AARHUS UNIVERSITY

Computer Engineering

Curvature based Racing line generation

Student
Alexander Magnus Frederik Bjerno

Supervisors
Lukas Esterle
Mirgita Frasheri

Date June 9, 2025

Aarhus University
Department of Electrical and Computer Engineering

Table of Contents

1 Introduction

Motivation

This paper explores a mathematical approach to calculating optimal racing lines for the F1Tenth project, a platform for autonomous racing at scale. By leveraging computational geometry and optimization techniques, we aim to develop an algorithm that, given a track's centerline data, can compute the fastest possible path through a race track.

F1tenth intro

This issue has been explored by others but a lot of the research is based on a machine learning approach so we wanted to explore using a more mathematical and analytical approach to create a general solution, as machine learning can be inefficient in some situations as it uses a lot of power and needs lots of training data to be accurate, it also poses risks in terms of overfitting and creating poor results when faced with problems that significantly differ from training data. Additionally machine learning models are generally not deterministic which means results can vary even with the exact same variables so making use of a purely mathematical approach should be able to create an algorithm which is consistent and will give the same result each time providing the variables are the same.

This approach has direct applications in both simulated and real-world racing scenarios. In the context of F1Tenth, optimizing the racing line can enhance the performance of autonomous vehicles in competitive settings. Additionally, the methodology explored here could be extended to full-scale motorsports, driver coaching, and race strategy optimization.

Our work considers key racing principles, such as corner classification (early, late, and geometric apex turns), track constraints, and vehicle dynamics, to generate a racing line that maximizes overall lap speed. By incorporating these factors, we aim to develop a robust algorithm that can adapt to different track layouts and optimize performance in a way that is both computationally efficient and practically viable for real-world implementation.

Research Question

This paper investigates whether an analytical model can be designed to efficiently and accurately compute an optimal racing trajectory, with a specific focus on the F1TENTH autonomous vehicle platform. The primary aim is to evaluate the feasibility of generating high-performance paths based solely on pre-existing map data, without requiring online learning or iterative optimization during deployment. In particular, this research emphasizes the critical role of cornering strategy in lap time optimization, aiming to develop an approach that generalizes across a wide variety of tracks. The central research question can thus be formulated as:

Can an analytical method be used to generate racing lines that optimize lap time on arbitrary tracks using only offline map data, and how effective is this method when implemented and tested on the F1TENTH autonomous car platform?

To answer this, the study explores the use of spline interpolation techniques and racing heuristics for path generation, and evaluates performance in terms of lap time, path length, curvature, and robustness across multiple track layouts.

Structure

In the following chapters we will be going through some of the background knowledge needed for this paper, particularly racing theory as well as some technical knowledge related to ros2 and the other tools used.

A detailed explanation of the work done including data extraction, feature analysis and path planning.

Details around the experiment, assumptions, variables and how it was conducted following with the results and discussion of the results.

A summary of related works including analytical and machine learning approaches which are worth reading.

Ending with a recap and our conclusions as well as a discussion on future work to be done in relation to this paper.

2 Background

Autonomous Racing and F1TENTH

F1TENTH is an organization and platform offering low cost 1/10 scale autonomous vehicles for research and education, this research is useful both for the racing scene but also for autonomous driving (self-driving cars) or other robotics fields that rely on navigating real world environments.

Certain challenges present themselves in the world of autonomous driving, particularly real-time perception, planning and control which all come with their own sets of requirements and challenges relevant not only for autonomous racing but also many other fields involving robotics and navigation. For this paper we will mainly be focusing on the planning using already made maps to plan an efficient route then using a path following algorithm to test these paths within the f1tenth simulation environment.

It is to be noted that the F1TENTH organization has been renamed as RoboRacer and can be found here.

Racing Line Optimization

A race line is the path that allows the fastest speed/time through a track, being able to properly plan a race line is therefore important in the world of racing to be able to clear tracks fast whilst accounting for what is possible within the constraints of the track and the vehicle.

Relevant to this paper is some of the classic racing principles apexing, minimizing curvature and maximizing exit speed, these pertain to the main problem in racing theory which is how to best approach corners. There are multiple approaches such as machine learning data driven approaches, minimum curvature or geometric methods, we will mainly be looking at the geometric methods trying to use the early and late apex rules to determine how to approach each corner, when and where to turn as well as where to exit. This is somewhat complicated as there are many different scenarios and depending factors such as the cars weight, motor, brakes and tires the best approach can change drastically. This is also why the paper is focused on the f1tenth platform giving us a more concrete environment to work within.

This graph showcases how the different geometric lines look, with the basic geometric line aiming to hit the apex (central inner point of the curve) in the middle where as early and late apex approaches aim to enter the corner earlier or later. In general in racing it is most optimal to enter a corner from the outer side hitting the inside around the middle of the corner and exiting on the outer side again unless the following corner goes in the opposite direction.

Path Planning and Spline Interpolation

Path planning is a fundamental component of autonomous navigation systems, enabling a vehicle or robot to move from an initial position to a desired destination while avoiding obstacles and adhering to dynamic and environmental constraints. Effective path planning is essential for ensuring both safety and efficiency, particularly in environments with limited space or complex geometry LaValle, 2006.

For vehicles such as autonomous cars, the path must not only avoid obstacles but also respect non-holonomic constraints and optimize performance criteria such as distance, curvature, or travel time Paden et al., 2016. To generate paths that are suitable for tracking, smooth, and dynamically

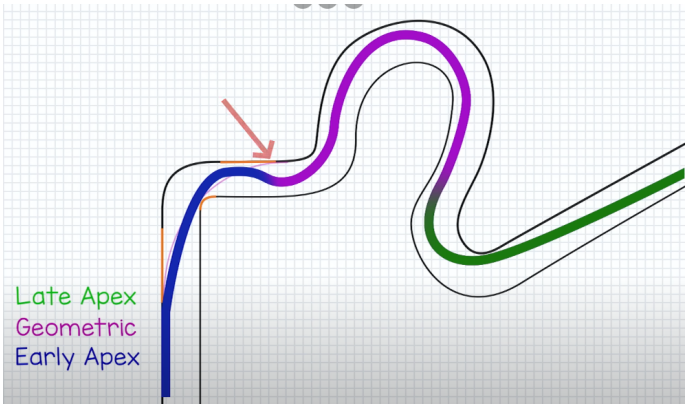


Figure 1: Apex example
Figure is from this article about racing lines link here.

feasible, spline interpolation techniques are frequently employed.

Spline interpolation involves constructing a smooth curve through a set of waypoints using piecewise polynomial functions. Commonly used splines include cubic B-splines and cubic Hermite splines, which provide continuous first and second derivatives, ensuring smooth transitions between segments Piegl and Tiller, 1996. This smoothness is critical for applications requiring stable and predictable vehicle motion, especially at higher speeds.

The advantages of spline-based path planning include:

- *Smoothness*: High-order continuity enables more stable and comfortable trajectories.
- *Flexibility*: Capable of accurately modeling complex, curved paths found in real-world driving scenarios.
- *Controllability*: Allows control over curvature and orientation, which is essential for satisfying dynamic constraints.

In the context of racing and performance driving, spline interpolation is often used to create an *optimal racing line*—a trajectory that minimizes lap time by maximizing speed through corners while respecting physical limits such as tire grip and turning radius Heilmeier et al., 2020. In contrast, a *centerline path* typically follows the geometric center of the track and serves as a baseline trajectory without regard for performance optimization.

The integration of path planning with spline-based interpolation thus provides a robust framework for autonomous navigation, balancing the competing demands of feasibility, safety, and performance.

