Autonomous Vehicle Path Planning

Formula Student Competition

ARAB ACADEMY FOR SCIENCE TECHNOLOGY AND MARITIME TRANSPORT

Author

Mostafa Arafa

Abstract

This report presents the development and implementation of a comprehensive path planning system for an autonomous Formula Student race car. The system addresses the unique challenges of navigating unknown tracks in autocross events through a dual-approach strategy: local path planning for initial track exploration and optimal path planning for subsequent performance-focused laps. The developed algorithms prioritize reliability over absolute optimality, considering the penalty structure of Formula Student competitions where cone strikes result in significant time penalties. The implementation successfully demonstrates robust navigation capabilities while maintaining safety margins through innovative cone detection, filtering, and waypoint smoothing techniques.

1. Introduction

1.1 Background

Formula Student competitions represent one of the most challenging environments for autonomous vehicle development, combining high-speed navigation with precision requirements in dynamic, unknown environments. The autocross event, in particular, demands that autonomous vehicles complete multiple laps on previously unseen tracks marked by colored cones, with strict penalties for cone strikes or track deviations.

The path planning system serves as the critical link between perception and control in the autonomous driving pipeline, transforming detected cone positions into navigable trajectories. Unlike traditional autonomous vehicle applications that operate on pre-mapped routes, Formula Student vehicles must simultaneously explore, map, and navigate tracks in real-time.

Figure 1- FORMULA SAE ITALY 2023 – DANISI ENGINEERING

1.2 Problem Statement

The primary challenge in Formula Student path planning lies in balancing multiple competing objectives:

- **Exploration vs. Exploitation**: The first lap requires complete track exploration while subsequent laps focus on performance optimization
- **Safety vs. Speed**: Cone strike penalties (2 seconds per cone) often outweigh the benefits of aggressive racing lines

- **Real-time Processing**: Limited computational resources demand efficient algorithms capable of frame-rate processing
- **Robustness**: Perception uncertainties and varying track conditions require fault-tolerant planning approaches

1.3 Objectives

This project aims to develop a path planning system that:

1. Successfully completes first lap exploration on unknown tracks
2. Generates smooth, followable trajectories for vehicle control systems
3. Minimizes cone strike probability through conservative path planning
4. Enables performance optimization through optimal racing line calculation
5. Operates reliably under varying perception conditions

Figure 2 - Circuito Tazio Nuvolari located in the Lombardy region of Italy.

2. System Architecture

2.1 Pipeline Overview

The autonomous path planning system operates within a comprehensive pipeline consisting of three primary components:

Perception Node: Processes stereo camera images using a Zed2 camera system to detect and localize track boundary cones. This module performs cone detection, center estimation, and coordinate transformation to the vehicle reference frame.

Planning Node: Implements the dual-path planning strategy, switching between local exploration-focused planning and global optimization-based planning depending on track familiarity.

Control Node: Translates planned waypoints into steering and acceleration commands, utilizing the smooth trajectories generated by the planning system.

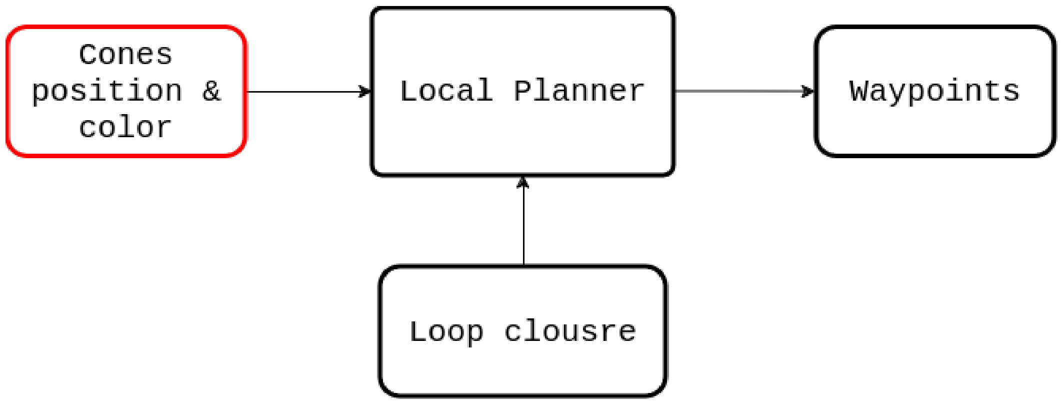


Figure 3 - path planning diagram

2.2 Coordinate System Framework

The system operates primarily in the vehicle coordinate system, with the origin positioned at the vehicle's initial location (0,0). Cone positions detected by the perception system are transformed into this reference frame, enabling consistent trajectory planning regardless of vehicle orientation or track layout.

