# Comparative Study of Delaunay Path Planner and RRT/RRT* for Path Planning in Autonomous Racing

Rodrigo Carrión Caro

March 18, 2025

# Contents

# Glossary

- **RRT** - Rapidly-Exploring Random Tree
- **ROS2** - Robot Operating System 2
- **Unity** - Motor de simulación
- **Path Planning** - Planificación de rutas

# Abstract

## 0.1 Contexto

Escribe aquí el contexto del problema.

## 0.2 Objetivo

Escribe aquí la descripción clara del propósito del proyecto.

## 0.3 Metodología

Describe el enfoque de desarrollo y herramientas utilizadas.

## 0.4 Resultados principales

Explica los principales hallazgos y comparación entre métodos.

## 0.5 Conclusión

Menciona el impacto del estudio y posibles mejoras.

# Acknowledgements

Escribe aquí los agradecimientos al supervisor, equipo OBRA, etc.

Chapter 1

# Introduction

## 1.1 Background

In autonomous vehicles, path planning is essential, as it allows for efficient and safe route planning in real time. Specifically, the RRT algorithm "RRT Rapidly exploring Random Tree"can explore large spaces efficiently. This makes it ideal for dynamic environments, such as an autonomous car race. These characteristics make it widely used in the field of robotics. The RRT (Rapidly exploring Random Tree) algorithm is a search method used to efficiently find paths in large spaces [1]. It is particularly useful in dynamic environments where viable paths need to be found quickly in real-time. Currently, the Oxford Brookes Racing Autonomous (OBRA) team uses a neural network path planner. A Delaunay path planner is being developed but has not yet been implemented. These approaches present some limitations in real-time situations. The development and implementation of an RRT [2] will allow for greater flexibility and adaptability. Ideally, this will improve the team's path planning capabilities. In addition to developing the RRT, this project will compare its performance with the current Delaunay path planner. By evaluating their efficiency and adaptability in dynamic scenarios, the goal is to determine which algorithm provides better results for autonomous racing.

## 1.2 Aim

The aim of this project is to develop and optimize two advanced path planning algorithms for autonomous racing: an improved Delaunay-based planner and an optimized RRT* algorithm. These planners will be designed to enhance adaptability, computational efficiency, and trajectory optimization in high-speed, dynamic environments.

This project also aims to conduct a detailed comparative analysis of both algorithms, evaluating their performance across key metrics such as computation time, path efficiency, adaptability to dynamic obstacles, and robustness under racing conditions.

By implementing and testing these algorithms within a ROS2-based simulation environment using Unity, this research seeks to identify the most effective path planning solution for the OBRA team's autonomous racing vehicle. The insights gained from this study will contribute to both the OBRA competition strategy and the broader field of autonomous vehicle navigation.

## 1.3 Objectives

- To conduct a comprehensive study of path planning techniques used in autonomous vehicle navigation, analyzing their advantages, limitations, and applications in high-speed racing scenarios. [3]
- To develop and implement an improved Delaunay-based path planner that enhances adaptability, computational efficiency, and trajectory smoothness in dynamic environments.
- To develop and implement an RRT algorithm from scratch for real-time path planning in autonomous vehicles, ensuring compatibility with the OBRA car's ROS2 framework.
- To optimize the RRT algorithm by integrating RRT* [4], improving its ability to generate efficient and dynamically adaptable routes.
- To validate both the optimized Delaunay and RRT* algorithms in a controlled simulation environment using Unity, testing their performance under varying racing conditions.
- To conduct a detailed comparative analysis of the optimized Delaunay and RRT* algorithms, evaluating key performance metrics such as computation time, path efficiency, adaptability to dynamic obstacles, and robustness in high-speed scenarios.
- To identify and address potential limitations of each algorithm, proposing refinements or hybrid approaches that could further enhance their performance.
- To integrate the most effective path planner into the OBRA team's autonomous racing pipeline, ensuring real-world applicability and alignment with competition requirements.

## 1.4 Product Overview

### 1.4.1 Scope

The objective of this project is to develop and optimize two new path planners for autonomous racing:

- Optimized Delaunay Path Planner – A modified version of the traditional Delaunay-based planner, improving its efficiency and adaptability for dynamic racing environments.
- RRT* – An enhanced Rapidly-exploring Random Tree algorithm that generates smoother and more efficient paths by reducing randomness and refining route selection.

Both planners will be developed in Python, ensuring seamless integration with the ROS2 framework used in the OBRA autonomous car. The testing and validation process will be conducted in simulation environments using Unity, allowing for extensive evaluation before potential real-world implementation. Once developed, these two new planners will be compared to determine which provides better performance in terms of adaptability, computational efficiency, and trajectory smoothness under high-speed, dynamic conditions.

