UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

UNIVERSITY OF MODENA AND REGGIO EMILIA

Department of Physical, Computer and Mathematical Sciences (FIM)

Master's Degree in Computer Science (LM-18)

Accounting for Other Agents Behaviour in Motion Planning and Control for Multi-Agent Autonomous Racing

Supervisor:
Prof. Marko Bertogna
Co-supervisors:
PhD. Alessandro Toschi
PhD. Ayoub Raji

Candidate:
Simone Bondi
Student ID 196792

ACADEMIC YEAR 2025/2026

Contents

Abstract

In Multi-Agent Autonomous Racing competitions, cars with totally different Autonomous Driving software stacks are driven at the limit of handling to compete side-by-side in races in which the first car to reach the finish line wins. Motion Planning and Control in this context is usually performed using a prediction of the behaviour of other agents, which is inherently uncertain or even wrong especially when the agents do not behave under the same policy. This prediction, however, is often used as ground truth because considering uncertainties directly is not an easy problem. This work describes an existing Motion Planning and Control software pipeline consisting of an high-level heuristic-based corridor selection algorithm which uses the other agents predictions as ground truth, followed by Model Predictive Control (MPC) based planner and controller, and then extends it to better account for the behaviour of the other agents.

Acknowledgments

I would like to thank all current and past members of the UNIMORE Racing team for their awesome work without which this work wouldn't be possible. Even if i haven't even been part of the team for a year as of now, i want to thank them for the time spent together both in the office and on track.

Chapter 1

Introduction

1.1 Autonomous Driving and Autonomous Racing

Advanced Driver Assistance Systems (ADAS) and Autonomous Driving are interconnected applications of robotics that attempt to either significantly simplify the task of driving a car for a human driver in the case of ADAS, or create cars that can drive completely without human intervention in the case of autonomous driving. Both of these applications have the potential to improve the safety of roads, their accessibility, and efficiency; possibly more importantly, implementing them is an engineering challenge that acts as a testbed and birthplace for developments in most fields related to robotics, which ultimately improve all aspects of modern life including safety. Autonomous Driving competitions are events well suited to be alternatives to experimenting directly on roads - in such closed environments, it is possible and encouraged to test scenarios and solutions that would be undesirable or impossible to test directly on the roads, driving innovation. Of particular interest for this thesis is the task of driving at the limit of handling while also having to account for the behaviour of the other agents, which is most often unpredictable.

Multiple Autonomous Driving competitions have been held over the years since the first DARPA Grand Challenge in 2005, in which the participant vehicles were tasked with traveling on a 212 km off-road course in the shortest time. Among the most prominent, the Formula Student Driverless (FSD) competition is an annual student engineering competition held in multiple locations where teams from universities all around the world are tasked with designing, building and finally racing the vehicles on small tracks delimited by cones. In both of those competitions, interactions between vehicles are restricted, either by staggered starts and manual pausing in the DARPA Grand Challenge (and thus other vehicles are at most a static obstacle, as was the case with the Stanford car avoiding the CMU vehicle [16]), or by just having a single car on track at a time in the case of FSD.

Figure 1.1: Roboracer, a low cost 1/10th scale autonomous racecar based on a modified Traxxas RC car.

More recently, the Roboracer [15] (previously F1-tenth [11]), Indy Autonomous Challenge (IAC) [8] and Abu-Dhabi Autonomous Racing (A2RL) [1] competitions have been held. In these three competitions, equal cars race at the same time on the track with completely different software, and therefore how a vehicle accounts for the behaviour of the other agents has an enormous impact on the outcome of the competition. Roboracer is for example is held at multiple venues every year, and its head-to-head trial sees two low-cost one-tenth scale electric cars (Figure 1.1) with nearly equal hardware (apart from the onboard computer) racing together at the same time to complete some number of laps of a track in the shortest time possible, with no particular rules governing the behaviour of the cars other than the fact that they must not actively hinder each other (i.e. no dirty racing). The IAC then scaled this concept to full-scale racecars (Figure 1.2) around oval tracks with passing competitions between two cars with equal hardware, in which the defender and attacker have predefined roles Finally, A2RL scaled this to annual races in which six full-scale racecars (Figure 1.3) with equal hardware race at the same time on road circuits, particularly in the Yas Marina Circuit in Abu-Dhabi, to complete the designated number of laps in the shortest time possible, in which the cars do not have a predefined role but are instead racing freely much like in traditional motorsport.

1.2 Indy Autonomous Challenge

Indy Autonomous Challenge started as a set of simulation-based challenges backing all the way to 2019, which peaked in a Simulated race in June 2021, in which vehicles raced against each other in the Indianapolis Motor Speedway oval circuit in a traditional race setting. The Simulation race was meant as a qualification to participate in the real world race that took place in October 2021 in the

Figure 1.2: IAC Dallara AV24, an autonomous racecar based on the Dallara IL-15 Indy NXT chassis.

Figure 1.3: A2RL Dallara EAV25, an autonomous racecar based on the Dallara SF23 Super Formula chassis.

same circuit. This first real world race consisted of a time-trial event, in which the vehicles were tasked with recording the fastest lap-time around the circuit, and an obstacle avoidance challenge, in which two inflatable obstacles were placed along the track and vehicles had to perceive and avoid them.

In January 2022, at the Computer Electronic Show (CES), a second event was held in a novel format, the ”overtaking games”: two cars drove on the track at the same time, with the predefined roles of attacker and defender. The defender had to move at a constant, predefined speed while staying in the inner line of the track - the attacker was tasked with overtaking the defender in a specific section of the track, then switch roles once the pass was complete. The speed of the defender increased in each round, until the attacker could not overtake anymore in the designated section. In this race, the defenders reached speeds up to 240km/h.

In November 2022, at Texas Motor Speedway and in January 2023 at Las Vegas Motor Speedway the overtaking games were held again, but in these events the defender was free to choose its trajectory and the attacker had to maintain minimum safety distances. The Right-of-Way rule was introduced at this point, which states that when an attacker is within a rules-dependent distance behind an opponent and laterally offset with respect to the defender the attacker acquires the so-called ”Right-of-Way” and the defender must leave a rules-dependent amount of free space from the boundary to the edge of the vehicle on that side.

In June 2023, at the Milano-Monza Motor Show (MIMO) the cars ran for the first time in a road course, holding a time-trial event due to the increased difficulty of running in such a course, both due to issues with GNSS coverage, but also due to the increased difficulty of performing multi-agent racing.

The overtaking games were again held in January 2024 at Las Vegas Motor Speedway during CES 2024 and in 2024 at Indianapolis Motor Speedway, and in July 2025, time trials were held at Laguna Seca.

1.3 Abu Dhabi Autonomous Racing League

The A2RL competition started with the first race being held in April 2024 in the GP layout of the Yas Marina Circuit, and held again in November 2025 with the North layout. In this competition, cars are not restricted to stay on particular lines or at limited speeds, and thus have to overtake the other vehicles at the limit of vehicle performance. Even though a lot of issues were encountered during the 2024 event, especially due to insufficient testing, the 2025 event was largely a success, also seeing an overtake being performed by the UNIMORE team (Figure 1.4) and a crash due to an

Figure 1.4: UNIMORE Racing overtaking an opponent in the braking zone of Turn 5 of Yas Marina Circuit North Layout, during the 2025 A2RL Grand Finale. The turn precedes the longest straight in the circuit, with the cars moving at 240km/h at the braking point, before braking hard and generating 2.5G of deceleration. Images taken from the A2RL official broadcast [10].

opponent stopping in the middle of the track, the reasons of which will be discussed in Chapter 4 (Figure 4.2).

The car is a Dallara EAV-25 (Figure 1.3) and is based on the SF23 chassis from Super Formula, the top open wheel racing series in Japan and can reach speeds of around 260km/h and have enough downforce to generate decelerations under heavy braking and lateral accelerations of values between 2.5G and 3.0G. Figure 1.4 shows an overtake being performed right after a braking zone during which such decelerations are generated.

This competition attempts to be as close as possible to human motorsport, with it being held on technical road circuits, with six cars racing together with a rolling start in the format where the first to complete a specific number of laps wins. For the purposes of this thesis, it is interesting to take A2RL as an example for the rulesets that the autonomous driving stacks running on the vehicles should respect while racing against each other. These rules define a ”Defender” and ”Attacker” as roles that can are dynamically obtained during interactions and not as static roles like in the case of IAC - if a faster car is attempting to overtake a slower one, then the faster car is an attacker, and the slower is a defender.

1. Defenders are allowed to make only one change of trajectory to defend their position, and that change must be planned in advance, not in reaction to the attacking opponent. On the same note, weaving, that is performing repeated changes of directions in order to block an opponent is prohibited - this rule is shared in common with human motorsport;

2. When an attacker is within 15m behind and laterally offset with respect to the defender the

attacker acquires Right-of-Way: the defender must leave 3.5m of free space from the boundary to the edge of the vehicle on that side.

3. Before the vehicles are longitudinally aligned with each other, that is, when viewed orthogonally from the side the vehicles overlap and assuming the defender respects rules 1 and 2, the responsibility of avoiding a collision lies solely on the attacker. After that point, the responsibility is shared between the opponents.

4. Forcing an opponent off-track is strictly prohibited. Also this rule is shared with human motorsport.

The competition is divided in three parts: practice, qualification, sprint race, and grand finale. During practice, cars either run individually or in small groups to test sensors and basic safety functionality by staying on different lines and performing various maneuvers such as stop tests. During qualifications, each team is allocated a timeslot to perform their fastest lap, and the leaderboard determines the starting order for the Sprint Race. The sprint race is performed in a format similar to the grand finale and the leaderboard again determines its starting order.

1.4 High-level overview of an Autonomous Racing Stack

An autonomous racing stack is usually composed of four subsystems:

1. perception, in which the data coming from the exteroperceptive sensors is used to detect opponents, obstacles, and sometimes local features like track boundaries relative to the car - a tracker then associates detections to tracked objects to estimate their state;

2. state estimation, which takes the data from all sensors to estimate the ego vehicle state;

3. planning, which makes the high-level decision the high-level decision of where and how to move the vehicle to reach the final goal of winning the competition, for example planning how fast the vehicle should move to follow another car at some specific distance before attempting an overtake, how to behave around another opponent to respect the race regulations, and how to overtake an opponent without crashing or losing control - this last point is particularly important, as in racing, the car is expected to drive at the limit of the vehicle handling. Although such a controlled environment may be more predictable than, say, public roads, accounting for the behaviour of the other opponents is still a difficult problem, as their behaviour is hard to predict even though the goal is known.

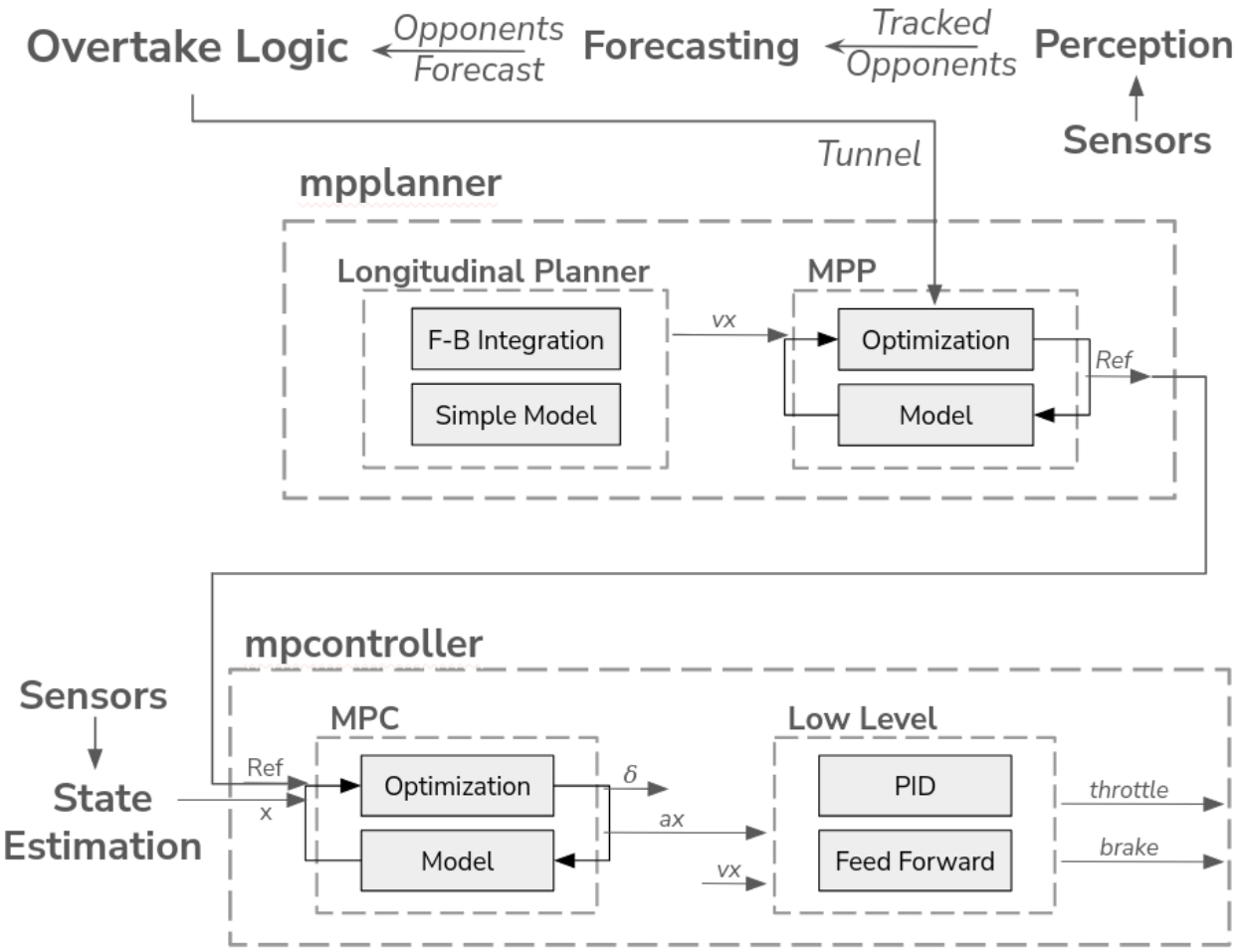


Figure 1.5: The ur.autopilot planning and control architecture.

