track as evaluated against motion capture data on the track setup outlined in Section 4.5.

5 Opponent Estimation

The *Opponent Estimation* module plays a pivotal role in consistently and precisely capturing moving objects, a capability imperative for successful planning and agile maneuvering throughout the race [Betz et al., 2022]. This module encompasses the processes of obstacle detection, classification, and tracking, extending to the computation of the opponent’s position and velocity during the *Head-to-Head* phase of the race.

5.1 Architecture

The opponent estimation module requires the precise localization of the racecar in relation to the race track, as well as the raw sensor readings of its main exteroceptive sensor, the LiDAR. Extensive trials and observations of the *ForzaETH Race Stack* under racing conditions have underlined the necessity of this module for critical downstream robotic tasks, such as trailing closely behind an opponent, where it ensures an uninterrupted and precise opponent estimation. This becomes even more vital when the opponent is out of Line of Sight (LoS), potentially hidden behind a curve, as it prevents unnecessary braking or potential collisions due to misjudgment of the opponent’s position. To meet these challenges, the opponent estimation architecture is designed with the following key objectives:

- I **High Precision Detection:** The detection submodule is designed to achieve a high True Positive Rate (TPR), ensuring consistent opponent detection whenever they are within LoS. This has been evaluated to be 96.8%, as in Section 5.2.1.
- II **Low False Detection Rate:** The detection submodule is optimized to minimize the number of False Discovery Rates (FDRs), enabling the racecar to maintain high speeds without unwarranted phantom breaking due to False Positives (FPs). This has been evaluated to be 1.6%, as in Section 5.2.1.
- III **Continuous Opponent Estimation:** The tracking submodule is tasked with providing a continuous estimation of the opponent’s position and velocity, even in the absence of LoS. The position and velocity estimation of the opponent has been evaluated to yield an RMSE of 0.17 m and 0.49 m s⁻¹ respectively, as in Fig. 12.
- IV **Low Latency:** The entire perception module is streamlined for efficiency, aiming to minimize latency and uphold reliable detection and tracking at high velocities. The latency has been evaluated to be 6.39 ms as later evaluated in Fig. 33.

5.1.1 Opponent Detection

The detection process begins with acquiring the complete 2D LiDAR scan data from the current LiDAR sweep. The data is then segmented into smaller clusters representing potential obstacles using an *Adaptive Breakpoint* method [Amin et al., 2022]. This method is based on the premise that objects on the track are formed by consecutive LiDAR points, and it segments the measured points of the LiDAR by identifying gaps between objects that exceed a threshold distance.

Following segmentation, the point clusters are filtered to ensure a low FP operation. By utilizing the known positions of the track boundaries and the ego-localization from state estimation, the algorithm eliminates clusters that represent the track itself. This is achieved by inflating the track boundaries by a predefined distance, which is a parameter that can be adjusted to minimize FP. This form of track filtering, represents a simple computation, as the curvilinear *Frenet* transformation allows for thresholding of the d coordinate. Additionally, clusters with an insufficient number of LiDAR points are discarded to further reduce the likelihood of FPs caused by LiDAR reflections.

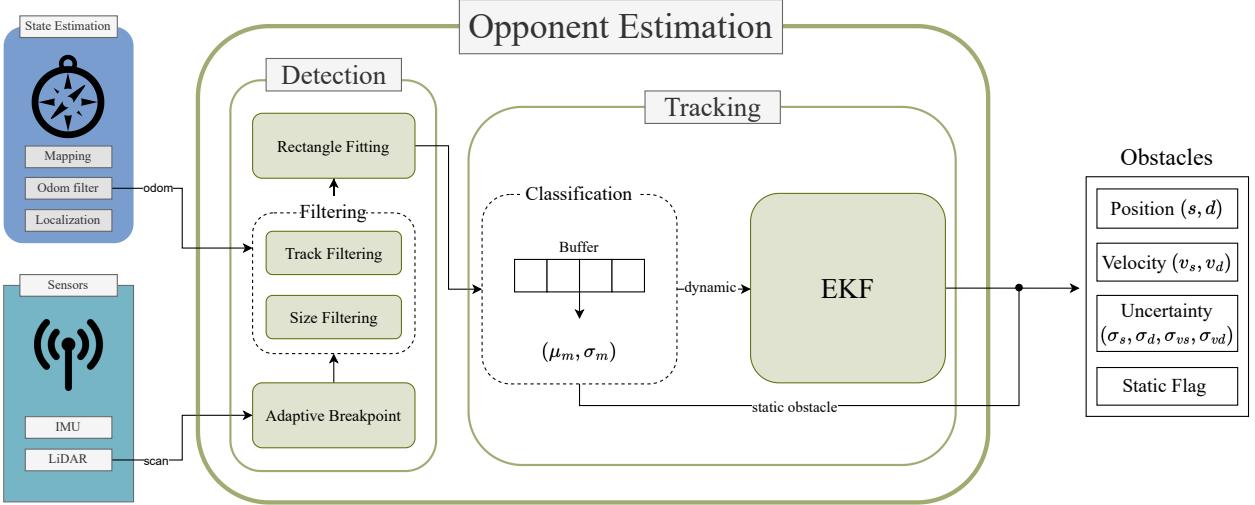


Figure 11: Overview of the proposed opponent estimation system architecture. The opponent estimation module incorporates the detection and tracking submodules and is integrated into the *ForzaETH Race Stack*, as depicted in Fig. 3.

The remaining clusters are then approximated with rectangles, on the 2D map space, facilitating better feature extraction for subsequent processes. These features, including the size, center coordinates, and rotation of the rectangle, are crucial for the planning algorithm. A final filtering step based on the estimated size of the obstacles, i.e. removal of objects that are smaller than a given threshold, ensures a further reduction in FDR, thus yielding an efficient detection with high TPR (97%) and low FDR (2%). The used parameters can be found in Appendix A.1.2.

5.1.2 Opponent Tracking

Ensuring precise and continuous tracking of the opponent is crucial for downstream robotic tasks, both when closely following an opponent and also when the opponent is out of sight, such as when navigating around a corner, necessitating a reliable estimation to maintain safety and performance. To address this, a tracking submodule is incorporated, utilizing a racing line-based motion model, i.e. assuming that the opponent progresses along the racing-line potentially with a lateral offset. The model is implemented based on the fitted obstacle rectangles and propagated within an EKF framework to offer consistent opponent estimation, regardless of LoS.

The tracking sub-module classifies detected objects as static or dynamic based on their temporal movement patterns. A voting system, which leverages the standard deviation of recent positional data, aids in this classification process. For static obstacles, their position is deduced from the average of recent (s_{opp}, d_{opp}) coordinate measurements. On the other hand, dynamic obstacles, such as the opponent's car, are tracked using an EKF, as in Fig. 11. This filter processes the measured (s_{opp}, d_{opp}) position and (v_s, opp, v_d, opp) velocity, ensuring a stable estimation robust to measurement noise. As it is known that during the *Head-to-Head* phase of a race, only one opponent will be present, only a single KF instance is initialized, meaning the velocity is only estimated for a single opponent. This could however be expanded for future multi-opponent racing scenarios.

This EKF operates similarly to a constant velocity KF, but it incorporates a normalization in the residual function to accommodate the cyclical nature of the *Frenet* coordinates. The state vector $\mathbf{x}_{opp} = [s_{opp}, v_s, opp, d_{opp}, v_d, opp]^T$ and the measurement vector $\mathbf{z}_{opp} = [z_s, opp, z_{v_s}, opp, z_d, opp, z_{v_d}, opp]^T$ are defined with their respective components representing position and velocity along the track, lateral displacement, and lateral velocity.

The behavior of the KF adapts depending on the visibility of the opponent car. When the opponent is within LoS, the behavior of the opponent is assumed to keep constant velocity along the *Frenet s* dimension, and the control input $\mathbf{u}_{\text{LoS}} = [0, -d_{opp}, -v_{d, opp}]^T$ is updated accordingly. On the other hand, when the opponent is not in LoS, the control input in the prediction step of the EKF is updated in order to drive the unseen opponent to the target velocity of the racing line $v_{s, \text{target}}$, i.e. $\mathbf{u}_{\text{nonLoS}} = [(v_{s, \text{target}} - v_{s, opp}), -d_{opp}, -v_{d, opp}]^T$. When considering the d -axis dynamics, both cases are instead treated the same way, with the state of the opponent being driven to zero both positionally in s and for the velocity in v_d . This prediction model ensures a more stable estimation in scenarios where direct measurements are not available, guaranteeing a more realistic behavior in unseen track sections.

The model is described by:

$$\begin{aligned} \mathbf{x}[k+1] &= \mathbf{Fx}[k] + \mathbf{Bu}[k] + \mathbf{w}_x[k], \quad \mathbf{F} = \begin{bmatrix} 1 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 & 0 & 0 \\ P_{v_s} & 0 & 0 \\ 0 & P_d & 0 \\ 0 & 0 & P_{v_d} \end{bmatrix}, \\ \mathbf{z}[k] &= \mathbf{Hx}[k] + \mathbf{w}_z[k], \quad \mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{u}[k] = \begin{cases} \mathbf{u}_{\text{LoS}}[k], & \text{if opponent within LoS} \\ \mathbf{u}_{\text{nonLoS}}[k], & \text{else} \end{cases}, \quad (1) \end{aligned}$$

where P_{v_s} , P_d , and P_{v_d} are the proportional gains for the respective control inputs to the states v_s , d , and v_d . Furthermore, Δt is the time between two updates, $\mathbf{w}_x \sim \mathcal{N}(0, \mathbf{Q})$ is the process Gaussian noise where $\mathbf{Q} \in \mathbb{R}^{4 \times 4}$ is the covariance matrix, and $\mathbf{w}_z \sim \mathcal{N}(0, \mathbf{R})$ is the input Gaussian noise where $\mathbf{R} \in \mathbb{R}^{4 \times 4}$ is the input covariance matrix. The exact parameters used can be found in Appendix A.1.2.

5.2 Opponent Estimation Results

In the experimental setup designed to assess the performance of the perception module, the ego-agent and an opponent racecar were positioned on a racetrack, each with the *ForzaETH Race Stack* deployed. The opponent vehicle was set to maintain the same racing line as the ego-agent but at a slower pace. This configuration enables the ego-agent to trail at close proximity (using the trailing controller later described in Section 7.2.2) while running the perception module. As a result, the detection and estimation of the ego-agent's perception module and the localization information of the opponent car can be directly recorded to evaluate the accuracy of the detector and tracker.

5.2.1 Opponent Detection Accuracy

Fig. 12 illustrates the performance of the detection submodule within the depicted racetrack environment. On the left side of the figure, spatial detections are mapped out across the racetrack, captured over several rounds of trailing behind a slower-moving autonomous opponent. The opponent's self-localization served as a reference for recording ground-truth positions over time, which are highlighted as red lines. The blue dots signify the classification of the detections, determined based on the mean and standard deviation of the buffer, as outlined in Fig. 11. The green dots indicate static detections; in this particular experiment, these instances are incorrectly classified since the opponent was consistently in motion. The purple dots, on the other hand, are labeled as detection anomalies, signifying that the spatial detections significantly deviated from the opponent's ground-truth position. For this experiment, a threshold of twice the measurement standard deviation, equivalent to 0.17 m, was set to identify a detection outlier.

It is worth mentioning, that even when the detection appears proximate to the opponent's ground-truth trajectory, the detection error over time-synchronized point-couples is taken into account and a longitudinal error is therefore present. On the right side of Fig. 12, we present the quantitative analysis of the spatial and qualitative detections shown on the left. This plot illustrates the longitudinal and lateral detection

error in meters with respect to the global coordinate frame. To enhance visibility, the plot employs a symmetric linear-logarithmic scale, with the grey-shaded region representing the logarithmic domain. The red cross symbolizes respectively the average X and Y detection error $\mu_{err} = (-0.08 \text{ m}, 0.01 \text{ m})$, while the red dotted circle surrounding the mean represents the standard deviation $\sigma_{err} = 0.08 \text{ m}$. The resulting RMSE is calculated to be 0.17 m .

In this evaluation, the performance of the perception module's detection submodule was quantitatively assessed using the TPR and the FDR. The TPR, calculated as $\frac{TP}{TP+FN}$, where TP is the number of True Positive (TP) and FN is the number of False Negative (FN), reflects the submodule's accuracy in correctly detecting the opponent car. The TPR for the system was found to be 96.8% out of 198 detections. On the other hand, the FDR, calculated as $\frac{FP}{TP+FP}$, where FP is the number of FP, provides insight into the proportion of false alarms among all the positive detections made by the system. The computed FDR for our system was 1.6% out of 198 detections. Here, TP are correct detections within $2 \times \sigma_{RMSE}$ of the ground-truth, resulting in 182 TP detections, while FP are incorrect detections beyond this range, resulting in 6 FP detections, of the total 198 detections. Lastly, FN are missed detections within this range. True Negative (TN) are not applicable in this context, as it would require defining true negative events, which are not present in this detection task.

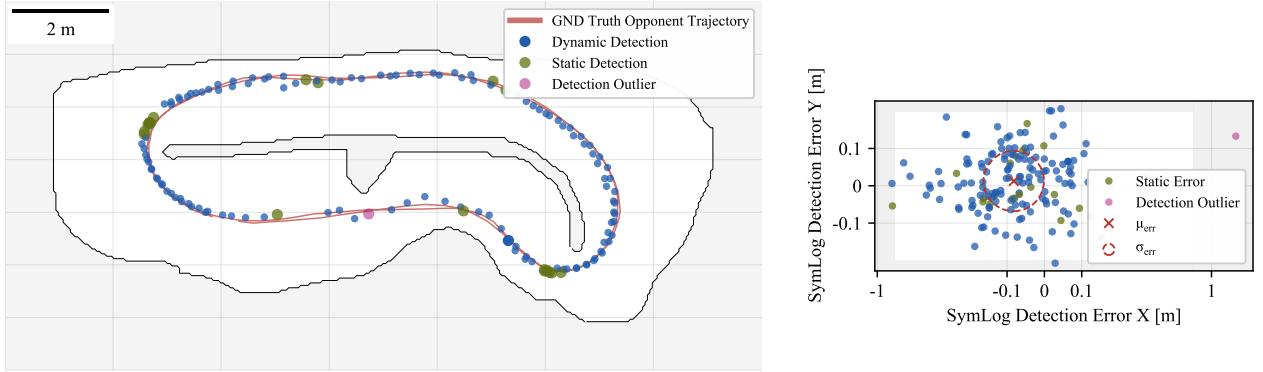


Figure 12: Evaluation of the spatio-temporal detection accuracy during multiple laps on a track, by trailing behind an opponent with the perception system active. Left: Spatial detections (blue), opponent trajectory (red), static detections (green), and detection outliers (purple). Right: Detection error in a symmetric linear-logarithmic plot; shaded regions indicate logarithmic scales and purple marks are outliers excluded from the RMSE calculation.

5.2.2 Opponent Estimation Accuracy

Fig. 13 highlights the performance of the velocity estimation capabilities, from the same experimental settings as the detection evaluations in Fig. 12. Red depicts the ground-truth velocities in (s, d) in m s^{-1} , blue is the velocity estimation with the shaded blue estimation covariances $\bar{\sigma}_{est}$ both from the EKF tracking submodule. Green-shaded represent the static misclassifications and purple-shaded the detection outliers.

The tracking submodule's estimation accuracy was evaluated through the computation of the RMSE for the longitudinal $v_{s,opp}$ and lateral $v_{d,opp}$ velocities. For non-static detections, the submodule exhibited a RMSE of 0.49 m s^{-1} for $v_{s,opp}$ and 0.37 m s^{-1} for $v_{d,opp}$, indicating a reliable accuracy in velocity estimation when the opponent car is in motion. However, when considering all detections, including static ones, the RMSE for $v_{s,opp}$ increased significantly to 1.00 m s^{-1} , while the RMSE for $v_{d,opp}$ remained relatively stable at 0.35 m s^{-1} . From this, it is visible, that the static misclassification significantly affects the longitudinal velocity estimation performance. Yet, it is to be mentioned, that during a race scenario within the context of *F1TENTH*, static obstacles within a race can be neglected, thus allowing the tracker submodule to focus only on dynamic obstacles, which enables high performance.

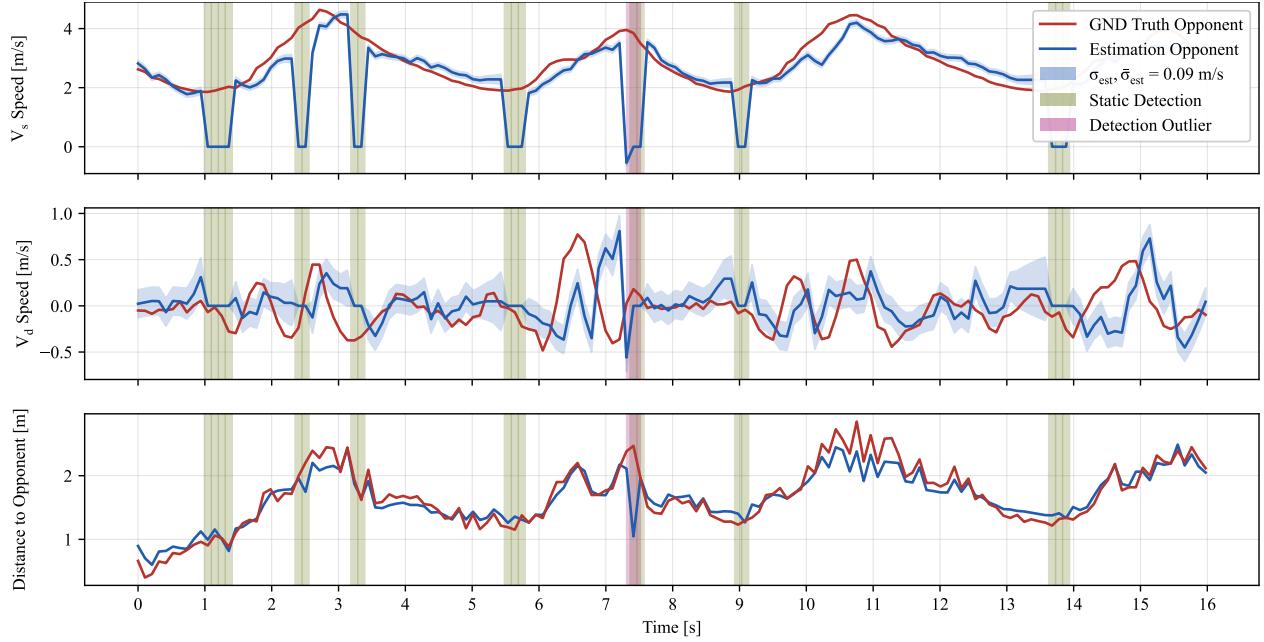


Figure 13: Opponent velocity estimation associated to the detections in Fig. 12. Ground-truth measurements of the opponent states are depicted in red, while the velocity estimation of the opponent is highlighted in blue. Green-shaded regions are static classifications of the opponent. Purple-shaded regions depict detection outliers, i.e. exceeding an error of twice the measurement standard deviation.