### 1.4.2 Audience

The primary audience for this study is the Oxford Brookes Racing Autonomous (OBRA) team, as the improved path planning algorithms will directly contribute to their autonomous racing performance. Additionally, this research is relevant to the academic community, particularly in robotics, AI, and autonomous vehicle navigation, by providing insights into optimization strategies for real-time path planning. From an industry perspective, this study holds significance for autonomous systems professionals, particularly those developing path planning solutions for high-speed and dynamic environments, such as self-driving vehicles, robotics, and UAV navigation.

# Chapter 2

# Background Review

## 2.1 Related Literature

| | |
|---|---|
| **Reference** | Zhao, H., Wu, Z., Li, Y., & Wang, J. (2021) 'Improved Bidirectional RRT* Path Planning Method for Smart Vehicle', Mathematical Problems in Engineering, pp. 1-14. doi: 10.1155/2021/6669728. |
| **Title** | Improved Bidirectional RRT* Path Planning Method for Smart Vehicle |
| **Summary** | The study proposes an improvement to the bidirectional RRT* algorithm to optimize route planning in intelligent vehicles, aiming for shorter and more efficient routes. |
| **Evaluation** | The approach is useful for static environments, but it does not sufficiently address the challenges in dynamic environments, which may limit its applicability in competitive vehicles. |
| **Reflection** | The proposed improvements could be applied to optimize the RRT* algorithm in my project, particularly in reducing computation time and optimizing routes. |
| **Main Themes** | RRT* optimization, route planning, autonomous vehicles. |

**Table 2.1:** Summary of Zhao et al. (2021)

| | |
|---|---|
| **Reference** | Gasparetto, A., Boscariol, P., Lanzutti, A., & Vidoni, R. (2015) 'Path Planning and Trajectory Planning Algorithms: A General Overview', Journal of Intelligent & Robotic Systems, pp. 1-33. doi: 10.1007/978-3-319-14705-5_1. |
| **Title** | Path Planning and Trajectory Planning Algorithms: A General Overview |
| **Summary** | The article provides an overview of the main trajectory and route planning algorithms in robotics. It analyses methods such as Roadmap, Cell Decomposition, and RRT, as well as their applications in industrial and autonomous environments. |
| **Evaluation** | The study offers a comprehensive overview of the algorithms but focuses more on static industrial systems, limiting its applicability to dynamic environments. However, the review of RRT is useful for improving my project. |
| **Reflection** | This article will be key to contextualizing my work, as it provides a solid foundation on traditional methods and suggests possible areas for improvement, such as applying them in more dynamic environments. |
| **Main Themes** | Route planning, RRT, optimization algorithms, autonomous robotics. |

**Table 2.2:** Summary of Gasparetto et al. (2015)

| | |
|---|---|
| **Reference** | Wang, H., Li, G., Hou, J., Chen, L., & Hu, N. (2022) 'A Path Planning Method for Underground Intelligent Vehicles Based on an Improved RRT* Algorithm,' Electronics, vol. 11, no. 3, p. 294. doi: 10.3390/electronics11030294. |
| **Title** | A Path Planning Method for Underground Intelligent Vehicles Based on an Improved RRT* Algorithm |
| **Summary** | The study proposes an improved RRT* method for route planning in underground intelligent vehicles, adjusting the dynamic step size and turn angle constraints. |
| **Evaluation** | The approach is innovative for underground environments and offers improvements in efficiency and safety, but it is limited to controlled spaces and does not address navigation in fully dynamic environments. |
| **Reflection** | This study is relevant to my project, as the proposed RRT* improvements could be applied to optimize the algorithm in more complex scenarios, such as autonomous racing. |
| **Main Themes** | RRT*, underground autonomous vehicles, route planning, optimization. |

**Table 2.3:** Summary of Wang et al. (2022)

| | |
|---|---|
| **Reference** | Sánchez-Ibáñez, J.R., Pérez-del-Pulgar, C.J., & García-Cerezo, A. (2021) 'Path Planning for Autonomous Mobile Robots: A Review', Sensors 2021, 21, 7898. doi: 10.3390/s21237898. |
| **Title** | Path Planning for Autonomous Mobile Robots: A Review. |
| **Summary** | The article reviews route planning algorithms for mobile robots, focusing on their classification and applicability in autonomous environments. |
| **Evaluation** | It provides a very useful overview of the main approaches, but it focuses on controlled scenarios and may be limited for dynamic environments such as competitions. |
| **Reflection** | This article provides a good foundation for comparing different approaches, which will help me justify the choice of RRT in my project. |
| **Main Themes** | Route planning, mobile robots, path search algorithms. |