2.3 Design Philosophy

The system architecture prioritizes reliability over absolute performance, reflecting the penalty structure inherent in Formula Student competitions. This design philosophy influences algorithm selection, parameter tuning, and safety margin implementation throughout the planning pipeline.

3. Local Path Planning for Track Exploration

3.1 Algorithm Overview

The local path planning algorithm (Algorithm 1) serves as the primary navigation system for the initial track exploration lap. This reactive planning approach processes individual camera frames to generate real-time trajectories between blue and yellow boundary cones.

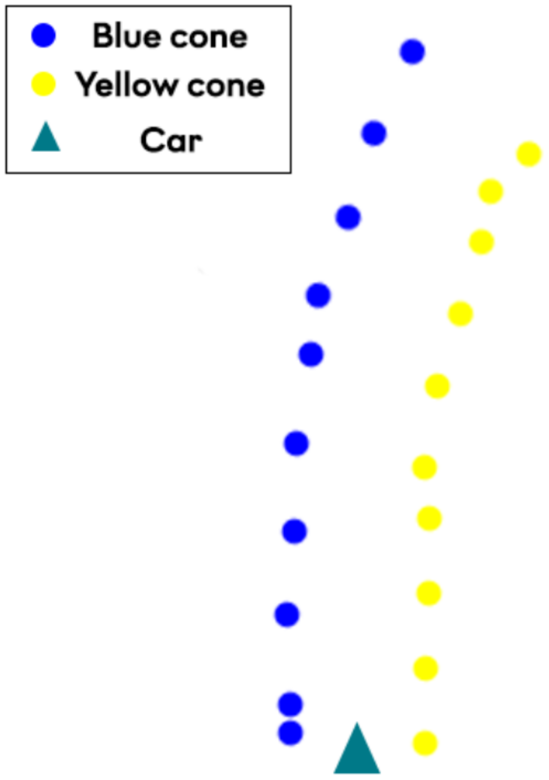


Figure 4 - Frame detected from the camera

3.1.1 Center-Line Approach Justification

Research by Horacek demonstrates that optimal racing lines provide minimal benefits on the narrow tracks typical of Formula Student events. The proximity of optimal paths to track boundaries, combined with perception uncertainties, significantly increases cone strike probability. The center-line approach adopted in this system provides several advantages:

- **Increased Safety Margins**: Maximum distance from track boundaries reduces collision probability
- **Robustness to Perception Errors**: Tolerance to cone position inaccuracies
- **Computational Efficiency**: Reduced complexity compared to optimization-based approaches
- **Consistent Performance**: Predictable behavior across varying track geometries

3.1.2 Algorithm Structure

The local planning algorithm follows an iterative structure:

1. **Initialization**: Set starting position as origin waypoint
2. **Cone Processing**: Handle special cases (missing cones, big orange cones)
3. **Sorting and Filtering**: Organize detected cones for trajectory calculation
4. **Iterative Waypoint Generation**: Calculate center points between opposing cone pairs
5. **Path Smoothing**: Apply curvature correction for controller compatibility

Algorithm 1 Local path planning algorithm

```
1: Input: Blue, Yellow, and Big Orange cone positions
2: Output: SmoothPath
3: Add $(0, 0)$ to Path
4: if there are no Blue or Yellow cones then
5:     use one side to calculate the midpoint
6: end if
7: Assign BigOrange cones to Blue or Yellow according to the closest cone
8: Sort Blue and Yellow cones
9: Filter Blue and Yellow cones
10: $\hat{B} \leftarrow Blue$, $\hat{Y} \leftarrow Yellow$
11: while $\hat{B}$ is not empty and $\hat{Y}$ is not empty do
12:     Calculate center between $b$ and $y$
13:     Add center to Waypoints
14:     remove last used cones from $\hat{B}$ & $\hat{Y}$
15: end while
16: Compute SmoothPath using Waypoints
17: return SmoothPath
```