4. control, which is tasked with determining the immediate action to follow the plan determined by the planner, by considering the current vehicle state and actuator and vehicle dynamics.

1.5 ur.autopilot

The contributions of this thesis are implemented in *ur.autopilot* [13], the autonomous driving stack of the UNIMORE Racing team. An architecture with some internal details of the components relevant to this thesis are shown in Figure 1.5 In this context, planning and control are two separate subsystems that run in real-time: at their boundary, planning computes a trajectory that considers opponents, track limits, vehicle dynamics and high-level instructions from humans, while control computes the vehicle input to track this trajectory, mostly focusing on vehicle and actuator dynamics.

1.5.1 Tracking References

Tracks are represented using splines in cartesian space representing the track borders, walls, and a reference line. Tracks are created offline: a human determines the track borders, walls, and some centerline, and then this data is passed to a global optimization algorithm which computes a laptime-optimal reference line. These reference lines serve both as an racing lines to track and as basis for constructing the local Frenet frame of references that are used in most of the stack.

All longitudinal planning is done online. *Longitudinal Planner*, the component used for this purpose, takes as input the curvature of a path and plans a speed profile with forward-backward planning, using a simplified model and a dynamic g-g diagram created analytically based on the Pacejka magic formula parameters and performance parameters decided either by a human or explored automatically. How the input curvature is computed will see an important contribution from this thesis in Chapter 2.

1.5.2 Planning and Control nodes

Each component of the stack is implemented as a ”node”, an independent Linux process that communicates with the others via message-passing routines. The control subsystem is implemented in the *mpcontroller* node, which takes the current state estimate of the ego vehicle from the state estimation subsystem and the planned trajectory from planning, and uses Model Predictive Control (MPC) to compute the optimal acceleration and steering angle that should be applied in order to track the planned trajectory, starting from the estimated state. In the same node, a low-level controller transforms the acceleration to throttle and brake inputs by a feedback controler based on a PI and feed-forward combination. Some details about the MPC problem formulation will be given shortly, while details about the MPC solver will be given in Chapter 4.

For what regards planning, there are two nodes of which knowledge is required:

- *forecasting*, which takes the current estimated positions of the opponents (estimated by the perception subsystem) to output the predicted trajectory each opponent will traverse;
- *mpplanner*, which takes the forecasting trajectories, determines a ”tunnel” (or corridor, a restriction of the track boundaries) to account for and overtake the other opponents, and uses MPC to find an optimal trajectory that moves inside this tunnel. Importantly, the starting state used by *mpplanner* is not the current ego state estimate, but its projection on the previously planned path. The starting state is only ever reset to the estimate if there is a large enough discrepancy, effectively operating almost exclusively in open-loop.

1.5.3 MPC Formulation

The MPC used in *ur.autopilot* is a Nonlinear MPC (NMPC). While the implementation details on the solver will be discussed in Chapter 4, the problem formulation is introduced here. The model used in the MPC is a non-linear dynamic bicycle model, in which the tyre forces are modeled with the Pacejka magic formula, and nonlinear soft constraints that limit the combined tyre forces are present [14]. The vector $\boldsymbol{x_i}$, that is the state at time step i, is the concatenation of the following vectors, omitting the subscript i for convenience of notation:

1. $[s, n, \mu]^T$: the pose in the world expressed in Frenet space. s is the progress along the reference path which defines the Frenet frame, while n and μ are the lateral offset and angular deviation with respect to that same frame.

2. $[v_x, v_y, r]^T$: the states pertaining to the vehicle dynamics, expressed in the body-fixed frame of reference and thus independent of the frame used for the pose. v_x and v_y are the longitudinal and lateral velocity respectively, and r the yaw rate.

3. $[\delta, D]^T$: the actuation states, which integrate the inputs $[d\delta, dD]^T$. δ is the front wheel steering angle according to the bicycle model, and D is the current acceleration.

The objective of MPC is tracking a trajectory, and therefore it includes tracking costs on all states. The most important ones for the contributions of this thesis are the costs on lateral deviation $(n_i - n_{ref_i})$, on angular deviation $(\mu_i - \mu_{ref_i})$, and on velocity error $(v_{x_i} - v_{x_{ref_i}})$. On one hand, *mpplanner* tracks the reference racing line that was computed offline (i.e. $n_{ref_i} = \mu_{ref_i} = 0$) and a longitudinal velocity profile that is computed with Longitudinal Planner from the racing line curvature, which this thesis will change in Chapter 2. On the other hand, *mpcontroller* tracks the output of *mpplanner* in all three states (i.e $\cdot_{ref_i} = \cdot_{mpp_i}$).

In *mpcontroller* uses an horizon long 63 steps with a time delta of 0.04s for a total of 2.52 seconds of horizon length, while *mpplanner* uses a longer horizon of 150 steps, for a total of 6 seconds. The *mpcontroller* nodes runs in a loop with a frequency of 100 Hz while *mpplanner* runs at 20Hz.

1.5.4 Overtake Logic

The ”tunnels” mentioned before are formulated as a box constraint on $\boldsymbol{n}$. The algorithm used to compute them and select one to be passed to MPC is called *Overtake Logic* [17], and it is separated in four ”blocks” which compose a pipeline:

1. *Interactions Manager*: interpolates track boundaries, forecasting trajectory and previous *mpplanner* horizon onto a common time horizon, specifically the time horizon that will be used in this iteration of MPC, and converts boundaries and forecasting to Frenet space. Then, it computes and maintains across iterations some flags that will be used for the heuristics in later stages, an important example of which will see a contribution from this work in Chapter 3. It then computes the ”interactions” between the ego vehicles and opponents, that is, the regions where ego is longitudinally aligned with opponents. Another example is marking each opponent as defender or attacker.

2. *Tunnels Generator*: constructs all tunnels, which is a combinatorial problem as each tunnel can pass to the right or to the left of each opponent, by shrinking the box constraint at each time step (dimension) in which there is an interaction with defender opponents, accounting for safety margins and the competition rules. Importantly, it accounts for giving the right-of-way space to attackers when required.

3. *Tunnels Filter*: determines whether tunnels are safe for use for overtaking, for example whether the car can fit through it. Importantly, it also decides whether an opponent is or will yield to respect the Right-of-Way or not; the sensitivity of this can be adjusted by changing an aggressiveness parameter, which will be used for an example scenario in Chapter 3.

4. *Tunnels Selector*: making use of heuristics and the data computed at the previous steps, it determines which tunnel should be used.

It must be noted that both *mpplanner* and *forecasting* use *Overtake Logic* - this can better account for more complex interactions.

1.6 Thesis overview

The contributions from this thesis are threefold, and focus on improving the behaviour of the stack in the presence of other agents.

1. a mechanism for computing the curvature signal dynamically for *Longitudinal Planner* is added in Chapter 2, to increase the stability in scenarios where *mpplanner* is required to plan paths that are very different from the reference line.

2. the mechanic with which one of the flags computed by *Overtake Logic Interactions Manager*, ”*Ego Localization*”, is changed in Chapter 3 to fix problems that were known to either hinder the performance or, worse, cause crashes in multi-agent scenarios.

3. the solver used for MPC is described in Chapter 4, and a work-in-progress experiment is discussed in which the optimizer is made aware of other opponents by adding a soft constraint on the distance to the other opponents - while in the current state it is not practical, it was included in the thesis due to the deep insights it provides on the current architecture, problems, and future works.

The first two contributions have been succesfully tested on-track at the A2RL 2025 Grand Finale. All three contributions have been tested in a high-fidelity simulation environment [9], based on a Functional Mockup Unit.

Chapter 2

Dynamic Curvature for Longitudinal Planning

2.1 Introduction

In the MPC problem formulation used in this work, a cost on tracking a reference speed profile is present. Tracking a prior speed profile provides two main benefits:

1. it is easy to obtain a desired behaviour from the vehicle: from simple profiles such as respecting some speed limit or slowing down gradually, to more complex profiles such as following another vehicle, staying at some performance level or in any case follow an arbitrary speed profile;

2. if the speed profile is approximately optimal and feasible, the numerical optimization can be attracted towards the limit of handling to increase the optimization solution stability and quality, resulting in higher performance.

In Chapter 1, the fact that *mpcontroller* tracks the speed profile (along with the path) planned by *mpplanner* was mentioned. However, what prior speed profile can be given to *mpplanner*? When driving at the limit of handling, the path traversed by a vehicle and the speed at which it is traversed are tightly coupled in such a way that this is a chicken-and-egg problem: computing an optimal speed profile requires knowledge of the optimal path the vehicle will take, but that is the result of the optimization problem which *mpplanner* is attempting to solve using MPC in the first place.

If the conditions can be predicted before driving, which in particular is partially true only when there are no other obstacles or other agents on the track, an optimal racing line can be computed

offline - then, assuming that the ego vehicle stays close to such racing line, the speed profile computed on such a path is approximately optimal.

Recall from Chapter 1 that *Long Planner* is the component used to plan a speed profile online. It takes $J \in \mathbb{N}$ samples $\boldsymbol{\kappa}_j$ of a signal $\kappa(s)$ representing the curvature of a path that starts at the current position of the vehicle sampled by arc length at uniform sampling interval δ_s, and assuming $J \cdot \delta_s$ is large enough such that the sampled signal covers the distance that will be traversed in the MPC horizon, outputs N_{MPC} samples $\boldsymbol{v}_i$ of the approximately optimal speed profile $v(t)$ uniformly spaced with intervals of the MPC time step, δ_t^{MPC}, with the constraint that $\boldsymbol{v}_0$ equals the current longitudinal velocity of the vehicle $x_{v_x}(t_0)$.

The existing implementation, for simplicity, assumed that the aforementioned ideal conditions held and directly sampled $\kappa(s)$ from the ideal racing line starting from the current progress $x_s(t_0)$ of the vehicle along the racing line; that is, $\boldsymbol{\kappa}_j = \kappa_{line}(x_s(t_0) + j\delta_s)$ where $\kappa_{line}(s)$ is the curvature of the racing line computed offline, for $j = 0 \ldots J$.

What, however, if these ideal conditions do not hold, for example what if there are other agents blocking some part of the track? The existing solution is inadequate when driving at the limit if the trajectories that do not lead to a crash have substantially larger curvature than the ideal racing line, and especially if a high velocity tracking cost is used, can lead to optimization instability, loss of control, or poor tracking performance, as the results will demonstrate. Figure 2.1 shows an example where the ego vehicle is restricted to travel in a tunnel about four meters wide on the left side of Turn 6, Yas Marina North Layout - this results in trajectories with higher curvature, which require very different speeds from the racing line. This can occur when an opponent is on the right hand side of the ego vehicle, for example.

2.2 Curvature Feedback

This thesis proposes a feedback mechanism to dynamically compute $\kappa(s)$ for *mpplanner*. At every execution, after the MPC solution is obtained, the resulting horizon is interpreted as a path. For each point i of this path the instantaneous curvature $\boldsymbol{\kappa}_i$ is computed then filtered with a low pass filter operating on the path arc length domain. Finally, the points $(\boldsymbol{x}_{\boldsymbol{s}_i}, \boldsymbol{\kappa}_i)$ are stored for the next iteration to serve as keypoints for interpolating $\kappa(s)$. This results in subsequent iterations of MPC converging to some a feasible path and speed profile.

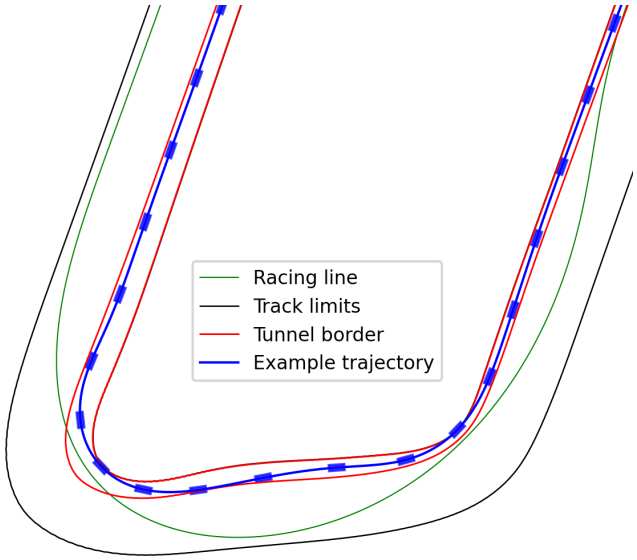


Figure 2.1: A narrow tunnel results in trajectories very different from the racing line.