Path Following Algorithms in ROS

While path planning determines a feasible or optimal trajectory for a vehicle, **path following** is concerned with ensuring that the vehicle adheres to this path during motion. Path following algorithms play a crucial role in converting the planned trajectory into actionable steering and speed commands while compensating for real-time deviations, external disturbances, and dynamic constraints.

The main objective of a path following controller is to minimize the cross-track error (the lateral deviation from the path) and heading error (the orientation difference between the vehicle and the path tangent) Snider, 2009. To achieve this, numerous control strategies have been proposed in literature and applied in autonomous driving applications.

Stanley Controller

One widely used method is the *Stanley Controller*, developed by Stanford Racing Team for the DARPA Grand Challenge Thrun et al., 2006. It is a geometric controller that computes the steering angle based on two components: the heading error between the vehicle and the path tangent, and the cross-track error scaled by a gain factor. The Stanley method is robust to measurement noise and can handle sharp turns, making it suitable for real-time driving.

Pure Pursuit

The *Pure Pursuit* algorithm is another geometric approach that directs the vehicle toward a lookahead point on the path, located at a fixed or dynamic distance ahead of the current position. The steering angle is calculated such that the vehicle follows an arc toward this point. While simple and effective at lower speeds, Pure Pursuit may suffer from instability or oscillation at high velocities or sharp turns Coulter, 1992.

Model Predictive Control (MPC)

Model Predictive Control (MPC) is a more advanced approach that formulates path following as an optimization problem over a finite prediction horizon. MPC accounts for the system's dynamics, constraints (e.g., steering limits, velocity bounds), and future path information Kong et al., 2015. It produces smoother and more optimal control signals but is computationally intensive and requires an accurate model of the vehicle.

Comparative Summary

Each path following method has trade-offs:

- **Stanley Controller**: Low computational cost, good for moderate-speed navigation and sharp curves.
- **Pure Pursuit**: Intuitive and easy to implement, works well in open spaces with smooth paths.
- **MPC**: High precision and optimality at the cost of computational complexity.

Choosing the appropriate path following algorithm depends on factors such as vehicle speed, track complexity, computational resources, and the required precision. In high-speed and racing contexts, geometric methods like Stanley and Pure Pursuit remain popular due to their simplicity and robustness.

3 Implementation

Data extraction

For this project centerline data was used, with the data coming from a f1tenth repository of scaled down f1 racing tracks Github repo, particularly using the Oschersleben map for development as it has a good variety of corners with different lengths of straights in between as well as sections of alternating turning through multiple corners.

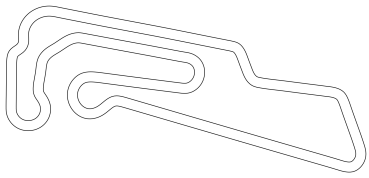


Figure 2: Oschersleben_map

The data came in a csv file so the python package pandas could be used to easily extract the specified data to get the middle line through the track, then using the information that the track was uniformly 2.2m wide we could calculate the borders of the track to restore it from the centerline data using normal vectors scaled to half the track width.

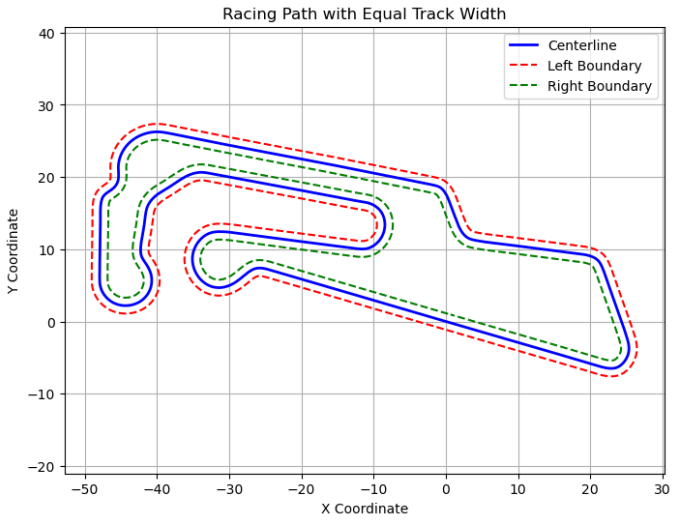


Figure 3: Restored Oschersleben map

With the centerline data as well as track bounds setup, we were able to start analyzing the track to identify corners and start on the developing a method for calculating an optimal route.

Feature identification

There are a couple characteristics or features we want to identify for every corner on a given track, these are the direction, length and curvature these features we use to identify the corners themselves using the curvature of the track and then we will later make decisions based on the length of the corners, their direction as well as adjacent corners.

Corners and straights

Firstly we needed to identify where the corners are, for this we will be taking the entire middle line and calculating the curvature at any given point to identify where corners start and end, for this we calculate the first and second derivatives for all the data points in the center-line then using this formula `curvature = np.abs(dx * ddy - dy * ddx) / (dx**2 + dy**2)**(3/2)` we can find the curvature, then using a threshold which from experimentation is around 0.06, we can then filter for corners based on if the curvature at any point is above our threshold and any other points will be counted as part of a straight. Then for future use the corners are collected into segments based on the first and last index of each corner relative to the track data so it is easy to manipulate the track around the corners by using the index numbers.

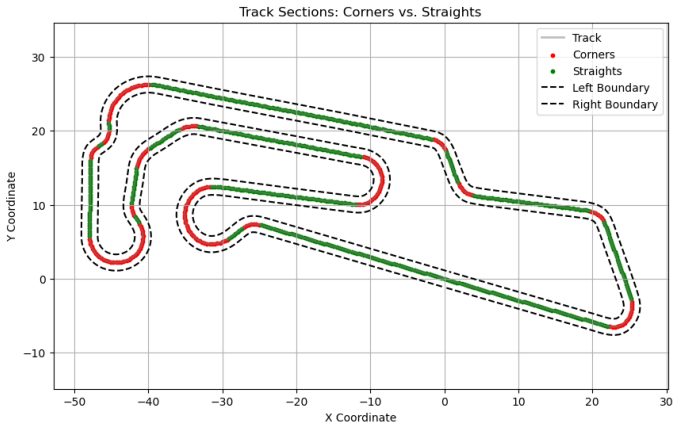


Figure 4: Plot of identified corners

Turn direction

Another piece of information we need for our optimizations are the directions of the corners, to do this we again use the first derivatives of the entire track data then we calculate the angle differences and wrap them around pi so the range at any point is between negative and positive pi, after that we can simply go through each corner segment we found earlier and find the average direction change, if the change is positive then it is a left turn and if the change is negative it is a right turn.

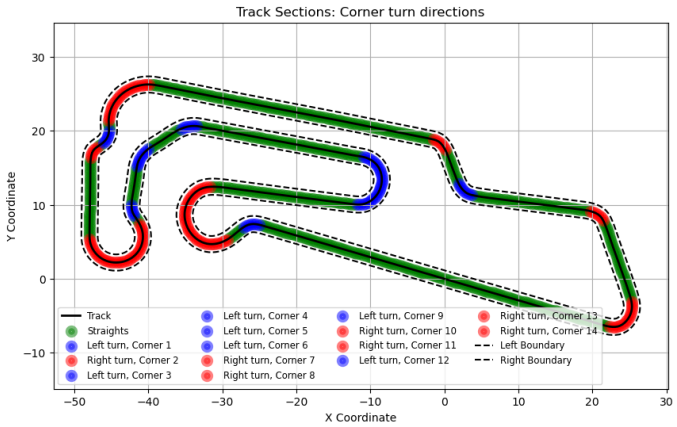


Figure 5: Corner directions
A plot showing identified directions on the Oschersleben track with blue corners being left turns and red corner being right turns.

