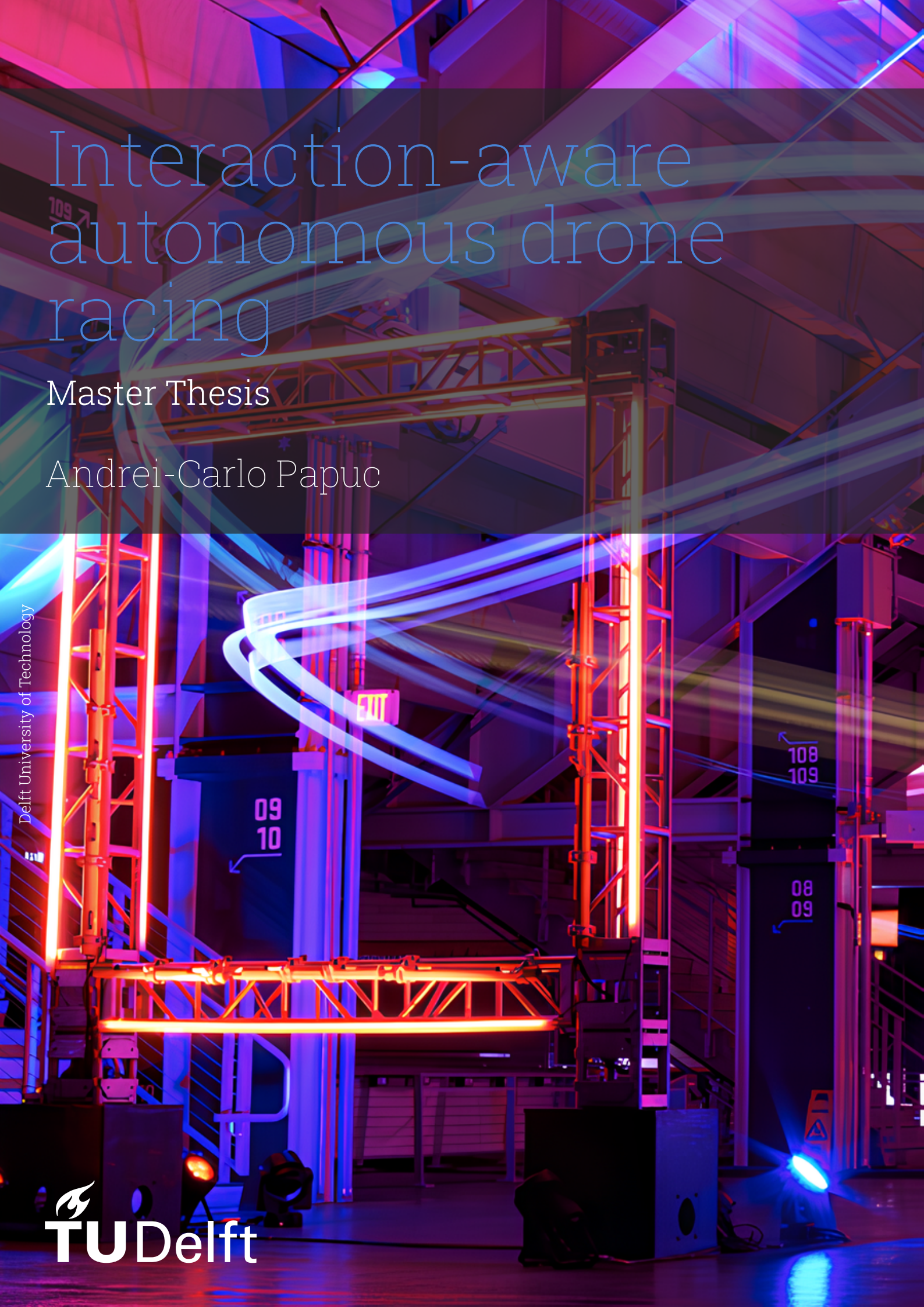
Interaction-aware autonomous drone racing

Master Thesis

Andrei-Carlo Papuc



Delft University of Technology

TU Delft

Interaction-aware autonomous drone racing

Master Thesis

by

Andrei-Carlo Papuc

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday 31 March, 2025 at 10:00 AM.

Student number:	4772385	
Project duration:	April 1, 2024 – March 31, 2025	
Thesis committee:	Prof. J. Alonso-Mora,	TU Delft, Main supervisor
	Ir. L. Peters,	TU Delft, Daily supervisor
	Prof. L. Ferranti,	TU Delft

This thesis is confidential and cannot be made public until June 31, 2025.