**Table 2.4:** Summary of Sánchez-Ibáñez et al. (2021)

| | |
|---|---|
| **Reference** | C. Messer, A. T. Mathew, N. Mladenovic and F. Renda, "CTR DaPP: A Python Application for Design and Path Planning of Variable-strain Concentric Tube Robots," 2022 IEEE 5th International Conference on Soft Robotics (RoboSoft), Edinburgh, United Kingdom, 2022, pp. 14-20, doi: 10.1109/RoboSoft54090.2022.9762088. |
| **Title** | CTR DaPP: A Python Application for Design and Path Planning of Variable-strain Concentric Tube Robots. |
| **Summary** | The study presents a modular platform in Python that uses the RRT* algorithm for route planning and design optimization in concentric tube robots. It focuses on trajectory planning in environments with torsion and curvature constraints. |
| **Evaluation** | The implementation in Python is relevant to my project, as it allows for the flexible use of planning and optimization algorithms. |
| **Reflection** | This article supports the use of Python in my project, demonstrating that it is an effective option for developing and testing algorithms like RRT*. |
| **Main Themes** | RRT*, route planning, concentric tube robots, design optimization, use of Python. |

**Table 2.5:** Summary of Messer et al. (2022)

|  |  |
|---|---|
| **Reference** | Kolski, S., Ferguson, D., Stachniss, C., & Siegwart, R. (2006) 'Autonomous Driving in Dynamic Environments', Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1-10. doi: 10.3929/ethz-a-010079481. |
| **Title** | Autonomous Driving in Dynamic Environments. |
| **Summary** | The study presents a hybrid autonomous navigation system that operates in both structured and unstructured environments, handling dynamic obstacles like pedestrians and other vehicles. |
| **Evaluation** | Unlike many approaches focused on static environments, this system is highly relevant for dynamic settings, such as autonomous car competitions, where real-time adjustments to moving obstacles are critical. |
| **Reflection** | This study is essential for my project as it highlights the importance of dynamic environments and provides useful insights for improving my route planning system. |
| **Main Themes** | Autonomous navigation, dynamic environments, route planning, moving obstacles, autonomous vehicles. |

**Table 2.6:** Summary of Kolski et al. (2006)

# Chapter 3

# Methodology

## 3.1 Research and Software Development Process

### 3.1.1 Application of Agile Methodology, aqui meto el diagrama de Agile

The project follows an Agile methodology, allowing for iterative progress and continuous refinement of the algorithms. This approach ensures flexibility and adaptability to changing requirements. Agile was chosen over traditional models such as Waterfall, as the latter requires a rigid, sequential structure that does not accommodate modifications once development begins.

Given the dynamic nature of this project, the Agile framework supports the parallel development of:

- Optimization and rewriting of the existing **Delaunay path planner**.
- Development of a new **RRT* algorithm** from scratch.
- Comparative testing and analysis of both approaches.

The project is structured into multiple iterations, ensuring continuous validation and improvement of the implemented methods.

## 3.2 Development Phases

### 3.2.1 Preparation Phase

Before development, the project involved:

- **Topic selection and meetings** with OBRA to align the objectives.

- **Proposal submission and feedback** to refine the scope.
- **System and environment preparation**, including setting up Unity and ROS2.
- **Progress report submission** to document early findings.
- **Research and design for Delaunay optimization**, establishing the baseline for improvements.

### 3.2.2 Iteration 1: Delaunay Optimization and Rewriting

This phase focused on improving the existing Delaunay-based path planner:

- **Algorithm rewriting** to enhance efficiency and adaptability.
- **Implementation of the optimized Delaunay path planner**.
- **Testing and validation in simulation**, ensuring improved path generation.
- **Demo presentation** of Delaunay results before moving to RRT.

### 3.2.3 Iteration 2: RRT Development

The second iteration introduced the development of RRT:

- **Theoretical study of RRT** to understand its principles and limitations.
- **Research and design for basic RRT**.
- **Implementation of the basic RRT algorithm**.
- **Testing and validation in simulation** to compare with Delaunay.

### 3.2.4 Iteration 3: RRT* Optimization and Comparison

Once the basic RRT was implemented, optimization and comparative analysis followed:

- **Research and design for RRT***, improving route efficiency.
- **Comparative analysis of Delaunay and RRT***, evaluating performance under racing conditions.

### 3.2.5 Final Validation and Integration

After comparative testing, the final phase focused on integration:

- **Integration into the OBRA car**, ensuring real-world applicability.