Corner types

We decided on grouping the corners into five types for this project and we will optimize them based on their classification, for more precision more types could be specified but this for this project 4 was enough. The types we have are `Early apex, Late apex, Geometric Apex, Hairpin and Sharp corner` which tell us how we want to take the corner. For example a corner labeled early apex will be optimized for earlier breaking and turning into the corner so the apex comes earlier, this can help in situation where a corner is followed by a long straight so positioning afterwards is not important and we just want to be able to accelerate as soon as possible. The other types follow a similar logic and we have a list of the basic things we are looking at to classify each corner. The sharp corner is an edge case for very short and sharp corners which are hard to handle and don't fit into the other categories.

- Distance to next corner
- Is the next corner turning in the opposite direction
- How long is the corner
- Curvature difference

If the distance is above a threshold then the corner will be an early apex, if not then we look if the direction of the next corner is the same, if it is not then the corner will be a late apex and otherwise it will be a geometric apex. Finally if any corner is above a certain length we will classify it as a hairpin regardless of any other factor (A hairpin is a long corner which ends in around a 180° turn).

```
for i, (start, end) in enumerate(corner_segments):
    if i + 1 >= len(corner_segments):
        n = 0
    else:
        n = i + 1
    start_n, end_n = corner_segments[n]
    distance = np.sqrt((x[end] - x[start_n]) ** 2 + (y[end] - y[start_n]) ** 2)

    if distance > 10:
        corner_type[i] = "Early Apex"
    elif corner_directions[i] != corner_directions[n]:
```

```
        corner_type[i] = "Late Apex"
    elif corner_directions[i] == corner_directions[n]:
        corner_type[i] = "Geometric Apex"

    # New hairpin definition
    heading_start = theta[start]
    heading_end = theta[end - 1]  # theta is len(x)-1
    total_heading_change = np.abs(np.arctan2(np.sin(heading_end - heading_start),
    np.cos(heading_end - heading_start)))
    corner_length = np.sqrt((x[end] - x[start])**2 + (y[end] - y[start])**2)

    # Define thresholds
    sharp_heading_threshold = np.deg2rad(90)
    hairpin_heading_threshold = np.deg2rad(150)
    length_threshold = 6  # in meters, adjust as needed

    if total_heading_change > hairpin_heading_threshold:
        corner_type[i] = "Hairpin"
    elif total_heading_change > sharp_heading_threshold and corner_length <
   length_threshold:
        corner_type[i] = "Sharp Turn"
```

Listing 1: Corner calssification

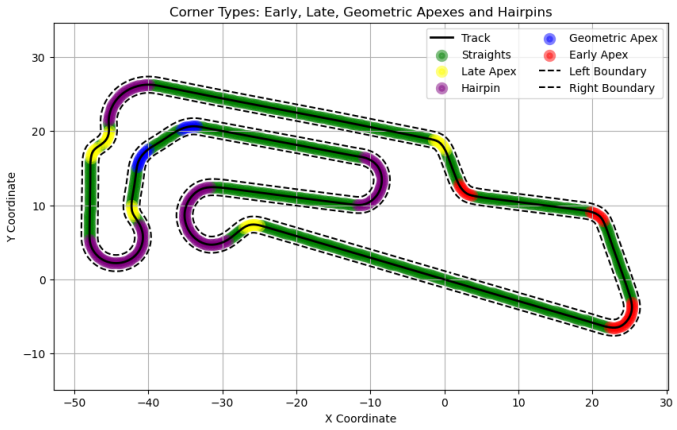


Figure 6: Corners classified

Path planning

Planning the path takes into consideration all the analysis we have already done on the centerline and then looks at each corner and set's way points for where the car should turn into the corner, where the apex should be and and a guiding way point for where to exit the corner. Where these points are set is based on the type of corner as well as some consideration for what comes after and the length of the corner, these points are selected from the centerline and then pushed towards a desired position much like how the boundaries where calculated. Once all points are set we interpolate that path to create points in between so we get a smooth path through the whole track then we check for invalid points which cross the boundaries and correct them before applying some smoothing as well as resampling the path for equal spacing along all points.

There are multiple variables that determine offset from centerline, braking distance, apex distance, end distance and other offsets which are determined based on the type of corner, they are the used

either to determine the index of the point we want to manipulate or how far this point should be offset from the center. This way it is also easy to modify and tune behavior of the algorithm with respect to different corner types and to add more types for better case coverage.

```
    # Direction offset
    dir_offset = -0.9 if corner_directions[i] == "Left Turn" else 0.9
    dir_offset2 = -0.2 if corner_directions[i] == "Left Turn" else 0.2
    exit_dist = 3
    exit_offset = 0
    # Corner type settings
    if corner_type[i] == "Late Apex":
        brake_dist, apex_pos, exit_dist = -2, 4, -1
        exit_offset = 0
    elif corner_type[i] == "Early Apex":
        brake_dist, apex_pos = 4, -2
        exit_offset = -1
        dir_offset2 = -0.1 if corner_directions[i] == "Left Turn" else -0.1
        exit_dist = 0
    elif corner_type[i] == "Geometric Apex":
        brake_dist, apex_pos = 3, 0
    elif corner_type[i] == "Hairpin":
        brake_dist, apex_pos = -4, 0
        dir_offset2, exit_dist = 0, -5
        exit_offset = 1
    elif corner_type[i] == "Sharp Turn":
        brake_dist, apex_pos = 1, -1
        dir_offset2 = 0
        exit_dist = -2
        exit_offset = -0.5
```

Listing 2: Planning variable determination

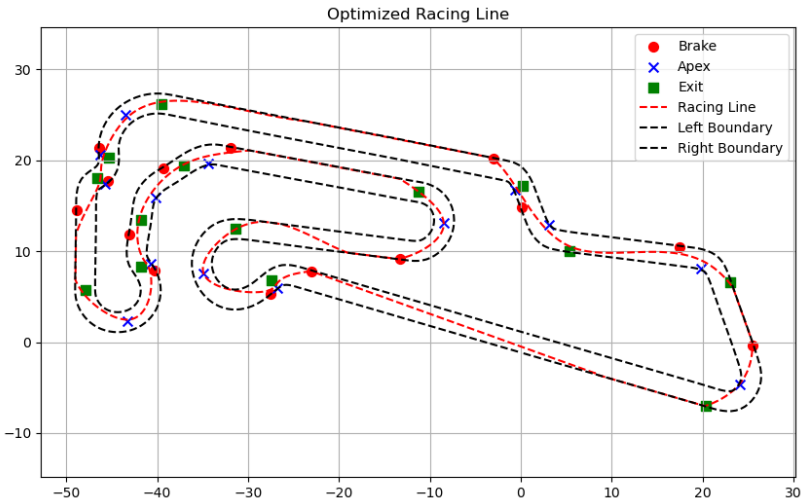


Figure 7: Racing line Oschersleben

To avoid the resulting race path intersecting with the track boundaries as can happen when relying on interpolation we are using the Polygon class from the Shapely python package to create bounds which can be used to easily check for points in the path which are outside the boundaries, these points are then pushed back towards to centerline until they are valid. To leave some amount of space between the bounds and the car a smaller bound is used for this than the 2.2m width of the track.

From experimentation a width of 1.2m works pretty well for this boundary using our controller, but it is dependent on the controller as a more precise controller should be able to handle paths much closer to the bounds of the track.