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Acknowledgements

This thesis marks the culmination of my studies at Delft University of Technology, a journey made possible by many wonderful individuals who supported me academically, professionally, and personally.

On a professional note, first and foremost, I want to express my deepest gratitude to my daily supervisor, Lasse Peters, for his exceptional guidance, patience, and dedication. I greatly appreciate all the time he spent explaining concepts, discussing progress, and collaboratively exploring ways to improve my work. Our conversations have always been insightful, enjoyable, and motivating. Beyond academia, I am deeply grateful for the invaluable personal and professional advice he provided, I couldn't have asked for a more involved mentor.

I also sincerely thank my co-supervisor, Dr. Sihao Sun, for his guidance, consistent support, and hands-on assistance throughout my lab experiments. His willingness to help me tackle numerous experimental challenges was essential to the success of this thesis.

I am grateful to my main supervisor, Prof. Javier Alonso-Mora, for his expert guidance and willingness to supervise this project. Though our interactions were less frequent, he always offered insightful questions, constructive feedback, and perspectives I had not considered. As he jokingly remarked, he often "pulled me out of local minima." Additionally, I would like to thank Prof. Laura Ferranti for kindly agreeing to participate in my thesis committee and providing valuable feedback on my work.

On a personal note, I must thank my partner, Alexandra, for patiently enduring my endless thesis-related rants and offering unconditional support along the way. Special thanks also go to my friends Alex, Alin, Madalina, and Georgia for their encouragement and for keeping me distracted exactly when I needed it. To my good old friends back in Romania, thank you for not forgetting me, even when I was radio silent or embarrassingly slow to reply; your friendship has meant more than you know. I would also like to thank my lab mates at AMR for making the challenging times of my Master's more enjoyable, whether through insightful discussions or a much-needed cup of coffee.

Of course, I owe immense gratitude to my family, especially my parents, for believing in me, encouraging my ambitions, and always welcoming me home whenever I needed a break.

Finally, I dedicate this thesis to my cat, Cow, for all his supportive meows and comforting purrs, and to my pet friends Albert, Apollo, Ava, and Pixel, who always managed to brighten even the toughest days.

Andrei-Carlo Papuc
Delft, March 2025

Abstract

Autonomous drone racing presents a unique challenge that requires both high-speed motion planning and strategic decision-making in a multi-agent setting. Prior work has primarily relied on model predictive control (MPC) methods that treat opponents as dynamic obstacles, limiting their ability to model strategic interactions. In this work, we formulate drone racing as a dynamic game and introduce game-theoretic planning methods that compute open-loop Nash equilibria, incorporate blocking strategies, and accelerate decision-making using learning-based techniques. These methods explicitly model opponent behavior, allowing drones to anticipate and react strategically in high-speed racing scenarios. To assess the effectiveness of our approach, we conduct a large-scale head-to-head tournament against MPC-based planners, demonstrating that interaction-aware planning enables more effective overtaking and defensive strategies, leading to a higher wining rate. However, computational delays in high-speed decision-making can limit performance, highlighting the need for efficient techniques that balance real-time feasibility with strategic adaptability. Our results show that learning-based acceleration significantly improves decision-making speed while preserving competitive advantages. Finally, high-fidelity simulations and real-world drone racing experiments validate the feasibility of these methods, confirming their ability to generate reliable and competitive strategies under practical racing conditions.

Contents

List of Figures

List of Tables

1

Introduction

1.1. Background

Autonomous racing has emerged as a significant challenge in robotics, where the goal is to develop intelligent systems capable of competing at or above human performance levels. This involves designing aerial vehicles that can navigate complex 3D environments at high speeds without human intervention, while making real-time decisions that account for the presence of competitors. Advancements in the fields of perception, planning, control, and decision-making are necessary for this, as demonstrated in previous research efforts focused on time-optimal flight (see Figure 1.1).

Similar to other autonomous navigation tasks, autonomous racing requires navigating environments filled with obstacles, making split-second decisions, and optimizing routes for efficiency. However, racing presents unique challenges such as the need for aggressive maneuvers and real-time responsiveness while operating at the limits of handling. These extreme conditions drive the development of robust algorithms that enhance not only racing performance but also broader applications in robotics and autonomous systems. For instance, decision-making algorithms designed for overtaking or collision avoidance in racing can improve safety and responsiveness in applications like urban driving, drone navigation, and robotic coordination. The ability to plan dynamically under uncertainty and adapt strategies in real time contributes to safer and more responsive autonomous systems across multiple domains.