### 3.2.6 Final Report and Presentation

The last step involved documentation and dissemination:

- **Final report draft** summarizing findings and results.
- **Final presentation preparation** for project submission.

## 3.3 Technology

### 3.3.1 Implementation Tools and Resources

The development of this project relies on several key technologies and tools to ensure efficient implementation and testing. The primary programming language used is Python, which serves as the backbone of the implementation. Python is used to develop the path-planning algorithms, integrate them into ROS2, and manage the necessary data processing tasks. Its extensive libraries and compatibility with machine learning and robotic frameworks make it ideal for rapid prototyping and testing.

For robotic integration, we utilize ROS2 (Humble), which facilitates seamless communication between different components of the autonomous system. ROS2 manages sensor data processing, real-time control, and path planning, ensuring that all modules operate in a synchronized manner. The middleware's efficiency in handling multiple nodes is crucial for real-time decision-making within the autonomous vehicle.

The project runs on Ubuntu 22.04, which provides a stable and well-supported environment for ROS2, Python-based applications, and simulation tools. Ubuntu ensures compatibility with the robotic frameworks used in this project and offers an extensive community for troubleshooting and optimization.

For version control, Git and GitLab serve as the central repository for managing code changes. The OBRA workspace on GitLab stores previous developments and is used to track new implementations through branches. Frequent commits help maintain a structured development workflow.

Simulation and testing are conducted using a structured environment that combines Unity for scenario modeling and Foxglove for real-time visualization of vehicle states and planned paths. The simulation outputs, along with test results, are stored in Google Drive, ensuring efficient data management and accessibility for analysis.

To maintain an organized workflow, Notion is used for task management and Agile planning, ensuring that development is well-structured and efficiently executed.

By integrating these technologies, we aim to develop a robust and scalable path-planning solution that seamlessly integrates with the OBRA autonomous vehicle.

### 3.3.2 Workspace, aquí puedo meter el graphq del ecosistema de OBRA

The OBRA workspace is structured to maintain a modular approach, ensuring that different path-planning algorithms can be developed and tested without interfering with the overall system architecture.

### 3.3.3 Simulation and Testing Tools

The simulation and testing phase plays a crucial role in validating the path-planning algorithms before deployment in the real OBRA autonomous vehicle. The core of the simulation environment is built using Unity, which provides a controlled and highly customizable setting where different track conditions, obstacles, and environmental variations can be modeled. Unity allows us to evaluate the performance of the algorithms under high-speed driving scenarios, tight turns, and unexpected obstacles, ensuring that the vehicle can navigate efficiently and safely.

To complement the simulation process, we use Foxglove for real-time visualization and analysis of key system data. Since our project is built on ROS2 (Humble), Foxglove acts as an interface to monitor ROS2 topics, displaying critical information such as vehicle state, planned trajectories, sensor readings, and dynamic obstacle detection. This visualization is essential for debugging, performance assessment, and fine-tuning the parameters of our path-planning algorithms.

The validation process is structured around a set of performance metrics, ensuring that each algorithm is assessed based on objective criteria. These metrics include:

- Path efficiency: Measuring the distance and curvature of the generated trajectory.
- Computational speed: Evaluating how quickly the algorithm produces a valid path.
- Adaptability to dynamic environments: Testing how well the planner reacts to changes such as moving obstacles or shifting track conditions.

Once the simulations are completed, the results, including logs, performance data, and video recordings, are stored in Google Drive for further analysis and comparison. This repository allows the team to track improvements over time and make data-driven decisions when optimizing the algorithms.

By integrating Unity for simulation, Foxglove for real-time monitoring, and Google Drive for data storage, we establish a structured and efficient testing framework. This ensures that our path-planning algorithms are rigorously validated before integration into the OBRA vehicle, minimizing risks and maximizing system reliability.

## 3.4 Version Management

### 3.4.1 Source Code and Data Management, aquí puedo meter la estructura del GitLab

Version control plays a crucial role in the development of this project, ensuring efficient collaboration, structured code management, and reliable tracking of modifications. For this purpose, we utilize Git alongside GitLab, where the OBRA team's centralized workspace is hosted. GitLab serves as the primary repository, storing all project files, code implementations, and historical data from previous years.

At the beginning of the project, the entire existing workspace was cloned from GitLab, providing access to all prior developments, including path-planning algorithms and other essential components used by the team in past seasons. This initial setup allowed us to build upon a solid foundation while ensuring compatibility with the existing autonomous system. Throughout the development process, every modification and improvement to the path planners has been systematically managed using branches within our GitLab repository. Each time a new version of a path-planning algorithm was implemented—whether modifications to the Delaunay planner or the development of the RRT/RRT* algorithm—a dedicated branch was created. This approach enabled parallel development and testing of different versions while preserving the stability of the main codebase.