Tuning

The values for offseting waypoints such as apex and braking point were manually tuned based both on heuristics and experimenting with the simulation tool this also means that the variables are not optimal for any specific track. For the development the Oschersleben track shown in the figures so far had been due to it being relatively small and simple, however we found that the manually tuned variables did not function very well for the larger F1 tracks, so we decided to use the F1 tracks for testing and the variables were tuned for those tracks instead.

```python
for i, (start, end) in enumerate(corner_segments):
    # Direction offset
    dir_offset = -1 if corner_directions[i] == "Left Turn" else 1
    dir_offset2 = dir_offset
    exit_dist = 4
    exit_offset = 0
    # Corner type settings
    if corner_type[i] == "Late Apex":
        brake_dist, apex_pos, exit_dist = 5, 4, 0
        exit_offset = 0
    elif corner_type[i] == "Early Apex":
        brake_dist, apex_pos = 7, -2
        exit_offset = -1
    elif corner_type[i] == "Geometric Apex":
        brake_dist, apex_pos = 4, 0
    elif corner_type[i] == "Hairpin":
        brake_dist, apex_pos = 4, 0
        dir_offset2, exit_dist = 0, 0
```

(a) Previous values for smaller tracks

```python
for i, (start, end) in enumerate(corner_segments):
    # Direction offset
    dir_offset = -0.7 if corner_directions[i] == "Left Turn" else 0.7
    dir_offset2 = -0.1 if corner_directions[i] == "Left Turn" else 0.1
    exit_dist = 3
    exit_offset = 0
    # Corner type settings
    if corner_type[i] == "Late Apex":
        brake_dist, apex_pos, exit_dist = -1, 3, -1
        exit_offset = 0
    elif corner_type[i] == "Early Apex":
        brake_dist, apex_pos = 4, -2
        exit_offset = -1
        dir_offset2 = -0.1 if corner_directions[i] == "Left Turn" else -0.1
        exit_dist = 0
    elif corner_type[i] == "Geometric Apex":
        brake_dist, apex_pos = 3, 0
    elif corner_type[i] == "Hairpin":
        brake_dist, apex_pos = -4, 0
        dir_offset2, exit_dist = 0, -5
        exit_offset = 1
    elif corner_type[i] == "Sharp Turn":
        brake_dist, apex_pos = 1, -1
        dir_offset2 = 0
        exit_dist = -2
        exit_offset = 1
```

(b) Current values for F1 tracks

Figure 8: Comparison of tuned variables

Also worth noting is the addition of the sharp corner classification in the updated Raceline algorithm.

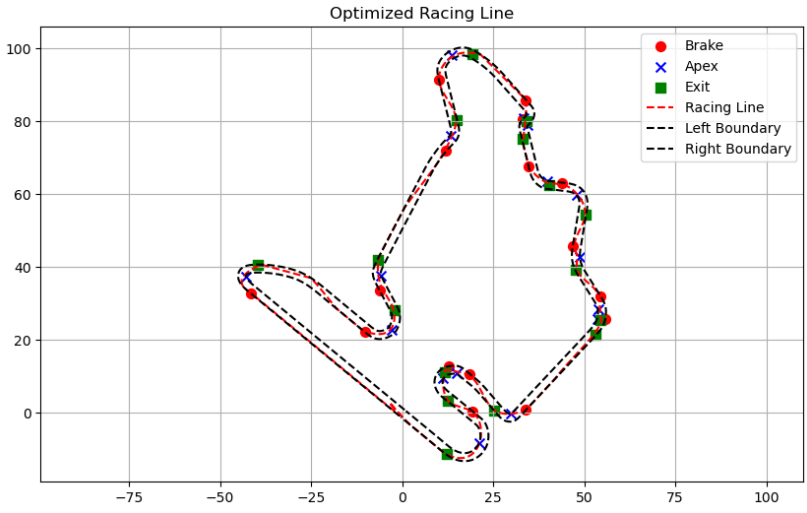


Figure 9: Racing line Budapest

4 Experimental setting

Simulation tool

The experiments was done using a slightly modified version of the F1tenth gym ros simulation tool, provided by the AU F1tenth community (find it here) which runs through docker containers simulating a ROS 2 car on a given map. This means the simulation both publishes and subscribes to a set of ROS 2 topics to simulate the real life equivalent.

Setup

The setup used for this paper was a windows machine running the containerized simulation with Docker desktop for windows using WSL integration to launch the containers from a Linux environment. Further information about installation and setup can be found on the Gitlab page.

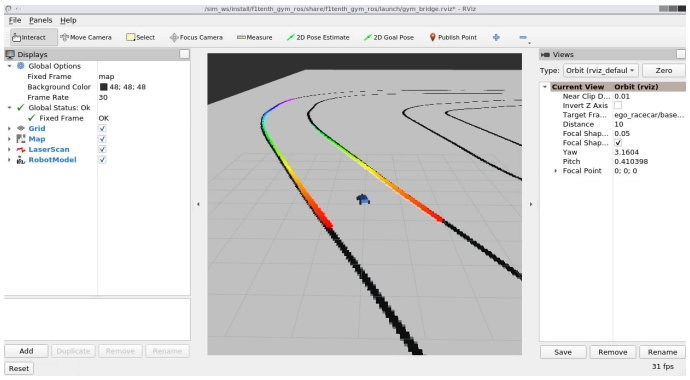


Figure 10: Simulation tool

Path following algorithm

To actually test the paths we made a path following algorithm based on Stanley controller, which while not ideal for exact path following is relatively simple and easy to implement. The algorithm uses a `lookahead` range to determine the goal at any point, we used a dynamic range to get more accuracy in corners while maintaining high speed in straights, as well as a dynamic speed which is limited by the magnitude of the steering angle. The algorithm is run as a Ros2 node, which initializes the car at $0, 0$ and then times the car until it has made a full lap. The algorithm also keeps track of the distance the car covers in each run.

The algorithm calculates and posts driving instructions in the Ackermann format which is for 4 wheeled robots with front steering and actual driving in the back wheels like a regular two wheel drive car, this type of steering is different from something like a differential controller where turning is done by setting a angular velocity in Rad/s, as here we are instead simply setting the angle of the wheels and need to adjust our linear velocity essentially to turn.

The algorithm has been somewhat modified to slow down the robot when the curvature of the path is high as well as to vary the look ahead distance based on the current speed and set look ahead gain, this both helps the controller follow the given path and simulate braking for and during corners.

```
1     curvature = 2.0 * y_veh / (lookahead ** 2)
2     curvature_gain = 4.0
```

```
    speed = max(0.1, min(5.0, 5.0 / (1 + curvature_gain * abs(curvature))))
```

Listing 3: Speed limitation

```
    def compute_dynamic_lookahead(self):
        return self.base_lookahead + self.lookahead_gain * self.current_speed
```

Listing 4: Dynamic lookahead

Experiment and variables

To strike a balance between time and a good amount of data it was decided that for each track we would do 20 runs of both the centerline path and the racing line path, for this purpose the controller was setup to automatically do 20 runs of both and exporting a csv file at the end containing the time and distance of each run.

For the experiment a number of assumptions were made either based on informed guesses or constraints as well as factors which are not taken into consideration which we will list here.

- No friction
- Maximum velocity of 5m/s
- Maximum turn angle of 0.34rad
- Gravity and wheight not considered
- Acceleration rate and braking time not considered

As mentioned earlier the algorithm is not ideal for following paths with high accuracy, but we tuned the variables to try and get as close as possible, this also means that the experiment is actually run with slightly different variables for centerline runs and racing line runs, as low lookahead distances makes the algorithm more accurate but only works well with the centerline. The results section will include some comparisons of the computed path to the one achieved during tests to show how accurate the runs actually were.

Here are the variable differences for the Stanley controller for racingline and centerline runs decided based on experimentation.

- Racingline: `base_lookahead=0.7, lookahead_gain=0.35, k=0.2, alpha=1.0`
- Centerline: `base_lookahead=0.2, lookahead_gain=0.25, k=0.2, alpha=1.0`

Below is a flowchart showing the process of creating a path and running the simulation, for real life testing the process would be much the same except replacing the simulation with an actual robot car running Ros2.