Beyond autonomous driving, the advancements made in autonomous racing research have significant benefits for a wide range of other applications. With the emphasis on high-speed operation, which directly relates to the concept of range efficiency, systems like autonomous drones and ground vehicles, can cover larger distances within the same energy or battery capacity [2]. This is particularly critical in time-sensitive applications such as search and rescue, where the ability to quickly scan large areas can mean the difference between success and failure.

1.2. Motivation

While much of the recent research in autonomous racing has focused on optimizing single-agent performance [3, 11], such as minimizing lap times, real-world racing scenarios often involve multiple competitors, each with their own strategies and goals. This creates a dynamic, multi-agent environment where decision-making is influenced by the actions of other participants.

The importance of considering opponent interactions in autonomous racing lies in the strategic depth it adds to the problem. In competitive racing, success is not only about speed but also about executing tactical maneuvers such as overtaking, blocking, and faking. These actions depend heavily on the behavior of other agents on the track, which makes interaction-aware decision-making critical. By incorporating opponent dynamics, autonomous systems can achieve a higher level of competitive performance, better mimicking the strategies used by human drivers or pilots in real-world racing scenarios.

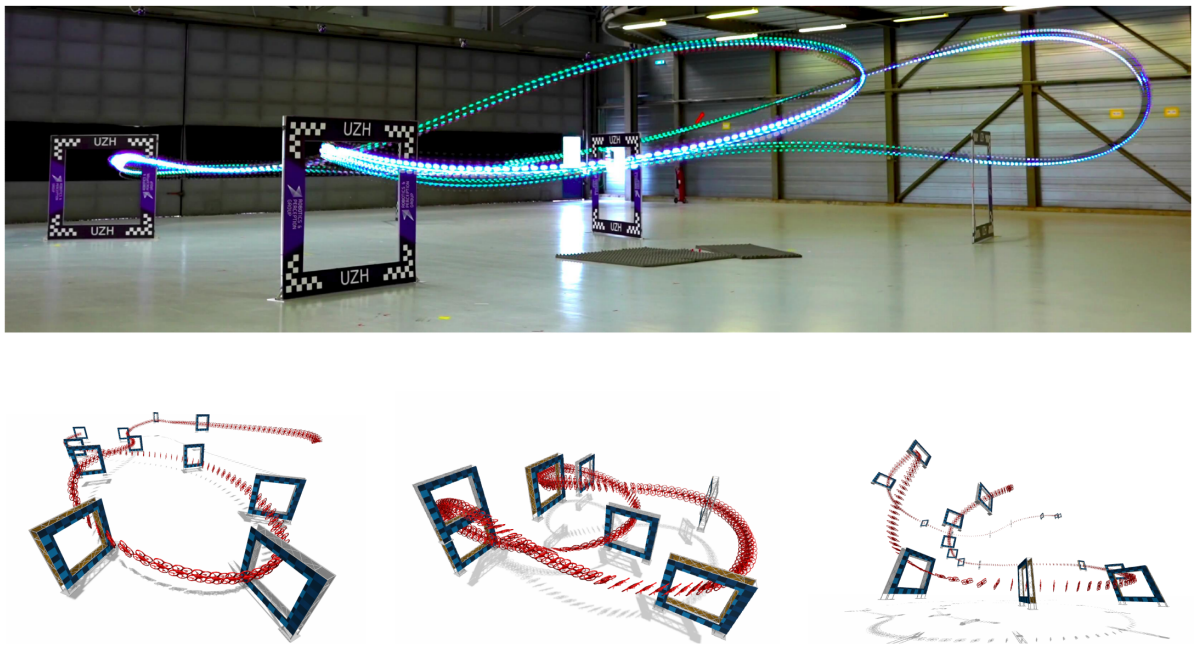


Figure 1.1: Examples of time-optimal drone racing strategies. The top image, from [27], and the bottom image, from [31], depict autonomous drones navigating race tracks using model predictive control and reinforcement learning methods to achieve minimal lap times.

Beyond the racing domain, solving the challenges posed by multi-agent autonomous racing has practical implications for broader autonomous driving applications [6]. In urban driving or highway scenarios, autonomous vehicles must interact with other vehicles, pedestrians, and cyclists, all of whom have independent goals and behaviors.