2.2.1 Computing the raw curvature from the horizon

First, interpret the MPC horizon as a path in Frenet space:

$$\boldsymbol{p}_i^{frenet} = (\boldsymbol{x}_{\boldsymbol{s}_i}, \boldsymbol{x}_{\boldsymbol{n}_i})$$

For convenience of implementation, in this work this path is converted to Cartesian space. Note that this conversion may be avoided by reformulating the formulas for euclidean distance and curvature in Frenet space. Anyhow, the Frenet to Cartesian conversion can be performed by letting $\vec{N}(s)$ and $\vec{P}(s)$ be, respectively, the vector normal to and the cartesian position of the racing line defining the Frenet space, which can be sampled efficiently (recall the racing line is represented with splines), and then computing:

$$\boldsymbol{p}_i = \vec{P}(\boldsymbol{x}_{\boldsymbol{s}_i}) + \boldsymbol{x}_{\boldsymbol{n}_i}\vec{N}(\boldsymbol{x}_{\boldsymbol{s}_i})$$

Recall the well known formula for computing curvature of a generic cartesian path parametrized by arc length:

$$\kappa(s) = \frac{x'(s)y''(s) - y'(s)x''(s)}{\left(x'(s)^2 + y'(s)^2\right)^{3/2}} \tag{2.1}$$

to compute these derivatives, a measure of arc length s is required, and by letting $s^{(0)} = 0$ it can be approximated as

$$\boldsymbol{s}_i = \boldsymbol{s}_0 + \|\boldsymbol{p}_i - \boldsymbol{p}_{i-1}\|_2 \quad i = 1 \ldots (k-1)$$

Because MPC horizon is sampled uniformly in the time domain, when the vehicle is very slow subsequent path samples will be very close together and the differences between subsequent s will

be very small, meaning that when the coordinates are numerically differentiated the noise generated both by numerical error and by model linearization is greatly amplified. In practice what happens at such low speeds is not relevant, as the objective is driving at the limit at high speeds: it is therefore possible to define a minimum delta arc length constant $\Delta_s^{min} \in \mathbb{R}^+$ such that, if in one time step the car hasn't traveled more than this distance, that step can be ignored. Additioanlly, the final interpolation will be performed on the racing line arc length x_s in order to have a global reference - it is therefore also necessary to make sure that the interpolation samples are not too close to each other, and thus also a minimum delta progress constant $\Delta_{prog}^{min} \in \mathbb{R}^+$ can be defined.

Specifically, find the first step k of the MPC horizon s.t.

$$\boldsymbol{s}_k - \boldsymbol{s}_{k-1} < \Delta_s^{min} \vee \boldsymbol{x}_{\boldsymbol{s}_k} - \boldsymbol{x}_{\boldsymbol{s}_{k-1}} < \Delta_{prog}^{min}$$

or let $k = N^{MPC}$ if there is no such step, and from now on only consider the path samples with indices $i = 0 \dots k-1$.

Finally, numerically differentiate $\boldsymbol{p}_{\boldsymbol{x}i}$ and $\boldsymbol{p}_{\boldsymbol{y}_i}$ by central differences, appropriately reverting to forward and backward difference for the first ($i = 0$) and last ($i = k-1$) samples respectively, and plug each resulting sample into Equation 2.1 to compute the raw curvature samples $\boldsymbol{\kappa}_{\boldsymbol{raw}_i}$.

2.2.2 Filtering the curvature

The signal $\kappa_{raw}(s)$ now includes noise due to numerical errors and model linearization, which all mainly appear as high frequency noise that can be filtered out with a low pass filter. Importantly, also note that $\boldsymbol{s}_i$ are non-uniformly spaced and thus the signal is non-uniformly sampled in the arc-length domain, while it is actually uniformly sampled in the time domain as it comes from the MPC horizon. However, it was still chosen to filter in the arc-length domain because the signal is fundamentally representing a geometric path, for which the arc-length domain is natural and exhibits a simpler behaviour the consequences of which can be more easily reasoned about, particularly because of its independence from speed.

In this work, a continuous-time third order Butterworth filter with cutoff frequency $\omega_c = 0.65$ in state-space representation is approximated numerically by discretizing it using the Bilinear transform (Tustin's method) described in [4]. This filter is applied twice, once backwards and once forwards, to obtain zero phase shift, which is fundamental to ensure the curvature signal is not shifted. The filter type, order and cutoff frequency was chosen by manual inspection of the resulting signals, while the approximation method was chosen because of its stability guarantees for any stable filter and timestep size, which was deemed an useful property for prototyping this work - although there are potentially more computationally efficient numerical integration methods, the Bilinear method

was kept even after prototyping because the execution time is acceptable. When feeding a sample into such a filter, it is also necessary to provide a step size for integration. In this case the step size associated with the sample $\boldsymbol{\kappa_{raw}}_i$ is $\boldsymbol{s}_{i+1} - \boldsymbol{s}_i$.

The state space representation of the third-order Butterworth filter used in this work is:

$$A = \begin{bmatrix} -\omega_c & 0 & 0 \\ \omega_c & -\omega_c & -\omega_c \\ 0 & \omega_c & 0 \end{bmatrix} \quad B = \begin{bmatrix} \omega_c \\ 0 \\ 0 \end{bmatrix} \quad C = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$$

$$\dot{h} = Ah + Bu \qquad y = Cu$$

Note how in each of the two passes the filter state h has to be initialized to some initial state such that the output response is similar to that of the actual signal right from the beginning. Although there are more sophisticated methods such as those employed by the MATLAB *filtfilt* function [7], a cheap way to initialize the state is computing the steady state resulting from a step input signal with amplitude equal to the curvature at the start of the signal - let κ_0 be the desired initial output value, and given $\dot{h} = Ah + bu$, set $\dot{h} = 0$ to obtain:

$$h_0 = A^{-1}B\kappa_0$$

This results in an output which begins with the correct value, but which has derivative close to zero. In addition to this, the filter can be "primed" with a sufficient amount of samples that approximate the real function: in practice for the backwards pass a natural extension of the signal is the racing line curvature κ_{line}, to which it is desirable to have a smooth transition at the end of κ_{raw} for extrapolation. Define sufficiently high constant padding size $n_{padding} \in \mathbb{N}$ and constant padding spacing $\Delta_s^{padding} \in \mathbb{R}^+$ and compute uniformly spaced samples of the padding signal

$$\boldsymbol{\kappa_{pad}}_i = \kappa_{line}(\boldsymbol{x}_{\boldsymbol{s}_{k-1}} + i\Delta_s^{padding}) \qquad i = 0 \ldots n_{padding} - 1$$

The backwards pass can then be initialized at the steady state value of $\boldsymbol{\kappa_{pad}}_{n_{padding}-1}$, then primed with the other values in reverse order, $\boldsymbol{\kappa_{pad}}_i$ for $i = n_{padding} - 2 \ldots 0$.

For the forwards pass, there is no clear value that approximates the real function, as the samples of $\boldsymbol{\kappa_{raw}}$ are noisy and may even contain outliers, while the racing line curvature is not representative of the instantaneous curvature the vehicle is currently traveling at. For this reason the backwards pass is applied first, which results in the backwards filtered signal $\boldsymbol{\kappa_{bwd}}$. Now that the signal is cleaner, it is possible to initialize the forward pass from a mirrored version of the backwards filtered signal: more specifically, find the first sample p such that $\boldsymbol{s}_p > n_{padding}\Delta_s^{padding}$, or let $p = N_{MPC} - 1$ if there is none, then mirror the first p samples across the point $(0, \boldsymbol{\kappa_{bwd}}_0)$:

$$\boldsymbol{\kappa_{bwd}}'_i = \boldsymbol{\kappa_{bwd}}_0 - (\boldsymbol{\kappa_{bwd}}_i - \boldsymbol{\kappa_{bwd}}_0) \qquad i = 0 \ldots p$$

Finally, initialize the filter at the steady state with $\boldsymbol{\kappa_{bwd}}'_p$ and then prime the filter with $\boldsymbol{\kappa_{bwd}}'_i$ in reverse order, for $i = p-1 \ldots 0$. After applying first the backwards pass then the forwards pass, the result is the final sample vector $\boldsymbol{\kappa}$.

2.2.3 Interpolating the curvature

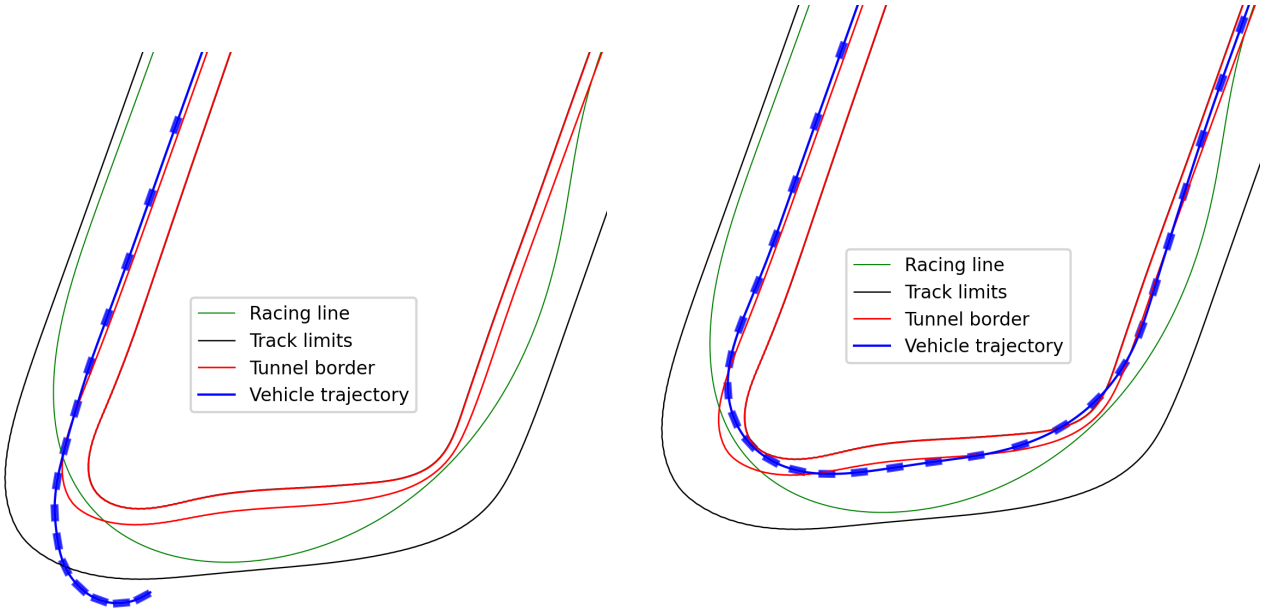
The sequence $S = ((\boldsymbol{x}_{s_i}, \boldsymbol{\kappa}_i))_{i=0}^{k-1}$ by construction satisfies $\boldsymbol{x}_{s_i} < \boldsymbol{x}_{s_{i+1}}$ for $i = 0 \ldots k-2$. It can therefore be used directly as keypoints for interpolation. Letting $L_S(s)$ be the well known linear interpolation function on S, the curvature at progress s can be computed as:

$$\kappa(s) = \begin{cases} L(s) & s \in [\boldsymbol{x}_{s_0}, \boldsymbol{x}_{s_{k-1}}] \\ \kappa_{\text{line}}(s) & \text{otherwise.} \end{cases}$$

2.2.4 Intuition behind the convergence properties

It's important to analyze the dynamics this feedback introduces. Assume the car is currently traveling on the long straight preceding the turn shown in 2.1, and focus on the iteration where the turn is just appearing in the MPC horizon. At the previous iteration the MPC predicted no corner, and therefore the curvature of the planned path was almost completely zero. To avoid leaving the tunnel, MPC will start slowing down and steering at the end of the horizon, but because the previous curvature was zero, the speed profile is attracting the MPC to continue accelerating. Because MPC is solving an optimization problem however, it will find a balance between the different costs and produce a speed prediction that is lower than the input speed profile. If this input speed profile is feasible for the path it was computed with, the MPC predicted path will have a higher curvature than the previous, resulting in a slower speed profile. This argument can be used for all iterations, so each subsequent iteration will result in a slower path and speed profile until convergence to a feasible pair.

Now assume that MPC starts with a suboptimal speed profile. The MPC problem formulation used in the work includes cost terms that can be argued to result in paths with lower curvature, and a progress cost term which can be argued to result in speeds larger than the speed profile. Therefore, the speed profile will not get worse, meaning there will be convergence towards a potentially optimal speed profile.



(a) Without (b) With

Figure 2.2: Without the proposed mechanism, the vehicle is too aggressive and locks up the front wheels, leaving the track. With it, the vehicle successfully navigates the scenario.

2.3 Results

To demonstrate the effectiveness of the approach, the same scenario previously shown in Figure 2.1 in which the car is forced to stay on the leftmost four meters of the track in the already tight Turn 6 of Yas Marina Circuit North layout, emulating an opponent blocking the right part of the road, is simulated with and without the proposed mechanism. Figure 2.2 shows that without the proposed mechanism, the vehicle fails to navigate the scenario. A snapshot of one of paths planned by MPC is shown in Figure 2.3, a comparison between the racing line and dynamic curvature and speed profiles are shown in Figure 2.4, and Figure 2.5 compares the raw curvature signal with the filtered one.

The algorithm was implemented from scratch in the C++ language, making heavy use of the Eigen library [6] for all linear algebra needs. As for computational performance, an one minute session of driving at high speed (to prevent the cutoff described in Section 2.2.1) was recorded: in this dataset, everything from converting the horizon to interpolating the curvature for *Longitudinal Planner* had an average execution time of 120 microseconds on a laptop with 32GB DDR4 RAM, an Intel i7-12700H with a maximum frequency of 4.7GHz, on Ubuntu 20.04 with the ”Performance” power mode. It’s important to note that no particular attention was posed to code performance optimization due to a lack of time, so the implementation can be easily improved.