The simulation was run for 6 different tracks which are shown in the figure below, the tracks are somewhat varied in size to get some variance in the results

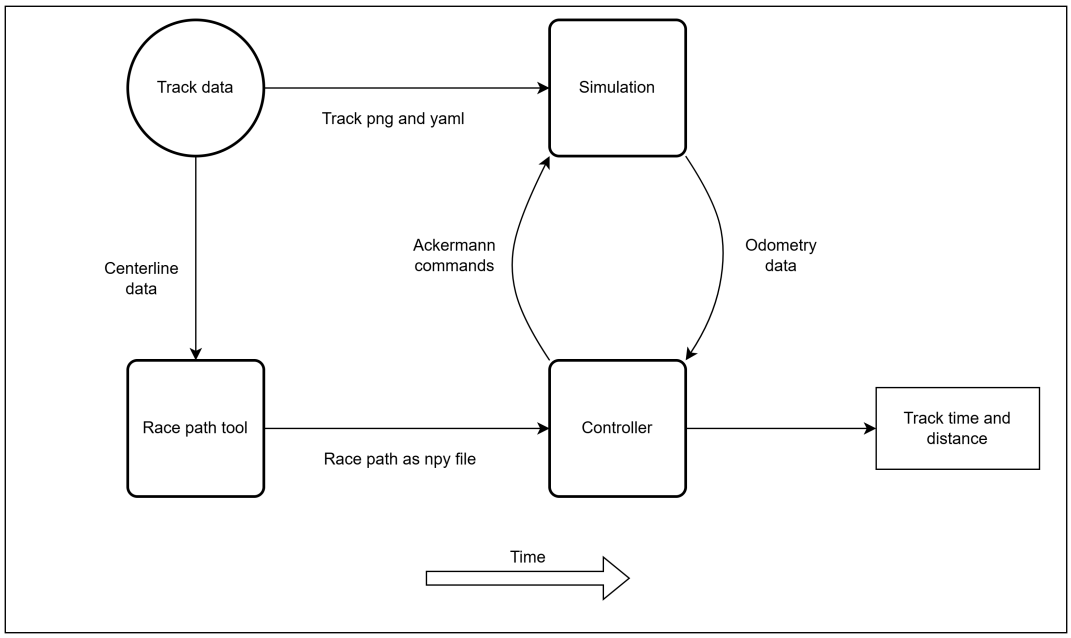
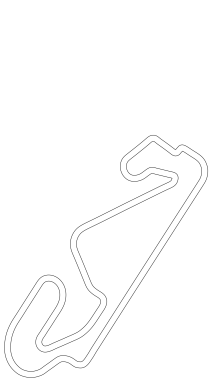
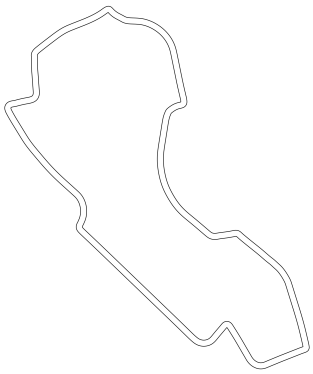


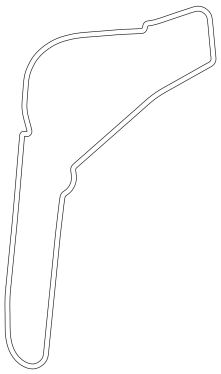
Figure 11: Flowchart of experiment process



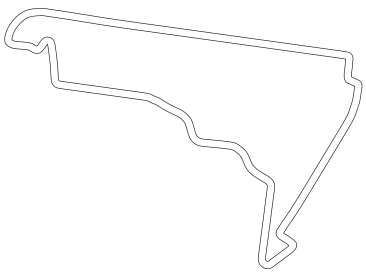
(a) Catalunya



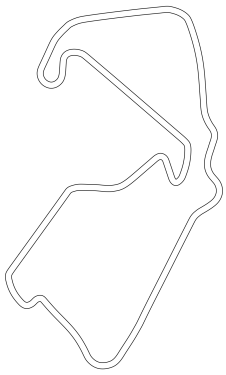
(b) Melbourne



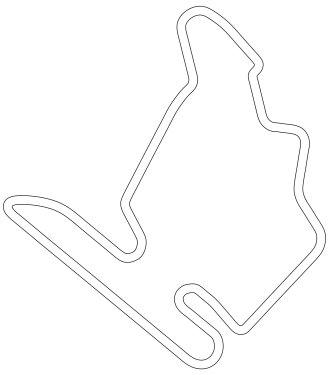
(c) Monza



(d) Mexico City



(e) Silverstone



(f) Budapest

Figure 12: Tracks used in experiment

Experiment data

The data which is collected is the Time and Distance for each lap, the process of timing a lap is pretty simple, once the car has been reset the time starts and once it is within 1m of starting position again the lap is ended and the time is saved. The total distance is collected with the below shown code which relies on the logged position of the car throughout the lap.

```
total_distance = sum(
        np.linalg.norm(np.array(self.log_pos[i + 1]) - np.array(self.log_pos[i]))
        for i in range(len(self.log_pos) - 1)
)
```

Listing 5: Lap distance calculation

The time and distance of each lap are saved along with the path type in a csv file for easy use later.

5 Results

This section presents and analyzes the outcomes from running both centerline and racing paths on six distinct tracks. The primary performance indicators are lap time, distance traveled, and efficiency (defined as time per unit distance). Table 1 presents the mean and standard deviation of these metrics for each track and path type. Table 2 shows the time saved by using the racing path compared to the centerline.

Track	Path Type	Time (s)		Distance (m)		Efficiency	
		mean	std	mean	std	mean	std
Budapest	centerline	112.54	1.394	400.583	0.074	0.281	0.003
Budapest	racing	97.91	1.93	399.855	0.27	0.245	0.005
Catalunya	centerline	113.19	0.743	416.836	9.682	0.272	0.007
Catalunya	racing	101.322	1.431	413.196	0.939	0.245	0.003
Melbourne	centerline	129.35	0.633	472.117	0.07	0.274	0.001
Melbourne	racing	114.845	4.085	472.188	0.13	0.243	0.009
MexicoCity	centerline	104.598	0.814	354.248	0.085	0.295	0.002
MexicoCity	racing	98.575	14.306	359.919	2.488	0.274	0.037
Monza	centerline	110.934	0.893	444.488	0.56	0.25	0.002
Monza	racing	101.405	1.097	445.162	0.257	0.228	0.002
Silverstone	centerline	116.569	0.272	455.932	0.082	0.256	0.001
Silverstone	racing	105.835	0.506	457.802	0.324	0.231	0.001

Table 1: Mean and standard deviation of results

Track	Mean time saved (s)
Budapest	14.630
Catalunya	11.869
Melbourne	14.505
MexicoCity	6.023
Monza	9.530
Silverstone	10.735

Table 2: Time saved

Analysis

Time Performance

On all tracks, the racing line consistently led to a shorter lap time than the centerline path. The most significant time savings were observed on the Budapest and Melbourne tracks, with improvements of

14.63s and 14.51s respectively. The smallest improvement was recorded on MexicoCity (6.02s), this was due to outliers caused by the only two times the car crashed in the simulation as these can also be observed in both figure 14 and the high standard deviation in table 1.

Distance comparison

Interestingly, the distance covered between the two paths was quite similar, with only slight variations. In some cases (e.g., Monza), the racing line even led to a longer path than the centerline, yet still resulted in faster lap times, highlighting the importance of optimal cornering and velocity profiles over strict path length minimization.