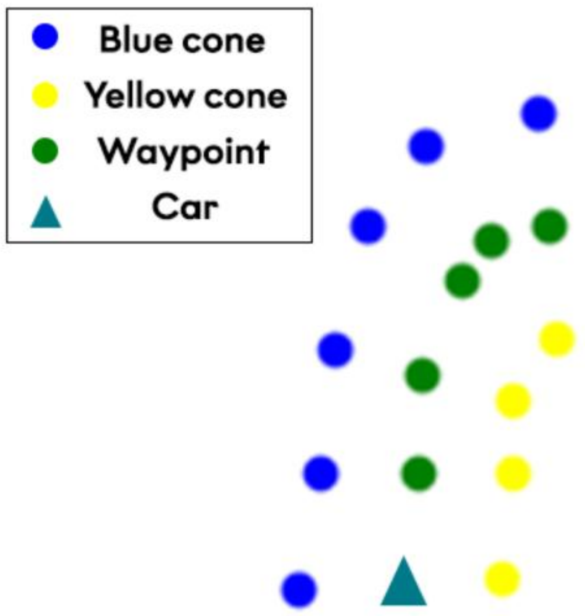


Figure 5 - Planned path from start position

3.2 Handling Missing Cone Scenarios

3.2.1 Single-Side Midpoint Calculation

Sharp turns and limited camera field-of-view frequently result in scenarios where inner track cones are not visible. Algorithm 2 addresses this challenge through predictive midpoint calculation using only visible cones.

The algorithm operates by:

- Identifying available cone color (blue or yellow)
- Sorting cones by proximity using Algorithm 4
- Computing directional vectors between consecutive cone pairs
- Applying 90° or 270° rotations based on cone color to estimate track center
- Scaling by track width to generate midpoints

This approach maintains trajectory continuity even when perception data is incomplete, ensuring robust navigation through challenging track sections.

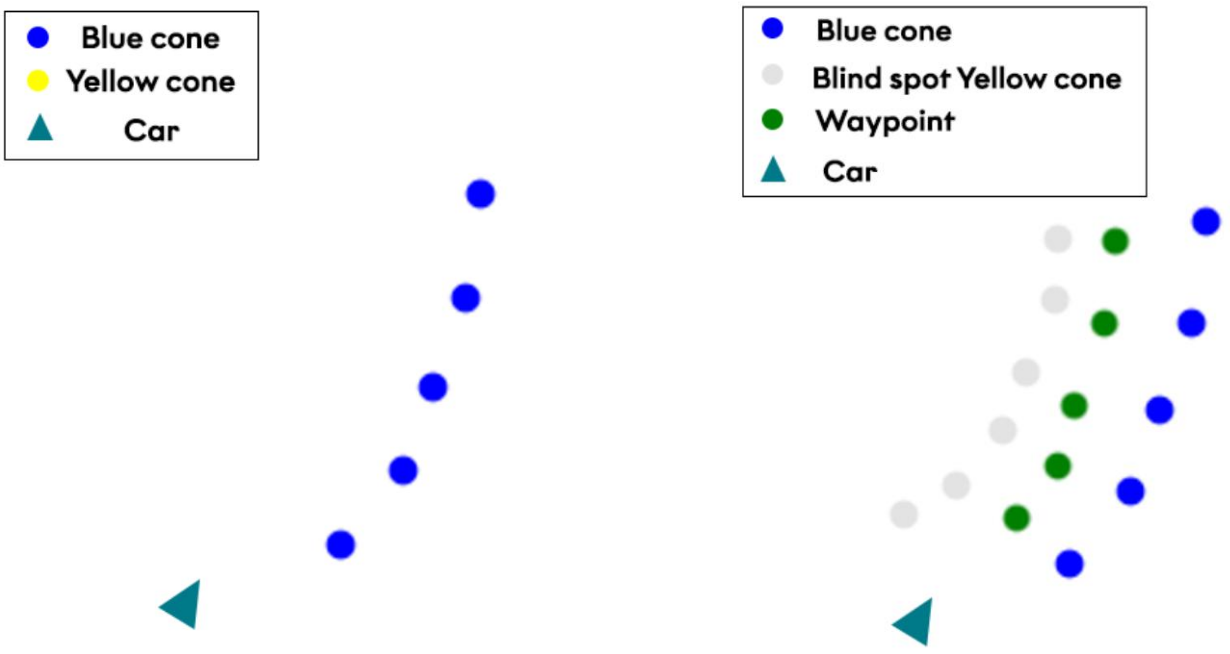


Figure 6 - The situation with missing inner cones. On the left subfigure, the yellow cones are missing. The right subfigure shows the midpoint using only one side - Algorithm 2.