To maintain an organized and up-to-date repository, commits have been made regularly, ensuring that every iteration and refinement is properly documented. This frequent commit practice has allowed us to track changes efficiently, revert to previous versions when necessary, and collaborate seamlessly within the team. By leveraging GitLab's version control features, we have established a structured workflow that facilitates code reviews, debugging, and continuous integration, ultimately enhancing the reliability of our path-planning system.

In addition to code management, we use Notion to coordinate team tasks and maintain an Agile workflow. Notion enables us to assign responsibilities, track progress, and ensure development efforts remain aligned with project goals. Furthermore, all generated data, including test results, simulation outputs, and performance evaluations, are stored in

Google Drive. This provides a centralized location for data accessibility, allowing for efficient documentation and analysis of algorithm performance over time.

### 3.4.2 Source Code Repository Link

URL or reference to the project repository.

# Chapter 4

# Results

## 4.1 Results and Testing

### 4.1.1 Simulation Setup

Explanation of the Unity + ROS2 environment configuration.

### 4.1.2 Test Scenarios

List of simulated scenarios used for validation.

## 4.2 Performance Metrics

### 4.2.1 Computation Time Evaluation

Data on execution times.

### 4.2.2 Path Quality Analysis

Metrics evaluating the efficiency and smoothness of generated paths.

### 4.2.3 Adaptability to Dynamic Obstacles

Explanation of testing under dynamic conditions.

## 4.3 Experimental Results

### 4.3.1 Comparison of Metrics Between Optimized Delaunay and RRT*

Presentation of comparative data.

### 4.3.2 Visualization of Results Through Graphs

Graphs and their interpretation.

# Chapter 5

# Professionalism

## 5.1 Project Management

### 5.1.1 Development Activities and Schedule

Project logs, reports, and Gantt charts.

### 5.1.2 Data Management

Storage and organization of research documents.

### 5.1.3 Project Deliverables

Summary of key milestones.

## 5.2 Risk Analysis

### 5.2.1 Identified Risks and Mitigation Strategies

Discussion of risks encountered and strategies used.

### 5.2.2 Updated Project Plan Based on Risk Evaluation

Adjustments made due to identified risks.

## 5.3 Legal, Ethical, and Environmental Considerations

### 5.3.1 Compliance with Professional Codes of Conduct

References to BCS, ACM, and industry standards.

### 5.3.2 Ethical and Environmental Impact of the Project

Analysis of social and environmental implications.

# Chapter 6

# Conclusion

## 6.1 Summary of Findings

### 6.1.1 Key Insights from the Algorithm Comparison

Main takeaways from testing and evaluation.

## 6.2 Future Work

### 6.2.1 Improvements in RRT* Implementation

Potential refinements to enhance algorithm performance.

### 6.2.2 Real-World Applications

Application of findings to actual autonomous vehicle scenarios.

# Chapter 7

# Bibliography

# Bibliography

[1] LaValle, S. (2006) 'Rapidly exploring Random Trees: Overview', Available at: https://lavalle.pl/rrt (Accessed: 10 October 2024).

[2] Bécsi, T. (2024) 'RRT-guided experience generation for reinforcement learning in autonomous lane keeping', Scientific Reports, 14, Article number: 24059. Available at: https://www.nature.com/articles/s41598-024-73881-z (Accessed: 16 October 2024).

[3] Muhsen, D.K., Raheem, F.A., and Sadiq, A.T. (2024) 'A Systematic Review of Rapidly Exploring Random Tree RRT Algorithm for Single and Multiple Robots', Cybernetics and Information Technologies, 24(3), pp. 78-101. Available at: https://doi.org/10.2478/cait-2024-0026 (Accessed: 19 September 2024).

[4] Fan, H., Huang, J., Huang, X., Zhu, H., and Su, H. (2024) 'BI-RRT*: An improved path planning algorithm for secure and trustworthy mobile robots systems', Heliyon, 24(e26403). Available at: https://doi.org/10.1016/j.heliyon.2024.e26403 (Accessed: 10 October 2024).

# Chapter 8

# Appendices

## 8.1 Supplementary Data

### 8.1.1 Source Code Repository (GitHub/GitLab)

Reference link to the source code.

### 8.1.2 ROS2 + Unity Configuration Details

Technical setup instructions.

### 8.1.3 Raw Simulation Results

Unprocessed data from testing and evaluation.