Efficiency Metric

The efficiency metric further supports the superiority of the racing path. On every track, the racing path achieved a lower time-per-meter value, confirming that the vehicle was not only faster in absolute terms but also more efficient in its traversal.

Statistical significance

To validate these findings, independent two-sample t-tests were performed. All tracks except MexicoCity showed statistically significant improvements in lap time using the racing path (p ¡ 0.001), confirming that the observed differences are unlikely due to chance. The result for MexicoCity (p = 0.0755) is not statistically significant at the 0.05 level, warranting further inspection of the path or track characteristics.

Track	T	p
MexicoCity	1.88	0.0755
Melbourne	15.69	0.0000
Monza	30.11	0.0000
Catalunya	32.91	0.0000
Silverstone	83.59	0.0000
Budapest	27.49	0.0000

Table 3: T-test

Overlap analysis

Figure 13 shows paths from the simulation on top of the given centerline and racing paths, as can be seen the controller does a near perfect job on the centerline but diverts a bit on the racing line path.

Box Plot Analysis

The box plots provide a clear comparative visualization of lap times and distances for both the centerline and racing line strategies across all tested tracks. Overall, the racing line consistently yields faster lap times, highlighting its effectiveness in minimizing curvature and allowing for more aggressive acceleration through corners. However, this improved performance comes at the cost of increased variability, as indicated by wider interquartile ranges and the presence of more outliers in

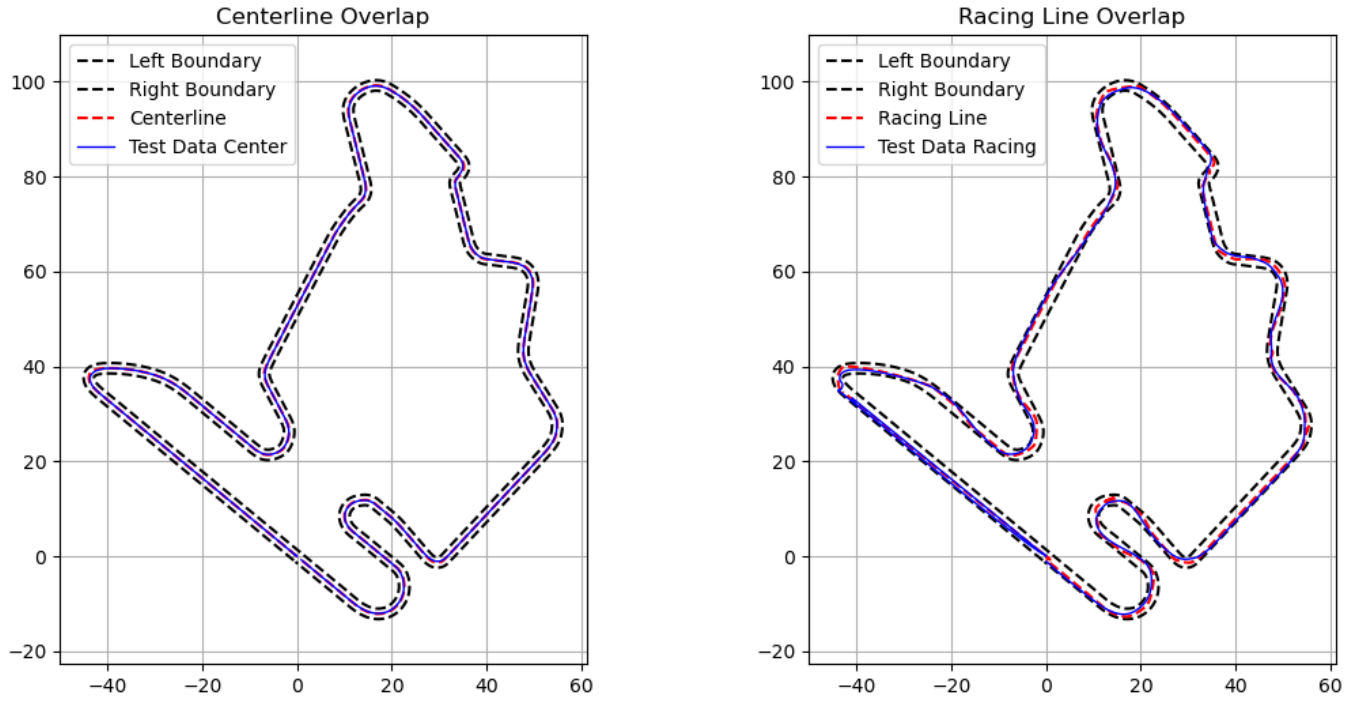


Figure 13: Budapest overlap

several tracks. This variability can partially be attributed to the controller as shown in figure 13 the controller is less accurate in following the racing line and therefore both time and distance deviates more for each lap for these.

Interestingly, while the racing line offers speed advantages, it is also, on average, slightly longer in total distance compared to the centerline. This supports the classic racing strategy trade-off: it is often more beneficial to carry more speed through a longer path than to take the shortest geometric route. This trade-off appears particularly advantageous on tracks with flowing, high-speed sections such as Monza and Silverstone.

An exception to this trend is observed in the Mexico City track, where two significant outliers appear in the racing line data. These correspond to crashes or controller instability during those runs, which negatively impacted lap times and highlight a potential drawback of the more aggressive trajectory. This suggests that while the racing line strategy is generally superior in controlled conditions, its success is more sensitive to system reliability and environmental stability.

Overall, the box plot analysis confirms the racing line's performance benefits while also underscoring the importance of robustness and precision in trajectory execution, especially on tighter or more complex circuits.