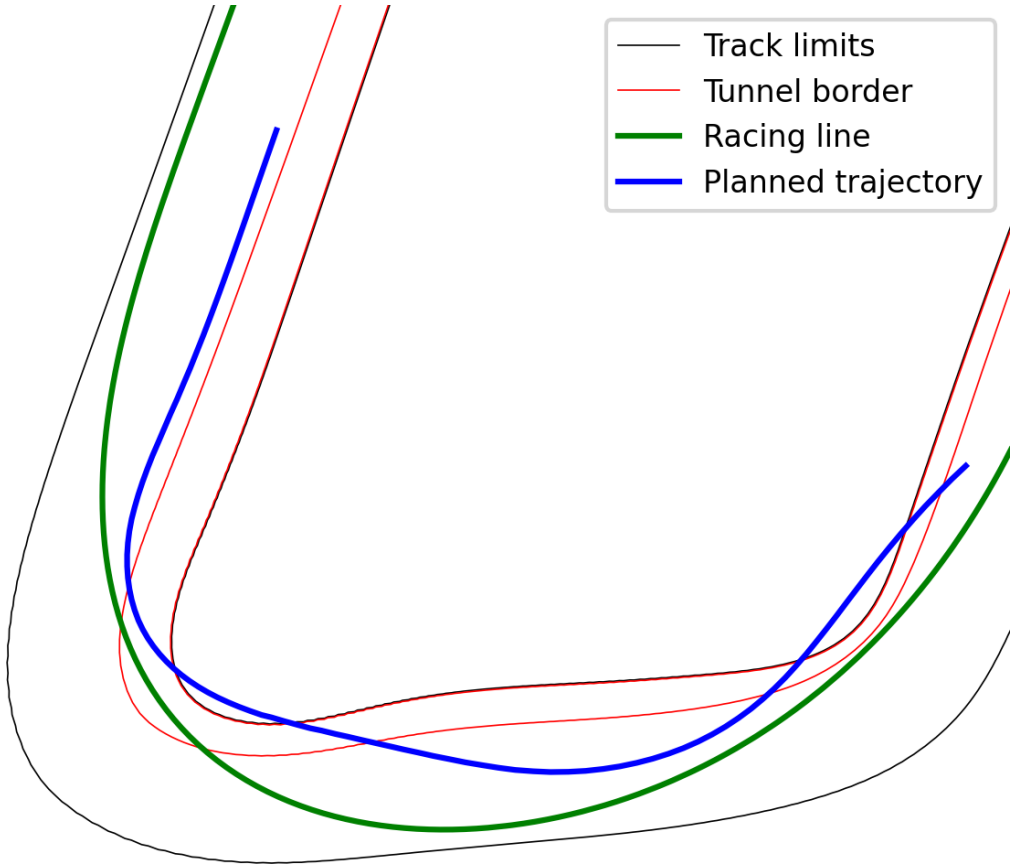


Figure 2.3: One of the paths planned by the MPC during the braking zone of the scenario described in Section 2.3, extracted from the same execution shown in Figure 2.2b, which was recorded with the dynamic curvature mechanism enabled. Importantly, compare the actual trajectory taken by the vehicle (Figure 2.2b) and this planned path: the planned path has not yet converged, which it will do in later iterations of MPC.

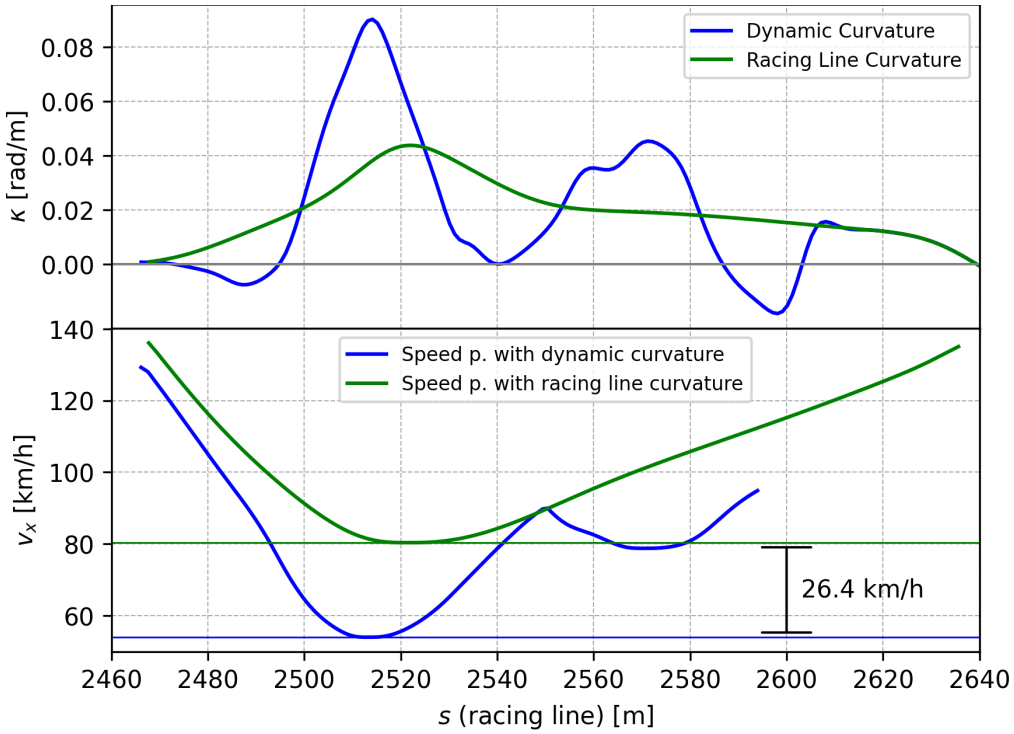


Figure 2.4: Comparison between the speed profiles computed on the racing line (green) and dynamic curvature (blue) for the path shown in Figure 2.3. Note the large difference between the (minimum) speeds at the first apexes of the two paths. If one believes that the path determined by the MPC horizon is optimal (which is highly subjective) the high speed difference can be interpreted as a measure of how exaggerated the racing line speed profile was for that path, showing how MPC was necessarily being attracted towards unfeasible solutions and explaining the loss of control.

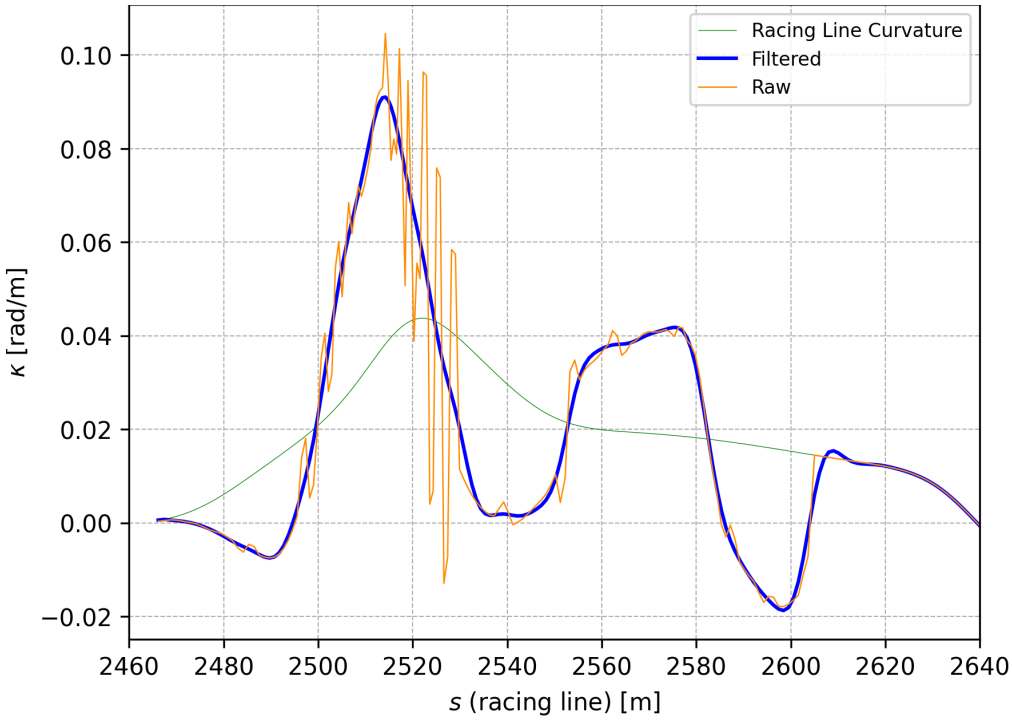


Figure 2.5: Comparison between the raw (orange) and filtered (blue) curvature for the path shown in Figure 2.3. Note that the filtered signal is not shifted with respect to the raw one, and how (subjectively) tightly follows the raw signal behaviour while filtering out what can be obviously considered noise.

Chapter 3

Forcing the overtaking side

3.1 Introduction

Recall from Chapter 1 that Overtake Logic determines the tunnel that will be passed to MPC. One of the most important choices it has to make is on what side to overtake each opponent, and in particular it is imperative that it does not switch overtaking side if that would cause a crash - this eventuality is exemplified by the scenarios shown in Figures 3.1 and 3.2, in which the ego vehicle causes a crash by switching tunnel while overtaking an opponent.

To be clear, the focus of this chapter is not on why the decision to switch tunnels was made, but on preventing decisions that could result in crashes. Unfortunately, this problem cannot be solved exactly without explicitly finding an optimal path that performs the switch, which is the problem that MPC solves in the first place - for example, with a single opponent two instances of MPC could be run in parallel to determine the feasibility of the tunnel switch and whether it is actually better to perform the switch, but this is computationally unfeasible. The approach currently used in *Overtake Logic* is based on heuristics, and this work proposes a modified version of the existing heuristic which will be shown to exhibit a list of desirable behaviours.

3.1.1 Ego Localization

The framework used to determine whether it is possible or not to switch sides is called *Ego Localization*. In this framework, there are four possible locations (”ego-locations”) where the ego vehicle is located with respect to each opponent: *back*, *front*, *left* and *right*. When the ego vehicle is *back* or *front* with respect to an opponent, it is free to choose the side on which to overtake the opponent. When an ego vehicle is instead *left* or *right* it is forced to select tunnels of which drivable areas are respectively on the left hand side or right hand side of the opponent. Clearly, the core of

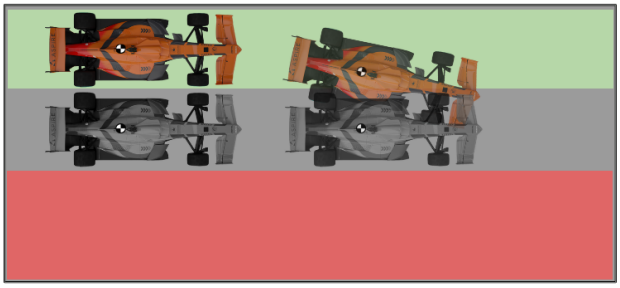


Figure 3.1: Ego (orange car) is side-by-side to an opponent (grey car) and both are moving at the same speed. Ego decides to switch overtaking side and switches from the green tunnel (left hand side) to the red one (right hand side), causing a collision.

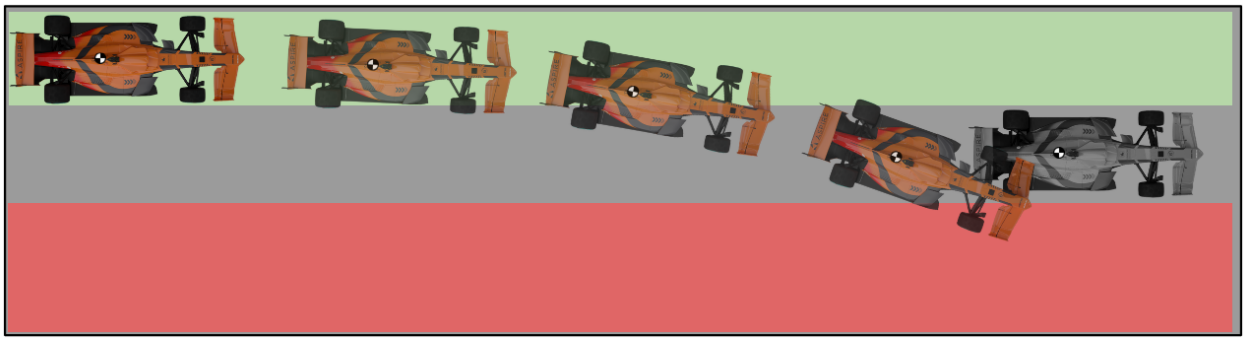


Figure 3.2: Ego is moving at high speed, planning to avoid on the left hand side the grey car, which is stationary in the middle of the track. Switching to the right hand side results in a crash because the car cannot physically swerve in time.

Ego Localization is the heuristic that determines the ego-location.

3.1.2 The Rectangle heuristic

The ego-location selection heuristic used prior to this work is shown in Figure 3.3 and is described in this section. Let x_{ego} be the position of the ego vehicle Center of Mass (CoM) and x_{opp} the position of the opponent CoM, both expressed in Frenet space with respect to the same racing line. Recall that the safety margins parameters in *Overtake Logic* determine the minimum distances expressed in Frenet space between the edges of the ego and opponent vehicles, and thus, when accounting for the sizes of the vehicles, they implicitly define a rectangular region around the opponent vehicle in which x_{ego} should not enter.

The track is first longitudinally divided into three complementary ranges, where the middle range coincides with the longitudinal range of the safety distance rectangle around the opponent. The front and back ranges fill all the remaining longitudinal extent of the track by dividing it into two equal parts. In case the ego CoM longitudinal progress $(x_{ego})_s$ is inside the front or back longitudinal range, the ego localization is determined to be *front* or *back* respectively.

When $(x_{ego})_s$ leaves the back or front longitudinal range and enters the middle one, the sign of the difference in lateral position $(x_{ego} - x_{opp})_n$ determines whether the left (positive sign) or the right (negative sign) ego-location is assigned. When $(x_{ego})_s$ is inside the middle longitudinal range in consecutive iterations, an hysteresis logic is applied to latch the ego-location: only if the ego vehicle completely moves to one side of the opponent the ego-location can switch between *left* and *right*. More specifically this happens when $|(x_{ego} - x_{opp})_n| > W$, where W is the minimum distance between the CoMs of two horizontal cars for them to touch - if the cars are equally sized, W coincides with one car width.