The nature of these challenges also varies across different racing platforms. Drone racing introduces additional complexity compared to ground-based racing, requiring the navigation of complex 3D environments, precise gate-passing, and real-time adaptation to aerodynamic disturbances. Unlike autonomous cars, which operate on relatively structured roads, drones move in free space with six degrees of freedom, making interaction-aware decision-making even more critical. Moreover, in first-person view (FPV) drone racing, human pilots actively anticipate and react to their competitors' actions, a capability that autonomous drones must develop to compete effectively.

However, designing effective multi-agent racing algorithms is inherently difficult. The decisions of each agent are coupled with those of their competitors, creating a problem where each vehicle's actions influence the strategies and outcomes of others. Finding a solution (or equilibrium) in such dynamic settings, where agents are constantly adjusting their actions in response to others, requires models that balance individual objectives with opponent predictions. Real-time performance is another critical challenge, as decisions need to be made rapidly in high-speed environments, whether on the racetrack or in the air.

1.3. Contributions

This thesis develops interaction-aware decision-making strategies for autonomous drone racing, leveraging game-theoretic methods to enhance competitive performance. The objective is to address the limitations of existing model predictive control (MPC)-based approaches, which often model opponents as dynamic obstacles without accounting for their strategic behavior. By integrating game-theoretic planning, we enable drones to anticipate and react strategically to opponents in high-speed racing scenarios. This allows for more adaptive and competitive decision-making in strongly dynamic environments.

The proposed methods are tested in both simulation and real-world racing environments, evaluating

their impact on key performance metrics such as number of wins, overtaking success, and blocking efficiency. We conduct a comparative analysis between game-theoretic methods and MPC in a tournament-style evaluation, assessing their effectiveness in competitive racing scenarios. The results provide insights into the advantages of interaction-aware decision-making and highlight the conditions under which game-theoretic approaches outperform traditional MPC-based planners.

The contributions of this thesis can be summarized as follows:

- We implement model predictive control (MPC) and model predictive games (MPG), along with tools for automating the transcription of 3D drone racing problems into these frameworks.
- We explore lifted game formulations to accelerate online computation, building on the approach proposed by Peters et al. [26], and introduce a specialized training procedure tailored for racing applications.
- We conduct a comparative analysis between game-theoretic methods and MPC in a tournament-style evaluation, assessing their effectiveness in win rate, overtaking, and blocking behaviors.
- We provide extensive experimental validation through high-fidelity simulations and real-world drone racing experiments, demonstrating the practical feasibility of game-theoretic planning in competitive racing scenarios.

1.4. Outline

This remainder of this thesis is organized as follows. Chapter 2 reviews existing research on multi-agent trajectory planning in autonomous racing. Chapter 3 provides an overview of fundamental game-theoretic concepts relevant to multi-agent decision-making in racing. Chapter 4 formally defines the racing problem, specifying the assumptions, rules, and the design of the referee. Chapter 5 describes the proposed game-theoretic planning methods, including their mathematical formulation, implementation details, and integration into the racing framework. Chapter 6 outlines the simulation environment and experimental setup used to evaluate the proposed methods, detailing the race tracks, agent dynamics, and evaluation metrics. Chapter 7 presents a comparative analysis of different planning approaches, discussing their performance in terms of speed, safety, and strategic interactions. Chapter 8 validates the findings through real-world drone racing experiments. Chapter 9 summarizes the contributions, highlights key insights, and discusses potential directions for future research.

2

Related work

This chapter presents a comprehensive review of state-of-the-art methods that specifically address multi-agent autonomous racing. Interaction-aware methods focus on how autonomous racing agents anticipate and respond to the actions of their opponents in real-time, aiming to maximize competitive performance while adhering to racing constraints. In literature most approaches fall into two main categories: model predictive control (MPC) based planners and game-theoretic (GT) methods, as shown in Figure 2.1.

In model predictive methods, the agent uses a receding horizon approach where it predicts the future evolution of the race based on its own actions and those of its opponents. The control decisions are formulated by solving an optimization problem at each time step, which typically includes various constraints like avoiding collisions, maximizing speed, and adhering to track limits. These methods manage interactions by predicting the opponents trajectories then considering them as static or dynamic obstacles.