6 Planning

Planning is a core robotic task within the *See-Think-Act* cycle [Siegwart et al., 2011]. The planning module leverages environmental data from the *Perception* module and ego-information from the *State Estimation* module to plan efficient and effective trajectories, adapting dynamically to environmental changes. These planned trajectories will then be supplied to the downstream *Control* task for execution.

For *F1TENTH*, this translates into two primary objectives: the computation of an optimal racing line, which can be computed offline, and to react to opponents during *Head-to-Head* races. This involves strategic planning for collision avoidance and executing overtaking maneuvers, ensuring both competitiveness and safety on the track.

6.1 Architecture

Fig. 14 depicts the architecture of the *Planning* module within the proposed *ForzaETH Race Stack*. The *Planning* module is subdivided into two submodules: a global and a local planner. The global planner leverages the occupancy grid, generated during the free practice session, and yields a performant racing line for the given track. The global planner operates offline, solely attributed to the calculation of a global trajectory that is stored for later use in the race. On the other hand, the local planner employs the global trajectory as a reference racing line and, based on it, generates a local trajectory designed to avoid obstacles, and thus allows the car to overtake an opponent.

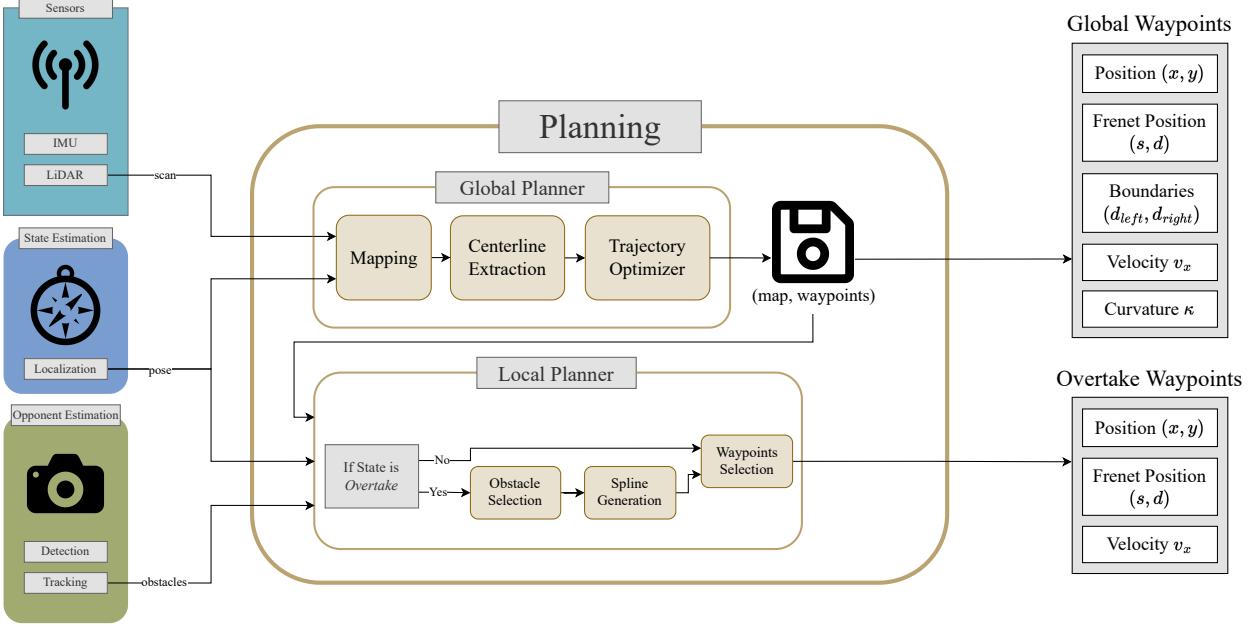


Figure 14: Overview of the proposed planning module. The system integrates a global planning submodule for computing an optimal racing line offline, alongside a local planner that dynamically responds to opponent estimations to compute overtaking trajectories.

6.2 Global Planner

The global planner is based on the work presented in [Heilmeier et al., 2020]. Their work describes the planning of a minimum curvature trajectory using a quadratic optimization problem formulation. To optimize a global path around a racetrack, it is necessary to acquire the centerline and the corresponding track boundaries from the map.

The occupancy grid, generated by SLAM, can be interpreted as an image by transforming each cell into a pixel as illustrated in Fig. 15a. In the first step, the occupancy grid is binarized. Subsequently, the binarized image is smoothed with a morphological open filter [Bovik, 2009] to mitigate most of the LiDAR scans located outside of the track. Following this, the centerline is extracted from the filtered image using the morphological skeleton method [Kong and Rosenfeld, 1996]. The centerline, characterized by its angular shape, can pose challenges for path optimization due to abrupt directional changes. Therefore, the centerline is smoothed using a Savitzky-Golay filter [Orfanidis, 1995], and the resulting centerline is depicted in Fig. 15b. Finally, the centerline, in combination with the *Watershed* algorithm [Bertrand, 2005] is utilized to derive the racetrack and, consequently, the distances along the normal vector to the track boundaries, crucial information that, being in *Frenet* frame, allows us to efficiently compute the distance of a specific coordinate to the boundary. This is useful, for example, to evaluate if candidate trajectories are safely within the track boundaries or if detected obstacles are inside the racetrack.

For the computation of the global trajectory, we employ the global trajectory optimization tool presented in [Heilmeier et al., 2020]. The centerline points $[x_i, y_i]$ and their corresponding distances to the track boundaries $w_{tr, left, i}, w_{tr, right, i}$ serve as the input. The optimization method used is the iterative minimum curvature optimization. This approach iteratively applies the standard minimum curvature optimization to address linearization errors occurring in the conventional problem formulation. The objective of the minimum curvature optimization is to minimize the sum of the discrete squared curvature κ_i^2 along the racing line with N points, which are interpolated through splines. Taking into account the vehicle width

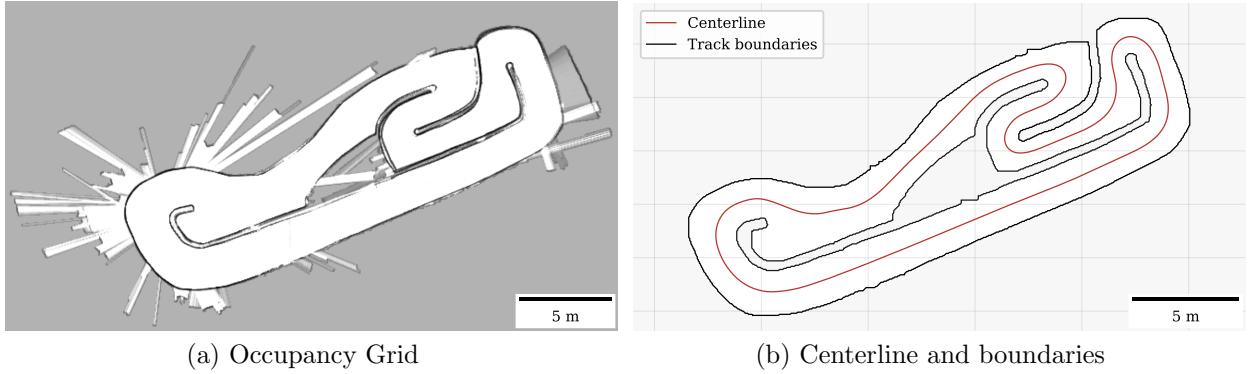


Figure 15: The centerline extraction step of the global planner shown on the track of the *ICRA Grand-Prix 2022*. Fig. 15a: Occupancy grid obtained by SLAM. Fig. 15b: Extracted centerline (red) including track boundaries (black).

w_{ego} , the optimization problem can be formulated as

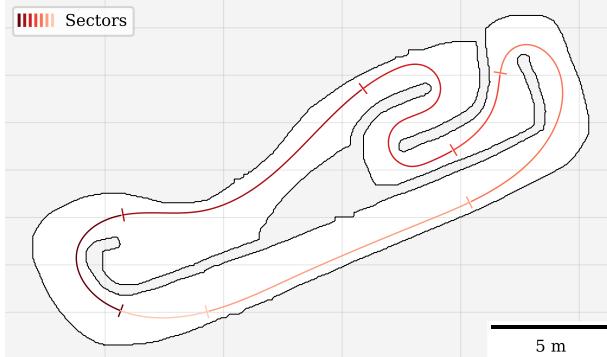
$$\begin{aligned}
 & \underset{[\alpha_1 \dots \alpha_N]}{\text{minimize}} \quad \sum_{i=1}^N \kappa_i^2 = \sum_{i=1}^N \frac{x_i'^2 y_i''^2 - 2x_i' x_i'' y_i' y_i'' + y_i'^2 x_i''^2}{(x_i'^2 + y_i'^2)^3} \\
 & \text{subject to} \quad \alpha_i \in [-w_{tr, left, i} + \frac{w_{ego}}{2}, w_{tr, right, i} - \frac{w_{ego}}{2}] \\
 & \quad \forall 1 \leq i \leq N.
 \end{aligned} \tag{2}$$

Given the cubic spline formulation of the points with respect to the parameter t as follows:

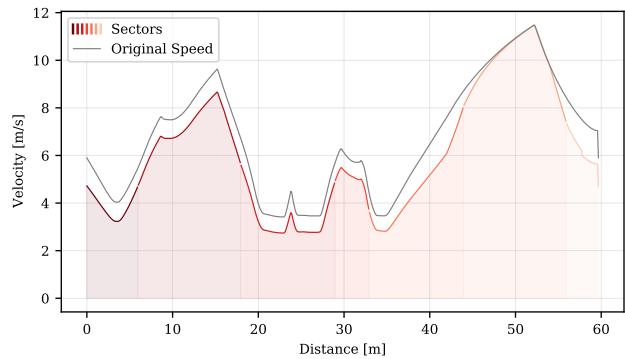
$$x_i = a_i + b_i t + c_i t^2 + d_i t^3, \tag{3}$$

the terms x'_i , x''_i are defined as $x'_i = \frac{dx_i}{dt}$, $x''_i = \frac{d^2x_i}{dt^2}$ (and similarly for the y_i terms). For more information on how the problem is setup and solved as a QP, refer to [Heilmeier et al., 2020]. In addition to path optimization, this tool also generates a velocity profile using a forward-backward solver, resulting in a global trajectory consisting of *Global Waypoints* as illustrated in Fig. 16. The optimization tool offers a range of configurable parameters, such as vehicle specifications or optimization-specific attributes. The most relevant parameters are reported in Appendix A.1.3.

Finally, to improve the tunability of the obtained racing line and handle mixed conditions such as floors with different friction levels, multiple *sectors* along its length are sequentially defined, based on track-by-track heuristics. Every i -th *sector* is then linked to a scaling parameter $\sigma_i^{\text{sector}} \in [0, 1]$, which then multiplies the speed of the global racing line, in order to scale it down. To ensure smooth transitions between the *sectors*, linear interpolation is applied for a range of 1 m at the junction of the *sectors*. An example of such *sectors* can be seen in Fig. 16. The *sector scalers* σ_i^{sector} can then be tuned, either by hand or automatically by means of Bayesian Optimization (BO). In this case, the technique is used to minimize a cost function which is a linear combination of lap time, racing line deviation, and minimum distance from the boundaries. The goal of the BO setup is therefore to find a mapping from the *sector scalers* to this cost function. The implementation is carried out with the *BayesOpt4ROS* package from [Fröhlich and Carron, 2021], and the specific acquisition function used is Expected Improvement (EI) [Jones et al., 1998], which was selected over the other implemented option Upper Confidence Bound (UCB) [Auer et al., 2002] after empirical assessments deemed it more sample efficient for our task, achieving faster performance when given the same amount of sample laps.



(a) Sectors along the racetrack



(b) Scaled velocity profile

Figure 16: Example *sectors* defined on the *Global Waypoints* for the *ICRA Grand-Prix 2022* racetrack. The *sectors* are shown with increasingly lighter hues in Fig. 16a, with a segment indicating the beginning of each *sector*. The effect of such *sectors* on the racing line’s speed profile is then displayed in Fig. 16b, with the original velocity profile in gray. The 7 chosen *sector scalers*, in this example, have as values [0.8, 0.9, 0.8, 0.9, 0.8, 1.0, 0.8], corresponding to the *sectors* ordered from the darkest to the lightest hue.

6.3 State Machine

Fig. 17 illustrates the devised state machine, with each state being physically represented on the vehicle through corresponding LED colors, thereby augmenting real-time visual feedback. This state machine is responsible for orchestrating different high-level behaviors through the information obtained from the different autonomy modules and supply the *Control* module with the correct waypoints. A detailed explanation of the states, with conditions for switching formatted as <cond>, is provided below:

- I **GBFree:** This state denotes an unobstructed global racing line, facilitating unimpeded trajectory tracking. *GBFree* denotes the GloBal racing line being free.
- II **Trailing:** This state denotes when an opponent, denoted as *opp*, is proximal to the forthcoming racing line, and the longitudinal controller is activated to keep a constant gap, ensuring safe and close trailing behind the opponent and positioning the robot well for potential overtaking scenarios.
- III **Overtake:** This state is engaged when the robot, after trailing an opponent, is presented with a valid overtaking solution (*ot*) by the local planner. The robot adheres to the proposed overtaking waypoints unless the solution becomes invalid, in which case it reverts to the *Trailing* state.
- IV **Reactive:** Serving as a safety net, this state is activated when poor state estimation jeopardizes safe operation (*ofc*), prompting the robot to employ a reactive scheme via the FTG controller. The state can be exited as soon as control has been regained, i.e. is in-control (*ic*).

6.4 Local Planner

As *F1TENTH* features unrestricted *Head-to-Head* competitions, dynamically avoiding obstacles is crucial, and local planning constitutes a fundamental step in this endeavor, by generating a feasible and performant trajectory in order for the controller to follow it. Aiming for simplicity and computational efficiency, the core algorithm revolves around generating a spline around the opponent that reconnects with the racing line.³

When no obstacle is within a predefined distance from the nominal trajectory, the local planner provides the controller described in Section 7 with the global waypoints. This state is the one described as *GBFree*

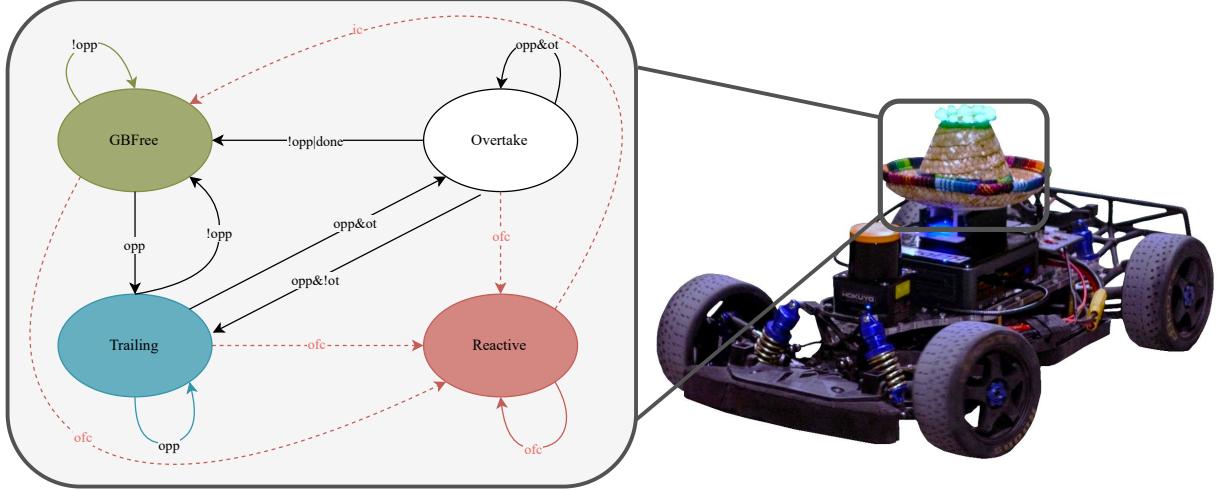


Figure 17: Visualization of the implemented state-machine with the corresponding state-transitions. The vehicle’s LED indicators correspond to these states, providing an immediate visual cue of the real-time status. The transition between states is determined by a set of boolean conditions: *opp* denotes the necessity to account for an opponent; *ot* signifies the feasibility of an overtake; *done* confirms the completion of an overtake; *ofc* indicates out-of-control, and *ic* signals the vehicle is in-control. For ease of presentation, the figure presented here is a simplified version; the full state machine is available in the open-source codebase.

in Section 6.3. When instead the detection of an opponent brings the car to the *Overtake* state, a local trajectory is needed and a detection of a target opponent is assumed to be available in the space in front of the agent. As discussed in Section 5.1.2, the provided opponent state is formulated in the *Frenet* frame as $\mathbf{x}_{\text{opp}} = [s_{\text{opp}}, v_s, v_d, d_{\text{opp}}, d_{\text{d}, \text{opp}}]^T$.