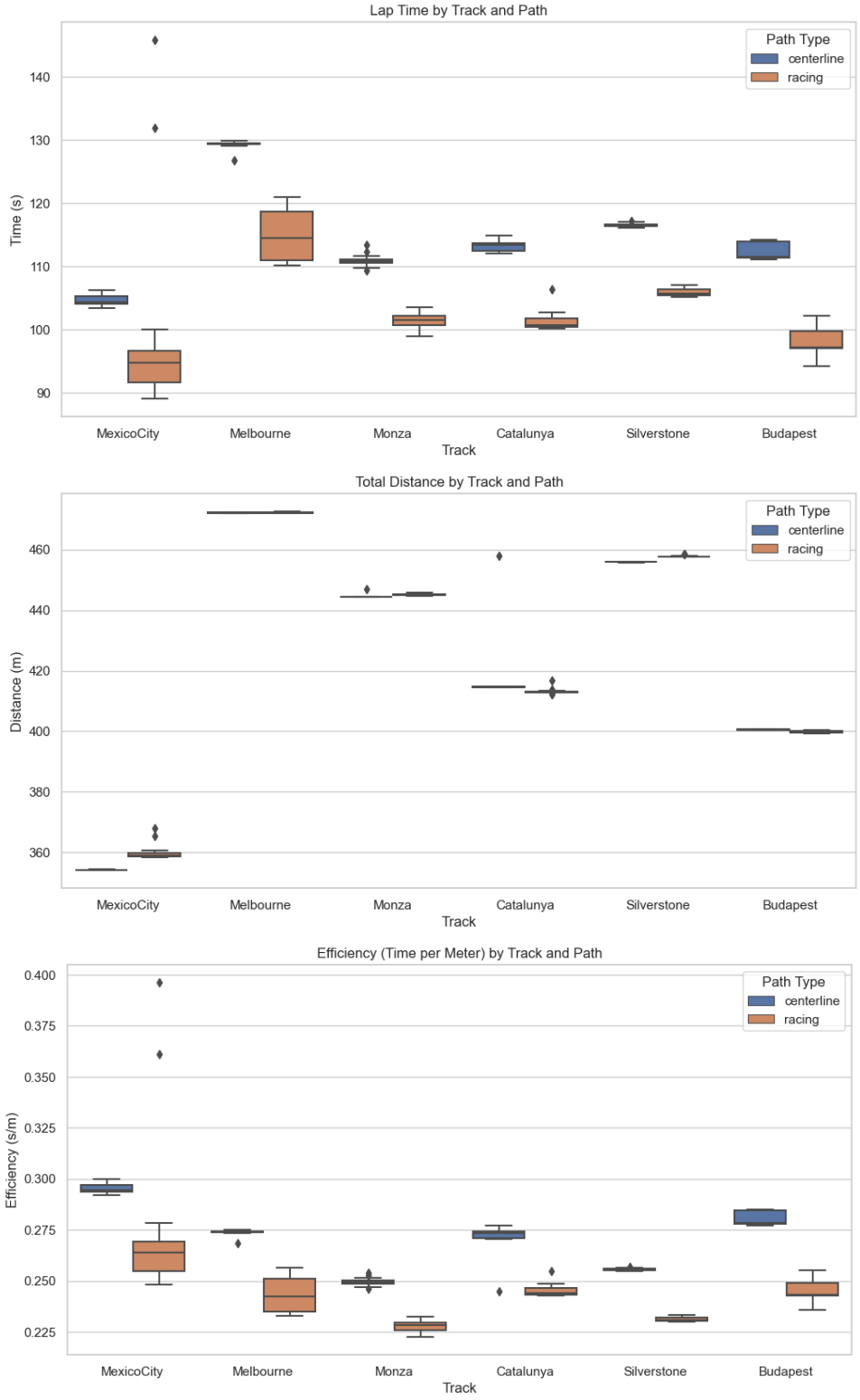


Figure 14: Box plot analysis

6 Related work

Path planning and path following are fundamental aspects of autonomous vehicle navigation. Numerous approaches have been proposed in the literature to generate and follow trajectories that optimize performance metrics such as safety, comfort, and lap time. Among these, spline-based path generation has become popular due to its smoothness and continuity, which are critical for safe and stable motion Muelling, Kretzschmar, and Zico Kolter, 2013. B-splines and cubic splines are widely used to represent paths due to their controllable curvature properties Zhang and Wang, 2018.

In the context of racing, the traditional approach uses the geometric centerline of the track, which simplifies navigation but often leads to sub-optimal lap times. Recent work has focused on generating so-called "racing lines," which optimize for performance, typically minimizing lap time by adjusting the path to exploit track width and curvature. For instance, the work by Liniger et al. Liniger, Domahidi, and Morari, 2015 uses optimization-based methods to derive racing lines considering vehicle dynamics and track geometry. Similarly, Rosolia and Borrelli Rosolia and Borrelli, 2019 introduce learning-based model predictive control to iteratively improve the racing line based on previous laps.

Path following controllers like Pure Pursuit, Stanley, and model predictive control (MPC) have been extensively compared in literature. Stanley, in particular, is known for its robustness to noisy position estimates and has been used effectively in many autonomous driving competitions Mike Montemerlo et al., 2006. However, most studies focus on controller performance without deeply analyzing how path curvature affects following efficiency and controller effort.

Several studies have also highlighted the importance of curvature in trajectory optimization. Curvature-based analysis can reveal insights into the handling characteristics of the vehicle along different segments of the path. In particular, comparing the curvature of centerline versus racing line can inform how path design influences both control smoothness and vehicle efficiency Seo, Lee, and Kim, 2017.

This project builds upon these existing works by performing a comparative analysis of centerline and racing line trajectories across various tracks. It incorporates curvature analysis and evaluates the impact of different path geometries on time, distance, and efficiency, thereby contributing to a more nuanced understanding of how path design influences autonomous racing performance.

7 Discussion

The experimental results provide several important insights into the performance and behavior of different path types—namely, the geometric centerline and the optimized racing line—across multiple tracks. Overall, the racing line significantly outperforms the centerline in terms of time efficiency on all tracks except for Mexico City, where the difference was not statistically significant.

Impact of Racing Line Optimization

The t-test results confirm that for most tracks (Catalunya, Budapest, Melbourne, Monza, Silverstone), the racing line results in a statistically significant improvement in lap time. This reinforces the importance of path optimization in high-speed autonomous navigation, where even small adjustments to the trajectory can lead to meaningful performance gains. The exception at the Mexico City track could suggest that the track's geometry either offers limited room for optimization or already aligns well with the centerline path in critical regions.

Controller Sensitivity and Parameter Tuning

Switching between paths with different curvature profiles necessitates careful adjustment of controller parameters such as lookahead distance, lookahead gain, and cross-track error gain (k). Without adaptive tuning, fixed parameters may be sub-optimal for specific path geometries. The results suggest that a controller tuned for a centerline path may underperform when following a racing line and vice versa. Future work could explore adaptive or learning-based methods to dynamically adjust controller parameters based on curvature or expected trajectory behavior.

Distance vs Time Trade-offs

Interestingly, while the racing line often results in slightly longer paths in terms of distance, the time savings indicate that the vehicle maintains higher average speeds. This highlights the effectiveness of racing lines in minimizing deceleration, especially through corners, by managing curvature and maximizing straight-line travel.

Limitations and Future Work

Some limitations of the current approach include the use of fixed controller parameters, limited vehicle dynamics modeling, and reliance on 2D path data without full kinematic constraints. Future extensions could include:

- Using a dynamic model predictive controller (MPC) to further exploit racing line advantages.
- Performing longitudinal-lateral coupling analysis to quantify trade-offs between speed and path accuracy.
- Including tire slip and friction modeling for more realistic racing simulations.
- Integrating dynamic variable adjustments to the path generator to better suit individual tracks based on size or the corners themselves.

- Expand usage to tracks of non-uniform width
- Introduce more realistic acceleration and braking constraints to the controller
- Real world experimentation

Extensions like using a MPC controller would allow for more accurate and in depth results with the current path generation whilst other extensions such as dynamic variable adjustments and acceleration and breaking constraints would help to create more accurate and detailed paths.

Overall, the combination of curvature analysis, statistical evaluation, and performance visualization provides a comprehensive framework for understanding how path geometry affects autonomous driving performance.

8 Discussion

The experimental results provide several important insights into the performance and behavior of different path types—namely, the geometric centerline and the optimized racing line—across multiple tracks. Overall, the racing line significantly outperforms the centerline in terms of time efficiency on all tracks except for Mexico City, where the difference was not statistically significant.

Impact of Racing Line Optimization

The t-test results confirm that for most tracks (Catalunya, Budapest, Melbourne2, Monza, Silverstone), the racing line results in a statistically significant improvement in lap time. This reinforces the importance of path optimization in high-speed autonomous navigation, where even small adjustments to the trajectory can lead to meaningful performance gains. The exception at the Mexico City track could suggest that the track's geometry either offers limited room for optimization or already aligns well with the centerline path in critical regions.

Curvature and Handling Complexity

Curvature analysis between the centerline and racing line reveals that the racing line tends to smooth out sharp corners and better exploit the full track width, reducing the curvature peaks that often challenge controller stability. This smoothing contributes to reduced controller effort and less aggressive steering adjustments. However, it also introduces trade-offs: in some tracks, increased curvature in certain segments may challenge tuning parameters of path-following controllers like Stanley or Pure Pursuit.

Controller Sensitivity and Parameter Tuning

Switching between paths with different curvature profiles necessitates careful adjustment of controller parameters such as lookahead distance, lookahead gain, and cross-track error gain (k). Without adaptive tuning, fixed parameters may be sub-optimal for specific path geometries. The results suggest that a controller tuned for a centerline path may underperform when following a racing line and vice versa. Future work could explore adaptive or learning-based methods to dynamically adjust controller parameters based on curvature or expected trajectory behavior.