On the other hand, game-theoretic methods model the interactions between racing agents explicitly as strategic decisions. These methods leverage the solution concepts of Nash and Stackelberg equilibrium, where agents assume rationality among all players, and adjust their strategies accordingly by taking into account their best responses. The planners reflect competitive settings where mutual interactions, rather than just individual performance, shape decision-making.

2.1. Model predictive approaches

Model predictive control (MPC) is widely used in autonomous racing due to its ability to handle complex constraints and optimize trajectories over a receding horizon. Predict-then-plan strategies are commonly employed in MPC-based approaches, where the ego vehicle first predicts the future behavior of opponents and subsequently computes an optimal trajectory under these assumptions. He et al. [12] proposed a hybrid strategy that switches between a learning-based trajectory planner for minimizing lap time and an optimization-based planner for handling interactions with other vehicles. This method integrates a low-level MPC controller with control barrier function constraints to ensure safety during overtaking maneuvers. Similarly, Kalaria et al. [15] developed a two-stage approach that first approximates a globally optimal trajectory before refining it using nonlinear MPC (NMPC). Their method incorporates inter-vehicle collision avoidance and drafting effects, demonstrating its effectiveness in high-speed racing environments.

Other notable contributions to MPC-based racing include Evans et al.'s [5] hybrid local planner, which integrates a path-following controller with a deep reinforcement learning (DRL) agent to improve obstacle avoidance while maintaining a reference trajectory. Liniger et al. [22] introduced a hierarchical two-level control framework, where a high-level planner generates feasible trajectories that are subsequently tracked by an NMPC controller. Their approach ensures real-time implementation on embedded control platforms. Rowold et al. [28] proposed a spatiotemporal graph search method for trajectory planning, emphasizing safety and competitiveness by incorporating track limits and opponent motion