A set of points $\mathbf{p}^{\text{spline}}$ is chosen around the opponent and a third-order spline is then fit through these points, to eventually resample at the same discretization step of the global waypoints. An example setup, describing the set of points and the final spline can be seen in Fig. 18. The first point obtained for $\mathbf{p}^{\text{spline}}$ is the *overtaking apex*, which is the point closest to our opponent where we plan to overtake. This point is defined as $p^{\text{apex}} = [s_{\text{opp}}, d^{\text{apex}}]^T$, where s_{opp} is the same as the target opponent and d^{apex} is defined to be either on the left or the right of the obstacle. The side is decided by evaluating the space from the opponent’s position to the track boundaries on the left and on the right. If one of the two sides is too small, the other one is selected for overtaking. If both have enough space, the side that yields an overtaking racing line closer to the global trajectory is chosen. The overtaking d^{apex} is then obtained in the following way:

$$\text{If the overtake is to the left: } d^{\text{apex}} = d_{\text{opp}} + \left(\frac{1}{2}(w_{\text{opp}} + w_{\text{ego}}) + \delta^{\text{apex}} \right), \quad (4)$$

$$\text{if the overtake is to the right: } d^{\text{apex}} = d_{\text{opp}} - \left(\frac{1}{2}(w_{\text{opp}} + w_{\text{ego}}) + \delta^{\text{apex}} \right), \quad (5)$$

where w_{ego} , w_{opp} indicate respectively the width of the ego-car and of the detected opponent, and δ^{apex} indicates a tunable distance at which we want to overtake, to account for robustness. Such a safety distance can be set based on the sum of our detection system’s positional RMSE plus the measured average lateral distance of our car from the racing line. Practically, this sum only constitutes a lower bound, and usually a value at least twice this bound is chosen. To obtain the remaining points of $\mathbf{p}^{\text{spline}}$, n^{spline} points before and after the *overtaking apex* are selected on the racing line (they are grouped in the two vectors $\mathbf{p}^{\text{preapex}}, \mathbf{p}^{\text{postapex}} \in \mathbb{R}^{2 \times n^{\text{spline}}}$). These points have all $d = 0$ and are used to fit the spline in a way that reconnects to the racing line. The distances along the \mathbf{s} dimension are then scaled with respect to the

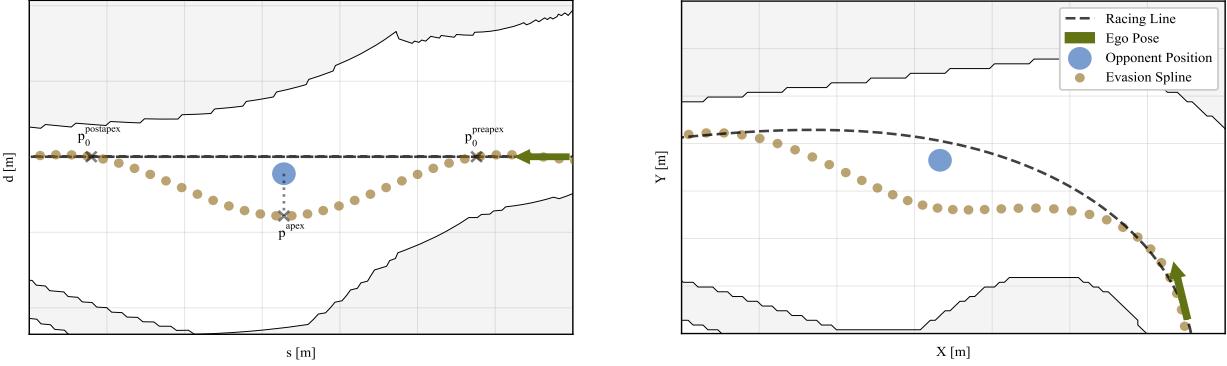


Figure 18: On the left, the planning setup in the *Frenet* coordinate system is described. The ego car’s pose is identified with a green arrow, the opponent’s position is identified with a blue dot, and the final avoidance trajectory around the opponent is in bronze. The points used to construct the spline are marked with black crosses. For ease of visualization, only three example points are used in the figure: the first points before and after the apex are, respectively, $p_0^{preapex}$, $p_0^{postapex}$, while the point at the apex is p^{apex} . The safety distance used to find the lateral apex distance from the opponent’s position is shown with a dotted gray line. On the right, the resulting overtaking trajectory is displayed in *Cartesian* coordinates.

ego-car velocity to account for the decreased steering capacity with the increasing longitudinal velocity. The coordinates of \mathbf{p}^{spline} are then defined as follows:

$$\mathbf{p}^{preapex} = \left[\begin{bmatrix} s_{opp} - \alpha_v \Delta_0^{preapex} \\ 0 \end{bmatrix}, \begin{bmatrix} s_{opp} - \alpha_v \Delta_1^{preapex} \\ 0 \end{bmatrix}, \dots, \begin{bmatrix} s_{opp} - \alpha_v \Delta_{n^{spline}}^{preapex} \\ 0 \end{bmatrix} \right], \quad (6)$$

$$p^{apex} = \begin{bmatrix} s_{opp} \\ d_{apex} \end{bmatrix}, \quad (7)$$

$$\mathbf{p}^{postapex} = \left[\begin{bmatrix} s_{opp} + \alpha_v \Delta_0^{postapex} \\ 0 \end{bmatrix}, \begin{bmatrix} s_{opp} + \alpha_v \Delta_1^{postapex} \\ 0 \end{bmatrix}, \dots, \begin{bmatrix} s_{opp} + \alpha_v \Delta_{n^{spline}}^{postapex} \\ 0 \end{bmatrix} \right], \quad (8)$$

$$\mathbf{p}^{spline} = [p_{n^{spline}}^{preapex}, \dots, p_0^{preapex}, p^{apex}, p_0^{postapex}, \dots, p_{n^{spline}}^{postapex}], \quad (9)$$

where α_v is the velocity scaler, scaling the length of the overtaking sector according to the current velocity, and $\Delta_i^{\{preapex, postapex\}}$ are the distances at which the spline points on the racing line are taken, defined as follows:

$$\alpha_v = 1 + \min \left(\frac{v_{s, ego}}{v_{max}}, 0.5 \right),$$

$$\Delta^{\{preapex, postapex\}} = [\Delta_1^{\{preapex, postapex\}}, \Delta_2^{\{preapex, postapex\}}, \dots, \Delta_{n^{spline}}^{\{preapex, postapex\}}],$$

$$\Delta_i^{\{preapex, postapex\}} \in \mathbb{R}, \quad i \leq j \Rightarrow \Delta_i^{\{preapex, postapex\}} \leq \Delta_j^{\{preapex, postapex\}} \quad \forall i, j \in \{1, 2, \dots, n^{spline}\},$$

where $v_{s, ego}$ is the component of our velocity tangential to the racing line, and v_{max} is the maximum racing line velocity. Given this set of points, a spline is then fit in order to regress the \mathbf{d} coordinate from the \mathbf{s} coordinate. An evasion trajectory is then resampled from the obtained spline with the same discretization step used for the global racing line, and its speed profile is also applied to the evasion trajectory. The avoidance trajectory is then substituted into the global racing line for $s \in [s_{opp} - \alpha_v \Delta_{n^{spline}}^{preapex}, s_{opp} + \alpha_v \Delta_{n^{spline}}^{postapex}]$ to provide waypoints all around the track. A diagram of the obtained trajectory is represented in both *Frenet* and *Cartesian* coordinates in Fig. 18.

6.5 Planning Results

This section presents the results obtained from the implementation of our planning strategies, focusing on both global and local planning aspects. Initially, we present the results of our global planning strategy,

displaying the obtained trajectory on different racetracks. This is followed by an examination of the local planning results, showcasing the local planner’s ability to perform overtaking maneuvers.

6.5.1 Global Planning Results

To showcase different trajectories from the global planner, we conducted qualitative assessments on various racetracks, as depicted in Fig. 19. Fig. 19a illustrates the racetrack from the *ICRA Grand-Prix 2022*, characterized by a combination of long straights and narrow turns. Conversely, Fig. 19b presents a test track, notable for its narrowness and frequent sharp turns with minimal straight segments. Across both tracks, the planner consistently generated a global trajectory that adhered to predefined constraints, such as vehicle width including a safety margin, thereby ensuring the vehicle remained within the track bounds without corner-cutting. As it can be seen from Fig. 19, the safety width is significantly larger than the vehicle width (0.3 m for the vehicle width and 0.8 m for the safety width). This was needed for mainly two reasons: first, to account for the additional size needed by a car in case it is not perfectly aligned with the racing line, and second to account for the limited tracking capabilities of our controller, as described in Section 7.

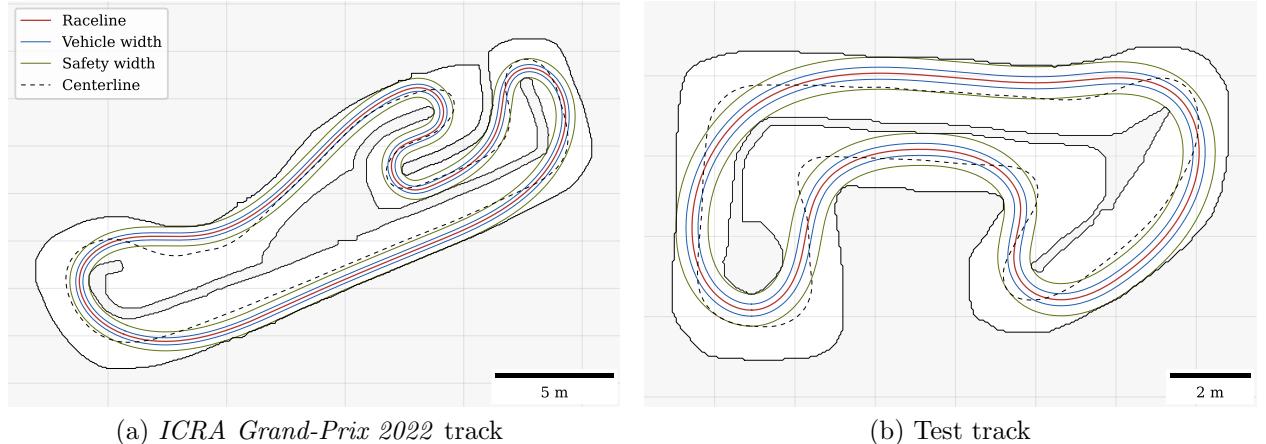


Figure 19: The global trajectory optimized with iterative minimum curvature on two different tracks. Besides the racing line (red), the figure also shows the centerline (dashed black) and the vehicle width without (blue) and with a safety distance (green). The illustration shows that even with the conservative safety distance the selected optimizer finds a feasible trajectory for typical *F1TENTH* competition tracks.

6.5.2 Local Planning Results

To qualitatively assess the overtaking capabilities of the proposed local planner, race simulations on real platforms have been conducted with two agents on a track, as depicted in Fig. 20. Both agents were equipped with the *ForzaETH Race Stack*. The ego-agent (green trajectory) was tasked with overtaking a slower opponent (blue trajectory), set to complete the track at a lap time 66% slower than the ego-agent.

- i At timestep t_0 , the ego-agent catches up with the slower opponent. The local planner identifies a feasible path for overtaking.
- ii By timestep t_1 , the ego-agent is actively executing the overtaking maneuver.
- iii At timestep t_2 , the overtaking is complete. The opponent is now behind the ego-agent, indicating a successful maneuver.
- iv Finally, at timestep t_3 , the ego-agent resumes regular racing operations.

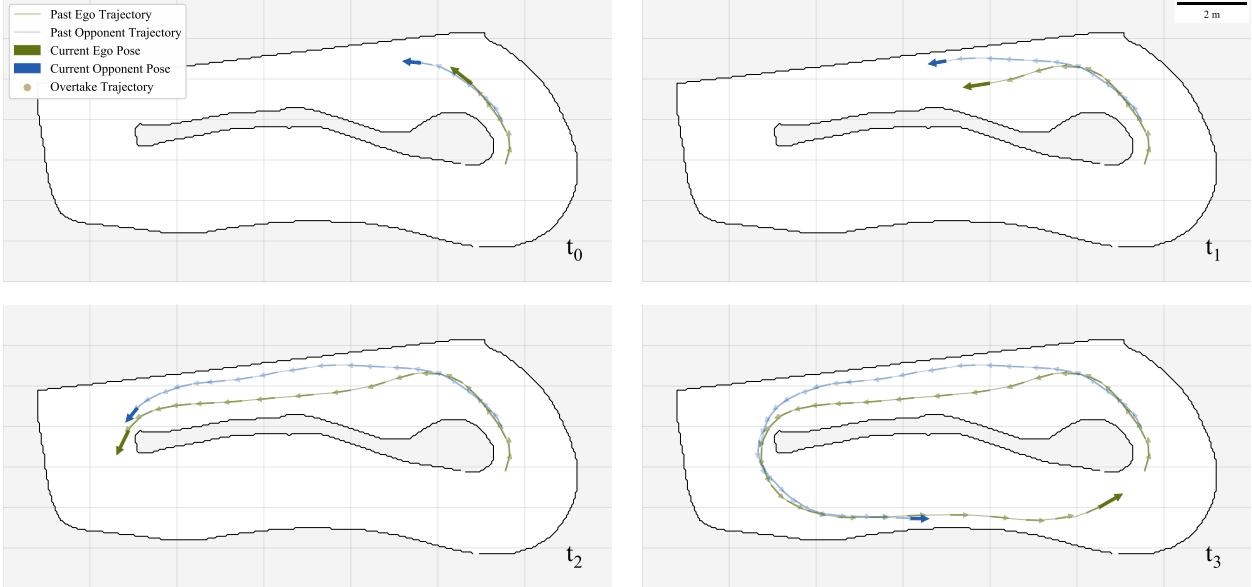


Figure 20: Sequential frames of an overtaking maneuver. The blue trajectory represents the opponent, while the green trajectory is that of the ego-agent. The figure also shows the ego and opponent poses, and the planned overtaking waypoints for each timestep. It can be seen that in the frames after the overtake has happened (t_2, t_3) the planned waypoints are on the nominal racing line.

7 Control

The goal of the *Control* module within the *ForzaETH Race Stack* is to enable performant, reliable, and consistent lap completions by accurately tracking a trajectory provided by the local planner. Therefore, key performance and safety metrics of the control algorithms are lap time, lateral deviation to the racing line, and deviation from the desired velocity. The velocity and steering commands are managed, respectively, by the longitudinal and lateral controllers.

7.1 Architecture

Fig. 21 depicts the architecture of the *Control* module. It relies on the upstream robotics modules: *State Estimation*, *Perception*, and *Planning*, to compute the final control outputs. The local trajectory, state estimation data, and opponent information are then used to calculate the desired steering angle and velocity. Lateral and longitudinal controllers are separated to simplify development and testing.

I Longitudinal Controller: This submodule computes the desired velocity of the racecar. This velocity is either obtained directly from the velocity of the local trajectory (nominal velocity controller, Section 7.2.1) or computed by the trailing controller (trailing velocity controller, Section 7.2.2) if the car is trailing an opponent and therefore in the *Trailing* state.

II Lateral Control: This computes the steering angle to enable accurate spatial trajectory tracking and is based on the Model- and Acceleration-based Pursuit (MAP) controller [Becker et al., 2023]. It finds the desired steering angle based on the desired lateral acceleration and by leveraging the tire model of the racecar.

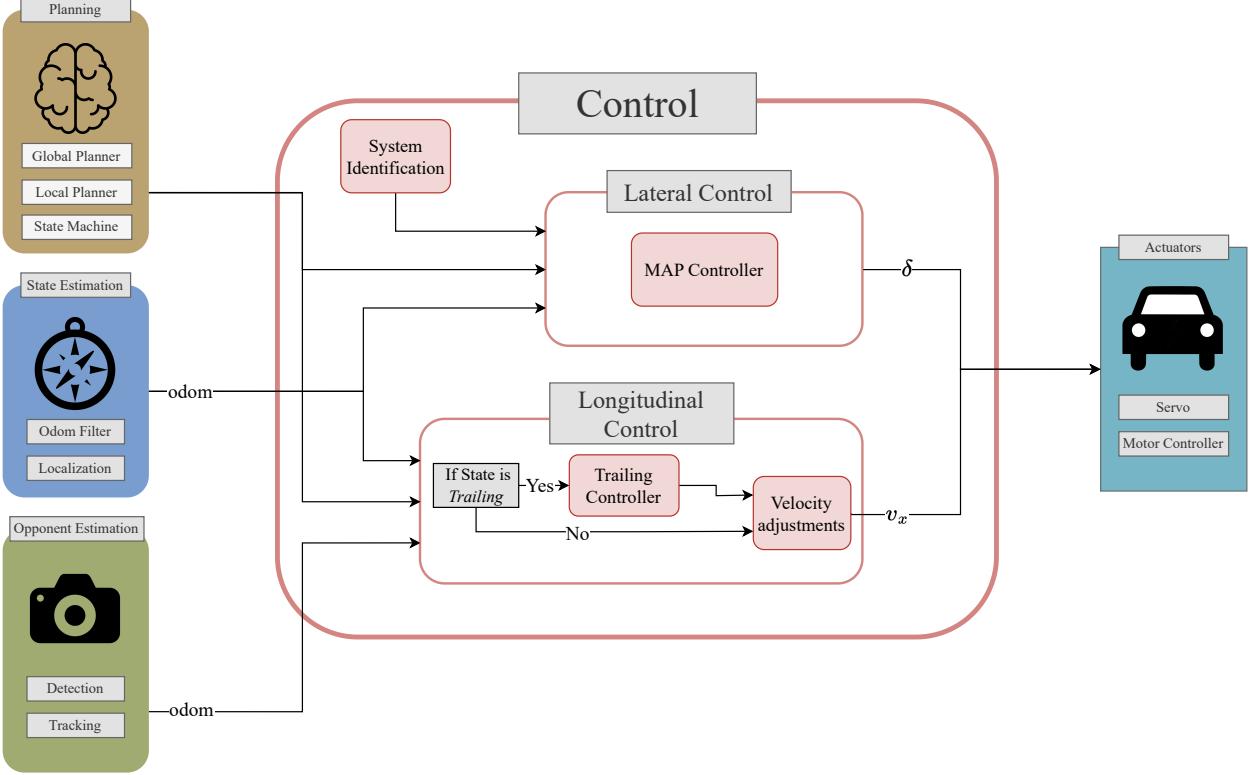


Figure 21: Overview of the proposed control module that depicts the separation into lateral and longitudinal control. The *Control* module, positioned at the end of the *See-Think-Act* sequence, utilizes upstream data from *State Estimation*, *Perception*, and *Planning* to generate control commands to finally execute it on the hardware actuators. In case the state provided by the state machine is not *Trailing*, the nominal velocity controller is used, described in Section 7.2.1. On the other hand, if the state is *Trailing*, the used controller is the trailing velocity controller, described in Section 7.2.2.

7.2 Longitudinal Controller

The longitudinal controller computes the velocity input to the *F1TENTH* racecar. This input is then commanded to the VESC actuator, which converts the desired velocity to the appropriate ERPM command with the help of the VESC internal low-level Proportional-Integral-Derivative (PID) controller.

In both the *GBFree* and *Overtake* states, the racecar derives its reference velocity from the local trajectory generated by the *Planning* module, this is referred to as the nominal velocity control, described in Section 7.2.1. However, when an opponent obstructs the racing line and overtaking is not immediately viable, the vehicle transitions into the *Trailing* mode, described in Section 7.2.2. In this state, it adjusts the velocity to maintain a safe following distance, thereby preventing potential collisions. This trailing strategy not only upholds safety by avoiding collision with the opponent car but also positions the ego vehicle close to seize an overtaking opportunity when it arises.

7.2.1 Nominal Velocity Controller

When operating in the *GBFree* or the *Overtake* state, the car adopts its target velocity from a corresponding point along the local trajectory. The reference velocity is first obtained by querying the velocity profile v_{traj} . As the global planner provides a discretized velocity profile, with a discretization step every 0.1 m, the s coordinate is first rounded, then used to obtain the required velocity. For ease of notation, the velocity

lookup on the precomputed racing line for a predetermined s is here noted with $v_{traj}(s)$.

To account for actuation and computation delay, the actual reference velocity is obtained from an s coordinate forward-propagated in time. A lookahead distance s_{la} is first calculated using a t_{la} lookahead time and the velocity of the car in the Frenet s coordinate v_s , then s_{la} is summed to the current car position in the s coordinate s_{ego} to evaluate the reference velocity v_{ref} :

$$v_{ref} = v_{traj}(s_{ego} + s_{la}). \quad (10)$$

We further apply the following heuristic, to scale down the reference velocity when the lateral deviation is not zero, using the term α_{lat_dev} defined as follows:

$$\begin{aligned} v_{des} &= \alpha_{lat_dev} \cdot v_{ref} \\ &= (1 + \lambda_{lat}(-1 + e^{-d_{norm} \cdot c_{norm}})) \cdot v_{traj}(s_{ego} + s_{la}). \end{aligned} \quad (11)$$

Here $\lambda_{lat} \in [0, 1]$ determines how much the lateral error is accounted for. A value of 0 means no accounting for lateral error and a value of 1 means maximal speed reduction on lateral error. Both d_{norm} and c_{norm} are normalized values in $[0, 1]$ representing the lateral deviation from the desired trajectory and the curvature of the trajectory at the current position. This adjustment helps the car slow down when it is not on the racing line, in order to reconnect more smoothly to the trajectory. The final value sent to the VESC for low-level actuation is then v_{des} defined as in Equation (11).