3.1.3 Issues with the Rectangle heuristic

The first issue with this approach is the sharp termination of the middle longitudinal region, and paradoxically, the most important issue is that it is too difficult to switch sides when inside it.

The sharp termination means that past the middle longitudinal region, no matter how laterally offset the ego vehicle is, it will always be free to choose side. To be competitive in road courses, small safety margins are strictly required, and while in the scenario shown in Figure 3.1 this approach works also for small margins, in the scenario shown in Figure 3.2 the vehicle is basically free to switch side until the very last moment.

For what regards the difficulty of switching sides when inside the middle longitudinal region,

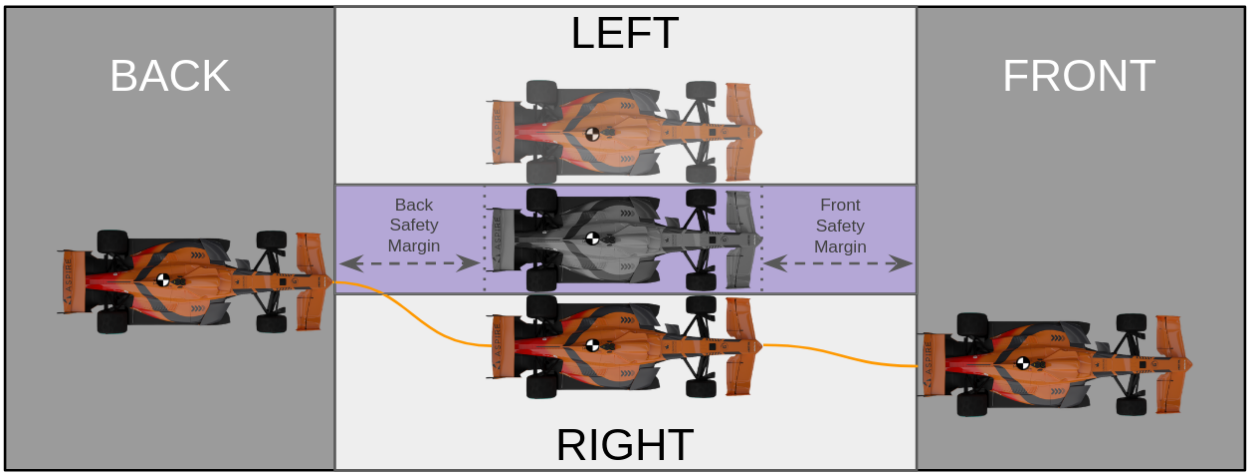


Figure 3.3: Rectangular ego-localization heuristic. The purple area the hysteresis region between *left* and *right*. Following the orange trajectory, the ego car goes through *back*, *right* and finally *front*.

Figure 3.4 shows an example where in Turn 2 of Yas Marina Circuit the much faster ego vehicle is attempting to overtake a relatively slow moving opponent on the inner part of the turn. Assuming the opponent will yield space given the competition rules, the ego vehicle decides to overtake on the left, and laterally offsets itself with respect to the opponent acquiring the *left* ego-location according to the rectangular ego-location approach. The opponent does not cooperate and continues to follow its racing line notwithstanding the approaching ego vehicle, closing the overtaking door and becoming laterally aligned with the ego vehicle: when laterally aligned, the ego vehicle maintains the *left* ego-location due to the hysteresis mechanism. In this situation, *Overtake Logic* would normally brake and follow the opponent until there is space on the left. For the sake of visualization this braking-and-waiting behaviour was overridden by increasing the aggressiveness of *Overtake Logic*, which then assumes the opponent will yield no matter what, causing a collision. In any case, the objectively better action is to instead switch to an overtake on the right.

3.2 The Cones heuristic

The proposed heuristic uses the idea that the closer the ego vehicle is to an opponent, the less lateral offset should be required to force a choice. This removes the sharp longitudinal region of the Rectangular heuristic, solving both of the problems described earlier.

Figure 3.5 shows the areas created by this heuristic. From the opponent CoM towards each of the four diagonal directions in Frenet space a ray is ”shot”, forming four conical zones. Whenever

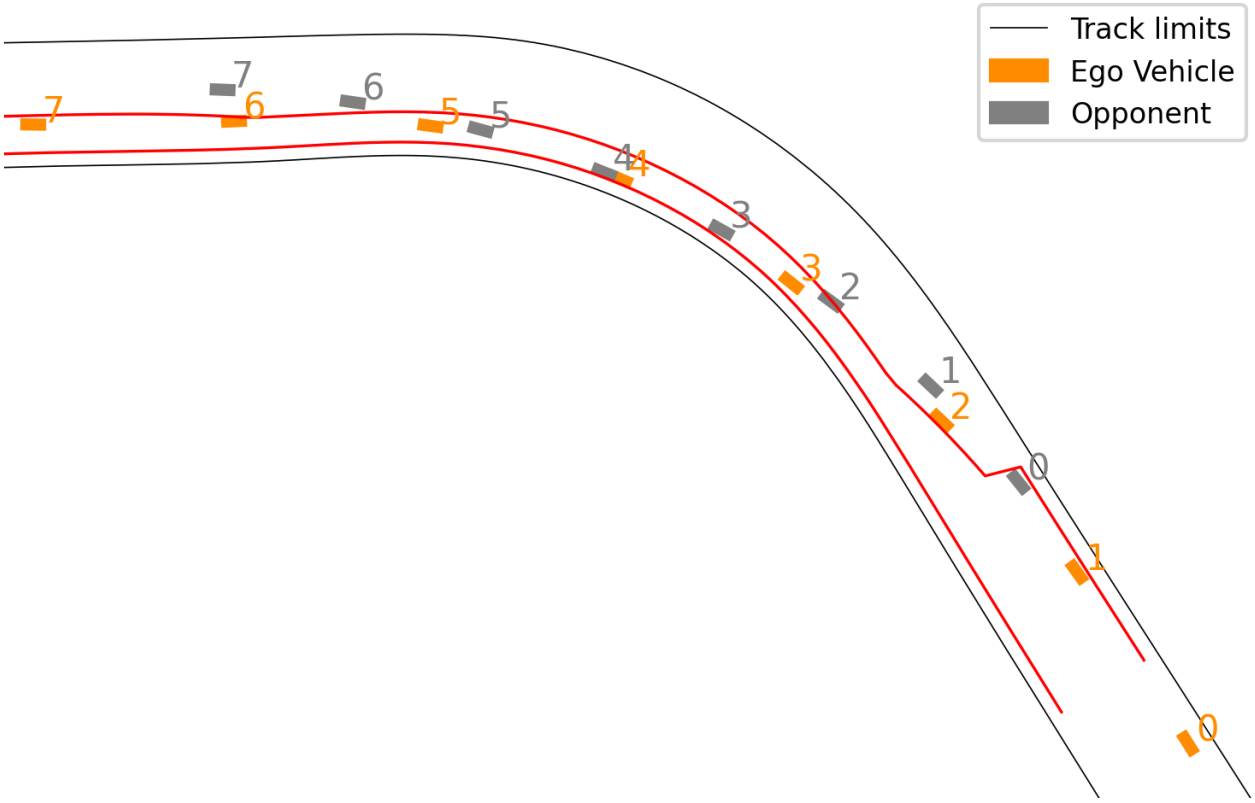


Figure 3.4: Overtake attempt by the ego vehicle on a non-collaborating opponent using the rectangular egoloc approach and the assumption the opponent will certainly yield space. Numbers show time steps k with a time delta of 1.6 seconds. The red lines show the tunnel chosen at $k = 0$, which is similar in shape to the tunnels chosen at the other time steps. Note how it overlaps with the opponent at $k >= 3$ because of the assumption that it will leave the overtaking door open as dictated by the Right-of-Way competition rule. At t=4, a collision occurs because the brake-and-wait behaviour was manually inhibited (the vehicles continue normally because collisions are not simulated).

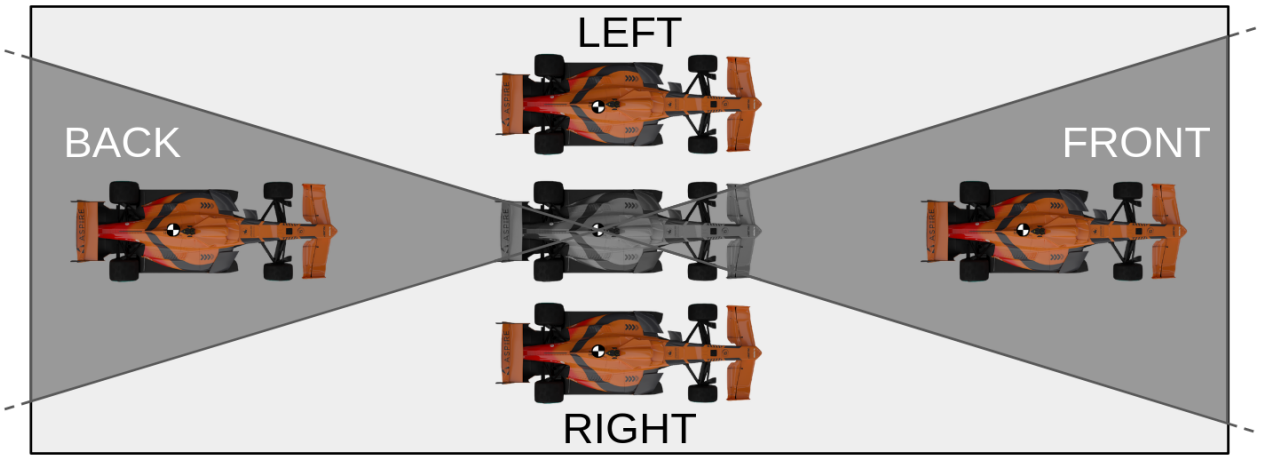


Figure 3.5: The cones ego-location heuristic.

the ego vehicle CoM is inside one of the lateral cones, the respective ego location is selected - an hysteresis is implemented when returning from a lateral (left/right) cone to one of the center (front/back) cones, such that the previous ego location is maintained if the vehicle is not completely inside one of the center cones, meaning it is overlapping with one of the rays.

Implementation

In practice, an approximation of this behaviour is implemented. Let m_{fl}, m_{fr}, m_{rl} and m_{rr} be the slopes of the front left, front right, rear left and rear right ray, respectively. Let $\Delta_s = x_{ego_s} - x_{opp_s}$ be the relative longitudinal offset between the ego vehicle and the opponent, and $\Delta_n = x_{ego_n} - x_{opp_n}$ the relative lateral offset. Choose the left and right slopes by checking the sign of Δ_s:

$$(m_l, m_r) = \begin{cases} (m_{fl}, m_{fr}), & \text{if } \Delta_s \geq 0 \\ (m_{rl}, m_{rr}), & \text{otherwise} \end{cases}$$

and sample the lateral offset of the left and right rays at Δ_s:

$$(n_l, n_r) = (m_l|\Delta_s|, -m_r|\Delta_s|)$$

Whether the ego vehicle is in the left cone corresponds to the expression $\Delta_n >= n_l$, while the right cone corresponds to $n_r >= \Delta_n$. Whenever one of these expressions is true, the ego-location changes to *left* or *right*, respectively. If this iteration is the first one in which the opponent is seen (that is, there is no previous ego-location) and both expressions are false, then the ego-location is set to *front* or *back* depending on the sign of Δ_s.

In subsequent iterations, an hysteresis mechanism is implemented for returning from the lateral cones to the center one. Let W be the width of the ego vehicle, the ego-location changes to *front* or

back if and only if $n_l - W/2 > \Delta_n > W/2 + n_r$, meaning the change happens only if ego is inside the center cone by at least half the width of the vehicle on the n axis. Figure 3.6 shows this heuristic and visualizes how the ego-location decision is made.

Dynamic slopes

An interesting property of this implementation is that the cone slopes can be computed dynamically. In this work, the front and rear slopes are made uniform to have two slopes m_f and m_r, that is $m_f = m_{fl} = m_{fr}$ and $m_r = m_{rl} = m_{rr}$.

These are computed as proportional to the speed difference between ego and opponent $\Delta_v = x_{ego_v} - x_{opp_v}$, each linearly interpolating between two points $(\Delta_{v_{min}}, m_{min}), (\Delta_{v_{max}}, m_{max})$, with separate parameters for the front and rear. When $\Delta_v < \Delta_{v_{min}}$ or $\Delta_v > \Delta_{v_{max}}$, m_{min} or m_{max} is maintained, respectively.

The points are parameters to be determined manually, and one clear guideline that can be identified is that the back slopes should decrease with larger Δ_v, i.e. less lateral offset should be required to become left or right, while the front slopes should increase with larger Δ_v, i.e. more lateral offset should be required to remain left or right after overtaking an opponent. Consider the example previously shown at the beginning of this chapter in Figure 3.2: Δ_v is very high, and intuitively ego vehicle should be considered left or right much earlier than if Δ_v were low - at the same time, after overtaking the opponent there should be as much freedom as possible.

Figure 3.6 shows the cones in the same scenario as Figure 3.1, where the opponent has come to a standstill in the middle of the track and the ego vehicle is moving at high speed. Note the size difference between the back and front cones.

Edge cases

An important edge case occurs when the slope parameters have high enough values: the ego-location may be *back* or *front* even when the vehicles are longitudinally aligned. An example of such edge edge case occurs in the previously shown Figure 3.6, when the ego vehicle acquires the *front* egoloc even if it is still longitudinally aligned with the opponent. This is particularly bad when $|\Delta_v|$ is small, as ego may swerve into the back or front of the opponent.