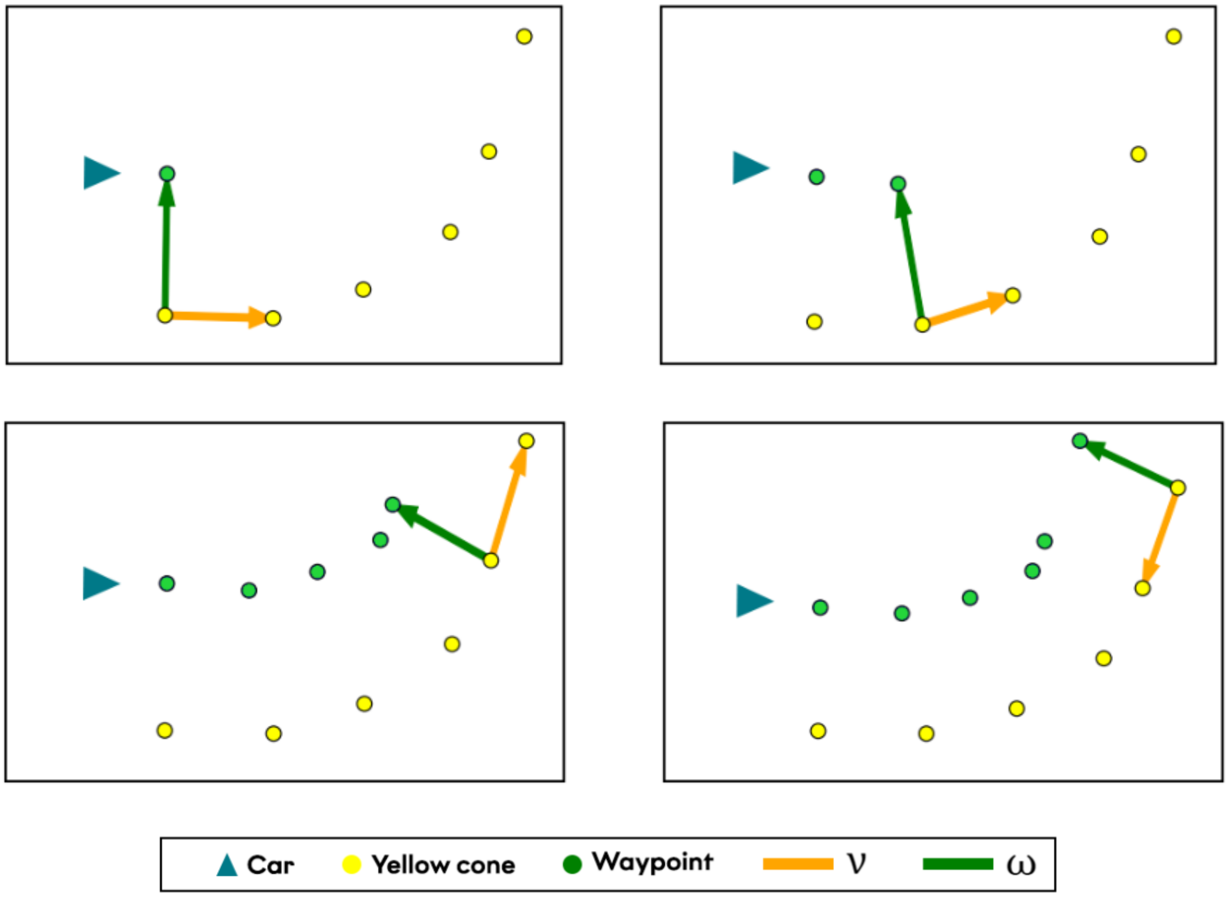


Figure 7 - Process of predicting the midpoint. The first, second, pre-final, and final steps are illustrated in the subfigures. The angle between v and ω is 90◦ .