7.2.2 Trailing Velocity Controller

If the car is in state *Trailing*, the goal is to adjust the ego-car velocity to the one of the to-be-trailed opponent, such that a constant distance gap can be achieved. This is achieved with a Proportional-Derivative (PD) controller with a feedforward term of the opponent's velocity as displayed in Fig. 22.

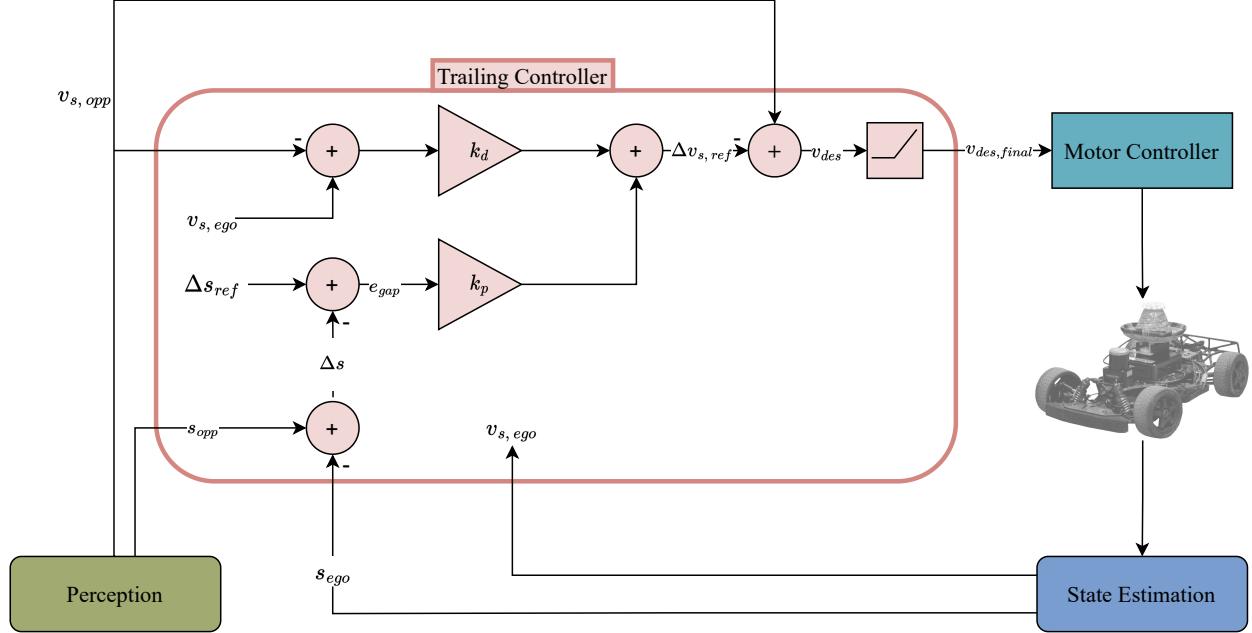


Figure 22: Diagram of the trailing control flow. The trailing controller represents a subset of the overall *Control* module of Fig. 21. It is responsible for maintaining a close and steady distance towards the opponent while avoiding collisions. This allows the agent to set itself up well for a possible overtaking maneuver.

The v_{des} for the trailing controller is then defined as follows:

$$v_{des} = v_{s, opp} - (k_p e_{gap} + k_d \Delta v_s), \quad (12)$$

where $v_{s, opp}$ is the velocity of the opponent in the *Frenet s* dimension. k_p is the gain of the proportional error term e_{gap} , which is the difference between the fixed target gap Δs_{ref} and the measured gap Δs . The gain k_d is applied to the derivative term, which is obtained by subtracting the opponent's velocity along the *Frenet s* dimension $v_{s, opp}$ to the ego-car's current velocity along the *Frenet s* dimension $v_{s, ego}$.

$$e_{gap} = \Delta s_{ref} - ((s_{ego} - s_{opp}) \% s_{max}) \quad (13)$$

$$\Delta v_s = v_{s, ego} - v_{s, opp}. \quad (14)$$

In Equation (13) % represents the modulo operator, to account for the *s* coordinate wrapping over multiple laps. On top of the core part, the PD controller, some considerations were made to improve the controller's performance in real-world scenarios:

- i Only the closest obstacle is considered
- ii If both static obstacles and dynamic obstacles are present, only dynamic obstacles are considered
- iii The velocity is clipped to a minimum velocity

Given the current *F1TENTH Head-to-Head* racing rules, which only allow for one opponent on the track, the first consideration should not affect the car's behavior, however, it still remains necessary when multiple opponents can be detected in free practice settings and more than two cars are present on the track at the same time. The second consideration then arises from the fact that some false positives can still arrive from the *Perception* module (e.g. caused by reflections on the floor, modified track boundaries, etc.), but since no static obstacle is present on the track during the *Head-to-Head* phase if both types of obstacles are present, static classifications are deemed false positives and dynamic obstacles are preferred. If, instead, only static obstacles are present, the trailing controller will choose the closest (and eventually stop in case the object does not disappear). The final consideration is then added to avoid rare cases where the trailing controller starts trailing very slow obstacles outside of LoS (e.g. in a hairpin) which could lead even to a halt, in case the detected velocity is low enough. When a minimum velocity v_{blind} is instead enforced, in the cases when there is no LoS of an obstacle, a minimum velocity will still be kept, in order to move to a position with a better view of the potential obstacles. The final velocity sent to the low-level controller is then obtained as follows:

$$v_{des, final} = \max(v_{blind}, v_{des}). \quad (15)$$

7.3 Lateral Controller

The lateral controller commands steering angles δ to the servo, with the goal of making the car follow a desired trajectory, as generated by the planner (Section 6). To achieve this, the MAP controller from [Becker et al., 2023] was chosen, due to its high tracking accuracy and little computational complexity. It outperforms state-of the art geometric controllers like *Pure Pursuit* [O'Kelly et al., 2020a], by incorporating tire-slip through a model of the car's steady state cornering behavior. The following sections lay out how this model is obtained experimentally and how the MAP controller is set up and tuned.

7.3.1 System Identification

The system identification follows the procedure outlined in [Becker et al., 2023]. Notably, the vehicle dynamics are modelled using a single-track model [Althoff et al., 2017]. Tire slip is incorporated through the *Pacejka* tire model [Pacejka and Bakker, 1992]. Most physical static model parameters can be measured, namely the mass m with a scale and the distances of the front and rear axle from the CG (l_r, l_f) by balancing the car on a string and measuring the distances from the front and rear axles. Furthermore, the height of the CG above the axles h_{cg} was obtained by first vertically balancing the car to identify the vertical position of the CG and then, to take suspensions into account, the car was put on the ground and the final h_{cg} was

obtained. The yaw moment of inertia I_{zz} is calculated based on the oscillation period in a bifilar pendulum experiment [Green, 1927].

The characterizing parameters of the tire model are obtained through steady-state cornering experiments, as described by [Voser et al., 2010]. The car is driven in circles at a constant speed and with slowly varying steering angles, resulting in a range of quasi-steady state measurements of the lateral acceleration for a sweep of tire slip angles. In particular, the steering angle is increased with 0.02 rad/s from neutral to the negative and the positive maximum for 3, 4, and 5 m/s respectively. Following equations (7 - 14) of [Becker et al., 2023], the lateral forces produced by the front and rear tires are related to the respective tire slip angles. The parameters of the tire model are obtained by solving a nonlinear least-squares problem, using the `least_squares` function from the Python library `scipy`. Outliers are removed through three steps k of expectation maximization, removing any measurements more than $\frac{10}{2^k}$ away from the previous estimate. To constrain the curves' shapes for high tire slip angles, the shape and curvature factor of the *Pacejka* model are limited to 1.5 and 0.8 respectively for the rear axle and 4.0 and 1.1 for the front axle. The scripts to automatically run the experiments and analyze the recorded data are part of the published race stack. In a racing scenario, such measurements would be conducted usually outside of the track, as a space of approximately 10 m by 5 m is needed.

7.3.2 MAP Controller

Similar to traditional geometric controllers like *Pure Pursuit*, MAP calculates steering commands intending to reach a lookahead point. This is a point on the trajectory at tunable distance L_d from the car, as depicted in Fig. 23a. The advantage of MAP over traditional geometric control lies in the incorporation of tire slip into the calculation of steering inputs. This has been shown to significantly improve its tracking performance in high-speed corners [Becker et al., 2023]. The MAP controller achieves this by computing the required centripetal acceleration a_c to reach the lookahead point and retrieves the corresponding steering angle from a precomputed lookup table.

The calculation of lateral acceleration is based on the L_1 guidance [Park et al., 2004], originally designed for fixed-wing Unmanned Aerial Vehicles (UAV)s. It finds the lateral acceleration a_c making the vehicle with longitudinal speed v_x follow a circular motion with radius R onto the lookahead point. This is obtained from the following relationship:

$$a_c = 2 \frac{v_x^2}{L_d} \sin(\eta) = \frac{v_x^2}{R}, \quad (16)$$

where η denotes the angle between the heading and the lookahead point as illustrated in Fig. 23a.

As there is no explicit solution to map steering angles to lateral accelerations, a lookup table is used, which is generated offline to keep computation lightweight. It is obtained by simulating the identified single-track model dynamics of Section 7.3.1 for a sweep of constant speeds and steering angles. These simulations, conducted for two seconds of simulated time, are repeated for velocities from 0.5 to 7 m/s at 0.1 m/s intervals and for a range of steering angles from 0 to 0.4 rad at steps of 0.01 rad. A finer step size of 0.0033 rad is used for the steering angles between 0 and 0.1 rad. The final steady-state lateral acceleration a_c is stored. If no steady state is reached the pair of velocity and steering angle is considered unstable. This results in a stable steering region illustrated as a surface in Fig. 23b.

As proposed in [Becker et al., 2023], the lookahead distance parameter L_d is scaled with the vehicle's current speed v using an affine mapping: $L_d = mv + q$. The tunable coefficients, m and q , are used to adjust the scaling.

This tuning of L_d and the scaling is critical to achieving stable and effective control. If the distance is set too short, especially at high speeds, it can result in undesirable vehicle oscillations. Conversely, setting the distance too large may tempt the vehicle to cut corners and potentially lead to collisions. Therefore, the goal is to keep L_d as small as possible without it causing oscillations, striking a balance between stability

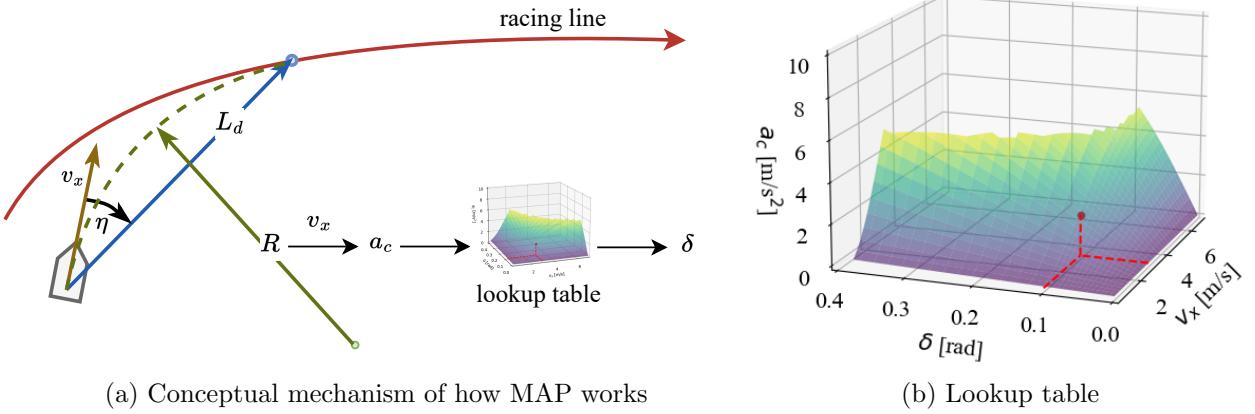


Figure 23: (a) MAP obtains required centripetal acceleration a_c to reach a lookahead point L_d using the L_1 guidance law. (b) The lookup table is used to retrieve the steering angle δ that results in the calculated a_c for the speed v_x .

and performance.

7.4 Control - Results

Here, the performance of the selected control algorithms is analyzed. Firstly the results of the longitudinal controller are examined, covering velocity tracking accuracy and the longitudinal distance-keeping outcomes of the trailing controller. Subsequently, the lateral control results, regarding the lateral tracking accuracy are explored.

7.4.1 Velocity Controller Results

In this section, the results for the nominal and trailing velocity controllers on the physical platform are reported. In both cases, the car was run on the racing track shown in Fig. 25. In the nominal case, the sector scalers were set uniformly to 60%. This scaler was chosen to drive fast lap times while also ensuring consistent driving behavior. In the second case, the ego car was set in a permanent *Trailing* state and a second car was run with a 40% sector scaler setup on the same racing line, to ensure continuous operation of the trailing controller. In both cases, data was recorded over ten laps, and both ego and opponent velocities were recorded running the *State Estimation* module respectively on the two cars. Data was then synchronized with the ROS timestamp.

In Fig. 24, the outcomes of the velocity control system are showcased. The longitudinal velocity directives are compared with the velocity estimates derived from state estimation. When driving in state *GBFree*, the actual velocity of the racecar as estimated by the *State Estimation* is compared to the racing line velocity determined by the global planner. As can be seen in Fig. 24 the computation and activation delay is compensated by taking the velocity of the racing line at a point in front of the racecar as described in Section 7.2. When driving in state *Trailing*, the actual velocity of the racecar is compared to the velocity of the opponent's car. Although the ego-car velocity generally tracks that of the opponent, the trailing controller is designed to maintain a constant gap to the opponent. Consequently, velocity divergence occurs at certain points to rectify and uphold the desired gap to the opponent.

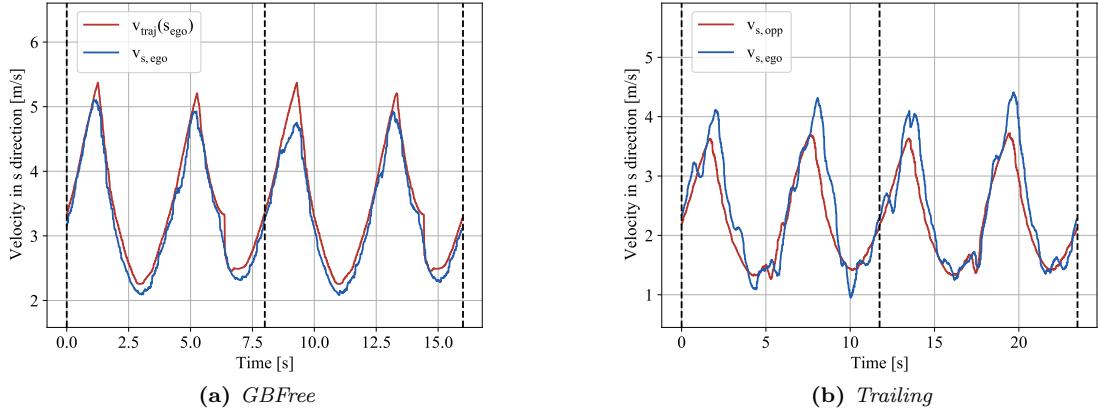


Figure 24: Velocity data over two consecutive laps, on the left the car is driving in the state *GBFree* and on the right trailing an opponent. The to-be-tracked reference velocity is the velocity of the global waypoints in the state *GBFree* and the opponent’s velocity in the state *Trailing*. The start of a new lap is marked by the black dashed line.

7.4.2 Trailing Controller Results

To test the trailing controller, the car was tuned to drive the fastest possible lap times without crashing, which was equivalent to a uniform 60% scaler. Then the trailing controller was tested against three slower opponents with different driving styles by driving four consecutive laps behind the opponent. The three different driving styles presented are a slower opponent with the same racing line and a 40% scaler, a FTG opponent with a constant speed, and a manually driven car.

The results presented in Fig. 25 show a consistent behavior over four consecutive laps, even against the less predictable manually driven opponent. Against the racing line opponent it can be observed, that the gap to the opponent overshoots at two sections on the track. These are the straights where the opponent is accelerating and the gap increases until the ego-car catches up. The mean deviation from the reference gap $\Delta\bar{\mu}_{Gap}$ and the maximal deviation from the reference gap $\Delta\mu_{Gap,max}$ are listed in Table 6. $\Delta\mu_{Gap,max}$ never exceeds 1.2 m, therefore a reference trailing gap of 1.5 m - 2 m, can be chosen to trail safely.

Opponent Style	$\Delta\mu_{Gap,max}$ [m]	$\Delta\bar{\mu}_{Gap}$ [m]
Racing Line Opponent	1.13	0.23
FTG Opponent	1.07	0.18
Manual Opponent	1.06	0.17

Table 6: Evaluation of the trailing controller against three different opponents.

7.4.3 Trajectory Tracking Results

The tracking performance of the MAP controller is illustrated in Fig. 26, depicting trajectories over ten laps on two different tracks, as well as the lateral tracking error and the velocity. The controller was tuned to maximize speed while accurately following the racing line (represented in red). The tuning process involved adjusting lookahead distance and scaling parameters (L_d , m and q) on the first track, scaling the trajectory speed to 75%. The speed was then incrementally increased until it reached the limits of the track bounds.