This can be fixed by introducing a longitudinal region in which the cars are forced to be left or right when they are longitudinally aligned, albeit with a much shorter region than what was used in the Rectangular heuristic. In this implementation, a small additional margin of $0.5m$ is used, that is, assuming the two vehicles are equally sized, when $-0.5m - W <= \Delta_s <= W + 0.5m$, a central ego-location is changed to a lateral ego-location depending on the sign of Δ_n, and switching from a

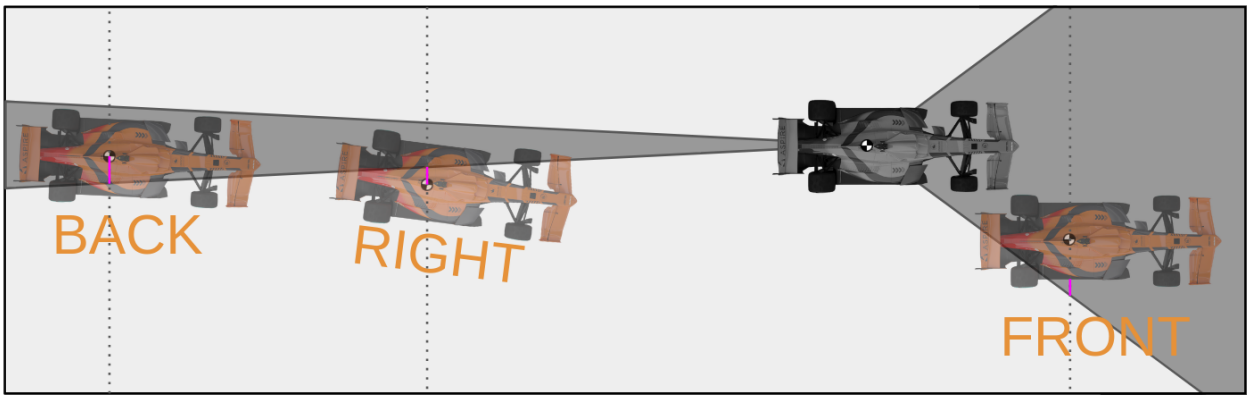


Figure 3.6: Cones ego-location during an avoidance at high Δ_v. The pink lines show how much Δ_n exceeds the threshold of the respective condition. Assume the opponent was first seen at the first step, and note how in the first iteration the full Δ_n is used, while in the last, half car width is removed due to the hysteresis mechanism.

lateral ego-locations to a central ego-location is inhibited. Switching between lateral ego-locations is possible, but an hysteresis mechanism is added such that this is possible only if $|\Delta_n| > W/2$.

3.3 Results

The same scenario shown in Figure 3.4 is simulated with the cones ego-location. The ego vehicle initially acquires ego-location *left*, then when the opponent laterally aligns with ego, the ego-location switches to *back* and ego is free to move to the right, comfortably completing the overtake without causing a collision or ever slowing down. The results are shown in Figure 3.8.

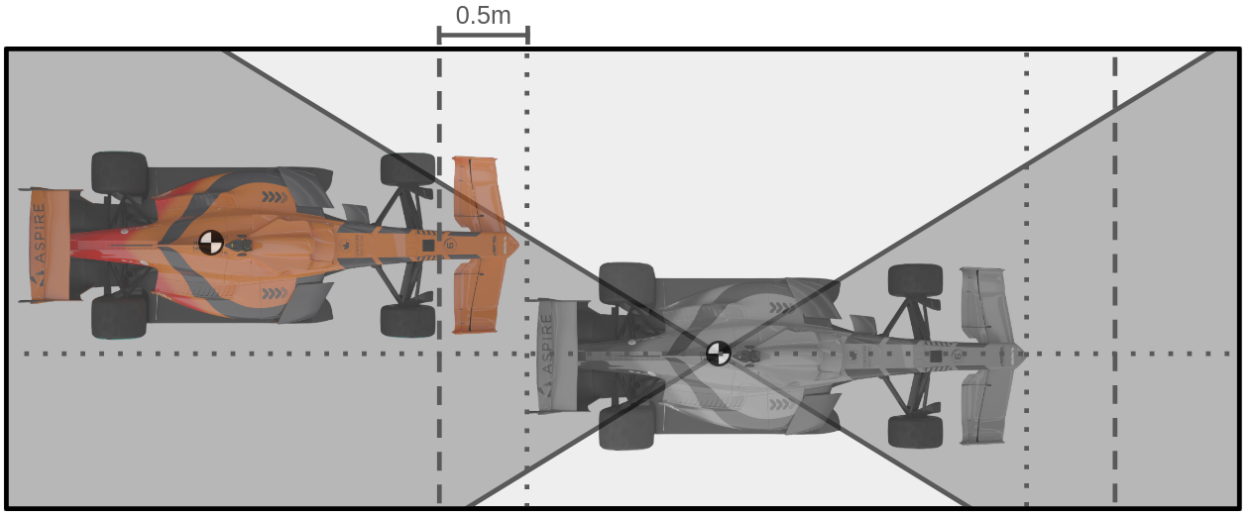
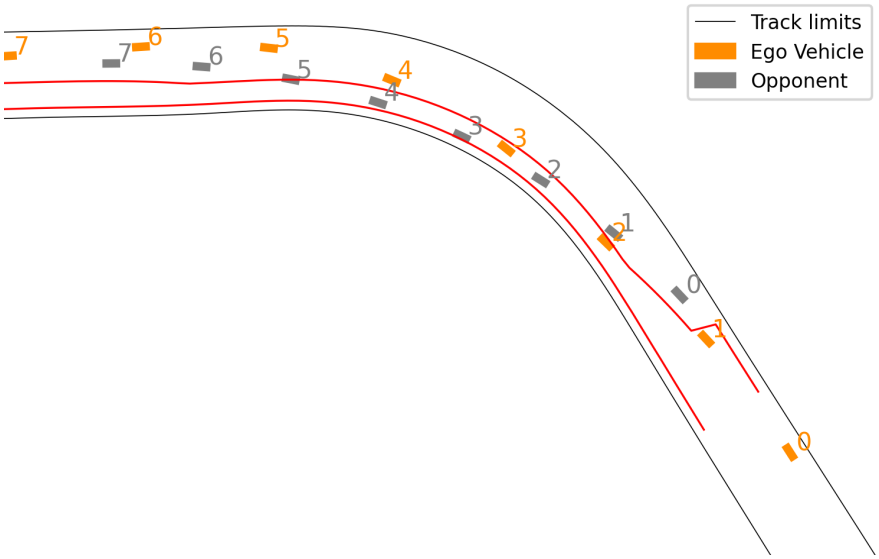
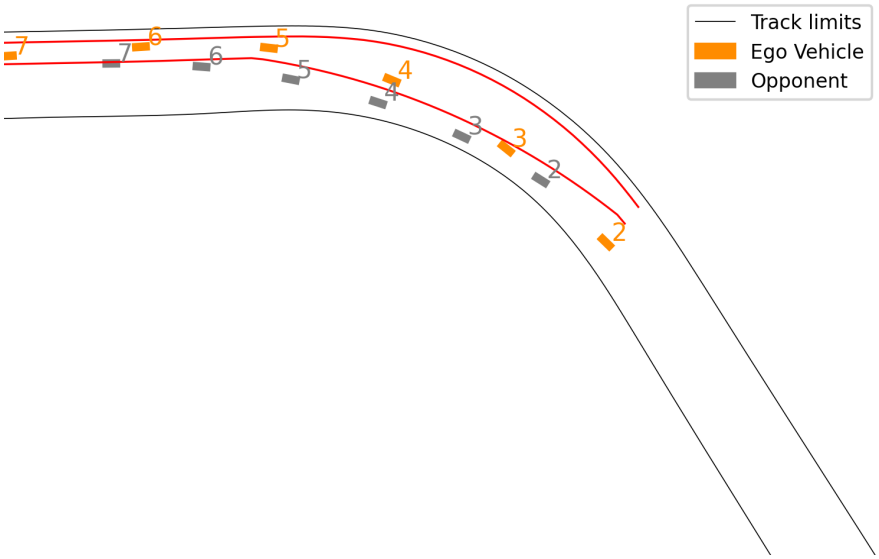


Figure 3.7: When the vehicles are longitudinally aligned (with some margin, in this case $0.5m$), a lateral ego-location is forced. Assuming there is no previous ego-location, the ego-location in this case is set to *left* because $\Delta_n > 0$, even though the ego vehicle CoM is not inside the left cone. Transitions to *back* or *front* are inhibited while the two vehicles are longitudinally aligned. Transitioning to right is still possible if the ego vehicle becomes laterally offset from the opponent to the right by at least half a car width - in the figure, the ego vehicle is offset more than half a car width to the left, in which case a *right* ego-location would be switched to *left*.



(a) Tunnel at $k = 0$



(b) Tunnel at $k = 2$

Figure 3.8: Figure (a) the intention of overtaking to the left until $k = 2$, when the tunnel is switched to the one shown in Figure (b). Between $k = 1$ and $k = 2$ ego acquires ego-location *left*. Shortly after $k = 2$ the opponent aligns laterally with the ego vehicle, which becomes *back* and switches to overtake on the right side. By $k = 4$, ego acquires ego-location *right*, and at $k = 7$, it becomes *front*. At no time step ego slows down, and the choice switch is performed smoothly.

Chapter 4

Overtaking with the optimizer

4.1 Introduction

The work in the previous chapters is mostly justified by the fact that with the current problem formulation, the optimizer is not aware of the problem of avoiding (and racing with) opponents. In this chapter, as part of work-in-progress experiments, a basic implementation of a distance soft-constraint between the ego vehicle and a single opponent is discussed along with its numerous problems. While this implementation is not meant to be integrated into the stack in its current state, it provides valuable insights on the current architecture and potential improvements to how opponents are accounted for.

Note that this idea is not meant to replace Overtake Logic. The overtaking problem is still a combinatorial, highly nonconvex problem which is difficult to solve directly with optimization, especially with the current solver used in the stack, about which more details are given in this chapter. Moreover, while there are various approaches for avoiding passive opponents directly in MPC, there are few that solve the problem of overtaking active agents that compete with the ego vehicle, for example using game-theoretical approaches [12], but they are mostly computationally inefficient. Even though Overtake Logic is fundamentally assuming that the behaviour of the opponent is not influenced by the behaviour of ego, it can efficiently account for more complex behaviours by using heuristics - an example was seen in Chapter 3, where it expected the opponent to yield space according to the competition regulations, only for then to switch overtaking side as soon as it was clear the opponent was not yielding - and also thanks to the fact that forecasting runs Overtake Logic to predict opponent behaviour, which generates more complex dynamics.

4.2 Basic implementation

The opponent forecasted trajectory is interpolated at the MPC horizon time steps to obtain the opponent position at each step. A soft lower bound on the distance to the opponent at each step is implemented and the slack variable is made to contribute to the cost function.

4.2.1 Distance function

A measure of distance between vehicles is required. For this prototype, vehicles are represented as Axis-Aligned Bounding Boxes (AABBs), that is, neither vehicle is rotated with respect to the racing line, and have equal width W and length L. While it is desirable to drop the AA assumption, the relative yaw between the vehicles can be accounted for by increasing the safety distance, which is reasonable for a prototype.

Unfortunately, using the simple euclidean distance function is unacceptable even for a prototype because cars in practice are about two and a half times longer than they are wide. Even scaling the longitudinal coordinate with some factor to obtain ellipsoidal variations is insufficient, as vehicle corners cannot be accounted for without increasing the safety distance to unacceptable levels.

The most precise distance function udner the AA assumption is the Signed Distance Function (SDF) between two AABBs, which is the same as the SDF between an AABB of doubled size and a point. Let $\boldsymbol{x_{ego}}$ and $\boldsymbol{x_{opp}}$ be the ego and opponent position expressed in Frenet space, respectively. In this paragraph, min, max, and the absolute value $|\cdot|$ are all applied component by component. Let $\boldsymbol{S} = \big(2L, 2W\big)$ be the size vector of the double-sized AABB, and define $\boldsymbol{\Delta} = |\boldsymbol{x_{ego}} - \boldsymbol{x_{opp}}| - \boldsymbol{S}$ be a vector such that when $\boldsymbol{\Delta} < 0$ the two AABBs are overlapping. The SDF between two equally sized AABBs is then:

$$d(\boldsymbol{x_{ego}}, \boldsymbol{x_{opp}}) = \|max(\boldsymbol{\Delta}, \boldsymbol{0})\|_2 + min(max(\boldsymbol{\Delta}_s, \boldsymbol{\Delta}_n), \boldsymbol{0}) \tag{4.1}$$

Figure 4.4 shows some level curves of this function, and issues related to the optimization with the gradient of this distance function will be brought up later.

4.2.2 Interpolation of the Forecasting trajectory

Forecasting produces a predicted cartesian trajectory of the opponent, with timestamp information for each point. Assume MPC is run on state information with timestamp t_0, and recall MPC has uniform step length Δ_t^{MPC}. It is therefore sufficient to sample the linear interpolator of the forecasting trajectory at timestamps $t_0 + i\Delta_t^{MPC}$ for $i = 0 \dots N^{MPC}$, then convert from cartesian

space to the Frenet space defined on the racing line currently being used as reference, to obtain the Frenet opponent position $\boldsymbol{x}_{\boldsymbol{opp}}^{(i)} = (\boldsymbol{s}_{\boldsymbol{opp}}, \boldsymbol{n}_{\boldsymbol{opp}})^{(i)}$ at each time step i. Because racing lines are a closed circuit, $\boldsymbol{s}_{\boldsymbol{opp}}^{(i)} \in [0, s_{max}]$ and therefore $\boldsymbol{s}_{\boldsymbol{opp}}$ has to be unwrapped to get a value in the same lap as ego (e.g. if $s_{max} > 2 \cdot 10$ and ego is at $s = s_{max} - 10$ and $\boldsymbol{s}_{\boldsymbol{opp}}^{(i)} = 1$, $\boldsymbol{s}_{\boldsymbol{opp}}^{(i)}$ should be unwrapped as $s_{max} + 1$).