Distance vs Time Trade-offs

Interestingly, while the racing line often results in slightly longer paths in terms of distance, the time savings indicate that the vehicle maintains higher average speeds. This highlights the effectiveness of racing lines in minimizing deceleration, especially through corners, by managing curvature and maximizing straight-line travel.

Limitations and Future Work

Some limitations of the current approach include the use of fixed controller parameters, limited vehicle dynamics modeling, and reliance on 2D path data without full kinematic constraints. Future extensions could include:

- Using a dynamic model predictive controller (MPC) to further exploit racing line advantages.
- Integrating curvature-aware adaptive gains for more robust controller performance.
- Performing longitudinal-lateral coupling analysis to quantify trade-offs between speed and path accuracy.
- Including tire slip and friction modeling for more realistic racing simulations.

Overall, the combination of curvature analysis, statistical evaluation, and performance visualization provides a comprehensive framework for understanding how path geometry affects autonomous driving performance.

9 Future Work

While this project has demonstrated the effectiveness of a mathematical curvature based approach to path planning and following using centerline and racing line trajectories, there are several avenues for future exploration and enhancement:

- **Dynamic Parameter Optimization:** Future systems could benefit from adaptive parameter tuning for the Stanley controller based on real-time feedback or predicted path curvature. Techniques such as reinforcement learning or model predictive control (MPC) could be employed to optimize control performance under varying conditions.
- **Generalization Across Tracks:** Although racing lines showed consistent performance gains, their generation was track-specific. Investigating generalizable strategies for racing line estimation or using neural networks to predict optimal trajectories across different layouts would extend applicability.
- **Sensor Noise and Localization Uncertainty:** The current setup assumes near-perfect localization. Incorporating noise models or testing with SLAM-based localization would help assess controller robustness in more realistic scenarios.
- **Higher-Fidelity Vehicle Models:** Expanding the controller to work with a more accurate vehicle dynamics model (e.g., including slip, tire forces, or acceleration time) could lead to better real-world transferability.
- **Competitive Multi-Agent Environments:** Future experiments could introduce multiple vehicles sharing the track, introducing constraints such as overtaking, dynamic blocking, and strategic racing line selection based on opponent behavior.

These extensions would provide a richer testbed for autonomous driving research and bring the system closer to real-world racing conditions and autonomy challenges.

Bibliography

Coulter, Robert C (1992). *Implementation of the Pure Pursuit Path Tracking Algorithm*. Tech. rep. Carnegie Mellon University, Robotics Institute.

Heilmeier, André et al. (2020). "Minimum curvature trajectory planning and control for time-optimal autonomous racing". In: *2020 American Control Conference (ACC)*. IEEE, pp. 571–578.

Kong, Jiliang et al. (2015). "Kinematic and dynamic vehicle models for autonomous driving control design". In: *IEEE Intelligent Vehicles Symposium (IV)*, pp. 1094–1099.

LaValle, Steven M (2006). *Planning algorithms*. Cambridge university press.

Liniger, Alexander, Andreas Domahidi, and Manfred Morari (2015). "Optimization-based autonomous racing of 1:43 scale RC cars". In: *European Control Conference (ECC)*, pp. 2459–2464.

Montemerlo, Mike et al. (2006). "The Stanford Racing Team's autonomous vehicle: "Junior"". In: *Journal of Field Robotics* 23.9, pp. 661–692.

Muelling, Katharina, Henrik Kretzschmar, and J. Zico Kolter (2013). "Minimum-time trajectory generation for quadrotors and ground vehicles". In: *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4002–4009.

Paden, Brian et al. (2016). "A survey of motion planning and control techniques for self-driving urban vehicles". In: *IEEE Transactions on intelligent vehicles* 1.1, pp. 33–55.

Piegl, Les and Wayne Tiller (1996). *The NURBS book*. Springer Science & Business Media.

Rosolia, Ugo and Francesco Borrelli (2019). "Learning model predictive control for autonomous racing". In: *IEEE Transactions on Intelligent Transportation Systems* 20.11, pp. 4081–4090.

Seo, Taeyoung, Dongsik Lee, and Hyun Myung Kim (2017). "Trajectory planning: A review". In: *IEEE International Conference on Control, Automation and Systems (ICCAS)*, pp. 1–8.

Snider, Jarrod M (2009). *Automatic steering methods for autonomous automobile path tracking*. Tech. rep. Carnegie Mellon University, Robotics Institute.

Thrun, Sebastian et al. (2006). "Stanley: The robot that won the DARPA Grand Challenge". In: *Journal of Field Robotics*. Vol. 23. 9. Wiley Online Library, pp. 661–692.

Zhang, Jian and Jia Wang (2018). "Curvature-constrained trajectory planning using Bezier curves". In: *Robotics and Autonomous Systems* 99, pp. 1–10.

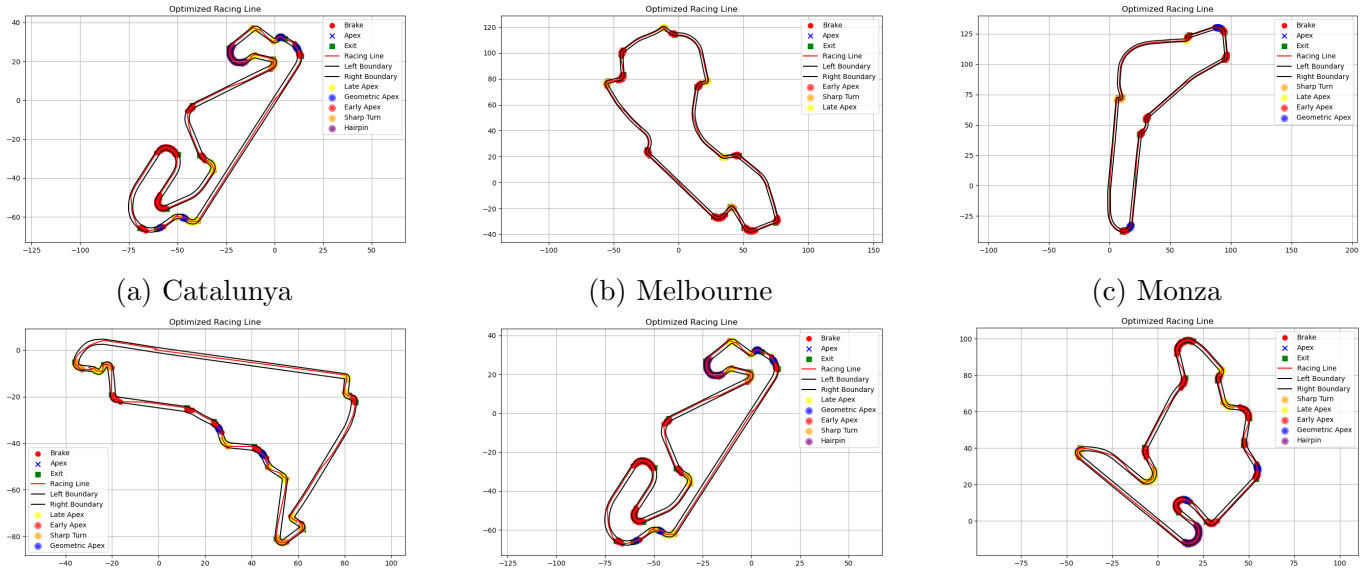
List of Figures

List of Tables

List of Codes

A Appendix

Racing lines



(a) Catalunya (b) Melbourne (c) Monza

(d) Mexico City (e) Silverstone (f) Budapest

Figure 15: Generated racing lines for tracks

Data

The full tables of data are available in this sheets document

Code

The full code for path generation and the stanley based controller can be found in this repository