Trajectory tracking results, with lap times and lateral errors, can then be seen in Table 7. On the *Serpentine Circuit*, MAP consistently reached 79% of the maximum speed without crashing. This map was used to

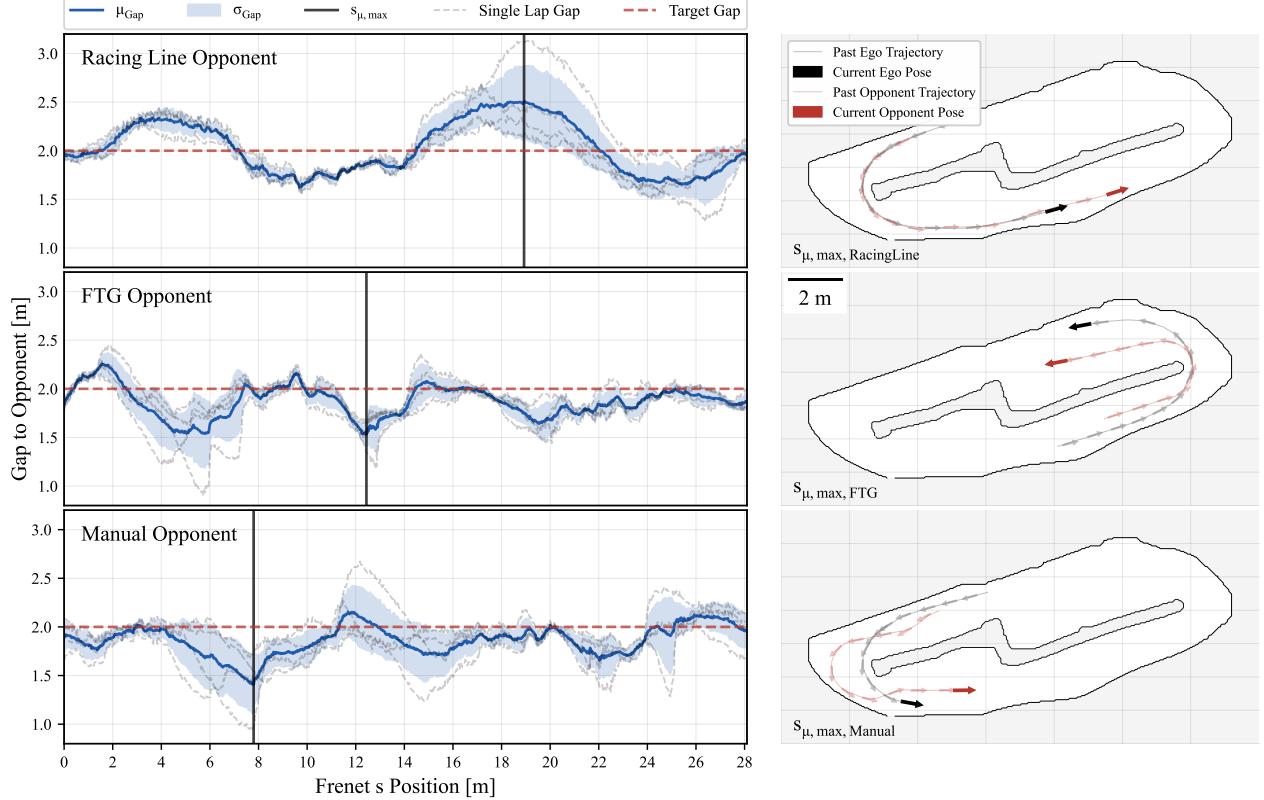


Figure 25: Trailing gap over four consecutive laps against three different opponents is shown. The reference gap is depicted in red, the mean and standard deviation from the reference gap in blue, and the measurements of the gap in each lap in grey.

perform tuning. On *Azure Ridge Track*, the tuning parameters, obtained from *Serpentine Circuit* were then used to demonstrate the adaptability of the controllers across different tracks. This track reaches higher velocities than *Serpentine Circuit* due to its long straight. On this track, MAP was able to consistently reach 81% of the maximum speed without crashing, due to the easier configuration of the circuit, presenting fewer corners. Fig. 26 shows the two tracks where the trajectory speed for both maps was scaled to 79% for comparison. This indicates that MAP can effectively adapt to different tracks when the tuning yields a sufficiently low lateral error (~ 10 cm in this case) and such error performance is maintained on a different track. When pushing to the limits, instead, and lateral error is higher, behavior deviates more from the expected racing line, and performance can be transferred only partially.

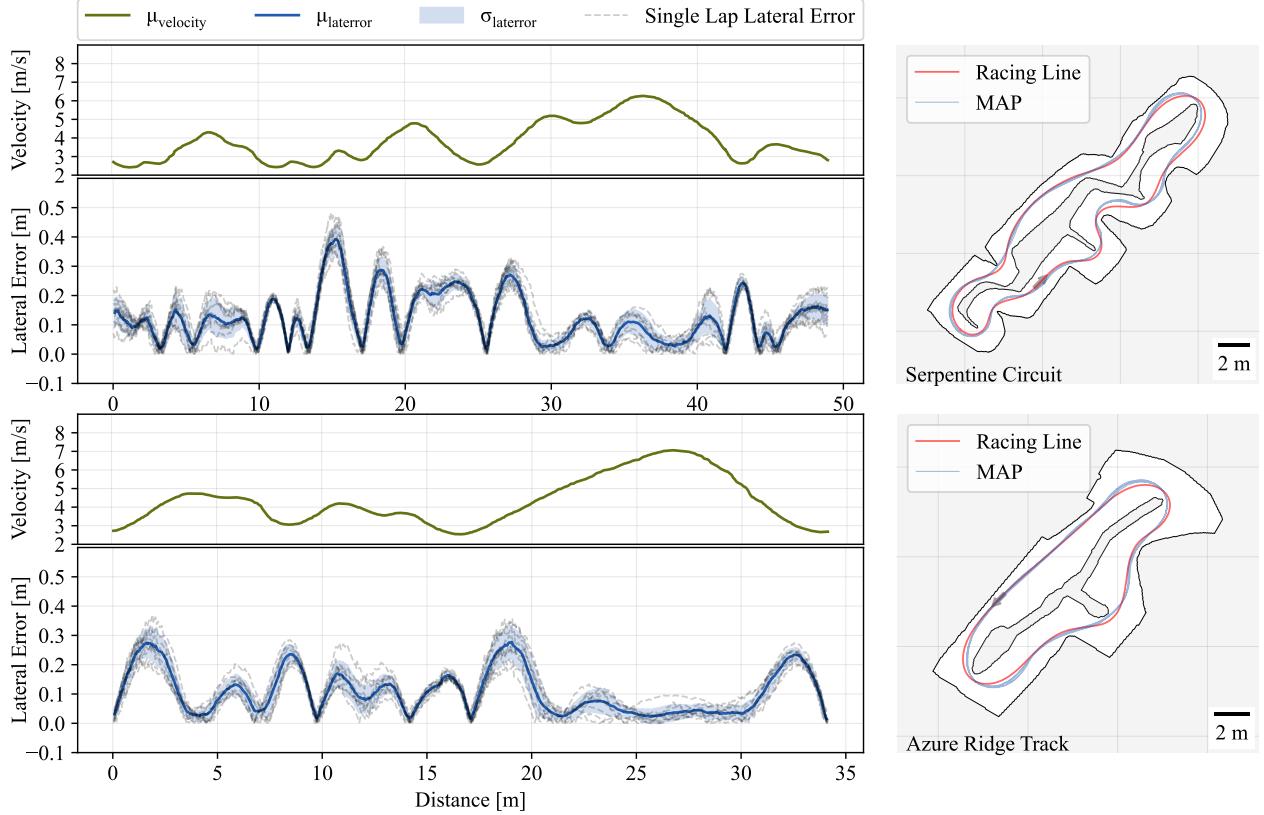


Figure 26: Ten laps of the MAP controller on two different real-world tracks tracking a reference racing line. The left side shows velocity and average and per lap lateral tracking error. On the right, the map and trajectories of the laps are shown.

Track (σ^{scaler})	Lap Time [s] ↓				Lateral error [cm] ↓		
	μ_t	σ_t	t_{min}	t_{max}	μ_{err}	σ_{err}	err_{max}
Serpentine Circuit (75%)	14.80	0.11	14.62	14.99	9.35	0.97	42.08
Serpentine Circuit (79%)	14.47	0.09	14.38	14.66	12.99	0.56	47.85
Azure Ridge Track (75%)	9.04	0.09	8.89	9.21	7.46	0.70	21.31
Azure Ridge Track (79%)	8.77	0.12	8.52	8.98	11.67	1.01	37.06

Table 7: The figure shows average, standard deviation, minimum, and maximum lap time, respectively as μ_t , σ_t , t_{min} , t_{max} . Furthermore, the average, standard deviation, and maximum lateral error with respect to the racing line, are shown, respectively, with μ_{err} , σ_{err} , err_{max} . The used uniform *sector scalers* are signaled with σ^{scaler} . Data was gathered on the tracks shown in Fig. 26. The statistics have been computed from ten laps each.

8 Full Stack Performance

The previous sections detailed the performance of individual components within the *See-Think-Act* cycle, specifically *State Estimation*, *Opponent Estimation*, *Planning*, and *Control*. These assessments aimed to quantify each module’s performance in an isolated context — as far as possible. This chapter presents the performance analysis of the full *ForzaETH Race Stack* when all modules operate interconnected. With the focus on evaluating the integrated system behavior during *Time-Trials* and *Head-to-Head* racing scenarios.

8.1 Time-Trials Performance

The *Time-Trials* stage in *F1TENTH* racing emphasizes achieving the fastest lap time while maintaining a high number of consecutive crash-free laps. This necessitates the ability to execute laps that are rapid, safe, and consistent. To assess these aspects, the *ForzaETH Race Stack* was configured as outlined and deployed on a test track (see Fig. 27). In this environment — free from obstacles and opponents — ten laps were recorded to evaluate racing performance, focusing on lap time, deviation from the optimal racing line, and consistency as indicated by the standard deviation of these metrics.

In racing, tuning the racing line to match specific environmental conditions, such as varying track surface friction coefficients, is crucial for achieving performant behavior. This necessitates adjusting the velocity profile of the racing line to align with the track layout and grip levels. The following tuning strategies, as outlined in Section 6.2, were employed to adjust the velocity scalers for each sector and refer to Fig. 28 and Fig. 29:

I Uniform Sector Scalers: Sectors are uniformly scaled by a constant factor, a quick but often sub-optimal method. Analysis shows a linear decrease in lap times up to a 78% scaler, then a minimum is obtained with a 79% scaler, yielding a lap time of $\mu_u = 6.34s$. The lateral error increases slightly up to 70% then sharply rises from 5.0 cm to 16.1 cm in the 70 – 79% range, indicating competitive lap times but with increasing safety risks. Higher sector scalers, from 80% onwards, led to collisions with the boundaries.

II Bayesian Optimized Scalers: Individual sector tuning using BO optimizes lap time and lateral deviation but requires multiple uninterrupted laps for convergence, which might not be feasible during a racing scenario. The cost function was acquired averaging three consecutive laps, and ten iterations were run, for a total of 30 laps. The optimum was obtained with the parameters from the fifth iteration. This method yields an average lap time of $\mu_{BO} = 6.35s$ with a significantly lower lateral deviation of 10.2 cm, about 1.5 times lower than uniform tuning, enhancing safety and consistency. The trajectories and speed profile resulting from such a tuning procedure can be seen in Fig. 27.

III Hand-Tuned Scalers: Manual adjustment based on racing expertise offers a balance between uniform and BO tuning, achieving $\mu_{HT} = 6.31s$ and a lateral deviation of 13.24 cm, a compromise between performance and safety. The trajectories and speed profile resulting from such tuning procedure can also be seen in Fig. 27, compared to the result from BO tuning.

Refer to Fig. 28 and Fig. 29 for a depiction of lap times and lateral deviations for these methods. This combined approach offers a comprehensive overview of each tuning strategy’s impact on lap time and lateral deviation, facilitating a nuanced understanding of their effectiveness and suitability in various racing scenarios.

The performance in *Time-Trials* was compared against other prevalent *F1TENTH* racing architectures to provide context for the proposed *ForzaETH Race Stack*:

I MAP: Uses the *ForzaETH Race Stack* with the MAP controller.

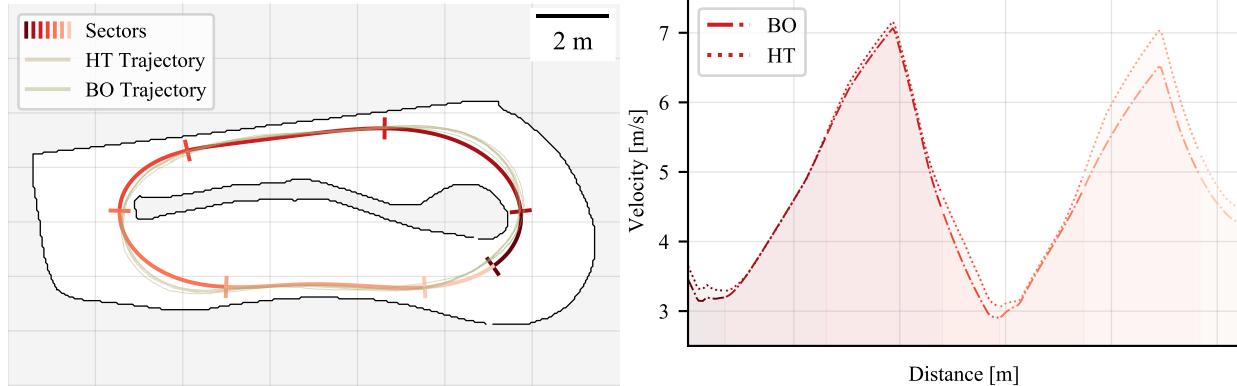


Figure 27: On the left, the image displays sectors as defined for the displayed circuit. Sectors are ordered from zero to six from the darkest hue to the lightest. A perpendicular segment is used to indicate the beginning of the sector. The trajectories following the hand-tuned (HT) sectors and the BO tuned sectors are also shown, respectively in bronze and green. On the right, the final velocity profiles resulting from the tuned sector scalers are displayed. The final values resulting from the BO setup were [0.76778, 0.76083, 0.78929, 0.78776, 0.75155, 0.75019, 0.77131]. The final values resulting from the hand-tuning setup were [0.81, 0.76, 0.80, 0.86, 0.76, 0.81, 0.81].

II PP: Employs the same stack but replaces the MAP controller with the widely used Pure Pursuit (PP) controller [Coulter, 1990], maintaining the same *State Estimation*, *Opponent Estimation*, and *Planning* setup.

III FTG: Utilizes the FTG controller [Özdemir and Sezer, 2017], a reactive method that bypasses the *See-Think-Act* cycle, responding directly to LiDAR data by navigating through the largest perceived gap. While simpler and autonomous, it lacks the performance and adaptability of systems that adhere to the *See-Think-Act* paradigm.

Fig. 30 presents the lap times and lateral deviations relative to the racing line across ten laps for different race stack configurations. To ensure a fair comparison, the PP and FTG parameters were hand-tuned to achieve the lowest possible lap times. Both the MAP and PP versions of the *ForzaETH Race Stack* employed a uniformly scaled racing line at 75%. The FTG, being a reactive strategy, does not adhere to a predefined racing line. As a result, its lateral deviation is less relevant and is accordingly greyed out in the figure.

The MAP controller within the proposed race stack demonstrates superior performance, achieving the most consistent and fastest lap time at $\mu_{MAP} = 6.47s$ and a lateral deviation of 9.23 cm. This represents a roughly 4.5% speed increase and a 50% reduction in lateral deviation compared to the PP setup, showcasing that the MAP configuration's enhanced precision and consistency in trajectory tracking at elevated speeds. In contrast to the FTG system, the proposed stack offers a significant 1.58 s advantage in lap time, approximating a 24.4% improvement. This, coupled with a 50% lower standard deviation, demonstrates that it is a significantly faster and more consistent racing approach.

8.1.1 Computation - Time-Trials

Fig. 31, on the left-hand side, displays the latency histogram for the *ForzaETH Race Stack* during the *Time-Trials* phase. In this phase, *Opponent Estimation* and *Planning* modules are deactivated, under the presumption of an opponent-free track. Local planning is simplified to waypoint sampling from a pre-computed global racing line and can be neglected. Thus, *State Estimation* and *Control* are the primary contributors to the system's latency. The latency associated with state estimation is measured separately for *Cartographer-SLAM* and *SynPF*, highlighting their distinct computational effects. The two different state estimation systems, however, are only used mutually exclusively. Under this setup, the race stack

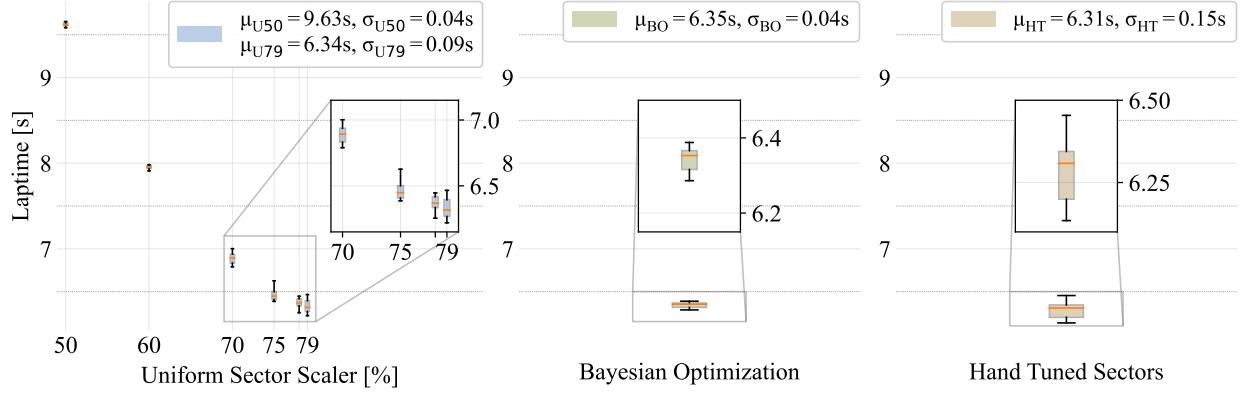


Figure 28: Lap times across 10 laps with the proposed *ForzaETH Race Stack* using the MAP controller. Uniform sector scalers in blue, BO obtained sector scalers in green, and hand-tuned sector scalers in bronze. The mean and standard deviation of all corresponding lap times are depicted in the top right, zoom in to enhance visibility when necessary.

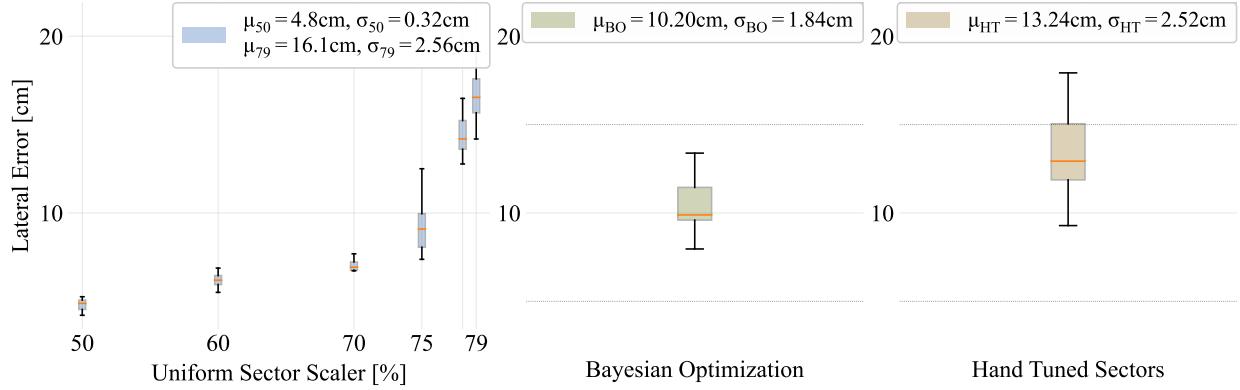


Figure 29: Lateral error across 10 laps, relative to the optimal racing line using the *ForzaETH Race Stack* with the MAP controller. Uniform sector scalers (blue), BO scalers (green), and hand-tuned scalers (bronze). The mean and standard deviation are in the top right.

demonstrates a maximum average latency of 7.5 ms, when using *Cartographer-SLAM* and evaluated on the Intel i5-10210U OBC.

Further, Fig. 31 on the right-hand side, showcases the CPU usage of the race stack, categorized by autonomy modules, based on data captured via the `cpu_monitor` library [alspitz, 2023], which internally uses the `psutil` process monitoring library. It encompasses various non-autonomy related `rosnodes` such as sensor drivers and dynamic reconfigure tools, categorized under *Sensors* and *Utils*. In this setup, the race stack's total CPU utilization reaches 268.23% or 169.41% for either the SLAM or the PF based localization backbone, on an Intel i5-10210U OBC (which can reach a maximum of 800% in terms of absolute CPU load). The *State Estimation* module is the primary computational load contributor, consuming 67.48% or 55.34% of the total racer stack CPU utilization (relative to the race stack computation) with *Cartographer-SLAM* or *SynPF*, respectively. In comparison, the *Control* module's computational demand is relatively low, accounting for approximately 7% of the total race stack utilization.