4.2.3 Details on the solver

Chapter 1 briefly introduced the MPC problem formulation. In this chapter, more detail is required. The problem is solved with the SQP methodology: at each MPC iteration, multiple sub-iterations are performed. In each sub-iteration, a convex Optimal Control Quadratic Programming sub-problem (OCP-QP) is constructed by linearizing the dynamics and constraints of the NMPC problem around the initial guess and reference trajectories $(\boldsymbol{x}_{\boldsymbol{guess}}, \boldsymbol{u}_{\boldsymbol{guess}}, \boldsymbol{s}_{\boldsymbol{guess}})$ and $(\boldsymbol{x}_{\boldsymbol{ref}}, \boldsymbol{u}_{\boldsymbol{ref}}, \boldsymbol{s}_{\boldsymbol{ref}})$. The OCP-QP sub-problem is then solved using the HPIPM [5] solver to compute the Newton direction for descent as a solution to this problem. Along this direction, a step is performed to compute the initial guess trajectory for the next iteration, and as a result, at each sub-iteration the initial guess moves towards a constrained local minimum of the NMPC problem. Figure 4.1 shows the OCP-QP problem formulation that can be solved with HPIPM. Some takeaways are important to take:

- all matrices can vary stage-wise, as indicated by the stage subscript n (a stage is one time step of the MPC horizon);
- all variables can have box bounds;
- it is possible to define polytopic constraints by adding a row to the matrices D_n and C_n, and each has its own lower and upper bound component;
- all constraints can be optionally softened with one slack variable per stage (the J matrices are just mathematical notation for "routing" the correct slack variables to the correct constraint, while the HPIPM API accepts a vector of constraint indices for each stage);
- there are both linear (lowercase vector) and quadratic (uppercase matrix) costs for all variables.
- because the OCP-QP problem is constructed as a local quadratic approximation of the NMPC problem, the OCP-QP problem variables $(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{s})$ are the Newton step direction, or in other words, the solution to the original problem can be reconstructed as $(\boldsymbol{x}^*, \boldsymbol{u}^*, \boldsymbol{s}^*) = (\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{s}) + (\boldsymbol{x}_{\boldsymbol{guess}}, \boldsymbol{u}_{\boldsymbol{guess}}, \boldsymbol{s}_{\boldsymbol{guess}})$.

$$
\begin{aligned}
\min_{x,u,s} \quad & \sum_{n=0}^{N} \frac{1}{2} \begin{bmatrix} u_n \\ x_n \\ 1 \end{bmatrix}^T \begin{bmatrix} R_n & S_n & r_n \\ S_n^T & Q_n & q_n \\ r_n^T & q_n^T & 0 \end{bmatrix} \begin{bmatrix} u_n \\ x_n \\ 1 \end{bmatrix} + \\
& + \frac{1}{2} \begin{bmatrix} s_n^{\mathrm{l}} \\ s_n^{\mathrm{u}} \\ 1 \end{bmatrix}^T \begin{bmatrix} Z_n^{\mathrm{l}} & 0 & z_n^{\mathrm{l}} \\ 0 & Z_n^{\mathrm{u}} & z_n^{\mathrm{u}} \\ (z_n^{\mathrm{l}})^T & (z_n^{\mathrm{u}})^T & 0 \end{bmatrix} \begin{bmatrix} s_n^{\mathrm{l}} \\ s_n^{\mathrm{u}} \\ 1 \end{bmatrix} \\
\text{s.t.} \quad & x_{n+1} = A_n x_n + B_n u_n + b_n \quad , \quad n \in \mathcal{H} \backslash \{N\} \\
& \begin{bmatrix} \underline{u}_n \\ \underline{x}_n \\ \underline{d}_n \end{bmatrix} \leq \begin{bmatrix} J_n^{b,u} & 0 \\ 0 & J_n^{b,x} \\ D_n & C_n \end{bmatrix} \begin{bmatrix} u_n \\ x_n \end{bmatrix} + \begin{bmatrix} J_n^{s,u} \\ J_n^{s,x} \\ J_n^{s,g} \end{bmatrix} s_n^{\mathrm{l}} \ , \ n \in \mathcal{H} \\
& \begin{bmatrix} J_n^{b,u} & 0 \\ 0 & J_n^{b,x} \\ D_n & C_n \end{bmatrix} \begin{bmatrix} u_n \\ x_n \end{bmatrix} - \begin{bmatrix} J_n^{s,u} \\ J_n^{s,x} \\ J_n^{s,g} \end{bmatrix} s_n^{\mathrm{u}} \leq \begin{bmatrix} \overline{u}_n \\ \overline{x}_n \\ \overline{d}_n \end{bmatrix} \ , \ n \in \mathcal{H} \\
& s_n^{\mathrm{l}} \geq \underline{s}_n^{\mathrm{l}} \ , \ n \in \mathcal{H} \\
& s_n^{\mathrm{u}} \geq \underline{s}_n^{\mathrm{u}} \ , \ n \in \mathcal{H} \\
& \text{where } H = \{0, 1, \ldots, N\}
\end{aligned}
$$

Figure 4.1: The OCP-QP problem structure solvable with the HPIPM solver, from [5]

4.2.4 Implementing the distance constraint

The nonlinear distance constraint can be implemented by local approximation as a polytopic soft constraint in the OCP-QP subproblem. Let d_s be a constant safety distance parameter, and $\boldsymbol{x}^* = \boldsymbol{x} + \boldsymbol{x}_{\boldsymbol{guess}}$ be the NMPC solution at any intermediate solution $\boldsymbol{x}$ of the OCP-QP problem. Let the distance functions between the ego and opponent at each time step be, for convenience of notation,

$$\boldsymbol{d}_i(\boldsymbol{x}) = d(\boldsymbol{x}^{*(i)}, \boldsymbol{x}_{\boldsymbol{opp}}^{(i)})$$

the non-linear constraint can then be formulated as, letting $\boldsymbol{s}_i$ be the slack variable for the safety distance soft constraint:

$$\boldsymbol{d}_i(\boldsymbol{x}) + \boldsymbol{s}_i >= d_s$$

which are linearized by taking the first order taylor expansion of each function in $\boldsymbol{d}$ around $\boldsymbol{x}_{\boldsymbol{guess}}$:

$$(\nabla \boldsymbol{d}_i(\boldsymbol{x}))^T(\boldsymbol{x} - \boldsymbol{x}_{\boldsymbol{guess}}) + \boldsymbol{s}_i >= d_s - \boldsymbol{d}_i(\boldsymbol{x}_{\boldsymbol{guess}})$$

and then integrated into the OCP-QP problem formulation as:

$$C_i = \nabla \boldsymbol{d}_i(\boldsymbol{x})$$
$$\underline{d}_i = d_s - \boldsymbol{d}_i(\boldsymbol{x}_{\boldsymbol{guess}})$$

where with a slight abuse of notation the equalities indicate that the right hand side is added as a row to the left hand side matrices rather than replacing them, in order to allow the use of other constraints. The slack variable is enabled for the constraint, the quadratic cost is added to the diagonal of Z_n and the linear cost as a new element of z_n.

The specification of the function d is written directly in C++ with the *CppAD* [2] auto-differentiation library, and the C++ source code to actually compute value and gradient of the function is generated with the third-party extension *CppADCodeGen* [3] and integrated into the build system.

In the minimum of d, that is when $|\Delta_s| <= L \wedge \Delta_n = 0$, the gradient is undefined. For this reason, before invoking the automatically generated code, Δ is manipulated to lie elsewhere. A generally safe choice is $\Delta = (L - \epsilon, 0)$ for some $\epsilon > 0$, as the gradient will point in the negative s direction, inducing the MPC to slow down at that step.

4.3 Issues

4.3.1 Overtaking side choice

Because MPC is performing a local optimization and because the distance function naturally splits the solution space in two (overtake on the left or on the right), MPC may converge on any side of the opponent and fail to explore the other side. Worse still, once MPC has converged to one side, the gradient of the distance function pushes the solution towards that side, making a switch highly unlikely.

Overtake Logic already performs the the choice of which side the opponent should be overtaken in. While it would be nice if MPC could correct mistakes committed by Overtake Logic on this part, the possibility (a very likely one, if high enough safety constraint costs are used) of getting stuck in one side is unacceptable (as an example scenario, return to Figure 3.4). As such, it is necessary to have the distance function not split the search space: in this implementation, d is modified such that Δ_n is clamped to the side the overtake is being performed in, to essentially extend (in other words "extrude") the distance function $d(\Delta_s, 0)$ on the n axis in the direction opposite to the overtake side, guaranteeing that the gradient never points in such direction. More formally:

$$d'(\Delta) = \begin{cases} d(\Delta_s, \min(\Delta_n, 0)) & \text{if overtaking on the right} \\ d(\Delta_s, \max(\Delta_n, 0)) & \text{if overtaking on the left} \\ d(\Delta_s, \Delta_n) & \text{otherwise (no decision has been made)} \end{cases} \tag{4.2}$$

4.3.2 Distance function gradient

For states that represent an ego vehicle which is laterally aligned with the opponent, but also if the modified distance function d' is used and ego is on the side opposite to the overtake, the gradient of the distance function has n component equal to zero. The safety distance constraint therefore induces the MPC to accelerate or decelerate without moving to the correct side of the overtake, effectively following the opponent instead of overtaking it if ego is on the back, or accelerating away from the opponent if ego is on the front. While the cost of the drivable area created by Overtake Logic adds its contribution on the n direction, it may still be desirable to modify the distance function so that the n component gradient is zero only when $\Delta_n = 0$.

Another issue related to the SDF distance function choice is that it is very non-linear near the vehicle corners. Although this nonlinearity is necessary to account for the corners exactly, the more nonlinear a function is the more SQP has issues converging.

An example of an alternative function would be to use a modified euclidean distance, by stretching it along the longitudinal axis to account for the vehicle length - let $a \in (0, 1]$ and $c \in \mathbb{R}^+$:

$$d_{euc}(\Delta) = \sqrt{(a\Delta_s)^2 + \Delta_n^2} - c \tag{4.3}$$

Unfortunately this distance function is too inaccurate to be practical, as it requires huge values of d_s to cover the vehicle corners. However, it is useful to show that it is necessary to move away from the accurate SDF to a more inaccurate distance function which exhibits better gradient properties.

4.3.3 Forecasting errors

Another weakness of this approach, which it shares with Overtake Logic, is that it relies on the forecasting trajectory as if it were ground truth.

With such an assumption, errors in both the longitudinal and lateral direction are made, and longitudinal errors are usually much more problematic than lateral errors. Longitudinal errors tend to accumulate and become much higher than the safety distance parameter: for example, a car may be stopping in the middle of the track, but if one makes the assumption it is going to accelerate, even the first few steps of the prediction become completely wrong due to the large difference in acceleration, and the errors accumulate in later time steps without any bound. Lateral errors, instead, have bounded magnitude given that the track width is finite and small, usually a few times the safety distance - their dynamics are usually also a bit slower and more predictable, as the other car is most likely tracking some predictable reference path, unless this path suddenly changes.

For what concerns lateral errors, due to their small magnitude and predictability it may even be possible to consider the error as gaussian noise - an idea could then consist in reducing the safety distance cost and increasing the safety distance at the steps in which the trajectory uncertainty is higher (which remains to be estimated!).

It is much more difficult to handle longitudinal errors, however. Due to their magnitude this solution can break down as they quickly become much larger than the safety distance itself, possibly even in the very first time steps. A real-world example of such breakdown is presented as happened with Overtake Logic. During the A2RL 2025 Final Race, the ego vehicle approached Turn 1 of the Yas Marina Circuit, followed by an opponent (”follower”). Another opponent was slowing down to a halt in the middle of Turn 1, but Forecasting predicted it would accelerate again, and as such even the very first steps of the forecasting trajectory had very large longitudinal errors. Overtake Logic decided to overtake the stopping opponent on the right, as it had to leave space on the inner part of the turn because of the following opponent, which was closing in to overtake the ego vehicle. As a result, Overtake Logic was setting the tunnel boundaries ahead in time, in the same way as

how this solution would compute the distance of an opponent ahead with respect to where it really is. When the tunnel boundaries finally overlapped with the opponent, it was too late and the ego vehicle collided with the stopped opponent, ending the race. Figure 4.2 shows the crash from the official broadcast of A2RL, while 4.3 as seen from the real telemetry data.

Unfortunately, assuming opponents are static (that is, using no forecasting), although it is the safest choice it is unacceptable to achieve good performance on track. Therefore, as part of future works, it is necessary to find a way to deal with longitudinal errors. Results on the bottom will show, however, the crash would've been avoided by combining static opponents and the distance cost presented in this chapter.

Figure 4.2: Frames of the crash from the A2RL official broadcast [10].

4.3.4 Phasing through opponents

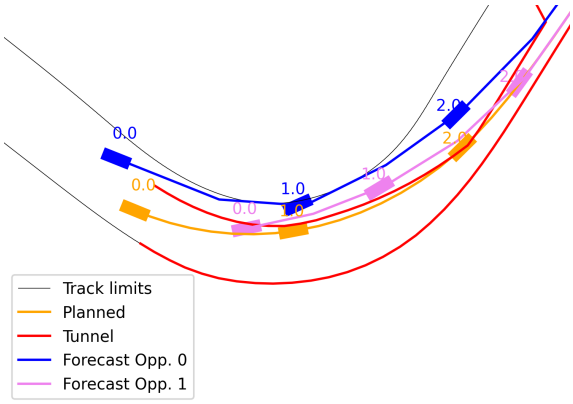
With the current approach, an issue correlated with the previous is that it is possible for a solution to phase through opponents. Assume the trajectory in a solution passes right through an opponent: the steps in front of the opponent have gradient with positive s direction, while the steps behind it will have it negative. If the safety distance cost in steps with positive component outweight the ones with negative component, MPC will accelerate towards the opponent. This does happen in

practice, especially when an opponent spawns on the ego planned path, which is possible due to the long prediction horizon. Figure 4.4 shows a diagram that visualizes the problem. Solving it is part of future works.