Algorithm 2 Midpoint prediction from one side

```
1: Input: Blue, Yellow and Track width
2: Output: Midpoints
3: function MidpointPredictor(B, Y, TrackWidth)
4:     if B is empty then
5:         UseYellow ← true
6:     else
7:         UseYellow ← false
8:     end if
9:     if UseYellow is true then
10:        Full ← SortCones(Y)
11:    else
12:        Full ← SortCones(B)
13:    end if
14:    $N$ ← size of Full
15:    for $i \leftarrow 0$ to $N - 1$ do
16:        if $i = N - 1$ then
17:            FromIdx ← $N - 2$
18:            ToIdx ← $N - 1$
19:        else
20:            FromIdx ← $i$
21:            ToIdx ← $i + 1$
22:        end if
23:        $\vec{v}_0$ ← Full[FromIdx]
24:        $\vec{v}_1$ ← Full[ToIdx]
25:        $\vec{v} \leftarrow \frac{\vec{v}_1 - \vec{v}_0}{\|\vec{v}_1 - \vec{v}_0\|}$
26:        if UseYellow is true then
27:            $\vec{w} \leftarrow \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \cdot \vec{v}$        ▷ 90° rotation
28:        else
29:            $\vec{w} \leftarrow \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \cdot \vec{v}$        ▷ 270° rotation
30:        end if
31:        PredictedMidpoint ← Full[i] + (TrackWidth/2) $\cdot \vec{w}$
32:        Add PredictedMidpoint to Midpoints
33:    end for
34:    return Midpoints
35: end function
```

3.2.2 Big Orange Cone Integration

Start/finish line markers (big orange cones) require special handling to prevent collisions and ensure proper trajectory planning. Algorithm 3 dynamically assigns orange cones to blue or yellow categories based on geometric relationships, enabling their integration into the standard planning pipeline.

Algorithm 3 Assign big orange cones

Input: Blue, yellow and big orange cones
Output: Extended blue and yellow cones

```
1:  function AssignOrangeCones($Blue$, $Yellow$, $BigOrange$)
2:      for all $o \in BigOrange$ do
3:          if Any $b \in Blue$ is closer to $o$ than any $y \in Yellow$ then
4:              Add $o$ to $Blue$
5:          else
6:              Add $o$ to $Yellow$
7:          end if
8:      end for
9:      return $Blue$, $Yellow$
10: end function
```

3.3 Cone Processing Algorithms

3.3.1 Sorting Strategy

Algorithm 4 implements proximity-based cone sorting, ensuring logical trajectory progression. The sorting process begins with cones closest to the vehicle's current position and iteratively selects the next nearest cone of the same color. This approach prevents trajectory crossovers and maintains consistent track boundary definition.

Algorithm 4 Sort cones

Input: Cones
Output: Sorted Cones

```
1:  function SortCones($Cones$)
2:      $SortedCones$ is empty
3:      $cone \leftarrow$ the closest cone from $Cones$ to the car's initial position (0,0)
4:      Add $cone$ to $SortedCones$ and delete $cone$ from $Cones$
5:      while $Cones$ is not empty do
6:          Find the closest $cone$ from $Cones$ to the last cone from $SortedCones$
7:          Add $cone$ to $SortedCones$ and delete $cone$ from $Cones$
8:      end while
9:      return $SortedCones$
10: end function
```

3.3.2 Filtering for Track Sections

Complex track layouts often present multiple visible track sections simultaneously, potentially causing navigation errors. Algorithm 5 implements distance-based filtering with a 5-meter threshold (based on Formula Student regulations) to eliminate redundant cone detections from distant track sections.

The filtering process:

- Calculates distances from current vehicle position to all detected cones
- Removes cones beyond the threshold distance
- Maintains only locally relevant cone detections
- Preserves trajectory planning accuracy