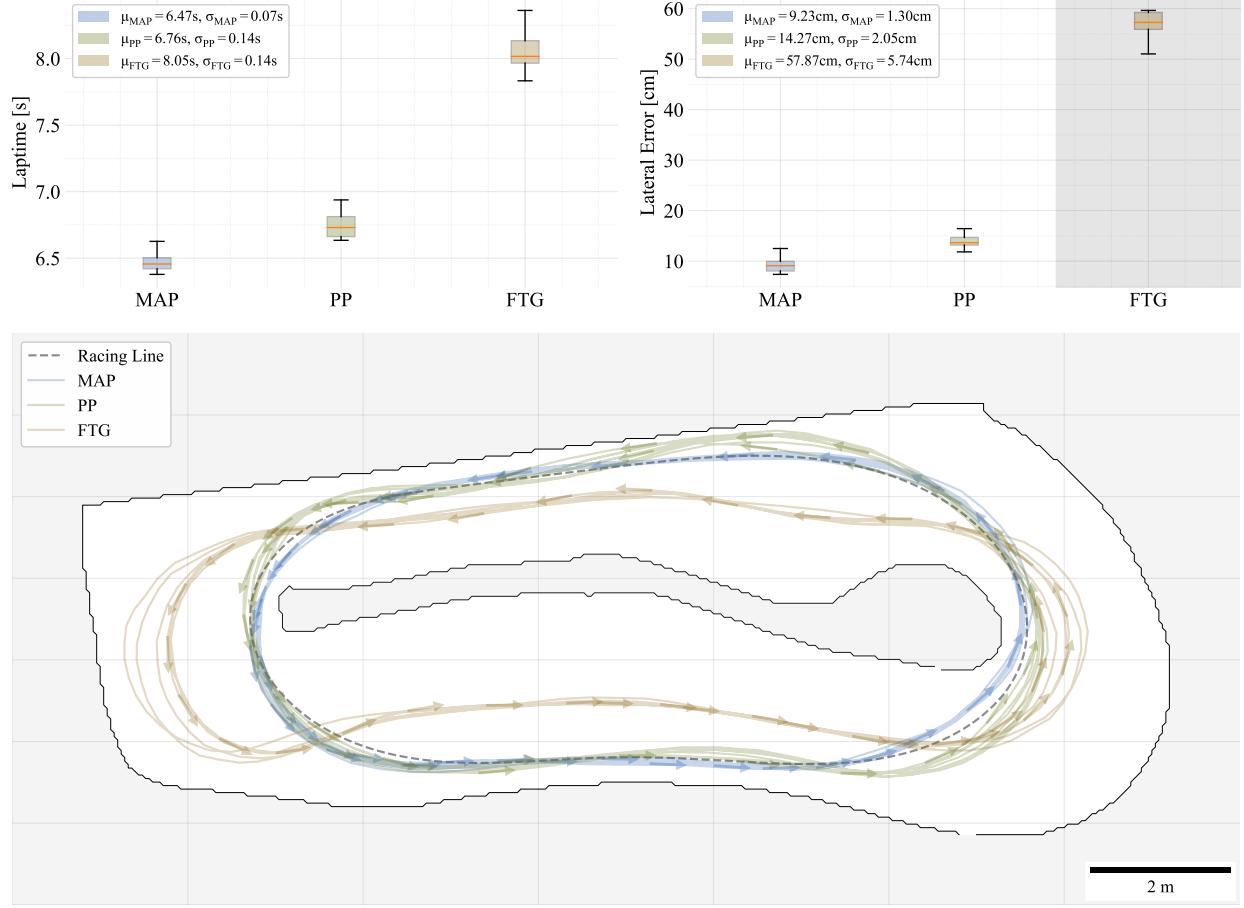


Figure 30: A lap time and lateral deviation comparison of the proposed *ForzaETH Race Stack* configuration against a different controller configuration of the race stack and a completely reactive FTG race stack. MAP denotes the proposed configuration, PP denotes a popular alternative for autonomous racing, and a fully reactive FTG race stack that does not adhere to the *See-Think-Act* cycle.

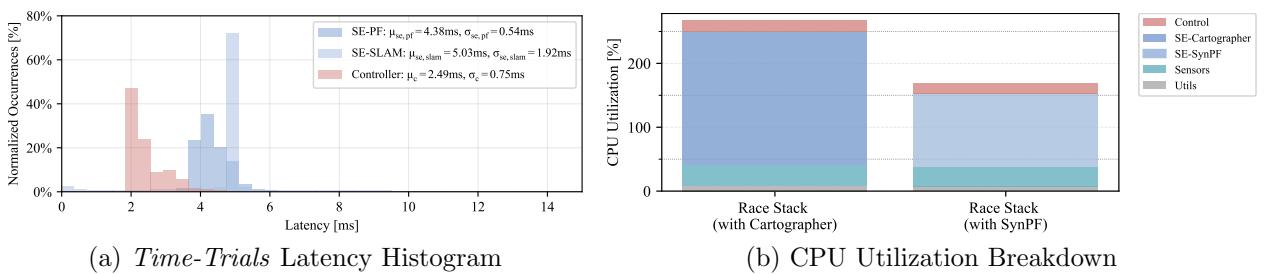


Figure 31: Latency histogram and CPU utilization breakdown of the *ForzaETH Race Stack* during the *Time-Trials* phase. Latencies of the *Control* module are obtained from the run with *Cartographer-SLAM*.

8.2 Head-to-Head Performance

The *Race Stack* was tested in a one-versus-one setup in the same racetrack as in Section 8.1. This assessment was used to understand under which circumstances the car can overtake an opponent driving a slower lap, and therefore understand when to enable the overtaking maneuvers. It is to be noticed, that in the setting of an *F1TENTH* competition overtaking is not a necessary move to complete in order to win a *Head-to-Head* race, as the rules state that the car that first completes a predefined number of laps (counted from the starting position) wins. Understanding the overtaking behavior of the car is however crucial in a future perspective, as multi-opponent racing — a natural next step in the *F1TENTH* competition — will enforce overtakes for winning.

In the experimental setup presented here, the ego-car used the nominal parameters described in the rest of this paper, exploiting the state machine described in Section 3.2 and with only minor modifications to the single components, detailed as follows. As a low deviation from the reference line is needed to robustly track the overtaking trajectories, the selected sector scalers were uniformly set at 75%, given the very close performance in terms of lap time (98% average speed over a lap when compared to the fastest settings, from Fig. 28) with the average lateral deviation below 10 cm (from Fig. 29). Furthermore, the trailing distance was set to the minimum value of 1.2 m, corresponding to the minimum recorded distance to the opponent of circa 1 m from Section 7.2, plus an additional 20 cm to account for robustness.

The opponent racecar was deployed with the same *Head-to-Head* capable *Race Stack* as the ego-car, with the only difference being the global scaler setup, which was set uniformly at lower values in order to match the lap time of the ego-car at lower percentages. The chosen opponents were as follows:

I Racing Line Opponent: The first opponent was chosen to be on the same racing line as ours. The velocity setups chosen corresponded to a speed over the lap 66%, 73%, and 80 % lower when compared to the fastest lap time achieved in Section 8.1.

II Altered Racing Line Opponent: The second opponent was on an altered racing line, which can be seen in Fig. 32a. This different racing line was generated by manually altering the track boundaries and then running the global planner described in Section 6.2. The velocity setups chosen corresponded to 66%, 73%, 80%, and 86% slower average velocity.

III FTG Opponent: The third and final opponent was set up to drive with an FTG algorithm, and the trajectory can be seen in Fig. 32b. The chosen FTG drove around at a speed comparable to circa 80% that of the fastest speed achieved on the track.

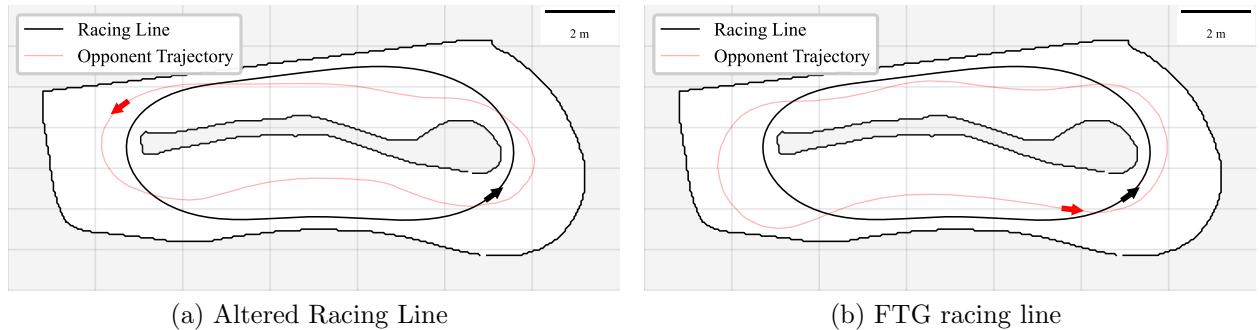


Figure 32: Opponent Racing Lines for the *Head-to-Head* analysis. The altered racing line was obtained by modifying the boundaries and then applying the minimum curvature racing line optimizer discussed in Section 6.2, while the FTG racing line is a result of the algorithm as described in Section 8.1.

The two trajectories different from the nominal racing line can be seen in Fig. 32. The cars were set up in the same racing scenario as the official *F1TENTH Head-to-Head* format, where the vehicles are positioned at opposite sides of the track and the first car finishing ten laps is deemed the winner. The main analyzed parameter was the overtake success. We defined an overtake maneuver as successful if the faster car managed to overtake the slower car by the end of the ten laps. An overtake is not successful if all the tentatives fail by the end of the test laps, and this result was caused either by a collision with the opponent or by lack of a speed advantage. In case of collision, the cars were reset from a standstill at predetermined points, with the offending car (the one causing the collision) situated behind the other. A further result reported is the eventual race outcome of the one-versus-one; it is however to be noted that since we tested with slower opponents the ego-car ended up winning all the runs — necessary with this algorithm, as without speed advantage it is not possible to overtake. The results of the overtaking experiments are shown in Table 8.

Opponent Type	[%]	Overtake Completed	Win
Same Racing Line	66	yes	yes
	73	yes	yes
	80	no ‡	yes
Altered Racing Line	66	yes	yes
	73	yes	yes
	80	yes	yes
	86	no †	yes
FTG	79	yes	yes

Table 8: Overtaking results of the full *Head-to-Head* system against opponents at slower velocity. †: the overtake was unsuccessful due to a collision. ‡: the overtake was unsuccessful due to a lack of speed advantage.

While it is clear that for slower opponents (66% and 73% average velocity) the overtaking procedure is carried out successfully, for faster opponents the ability to overtake is dependent on the racing line. For the altered racing line and the FTG opponent, a setup with higher velocity can still be overtaken, as the velocity advantage in the corner allows for the maneuver. For the opponent on the same racing line, instead, the overtake is carried out with a straight-line speed advantage, which in the case of this specific track is not enough to achieve enough advantage to safely complete an overtake when the opponent is set at a velocity approximately 80% of the ego-car.

While disabling overtake and setting the trailing gap at a safe 2 m length still remains a dominant strategy that would allow winning every *Head-to-Head* confrontation with an opponent at a slower average velocity than the ego agent, assessing the overtake capacities remains important. In a setting where overtaking is mandatory, the evaluation carried out here can help in deciding how to set the vehicle’s parameters. In case the opponent keeps a velocity comparable to 70% of ours, the overtaking strategy can be deployed effectively, while, in case of a faster opponent, a clear section with a speed advantage should be first identified to ensure that an overtake is feasible, and with any opponent above 80% of our average velocity, overtake should be avoided.

8.2.1 Computation - Head-to-Head

Fig. 33 displays the latency histogram of the *Head-to-Head* normalized to the total occurrences. Here all autonomy modules within the *See-Think-Act* cycle are active and the latency histogram gives insights into the robot’s operation. The highest average latency can be seen from the *Opponent Estimation* module with 6.39 ms. Interestingly to note, is the distribution shape of the *Planning* latency, as there are two noticeable peaks, one around 0 ms and then one after 3 ms. This is due to the fact that the local planner is only active if an opponent is present in front of the car.

During the *Head-to-Head* phase, all autonomy modules are involved and thus contribute towards the total computation, which amounts to 295.96% or 245.48% psutil CPU utilization on the Intel i5-10210U OBC,

respectively for a SLAM or PF based *State-Estimation* backbone. For the SLAM based localization system, relative to the total compute, *State-Estimation* accounts for 51.97%, *Opponent Estimation* for 14.45%, *Planning* for 10.51%, and *Control* for 6.66% of the entire race stack. The remainder is used for utility functionalities and sensor drivers. In the PF based system, the relative compute utilization is nearly identical as in the aforementioned SLAM scenario, albeit of course with a lower total consumption.

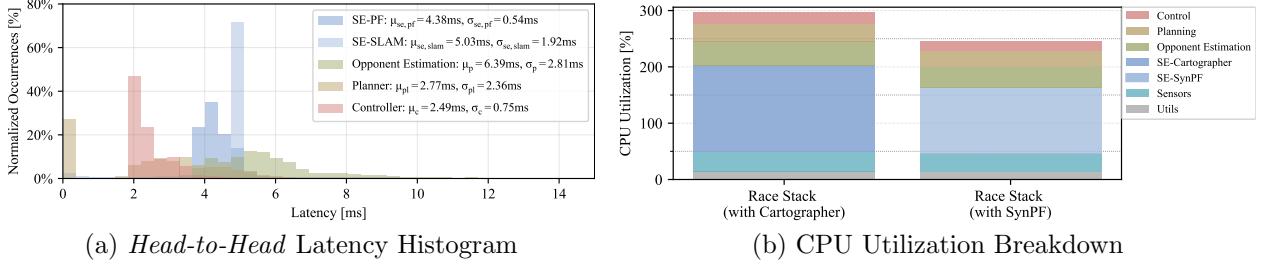


Figure 33: Latency histogram and CPU utilization breakdown of the *ForzaETH Race Stack* during the *Head-to-Head* phase. Latencies of the *Opponent Estimation*, *Planning*, and *Control* module are obtained from the run with *Cartographer-SLAM*.

9 Lessons Learned

Throughout the development of the *ForzaETH Race Stack*, many learnings could be drawn that improve the overall performance of the race stack. In this chapter, we distill the key lessons learned throughout this process. These lessons span various aspects from the development process to technical decision-making and have played pivotal roles in shaping the stack’s performance and reliability. By sharing these insights, we hope to provide valuable guidance for similar endeavors in the field of autonomous racing, aiding in the advancement of technology and operational strategies.

- I Upstream Affects Downstream:** The system architecture depicted in Fig. 3 demonstrates the inter-dependent nature of autonomy modules within the *See-Think-Act* cycle. The performance of upstream modules, particularly *State Estimation*, is critical as it lays the foundation for subsequent processing stages. For instance, inaccuracies in *State Estimation* will propagate errors into the *Control* module, regardless of the controller’s capabilities. This interdependence necessitates a holistic approach to the development and iterative refinement of all autonomy modules to enhance the overall system performance.
- II Balance Development Simplicity with Latency:** While a lower code execution latency is always desired, it is essential to strike a balance between development simplicity and performance. Rapid prototyping with `Python` can identify where low latency is critical and if deemed necessary `C++` for performance optimization can be used. This approach helps in efficiently allocating development resources and ensures that efforts are directed towards modules where speed is crucial, without unnecessarily complicating the system and the code maintenance.
- III Operational Simplicity:** The dynamic and unpredictable nature of real-world racing events demands an autonomy stack designed for operational robustness. Unanticipated variables, such as variations in surface traction or environmental conditions affecting sensor performance, must be accounted for. The stack should facilitate straightforward parameter adjustments, providing operators with the ability to swiftly adapt to changes in the racing environment. A transparent system with tunable knobs that produce predictable changes is essential for rapid adaptation and informed decision-making through key parameters in autonomous racing scenarios.
- IV Importance of Testing:** To build upon the previous point, figuring out the key parameters that yield full operational control of the robot, is only possible through thorough testing of the system. Testing is

not only vital for obtaining technical insights but also for training the operational handling of the pit staff. Therefore, fully rehearsing the race can allow the operators to spot technical, as well as operational flaws, which are invaluable information to increase performance during the development phase as well as the race itself. Leverage the simulation environment for as much as possible, but evaluate racing performance only on the physical platform, to not be misled by the *Sim-to-Real* gap.

- V **Team Spirit:** As this project has built upon many different student theses throughout their academic studies, we believe that upholding team spirit and passion for autonomous racing is a key enabler to achieving strong performance.

10 Conclusion and Future Work

This work presents the *ForzaETH Race Stack*. The presented system is designed for CotS hardware as proposed by the official *F1TENTH* competition. The system is fit for highly competitive autonomous racing in both a *Time-Trials* setting and *Head-to-Head* races. In this paper, we describe the essential components which are *Perception*, *Planning*, and *Control*. We detail the subsystems of each component, giving insights on how to configure those subsystems including information regarding the performance that can be expected from each system. The system has been extensively tested under various track conditions and has proven race-winning in official *F1TENTH* competitions.

The primary goal of this paper is to provide the first complete, robust, and accessible autonomy stack on CotS hardware for autonomous racing and research communities. By offering a race-proven system, we lower barriers to entry, enabling teams and researchers to focus on innovation and performance enhancement without the need for extensive resource investment. This approach enables inclusive access to advanced autonomous racing technologies, for growth and experimentation in the field.

Moving forward, the *Head-to-Head* racing domain offers expansive prospects for innovation. The *ForzaETH Race Stack* lays a robust foundation for delving into multi-opponent racing dynamics, a promising research field poised to yield significant advancements in autonomous racing strategies and technologies. Additionally, exploring the scalability of this stack for full-scale autonomous vehicles presents an interesting topic for future research, potentially broadening its applicability and impact in the domain of general autonomous systems.

References

- [Alcalá et al., 2020] Alcalá, E., Puig, V., Quevedo, J., and Rosolia, U. (2020). Autonomous racing using linear parameter varying-model predictive control (lpv-mpc). *Control Engineering Practice*, 95:104270.
- [alspitz, 2023] alspitz (2023). Cpu monitor. https://github.com/alspitz/cpu_monitor.
- [Althoff et al., 2017] Althoff, M., Koschi, M., and Manzinger, S. (2017). Commonroad: Composable benchmarks for motion planning on roads. In *Proc. of the IEEE Intelligent Vehicles Symposium*.
- [Amin et al., 2022] Amin, D. E., Priandana, K., and Hardhienata, M. K. D. (2022). Development of adaptive line tracking breakpoint detection algorithm for room sensing using lidar sensor. *International Journal of Advanced Computer Science and Applications*, 13(7).
- [Auer et al., 2002] Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256.
- [BARC, 2023] BARC (2023). Berkeley autonomous racecar. <https://goldeneye.berkeley.edu/barc.html>. [Online; accessed 6-November-2023].
- [Becker et al., 2023] Becker, J., Imholz, N., Schwarzenbach, L., Ghignone, E., Baumann, N., and Magno, M. (2023). Model- and acceleration-based pursuit controller for high-performance autonomous racing. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5276–5283.