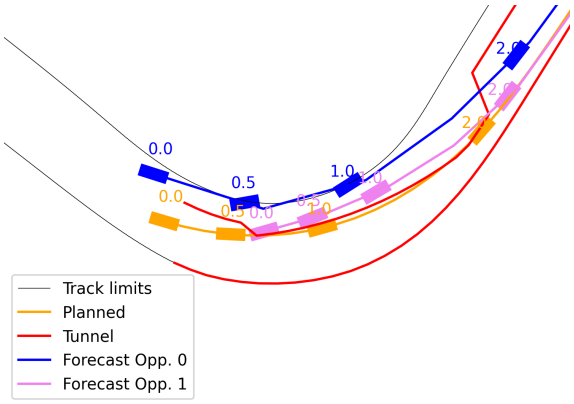
4.4 Results

The scenario of 4.3 is reconstructed with the opponents assumed static, meaning that forecasting assumes the opponents have zero velocity and thus all time steps have the same opponent position. Additionally, the distance function is modified as shown in Equation 4.2, and all costs and the safety distance are set to the same values in all simulations. When forecasting is considered, the result is analogous to the telemetry data seen in Figure 4.3, so no results will be given for that configuration. The scenario is run with the modified euclidean distance function (Equation 4.3), and Figure 4.5 shows the result.

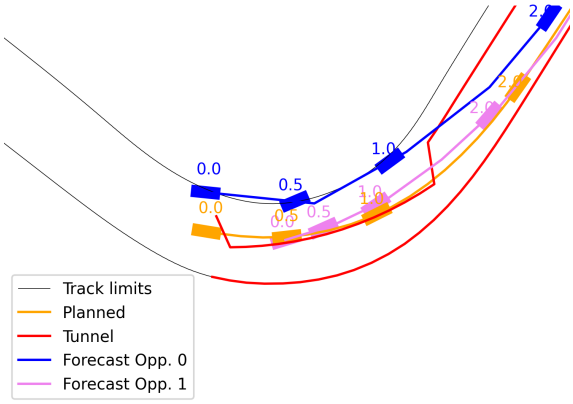
The results show allow two important conclusions to be made: first, the longitudinal error is fatal if not accounted properly, and not using forecasting at all is safer, but has unacceptable performance on track; second, such a distance cost, even with a temporary implementation allows to correct the mistakes made by Overtake Logic, as can be seen from the planned trajectory always lying at a safer distance than the tunnel boundary.



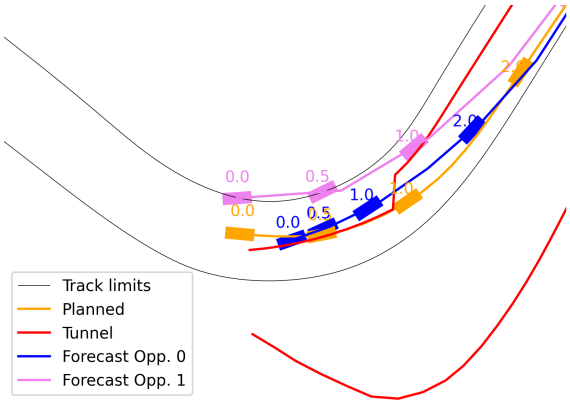
(a) T−0.8s



(b) T−0.645s



(c) T−0.35s



(d) T−0.15s

Figure 4.3: Evolution of the planned path, forecasting and tunnel during the real crash, as seen from downsampled telemetry data. Colored numbers show the seconds passed since the snapshot was taken. In the captions, T−Xs means that the snapshot was taken X seconds before the crash - note that the whole dynamic lasts less than 800ms. In all steps, forecasting predicts the stopping opponent (pink) is going to accelerate, and therefore the tunnel is shifted to the front - note how the opponent lies outside the tunnel until step (b). At step (c), it is clear that the tunnel cost is not enough to overcome the rest of the costs, and at (d) it is too late to do anything - at this point Overtake Logic also enables the runoff areas due to some safety checks, as can be seen by the right border being moved to the wall.

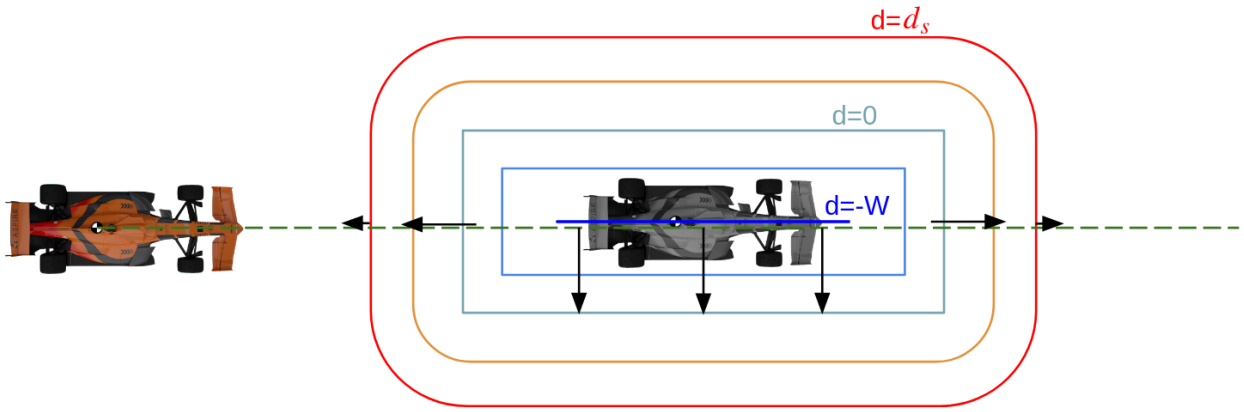
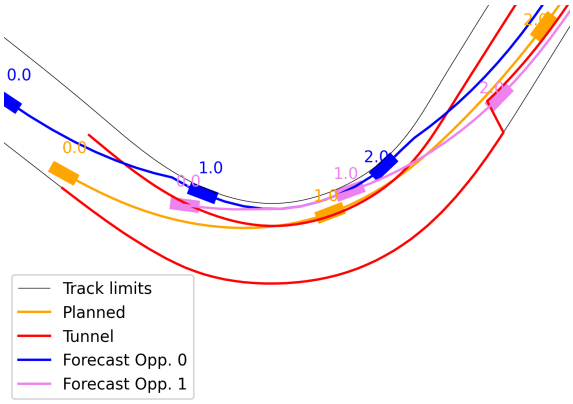
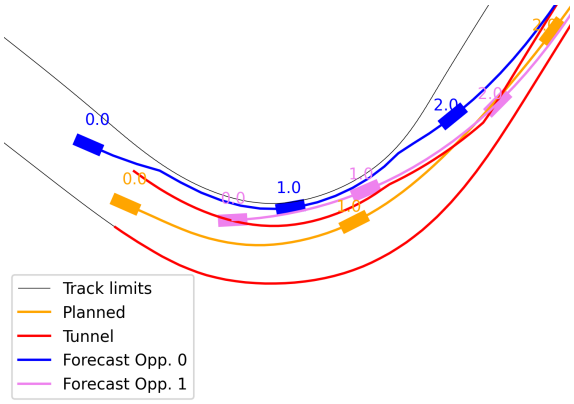


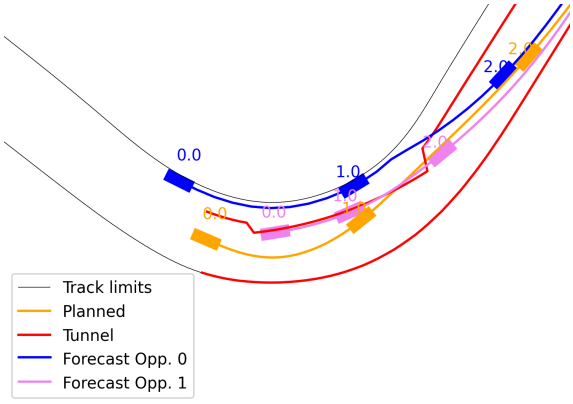
Figure 4.4: The figure shows some level curves of the safety distance function, and the black arrows show the gradient of the constraint at some MPC time steps. Note how the front states have gradient in the positive x direction.



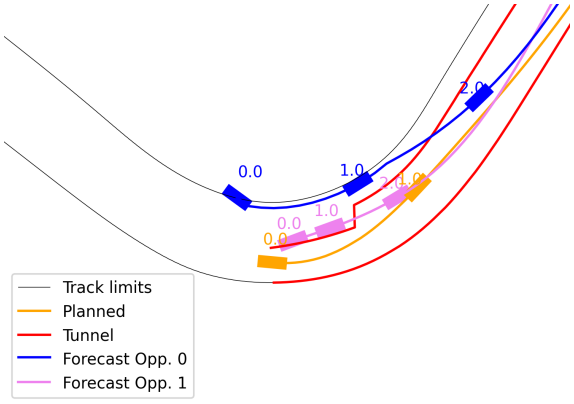
(a) T+0s



(b) T+0.5s



(c) T+1.0s



(d) T+1.5s

Figure 4.5: Reconstruction of Figure 4.3 with the safety distance cost using the modified euclidean distance formulation. The forecasting trajectory used by MPC is overriden with the position at Δ_t=0, while the figures show the full forecasting for reference. Note how the planned path gets a larger and larger radius to avoid the opponent, even though forecasting keeps accelerating. The first snapshot is taken at time T.

Conclusions and Future Works

In this work, two contributions that were validated both in the real world and in simulations improved the reliability and performance of an autonomous driving racecar in the presence of other agents: specifically, the ability to change the reference speed profile used by the planning MPC to account for areas that are undrivable due to the presence of other agents, and improving the overtaking performance by removing assumptions that only worked in oval racetracks.

Future works include properly handling the longitudinal forecast uncertainties and improving the collaboration between *Overtake Logic* and *mpplanner* - it was clearly seen in chapter 4 that a full separation is insufficient, and bringing down some knowledge from the high level decision maker to the optimizer can result in better handling edge cases, although formulating the problem correctly is an open issue.

Bibliography

[1] *Abu-Dhabi Autonomous Racing League*. https://a2rl.io/.

[2] COIN-OR. *CppAD*. https://projects.coin-or.org/CppAD.

[3] *CppADCodeGen*. https://github.com/joaoleal/CppADCodeGen.

[4] L. Fesquet and B. Bidégaray-Fesquet. "IIR digital filtering of non-uniformly sampled signals via state representation". In: *Signal Processing* 90.10 (2010), pp. 2811–2821. ISSN: 0165-1684. DOI: https://doi.org/10.1016/j.sigpro.2010.03.030. URL: https://www.sciencedirect.com/science/article/pii/S0165168410001349.

[5] Gianluca Frison and Moritz Diehl. *HPIPM: a high-performance quadratic programming framework for model predictive control*. 2020. arXiv: 2003.02547 [math.OC]. URL: https://arxiv.org/abs/2003.02547.

[6] Gaël Guennebaud, Benoît Jacob, et al. *Eigen*. https://libeigen.gitlab.io. 2010.

[7] F. Gustafsson. "Determining the initial states in forward-backward filtering". In: *IEEE Transactions on Signal Processing* 44.4 (1996), pp. 988–992. DOI: 10.1109/78.492552.

[8] *Indy Autonomous Challenge*. https://www.indyautonomouschallenge.com/.

[9] Giovanni Lambertini et al. *Fast and Realistic Automated Scenario Simulations and Reporting for an Autonomous Racing Stack*. 2025. arXiv: 2512.24402 [cs.RO]. URL: https://arxiv.org/abs/2512.24402.

[10] ©Abu-Dhabi Autonomous Racing League. *A2RL 2025 Grand Finale Official Broadcast*. https://www.youtube.com/watch?v=d9LLZ5mb5cA.

[11] Matthew O'Kelly et al. *F1/10: An Open-Source Autonomous Cyber-Physical Platform*. 2019. arXiv: 1901.08567 [cs.RO]. URL: https://arxiv.org/abs/1901.08567.

[12] Francesco Prignoli et al. *Regulation-Aware Game-Theoretic Motion Planning for Autonomous Racing*. 2025. arXiv: 2508.20203 [eess.SY]. URL: https://arxiv.org/abs/2508.20203.

[13] Ayoub Raji et al. "er.autopilot 1.1: A Software Stack for Autonomous Racing on Oval and Road Course Tracks". In: *IEEE Transactions on Field Robotics* 1 (2024), pp. 332–359. DOI: 10.1109/TFR.2024.3501252.

[14] Ayoub Raji et al. "Motion Planning and Control for Multi Vehicle Autonomous Racing at High Speeds". In: *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*. 2022, pp. 2775–2782. DOI: 10.1109/ITSC55140.2022.9922239.

[15] *Roboracer*. https://roboracer.ai/.

[16] Sebastian Thrun et al. “Stanley: The robot that won the DARPA Grand Challenge”. In: *Journal of Field Robotics* 23.9 (2006), pp. 661–692. DOI: https://doi.org/10.1002/rob.20147. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.20147. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20147.

[17] Alessandro Toschi, Francesco Prignoli, and Marko Bertogna. “Modular Decision-Making and Drivable Areas for Multi-Agent Autonomous Racing”. In: Oct. 2025, pp. 12435–12441. DOI: 10.1109/IROS60139.2025.11246897.