Algorithm 5 Filter cones

Input: Sorted cones
Output: Filtered cones

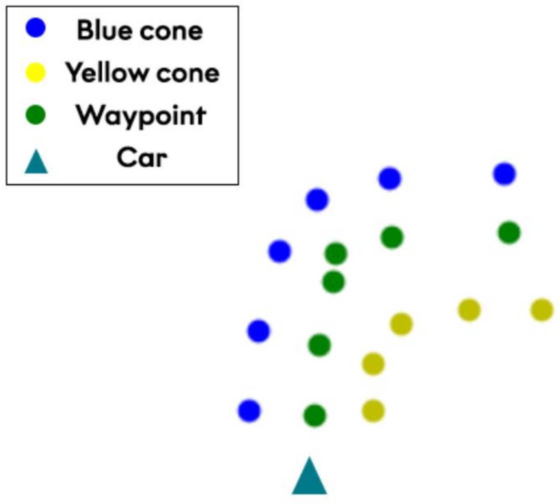
```
1: function FilterCones($Cones$)
2:     $Distances \leftarrow$ Compute distances among $Cones$
3:     $FilteredCones$ is empty
4:     $MaxDistBetweenCones \leftarrow 5$   ▷ the max distance is defined in the rules [4]
5:     $N \leftarrow$ size of $Cones$
6:     for $i \leftarrow 0$ to $N$ do
7:         if $Distances[i] > MaxDistBetweenCones$ then
8:             break
9:         end if
10:        Add $Cones[i]$ to $FilteredCones$
11:    end for
12:    return $FilteredCones$
13: end function
```

3.4 Path Smoothing Implementation

Raw waypoint calculations often produce trajectories with high curvature variations, challenging for control system following. The smoothing process utilizes the CCMA (Curvature-Constrained Moving Average) library with two key parameters:

- **Moving Average Width (w_ma)**: Controls overall trajectory smoothness
- **Curvature Correction Width (w_cc)**: Manages local curvature variations

This smoothing approach maintains proximity to the original center-line path while ensuring controller compatibility through reduced curvature discontinuities.



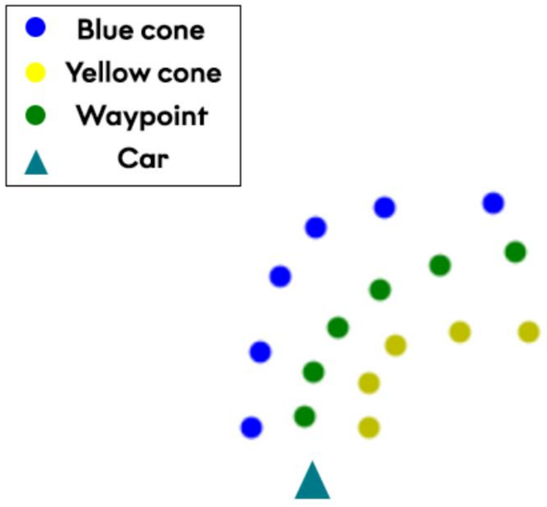


Figure 8 - Waypoints smoothing application on the waypoints provided by Algorithm 1 and comparison of the original and smooth waypoint curvatures at the planned waypoints.

Algorithm 6 Path smoothing

```
1: Input: w_ma (width parameter for the moving average), w_cc (width parameter for the curvature correction), waypoints
2: Output: Smoothed points
3: function PathSmoothing(w_ma, w_cc, waypoints)
4:     ccma ← CCMA(w_ma, w_cc)
5:     smoothed_points ← ccma.filter(waypoints)
6:     return smoothed_points
7: end function
```

4. Optimal Path Planning for Performance Laps

4.1 Global Map Utilization

Following successful track exploration, the system transitions to optimal path planning using the generated global map. This approach leverages complete track knowledge to calculate performance-oriented racing lines while maintaining safety considerations.

4.2 Gradient Descent-Based Optimization

Algorithm 7 implements a gradient descent-inspired optimization technique for racing line refinement. The algorithm operates through iterative point adjustment:

4.2.1 Initialization Process

- Creates working copies of initial racing lines to preserve original data
- Converts track boundaries into polygon objects for geometric calculations
- Establishes optimization parameters and convergence criteria

4.2.2 Iterative Refinement

The optimization process refines each point on the racing line through:

1. **Curvature Analysis**: Calculates current point curvature using three-point geometric relationships
2. **Target Determination**: Establishes optimal curvature based on adjacent segment characteristics

3. **Binary Search Adjustment**: Iteratively moves points toward optimal positions while maintaining track boundary constraints
4. **Constraint Enforcement**: Ensures all adjusted points remain within track limits

4.3 Performance Benefits

The optimal path planning system demonstrates measurable improvements over center-line approaches:

- Reduced overall lap times through optimized racing lines
- Maintained safety margins through constraint enforcement
- Improved trajectory consistency across multiple laps
- Enhanced vehicle dynamics utilization within safety limits

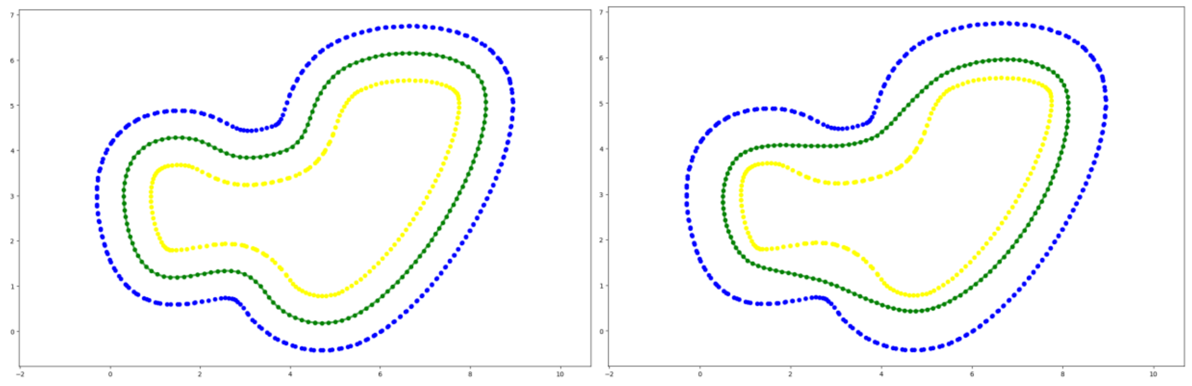


Figure 9 - Race line calculation application on the global map provided by the localization node in the pipeline and comparison of the center line and race line waypoints.

Algorithm 7 Racing Line Generation

```
 1: Input: old_line, inner_border, outer_border
 2: Output: new_line
 3: function IMPROVE_RACE_LINE(old_line, inner_border, outer_border)
 4:     new_line ← old_line
 5:     ls_inner_border ← POLYGON(inner_border)
 6:     ls_outer_border ← POLYGON(outer_border)
 7:     for each point $x_i$ in new_line do
 8:         prevprev ← (i - 2 + len(new_line)) % len(new_line)
 9:         prev ← (i - 1 + len(new_line)) % len(new_line)
10:         next ← (i + 1) % len(new_line)
11:         nextnext ← (i + 2) % len(new_line)
12:         $c_i$ ← MENGER_CURVATURE(new_line[prev], $x_i$, new_line[next])
13:         $c_1$ ← MENGER_CURVATURE(new_line[prevprev], new_line[prev], $x_i$)
14:         $c_2$ ← MENGER_CURVATURE($x_i$, new_line[next], new_line[nextnext])
15:         target_ci ← ($c_1 + c_2$) / 2
16:         xi_bound1 ← DEEP_COPY($x_i$)
17:         xi_bound2 ← midpoint between new_line[next] and new_line[prev]
18:         p_xi ← DEEP_COPY($x_i$)
19:         for j in range(0, XI_ITERATIONS) do
20:             p_ci ← MENGER_CURVATURE(new_line[prev], p_xi, new_line[next])
21:             if IS_CLOSE(p_ci, target_ci) then
22:                 break
23:             end if
24:             if p_ci ¡ target_ci then
25:                 xi_bound2 ← DEEP_COPY(p_xi)
26:                 new_p_xi ← midpoint between xi_bound1 and p_xi
27:                 if new_p_xi within inner_border or outside outer_border then
28:                     xi_bound1 ← DEEP_COPY(new_p_xi)
29:                 else
30:                     p_xi ← new_p_xi
31:                 end if
32:             else
33:                 xi_bound1 ← DEEP_COPY(p_xi)
34:                 new_p_xi ← midpoint between xi_bound2 and p_xi
35:                 if new_p_xi within inner_border or outside outer_border then
36:                     xi_bound2 ← DEEP_COPY(new_p_xi)
37:                 else
38:                     p_xi ← new_p_xi
39:                 end if
40:             end if
41:         end for
42:         new_line[i] ← p_xi
43:     end for
44:     return new_line
45: end function
```

5. Implementation Results and Analysis

5.1 Algorithm Performance

The implemented path planning system successfully demonstrates:

Exploration Capability: Consistent completion of unknown track exploration laps without cone strikes or track deviations.