- [Benjumea et al., 2021] Benjumea, A., Teeti, I., Cuzzolin, F., and Bradley, A. (2021). YOLO-Z: improving small object detection in yolov5 for autonomous vehicles. *CoRR*, abs/2112.11798.
- [Bertrand, 2005] Bertrand, G. (2005). On topological watersheds. *Journal of Mathematical Imaging and Vision*, 22(2-3):217–230.
- [Betz et al., 2023] Betz, J., Betz, T., Fent, F., Geisslinger, M., Heilmeier, A., Hermansdorfer, L., Herrmann, T., Huch, S., Karle, P., Lienkamp, M., Lohmann, B., Nobis, F., Ögretmen, L., Rowold, M., Sauerbeck, F., Stahl, T., Trauth, R., Werner, F., and Wischnewski, A. (2023). Tum autonomous motorsport: An autonomous racing software for the indy autonomous challenge. *Journal of Field Robotics*, 40(4):783–809.
- [Betz et al., 2019] Betz, J., Wischnewski, A., Heilmeier, A., Nobis, F., Stahl, T., Hermansdorfer, L., and Lienkamp, M. (2019). A software architecture for an autonomous racecar. In *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, pages 1–6.
- [Betz et al., 2022] Betz, J., Zheng, H., Liniger, A., Rosolia, U., Karle, P., Behl, M., Krovi, V., and Manghamaram, R. (2022). Autonomous vehicles on the edge: A survey on autonomous vehicle racing. *IEEE Open Journal of Intelligent Transportation Systems*, 3:458–488.
- [Bovik, 2009] Bovik, A. C. (2009). Basic Binary Image Processing. In *The Essential Guide to Image Processing*, pages 69–96. Elsevier, second edition.
- [Brunnbauer and Bader, 2019] Brunnbauer, A. and Bader, M. (2019). Traffic cone based self-localization on a 1 : 10 race car. In *Proceedings of the ARW & OAGM Workshop 2019*.
- [Brunnbauer et al., 2022] Brunnbauer, A., Berducci, L., Brandstätter, A., Lechner, M., Hasani, R., Rus, D., and Grosu, R. (2022). Latent imagination facilitates zero-shot transfer in autonomous racing. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 7513–7520.
- [Buckman et al., 2022] Buckman, N., Hansen, A., Karaman, S., and Rus, D. (2022). Evaluating autonomous urban perception and planning in a 1/10th scale minicity. *Sensors*, 22(18).
- [Bulsara et al., 2020] Bulsara, A., Raman, A., Kamarajugadda, S., Schmid, M., and Krovi, V. N. (2020). Obstacle avoidance using model predictive control: An implementation and validation study using scaled vehicles. In *WCX SAE World Congress Experience*. SAE International.
- [Burnett et al., 2021] Burnett, K., Qian, J., Du, X., Liu, L., Yoon, D. J., Shen, T., Sun, S., Samavi, S., Sorocky, M. J., Bianchi, M., Zhang, K., Arkhangorodsky, A., Sykora, Q., Lu, S., Huang, Y., Schoellig, A. P., and Barfoot, T. D. (2021). Zeus: A system description of the two-time winner of the collegiate sae autodrive competition. *Journal of Field Robotics*, 38(1):139–166.
- [Carron et al., 2023] Carron, A., Bodmer, S., Vogel, L., Zurbrügg, R., Helm, D., Rickenbach, R., Muntwiler, S., Sieber, J., and Zeilinger, M. N. (2023). Chronos and crs: Design of a miniature car-like robot and a software framework for single and multi-agent robotics and control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1371–1378.
- [Chalaki et al., 2022] Chalaki, B., Beaver, L. E., Mahbub, A. I., Bang, H., and Malikopoulos, A. A. (2022). A research and educational robotic testbed for real-time control of emerging mobility systems: From theory to scaled experiments [applications of control]. *IEEE Control Systems Magazine*, 42(6):20–34.
- [Costa et al., 2023] Costa, G., Pinho, J., Botto, M. A., and Lima, P. U. (2023). Online learning of mpc for autonomous racing. *Robotics and Autonomous Systems*, 167:104469.
- [Coulter, 1990] Coulter, R. (1990). Implementation of the pure pursuit path tracking algorithm.
- [Dhall et al., 2019] Dhall, A., Dai, D., and Van Gool, L. (2019). Real-time 3d traffic cone detection for autonomous driving. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 494–501.

- [Evans et al., 2023] Evans, B. D., Engelbrecht, H. A., and Jordaan, H. W. (2023). High-speed autonomous racing using trajectory-aided deep reinforcement learning. *IEEE Robotics and Automation Letters*, 8(9):5353–5359.
- [F1TENTH, 2023] F1TENTH (2023). F1tenth. <https://f1tenth.org/>. [Online; accessed 6-November-2023].
- [Fabian Christ and Lohmann, 2021] Fabian Christ, Alexander Wischnewski, A. H. and Lohmann, B. (2021). Time-optimal trajectory planning for a race car considering variable tyre-road friction coefficients. *Vehicle System Dynamics*, 59(4):588–612.
- [Finn, 2021] Finn, M. (2021). From accelerated advertising to fanboost: mediatized motorsport. *Sport in Society*, 24(6):937–953.
- [Foote and Purvis, 2010] Foote, T. and Purvis, M. (2010). REP: 103 standard units of measure and coordinate conventions. ROS Enhancement Proposals (REPs). [Online; accessed 12-Dec-2023].
- [Fröhlich and Carron, 2021] Fröhlich, L. P. and Carron, A. (2021). Bayesopt4ros: A Bayesian optimization package for the robot operating system. <https://github.com/IntelligentControlSystems/bayesopt4ros>. DOI 10.5281/zenodo.5560966.
- [Gerkey, 2023] Gerkey, B. (2023). amcl - ros wiki. [Online; accessed 19-June-2023].
- [Ghignone et al., 2023] Ghignone, E., Baumann, N., and Magno, M. (2023). TC-driver: A trajectory conditioned reinforcement learning approach to zero-shot autonomous racing. *Field Robotics*, 3(1):637–651.
- [Green, 1927] Green, M. (1927). Measurement of the moments of inertia of full scale airplanes. Technical report, Langley Aeronautical Laboratory.
- [Hart et al., 2014] Hart, K., Montella, C., Petitpas, G., Schweisinger, D., Shariati, A., Sourbeer, B., Trephan, T., and Spletzer, J. (2014). Roscar: Robot stock car autonomous racing. In *Proceedings of the 2014 Workshop on Mobile Augmented Reality and Robotic Technology-Based Systems*, MARS ’14, page 3–8, New York, NY, USA. Association for Computing Machinery.
- [Heetmeyer et al., 2023] Heetmeyer, F., Paluch, M., Bolliger, D., Bolli, F., Deng, X., Filicicchia, E., and Delbruck, T. (2023). Rpgd: A small-batch parallel gradient descent optimizer with explorative resampling for nonlinear model predictive control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3218–3224.
- [Heilmeier et al., 2020] Heilmeier, A., Wischnewski, A., Hermansdorfer, L., Betz, J., Lienkamp, M., and Lohmann, B. (2020). Minimum curvature trajectory planning and control for an autonomous race car. *Vehicle System Dynamics*, 58(10):1497–1527.
- [Hess et al., 2016] Hess, W., Kohler, D., Rapp, H., and Andor, D. (2016). Real-time loop closure in 2d lidar slam. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1271–1278.
- [Hyldmar et al., 2019] Hyldmar, N., He, Y., and Prorok, A. (2019). A fleet of miniature cars for experiments in cooperative driving. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3238–3244.
- [IAC, 2023] IAC (2023). Indy autonomous challenge. <https://www.indyautonomouschallenge.com/>. [Online; accessed 6-November-2023].
- [Jain and Morari, 2020] Jain, A. and Morari, M. (2020). Computing the racing line using Bayesian optimization. In *Proceedings of the 59th IEEE Conference on Decision and Control (CDC)*.
- [Jarvenpaa and Standaert, 2017] Jarvenpaa, S. and Standaert, W. (2017). Emergent ecosystem for radical innovation: Entrepreneurial probing at formula e. In *Proceedings of the 50th HICSS*.

- [Jones et al., 1998] Jones, D. R., Schonlau, M., and Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13:455–492.
- [Jung et al., 2023] Jung, C., Finazzi, A., Seong, H., Lee, D., Lee, S., Kim, B., Kang, G., and Shim, H. (2023). An autonomous racing system: Design, implementation, and analysis; team KAIST at the IAC. *Field Robotics*, 3(1):766–800.
- [Kabzan et al., 2020] Kabzan, J., Valls, M. I., Reijgwart, V. J. F., Hendrikx, H. F. C., Ehmke, C., Prajapat, M., Bühler, A., Gosala, N., Gupta, M., Sivanesan, R., Dhall, A., Chisari, E., Karnchanachari, N., Brits, S., Dangel, M., Sa, I., Dubé, R., Gawel, A., Pfeiffer, M., Liniger, A., Lygeros, J., and Siegwart, R. (2020). Amz driverless: The full autonomous racing system. *Journal of Field Robotics*, 37(7):1267–1294.
- [Karaman et al., 2017] Karaman, S., Anders, A., Boulet, M., Connor, J., Gregson, K., Guerra, W., Guldner, O., Mohamoud, M., Plancher, B., Shin, R., and Vivilecchia, J. (2017). Project-based, collaborative, algorithmic robotics for high school students: Programming self-driving race cars at mit. In *2017 IEEE Integrated STEM Education Conference (ISEC)*, pages 195–203.
- [Karle et al., 2023] Karle, P., Török, F., Geisslinger, M., and Lienkamp, M. (2023). Mixnet: Physics constrained deep neural motion prediction for autonomous racing. *IEEE Access*, 11:85914–85926.
- [Kong and Rosenfeld, 1996] Kong, T. Y. and Rosenfeld, A. (1996). *Topological Algorithms for Digital Image Processing*, volume 19 of *Machine Intelligence and Pattern Recognition*. Elsevier Science.
- [Konstantinidis, 2020] Konstantinidis, K. (2020). Development of a detection and tracking of moving vehicles system for 2d lidar sensors.
- [Li et al., 2023] Li, B., Fang, Y., Xu, T., Ma, S., Wang, H., Wang, Y., Li, X., Zhang, T., Bian, X., and Wang, F.-Y. (2023). Toward fair and thrilling autonomous racing: Governance rules and performance metrics for the autonomous one. *IEEE Transactions on Intelligent Vehicles*, 8(8):3974–3982.
- [Lim et al., 2024] Lim, T. Y., Ghignone, E., Baumann, N., and Magno, M. (2024). Robustness evaluation of localization techniques for autonomous racing.
- [Loetscher et al., 2023] Loetscher, M., Baumann, N., Ghignone, E., Ronco, A., and Magno, M. (2023). Assessing the robustness of lidar, radar and depth cameras against ill-reflecting surfaces in autonomous vehicles: An experimental study.
- [Lu et al., 2023] Lu, T., Ding, X., Liu, H., Wu, G., and Wang, L. (2023). Link: Linear kernel for lidar-based 3d perception. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1105–1115.
- [Macenski and Jambrecic, 2021] Macenski, S. and Jambrecic, I. (2021). Slam toolbox: Slam for the dynamic world. *Journal of Open Source Software*, 6(61):2783.
- [McElhoe, 1966] McElhoe, B. A. (1966). An assessment of the navigation and course corrections for a manned flyby of mars or venus. *IEEE Transactions on Aerospace and Electronic Systems*, AES-2(4):613–623.
- [Moore and Stouch, 2014] Moore, T. and Stouch, D. (2014). A generalized extended kalman filter implementation for the robot operating system. In *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer.
- [Muzzini et al., 2023] Muzzini, F., Capodieci, N., Ramanzin, F., and Burgio, P. (2023). Optimized local path planner implementation for gpu-accelerated embedded systems. *IEEE Embedded Systems Letters*, pages 1–1.
- [O'Kelly et al., 2019] O'Kelly, M., Sukhil, V., Abbas, H., Harkins, J., Kao, C., Pant, Y. V., Mangharam, R., Agarwal, D., Behl, M., Burgio, P., and Bertogna, M. (2019). F1/10: An open-source autonomous cyber-physical platform. *ArXiv*, abs/1901.08567.
- [Orfanidis, 1995] Orfanidis, S. J. (1995). *Introduction to signal processing*. Prentice-Hall, Inc.

- [O’Kelly et al., 2020a] O’Kelly, M., Zheng, H., Jain, A., Auckley, J., Luong, K., and Mangharam, R. (2020a). Tunercar: A superoptimization toolchain for autonomous racing. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5356–5362.
- [O’Kelly et al., 2020b] O’Kelly, M., Zheng, H., Karthik, D., and Mangharam, R. (2020b). F1tenths: An open-source evaluation environment for continuous control and reinforcement learning. In *NeurIPS 2019 Competition and Demonstration Track*, pages 77–89. PMLR.
- [Pacejka and Bakker, 1992] Pacejka, H. B. and Bakker, E. (1992). The magic formula tyre model. *Vehicle system dynamics*, 21(S1):1–18.
- [Park et al., 2004] Park, S., Deyst, J., and How, J. (2004). A new nonlinear guidance logic for trajectory tracking. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Reston, Virginia. American Institute of Aeronautics and Astronautics.
- [Raji et al., 2023] Raji, A., Caporale, D., Gatti, F., Giove, A., Verucchi, M., Malatesta, D., Musiu, N., Toschi, A., Popitanu, S. R., Bagni, F., Bosi, M., Liniger, A., Bertogna, M., Morra, D., Amerotti, F., Bartoli, L., Martello, F., and Porta, R. (2023). er.autopilot 1.0: The full autonomous stack for oval racing at high speeds.
- [Raji et al., 2022] Raji, A., Liniger, A., Giove, A., Toschi, A., Musiu, N., Morra, D., Verucchi, M., Caporale, D., and Bertogna, M. (2022). Motion planning and control for multi vehicle autonomous racing at high speeds. In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, pages 2775–2782.
- [Roborace, 2023] Roborace (2023). Roborace. <https://roborace.com/>. [Online; accessed 6-November-2023].
- [Seong et al., 2023] Seong, H., Chung, C., and Shim, D. H. (2023). Model parameter identification via a hyperparameter optimization scheme for autonomous racing systems. *IEEE Control Systems Letters*, 7:1652–1657.
- [Sezer and Gokasan, 2012] Sezer, V. and Gokasan, M. (2012). A novel obstacle avoidance algorithm: “follow the gap method”. *Robotics and Autonomous Systems*, 60(9):1123–1134.
- [Siegwart et al., 2011] Siegwart, R., Nourbakhsh, I. R., and Scaramuzza, D. (2011). *Introduction to Autonomous Mobile Robots*. The MIT Press, 2nd edition.
- [Srinivasa et al., 2019] Srinivasa, S. S., Lancaster, P., Michalove, J., Schmittie, M., Summers, C., Rockett, M., Smith, J. R., Chouhury, S., Mavrogiannis, C., and Sadeghi, F. (2019). MuSHR: A low-cost, open-source robotic racecar for education and research. *CoRR*, abs/1908.08031.
- [Srinivasan et al., 2020] Srinivasan, S., Sa, I., Zyner, A., Reijgwart, V., Valls, M. I., and Siegwart, R. (2020). End-to-end velocity estimation for autonomous racing. *IEEE Robotics and Automation Letters*, 5(4):6869–6875.
- [Stahl et al., 2019a] Stahl, T., Wischnewski, A., Betz, J., and Lienkamp, M. (2019a). Multilayer graph-based trajectory planning for race vehicles in dynamic scenarios. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 3149–3154.
- [Stahl et al., 2019b] Stahl, T., Wischnewski, A., Betz, J., and Lienkamp, M. (2019b). Ros-based localization of a race vehicle at high-speed using lidar. *E3S Web of Conferences*, 95:04002.
- [Strobel et al., 2020] Strobel, K., Zhu, S., Chang, R., and Koppula, S. (2020). Accurate, low-latency visual perception for autonomous racing: Challenges, mechanisms, and practical solutions. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1969–1975.

- [Thrun et al., 2007] Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L.-E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., and Mahoney, P. (2007). *Stanley: The Robot That Won the DARPA Grand Challenge*, pages 1–43. Springer Berlin Heidelberg, Berlin, Heidelberg.
- [Urmson et al., 2009] Urmson, C., Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Clark, M. N., Dolan, J., Duggins, D., Galatali, T., Geyer, C., Gittleman, M., Harbaugh, S., Hebert, M., Howard, T. M., Kolski, S., Kelly, A., Likhachev, M., McNaughton, M., Miller, N., Peterson, K., Pilnick, B., Rajkumar, R., Rybski, P., Salesky, B., Seo, Y.-W., Singh, S., Snider, J., Stentz, A., Whittaker, W. R., Wolkowicki, Z., Ziglar, J., Bae, H., Brown, T., Demirish, D., Litkouhi, B., Nickolaou, J., Sadekar, V., Zhang, W., Struble, J., Taylor, M., Darms, M., and Ferguson, D. (2009). *Autonomous Driving in Urban Environments: Boss and the Urban Challenge*, pages 1–59. Springer Berlin Heidelberg, Berlin, Heidelberg.
- [Voser et al., 2010] Voser, C., Hindiyeh, R. Y., and Gerdes, J. C. (2010). Analysis and control of high sideslip manoeuvres. *Vehicle System Dynamics*, 48(sup1):317–336.
- [Vázquez et al., 2020a] Vázquez, J. L., Brühlmeier, M., Liniger, A., Rupenyan, A., and Lygeros, J. (2020a). Optimization-based hierarchical motion planning for autonomous racing. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2397–2403.
- [Vázquez et al., 2020b] Vázquez, J. L., Brühlmeier, M., Liniger, A., Rupenyan, A., and Lygeros, J. (2020b). Optimization-based hierarchical motion planning for autonomous racing. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2397–2403.
- [Walsh and Karaman, 2018] Walsh, C. H. and Karaman, S. (2018). Cddt: Fast approximate 2d ray casting for accelerated localization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3677–3684.
- [Wan and Van Der Merwe, 2000] Wan, E. and Van Der Merwe, R. (2000). The unscented kalman filter for nonlinear estimation. In *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*, pages 153–158.
- [Wang et al., 2021] Wang, R., Han, Y., and Vaidya, U. (2021). Deep koopman data-driven optimal control framework for autonomous racing. In *ICRA 2021 Workshop Opportunitinties and Challenges with Autonomous Racing*.
- [Werling et al., 2010] Werling, M., Ziegler, J., Kammel, S., and Thrun, S. (2010). Optimal trajectory generation for dynamic street scenarios in a frenét frame. In *2010 IEEE International Conference on Robotics and Automation*, pages 987–993.
- [Wilson et al., 2020] Wilson, S., Glotfelter, P., Wang, L., Mayya, S., Notomista, G., Mote, M., and Egerstedt, M. (2020). The robotarium: Globally impactful opportunities, challenges, and lessons learned in remote-access, distributed control of multirobot systems. *IEEE Control Systems Magazine*, 40(1):26–44.
- [Wischniewski et al., 2021] Wischniewski, A., Euler, M., Gümüs, S., and Lohmann, B. (2021). Tube model predictive control for an autonomous race car. *Vehicle System Dynamics*, 60:1–23.
- [Wischniewski et al., 2022] Wischniewski, A., Geisslinger, M., Betz, J., Betz, T., Fent, F., Heilmeier, A., Hermansdorfer, L., Herrmann, T., Huch, S., Karle, P., et al. (2022). Indy autonomous challenge-autonomous race cars at the handling limits. In *12th International Munich Chassis Symposium 2021: chassis. tech plus*, pages 163–182. Springer.
- [Wischniewski et al., 2023] Wischniewski, A., Herrmann, T., Werner, F., and Lohmann, B. (2023). A tube-mpc approach to autonomous multi-vehicle racing on high-speed ovals. *IEEE Transactions on Intelligent Vehicles*, 8(1):368–378.