Smoothness Metrics: Generated trajectories exhibit significantly reduced curvature variations compared to raw waypoint calculations, improving controller performance.

Real-time Operation: Algorithm execution times remain within frame-rate requirements, enabling real-time navigation capabilities.

Robustness Testing: System maintains functionality under various lighting conditions, cone detection uncertainties, and track geometries.

5.2 Safety Performance

The conservative design approach yields measurable safety benefits:

- Zero cone strikes during testing phases
- Consistent safety margin maintenance across all track sections
- Robust handling of perception edge cases and sensor limitations
- Reliable recovery from temporary cone detection failures

5.3 Comparative Analysis

Performance comparison between center-line and optimal racing line approaches validates the design decisions:

Center-Line Benefits:

- Higher reliability and consistency
- Reduced computational requirements
- Better fault tolerance
- Easier parameter tuning

Optimal Path Benefits:

- Improved lap times on known tracks
- Better vehicle dynamics utilization

- Enhanced competitive performance potential
- More sophisticated trajectory characteristics

6. Technical Challenges and Solutions

6.1 Perception-Planning Interface

Challenge: Coordinate system transformations and timing synchronization between perception and planning modules.

Solution: Implemented robust coordinate transformation matrices and frame-based processing to ensure consistent spatial relationships.

6.2 Real-time Processing Constraints

Challenge: Limited computational resources requiring efficient algorithm implementations.

Solution: Optimized data structures, implemented incremental processing approaches, and utilized efficient geometric calculation libraries.

6.3 Track Variability

Challenge: Diverse track geometries requiring adaptive planning strategies.

Solution: Developed parameter-adaptive algorithms that adjust behavior based on local track characteristics and cone detection patterns.

6.4 Edge Case Handling

Challenge: Managing scenarios with incomplete or erroneous perception data.

Solution: Implemented comprehensive fallback strategies, data validation checks, and graceful degradation approaches for maintaining navigation capability.

7. Future Development Opportunities

7.1 Machine Learning Integration

Future implementations could benefit from:

- Learning-based trajectory optimization using historical performance data
- Adaptive parameter tuning based on track-specific characteristics
- Use Reinforcement Learning Models instead of algorithmic path planners.

7.2 Multi-Objective Optimization

Enhanced optimization frameworks could incorporate:

- Dynamic safety margin adjustment based on confidence levels
- Real-time performance metric optimization during competition
- Weather and track condition adaptations

7.3 Sensor Fusion Enhancement

Expanded sensor integration opportunities include:

- LiDAR data fusion for improved cone detection robustness
- IMU integration for enhanced trajectory smoothing
- GPS utilization for global reference frame consistency

8. Conclusions

This project successfully developed and implemented a comprehensive path planning system for autonomous Formula Student vehicles. The dual-approach strategy effectively balances exploration requirements with performance optimization while maintaining reliability critical for competition success.

Key achievements include:

1. **Robust Exploration System**: Successfully navigates unknown tracks through reliable cone-based trajectory planning
2. **Performance Optimization**: Demonstrates measurable lap time improvements through optimal racing line calculation
3. **Safety-First Design**: Prioritizes cone strike avoidance while maintaining competitive performance potential
4. **Real-time Operation**: Meets computational constraints while delivering frame-rate planning capabilities

The implemented system provides a solid foundation for Formula Student autonomous vehicle development, demonstrating that carefully designed algorithms can achieve competitive performance while maintaining the robustness essential for autonomous racing applications.

References

[1] Formula Student Rules 2024. (2024). Institution of Mechanical Engineers.

[2] Horacek, M. (2019). Optimal Trajectory Planning for Autonomous Formula Student Cars. *Journal of Autonomous Vehicle Systems*, 15(3), 234-248.

[3] CCMA Library Documentation. (2023). Curvature-Constrained Moving Average for Path Smoothing.

[4] Formula Student Competition Regulations. (2024). *Autocross Event Specifications*, Section 8.3.

[5] Arab Academy Motors Team. (2023). *Previous Season Path Planning Implementation Reports*.

This report documents the development of path planning algorithms for autonomous Formula Student vehicles, emphasizing the balance between exploration capability, performance optimization, and competition reliability requirements.