- [Wischnewski et al., 2019] Wischnewski, A., Stahl, T., Betz, J., and Lohmann, B. (2019). Vehicle dynamics state estimation and localization for high performance race cars**research was supported by the basic research fund of the institute of automotive technology of the technical university of munich. *IFAC-PapersOnLine*, 52(8):154–161. 10th IFAC Symposium on Intelligent Autonomous Vehicles IAV 2019.
- [Zhang et al., 2024] Zhang, T., Sun, Y., Wang, Y., Li, B., Tian, Y., and Wang, F.-Y. (2024). A survey of vehicle dynamics modeling methods for autonomous racing: Theoretical models, physical/virtual platforms, and perspectives. *IEEE Transactions on Intelligent Vehicles*, pages 1–24.
- [Zhu et al., 2022] Zhu, E. L., Busch, F. L., Johnson, J., and Borrelli, F. (2022). A gaussian process model for opponent prediction in autonomous racing. *arXiv preprint arXiv:2204.12533*.
- [Zong et al., 2023] Zong, Z., Jiang, D., Song, G., Xue, Z., Su, J., Li, H., and Liu, Y. (2023). Temporal enhanced training of multi-view 3d object detector via historical object prediction.
- [Özdemir and Sezer, 2017] Özdemir, A. and Sezer, V. (2017). A hybrid obstacle avoidance method: Follow the gap with dynamic window approach. In *2017 First IEEE International Conference on Robotic Computing (IRC)*, pages 257–262.

A Appendix

A.1 Parameters

This section aims to supply the reader with relevant hyperparameters used in each autonomy module.

A.1.1 State Estimation

I Localization Parameters:

Both the *Cartographer* and *SynPF* algorithms were tuned to minimize the lap time in a *Time-Trials* scenario over multiple maps. The parameters are shown as [min, max, tuned]:

- (a) **Cartographer:** *Cartographer* parameters are identical for the mapping and race stages of competition. However, at race-time, *Cartographer* is run in *localization-only* mode, which runs with lower latency and compute usage. It then uses the map of the racetrack obtained from the mapping stage.
 - `POSE_GRAPH.optimization_problem.odometry_rotation_weight` [0, ∞ , 0] - Weight of rotating incoming LiDAR scans away from the predicted pose given by odometry input in the scan-matching optimization problem.
 - `POSE_GRAPH.optimization_problem.odometry_translation_weight` [0, ∞ , 0] - Weight of translating incoming LiDAR scans away from the predicted pose given by odometry input in the scan-matching optimization problem.
 - `TRAJECTORY_BUILDER_2D.ceres_scan_matcher.rotation_weight` [0, ∞ , 8] - Cost of rotational deviation from the prior position in the scan matcher, with higher values increasing the penalty for rotational discrepancies.
 - `TRAJECTORY_BUILDER_2D.ceres_scan_matcher.translation_weight` [0, ∞ , 5] - Cost of translational deviation from the prior position in the scan matcher, with higher values increasing the penalty for translational discrepancies.
- (b) **SynPF** These parameters affect the variance of noise applied to the *SynPF* motion model and correspond to those in [Lim et al., 2024].
 - α_1 [0.0, 1.0, 0.5] - how much rotation affects rotation variance.
 - α_2 [0.0, 0.05, 0.01] - how much translation affects rotation variance.
 - α_3 [0.0, 1.0, 0.1] - how much translation affects translation variance.
 - α_4 [0.0, 5.0, 1.0] - how much rotation affects translation variance.
 - `lam_thresh` [0.01, 0.2, 0.1] - minimum translation for the TUM model to become effective.

II Extended Kalman Filter Parameters: The parameters relevant to the EKF, responsible for generating the velocity estimates, can in this case be subdivided into two categories: the covariances $\sigma_i^2 \in [0, \infty)$ associated with observations (zero elements are handled by the `robot_localization` package) and the configuration parameters which dictate what sources are to be considered by the EKF.

The `robot_localization` sensor fusion EKF configuration parameters are defined according to [Moore and Stouch, 2014] as:

$$\text{config} = \begin{bmatrix} x & y & z \\ roll & pitch & yaw \\ vx & vy & vz \\ vax & vay & vaz \\ alx & aly & alz \end{bmatrix}$$

where each entry is of type boolean, indicating whether or not the associated data source should be considered by the filter.

The EKF parameters intended for tuning are summarized as follows:

- $[\sigma_{VESC,x}, \sigma_{VESC,y}, \sigma_{VESC,\psi}] = [0.25, 0.5, 0.4]$ - the covariances associated with VESC control odometry x position, y position, and yaw angle, respectively.
- $[\sigma_{VESC,vx}, \sigma_{VESC,vy}, \sigma_{VESC,vaz}] = [0.02, 0.05, 0.0]$ - the covariances associated with VESC control odometry longitudinal, lateral, and angular yaw velocities, respectively.
- $[\sigma_{IMU,al}, \sigma_{IMU,va}, \sigma_{IMU,q}] = [0.0, 0.0, 0.0]$ - the covariances associated with IMU linear acceleration, angular velocity, and orientation measurements, respectively.
- The EKF configuration matrix for the VESC control odometry is defined as:

$$\text{config_odom} = \begin{bmatrix} \text{false} & \text{false} & \text{false} \\ \text{false} & \text{false} & \text{false} \\ \text{true} & \text{true} & \text{false} \\ \text{false} & \text{false} & \text{false} \\ \text{false} & \text{false} & \text{false} \end{bmatrix}$$

- The EKF configuration matrix for the IMU is defined as:

$$\text{config_imu} = \begin{bmatrix} \text{false} & \text{false} & \text{false} \\ \text{false} & \text{false} & \text{true} \\ \text{false} & \text{false} & \text{false} \\ \text{false} & \text{false} & \text{true} \\ \text{false} & \text{false} & \text{false} \end{bmatrix}$$

A.1.2 Opponent Estimation

I **Detection:** The parameters for the detection algorithm that were used to achieve the documented results are shown below as [`used`, `minimum`, `maximum`]:

- `min_obs_size` [40, 5, 300] - Minimum number of cloud points of an obstacle
- `max_obs_size` [0.5, 0.1, 1] - Maximum size of an obstacle in meters
- `max_viewing_distance` [9, 3, 10] - Maximal reliable distance of LiDAR measurements in meters

II **Tracking:** The parameters for the tracking algorithm that were used to achieve the documented results are shown below:

- The process Gaussian noise is defined as $Q = \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix}$ where $Q_1 = \begin{bmatrix} 1.95 \cdot 10^{-7} & 1.56 \cdot 10^{-5} \\ 1.56 \cdot 10^{-5} & 1.25 \cdot 10^{-3} \end{bmatrix}$ and $Q_2 = \begin{bmatrix} 7.81 \cdot 10^{-7} & 6.25 \cdot 10^{-5} \\ 6.25 \cdot 10^{-5} & 5 \cdot 10^{-3} \end{bmatrix}$
- The input Gaussian noise is defined as $R = \begin{bmatrix} 0.002 & 0 & 0 & 0 \\ 0 & 0.2 & 0 & 0 \\ 0 & 0 & 0.002 & 0 \\ 0 & 0 & 0 & 0.2 \end{bmatrix}$
- Proportional gains for Input: $P_{v_s} = 0.2$, $P_d = 0.02$, $P_{v_d} = 0.2$
- Target speed $v_{s,target} = v_{s,ego}(s) \cdot r$, with $v_{s,ego}(s)$ being the car's speed at the opponent s position and r a configurable ratio of this speed, set to 0.6

A.1.3 Planning

I **Global Planner:** The parameters for the global planner that were used to achieve the documented racing line are shown below as [`minimum`, `maximum`, `tuned`]:

- `curvlim` [0, ∞ , 1.0] - Maximum curvature of the vehicle in radians per meter
- `iqp_curverror_allowed` [0, ∞ , 0.1] - Maximum curvature error in radians per meter allowed between the curvature of the optimized path and `curvlim`
- `width_opt` [0, ∞ , 0.8] - Vehicle width in meters including a safety distance

- `stepsize_reg` [0, ∞ , 0.2] - Distance in meters between two points on the reference line during optimization

II Local Planner:

- $n^{spline} = 3$ - number of *preapex/postapex* points before and after the opponent's position, selected on the racing line to construct the overtaking spline.
- $(\Delta_1^{preapex}, \Delta_2^{preapex}, \Delta_3^{preapex}) = (2, 3, 4)$ - baseline distances of the *preapex* points from the opponent's d position.
- $(\Delta_1^{postapex}, \Delta_2^{postapex}, \Delta_3^{postapex}) = (4.5, 5, 5.5)$ - baseline distances of the *postapex* points from the opponent's d position.
- $\delta^{apex} = 0.4$ - extra lateral distance of the overtaking apex, to account for the controller's imperfections in tracking the reference spline.

A.1.4 Control

The controller parameters that were used to achieve the documented results are shown below as [`minimum`, `maximum`, `tuned`]:

I Longitudinal Controller:

- t_{la} [0, ∞ , 0.25] - Lookahead time in seconds to account for actuation and computation delay.
- λ_{lat} [0, 1, 1] - How much of the lateral error is taken into account to smoothly rejoin the trajectory. Higher values increase the dependence of the lateral error on the speed reduction.
- k_p [0, ∞ , 1] - Proportional gain for the error term e_{gap} in the calculation of v_{des} in the *Trailing* state.
- k_d [0, ∞ , 0.2] - Gain for the derivative term in the calculation of v_{des} in the *Trailing* state.
- v_{blind} [0, ∞ , 1.5] - Minimum velocity in m/s in the case when there is no LoS of an obstacle.
- g_{tar} [0, ∞ , 2] - Target gap in m to the to-be-trailed opponent.

II Lateral Controller:

- m [0, ∞ , 0.6] - Proportional term for the affine mapping of the velocity to the lookahead distance for the MAP controller.
- q [- ∞ , ∞ , -0.18] - Offset term for the affine mapping of the velocity to the lookahead distance.

A.2 Extended Kalman Filter Definition of Odometry Filter

The EKF model used for the odometry filter consists of an omnidirectional, three-dimensional, point-mass motion model. The state X and the discrete-time transfer function $f(X)$ used in the library are defined in the following equations:

$$\begin{aligned} X &= [x, y, z, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi}, \ddot{x}, \ddot{y}, \ddot{z}]^\top \\ X^+ &= f(X) = [x^+, y^+, \dots, \ddot{z}^+]^\top \\ x^+ &= x + (\dot{x} \cos \psi \cos \theta + \dot{y} (\cos \psi \sin \theta \sin \phi - \sin \psi \cos \phi) + \dot{z} (\cos \psi \sin \theta \cos \phi + \sin \psi \sin \phi)) \Delta_t \\ &\quad + \frac{1}{2} (\ddot{x} \cos \psi \cos \theta + \ddot{y} (\cos \psi \sin \theta \sin \phi - \sin \psi \cos \phi) + \ddot{z} (\cos \psi \sin \theta \cos \phi + \sin \psi \sin \phi)) \Delta_t^2 \\ y^+ &= y + (\dot{x} \sin \psi \cos \theta + \dot{y} (\sin \psi \sin \theta \sin \phi + \cos \psi \cos \phi) + \dot{z} (\sin \psi \sin \theta \cos \phi - \cos \psi \sin \phi)) \Delta_t \\ &\quad + \frac{1}{2} \left(\ddot{x} \sin \psi \cos \theta + \frac{1}{2} \ddot{y} (\sin \psi \sin \theta \sin \phi + \cos \psi \cos \phi) + \frac{1}{2} \ddot{z} (\sin \psi \sin \theta \cos \phi - \cos \psi \sin \phi) \right) \Delta_t^2 \end{aligned}$$

$$\begin{aligned}
z^+ &= z + (-\dot{x} \sin \theta + \dot{y} \cos \theta \sin \phi + \dot{z} \cos \theta \cos \phi) \Delta_t + \frac{1}{2} (-\ddot{x} \sin \theta + \ddot{y} \cos \theta \sin \phi + \ddot{z} \cos \theta \cos \phi) \Delta_t^2 \\
\phi^+ &= \phi + (\dot{\phi} + \dot{\theta} \sin \phi \tan \theta + \dot{\psi} \cos \phi \tan \theta) \Delta_t \\
\theta^+ &= \theta + (\dot{\theta} \cos \phi - \dot{\psi} \sin \phi) \Delta_t \\
\psi^+ &= \psi + \left(\dot{\theta} \frac{\cos \phi}{\cos \theta} + \dot{\psi} \frac{\sin \phi}{\cos \theta} \right) \Delta_t \\
\dot{x}^+ &= \dot{x} + \ddot{x} \Delta_t \\
\dot{y}^+ &= \dot{y} + \ddot{y} \Delta_t \\
\dot{z}^+ &= \dot{z} + \ddot{z} \Delta_t
\end{aligned}$$

where (x, y, z) represent the position, (ϕ, θ, ψ) represent the orientation as roll, pitch, yaw respectively, and, for ease of notation, the superscript $+$ represents the next-timestep state, e.g. $X^+ := X[k+1] = f(X[k])$. Furthermore, the next-states not present in the equation are assumed to be constant.

A.3 Race Results

By the end of 2023, the *ForzaETH* team did participate in three of the official *F1TENTH* competitions. The first participation was at the *ICRA Grand-Prix 2022* in Philadelphia where the team was able to reach fourth place. With the learnings of the first competition, the team was able to further improve the *Race Stack* and reach first place at the *German Grand-Prix 2022* which was held next to the Lausitzring (full-scale) race track. At the *ICRA Grand-Prix 2023* in London, the *ForzaETH* team was able to confirm its first place against a broader and more international range of opponents. Table 9 shows the official *F1TENTH* race rankings that the proposed race stack has been able to achieve — keeping in mind, that the stack improved throughout time.

The *Race Stack* presented in this paper is for most parts the stack that was used in the *ICRA Grand-Prix 2023*. The subsequent subsections describe in greater detail how the *Race Stack* configurations perform for both the *Time-Trials* and *Head-to-Head* phases of the competitions with performance evaluations for various test tracks, as well as the achieved race results of the *ForzaETH* team.

Competition	Year	Venue	# Teams	Ranking
ICRA Grand-Prix	2022	Philadelphia, PA, USA	20	4
German Grand-Prix	2022	Lausitzring, Germany	6	1
ICRA Grand-Prix	2023	London, UK	22	1

Table 9: Overview of the *F1TENTH* competitions where the *ForzaETH* team did compete. Note that at *ICRA Grand-Prix 2022* the team did start under the name *ForzaPBL*.

Competition Results

In the following, we are presenting the results of the most recent competition *ForzaETH* competed in, the *ICRA Grand-Prix 2023*. As previously mentioned the *Grand-Prix* style competitions consist of two phases, a *Time-Trials* phase and a *Head-to-Head* phase.

In the *Time-Trials* phase the teams have a predefined time window (typically 5 minutes) to achieve two goals. Firstly the teams need to achieve the fastest lap time and secondly, they need to reach the highest possible number of consecutive (uninterrupted) laps. The teams get two attempts (heats) to reach these goals. Only the better heat where both the lap time and number of consecutive laps are considered is used for the ranking. The ranking of the top 10 teams of the *Time-Trials* phase at *ICRA Grand-Prix 2023* is

given in Table 10. The results show that both a fast and robust system are required to reach first place. As can be seen from Table 10 most teams perform well in one of the two criteria, e.g. *HiPeRT Modena* reached second place in the number of consecutive laps, but only fourth place in the lap time and even more strikingly *Scuderia Segfault* reached second place in the lap time, but only eleventh place regarding the consecutive laps. Hence to score well in *F1TENTH*, one's car needs to both be fast and follow a trajectory consistently.

Team Name	Heat Nr.	# Cons. Laps ↑	Fastest Lap [s] ↓	Score			Qualifying Rank
				Cons. Laps ↑	Laptim ↑	Total ↑	
ForzaETH	1	25	11.54	15 (1)	20 (1)	35	1
HiPeRT Modena	2	22	13.37	14 (2)	17 (4)	31	2
AUTh Dependables	1	21	13.11	13 (3)	18 (3)	31	2
PUT-PPI	2	21	13.87	13 (3)	16 (5)	29	4
VAUL	1	20	14.84	12 (4)	14 (7)	26	5
Suzlab	1	19	14.36	11 (5)	15 (6)	26	5
Scuderia Segfault	1	8	12.83	5 (11)	19 (2)	24	7
UT AUTOMata	1	17	16.61	10 (6)	11 (10)	21	8
HMCar	1	15	16.60	9 (7)	12 (9)	21	9
HUMDA-SZE 2.	2	15	18.37	9 (7)	9 (12)	18	9
...

Table 10: *Time-Trials* results of the top 10 teams at the *ICRA Grand-Prix 2023*. The table only shows the better of the two heats for each team. Both the fastest achieved lap time and the number of uninterrupted laps are given. In separate columns, the achieved score for both the number of consecutive laps and the lap times are given, and the relative rankings for both categories are given in brackets.

During the *Head-to-Head* phase, two teams race in a 1v1 mode where the goal is to finish a given number of laps (typically 10) before the other team. To check for consistent driving of the teams the *Head-to-Head* race is done in a best-of-three mode for each pairing. In the current mode, the teams start on opposite sides of the race track, so overtaking can be avoided. In Table 11 the results of the *ICRA Grand-Prix 2023* are shown, as one can see *ForzaETH* was able to win each of the brackets in the first two races and never had to rely on a tie-breaker round.

Round	Team 1		Team 2		Score
Round of 16	1	ForzaETH	16	Technion F1Tenth Team 1	2 - 0
Quarter Final	1	ForzaETH	9	HMCar	2 - 0
Semi Final	1	ForzaETH	4	PUT-PPI	2 - 0
Final	1	ForzaETH	7	Scuderia Segfault	2 - 0

Table 11: Results of the *Head-to-Head* phase at the *ICRA Grand-Prix 2023*. For the teams, the Qualifying Rank and the team names are given. One can see that *ForzaETH* managed to win in all *Head-to-Head* pairings within the first two races of the best-of-three setting, never relying on a tie-breaker race.