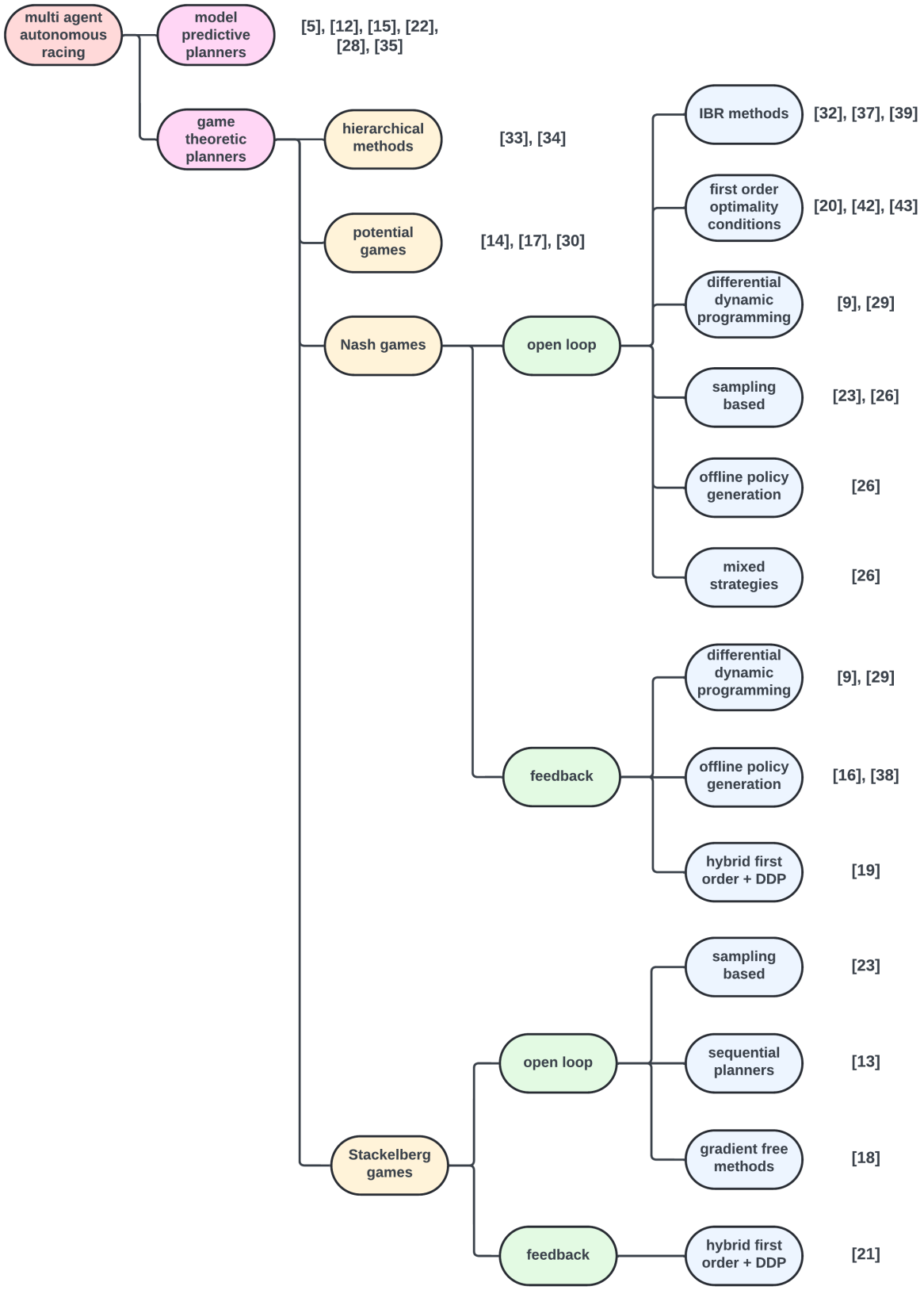


Figure 2.1: Overview of different autonomous racing problems found in literature along with the proposed methods of solving them

predictions into the cost function. Vázquez et al. [35] extended this hierarchical approach by introducing a terminal constraint in the high-level planner's solution, ensuring optimality even with limited prediction horizons.

Despite their effectiveness, MPC methods generally rely on treating opponents as dynamic obstacles rather than strategic agents. This limitation reduces their ability to capture adversarial behaviors, such as blocking and strategic overtaking, which are essential in competitive racing. Furthermore, MPC approaches often struggle with long prediction horizons due to high computational demands. Recent work attempts to address these limitations by integrating reinforcement learning and imitation learning components within MPC frameworks, enabling more adaptive and dynamic decision-making.

2.2. Game-theoretical planners

Game-theoretic approaches explicitly model strategic interactions between agents by leveraging equilibrium concepts such as Nash and Stackelberg equilibria. These methods provide a more principled approach to competitive decision-making by ensuring that each agent accounts for the responses of others when computing its optimal strategy.

2.2.1. Nash games

Nash equilibrium-based planners compute strategies where no agent can unilaterally improve its outcome. Iterative best response (IBR) is a popular method for approximating Nash equilibria, where agents iteratively adjust their strategies based on the fixed responses of opponents. Spica et al. [32] employed IBR with sensitivity analysis to predict and exploit opponent reactions, demonstrating superior performance compared to MPC-based approaches. Wang et al. [37] extended this method to 3D racing using a sensitivity-enhanced IBR (SE-IBR), which guarantees convergence under specific conditions. Williams et al. [39] applied best-response model predictive path integral control (BR-MPPI) for real-world experiments in autonomous racing.

Other approaches use first-order optimality conditions derived from the Karush-Kuhn-Tucker (KKT) conditions to compute Nash equilibria. Le Cleac'h et al. [20] introduced the ALGAMES solver, which applies an augmented Lagrangian method to enforce state and input constraints. Zhu and Borrelli [42, 43] developed a sequential quadratic programming (SQP) approach for constrained dynamic games, demonstrating improved success rates in racing scenarios. Additionally, hierarchical approaches that blend Nash equilibria with reinforcement learning techniques [33, 34] are being explored to provide computationally efficient yet adaptive planning strategies.

Differential dynamic programming (DDP) techniques, such as iterative linear-quadratic games (iLQGames)[9], refine Nash equilibrium strategies by iteratively solving local approximations of the game. Rowold et al. [29] extended iLQGames to racing scenarios, showing that feedback solutions lead to more aggressive and strategic behaviors, such as blocking. However, iLQGames struggles with constraint handling, requiring soft penalty terms to enforce track boundaries and acceleration limits.

2.2.2. Stackelberg games

Stackelberg games introduce a leader-follower hierarchy where one agent commits to a strategy first, and others respond optimally. Liniger and Lygeros [23] demonstrated that Stackelberg equilibria are particularly useful for enforcing blocking maneuvers in racing. Hu et al. [13] developed a method for optimizing the order of play in Stackelberg games, using a mixed-integer optimization framework to determine socially optimal leader-follower assignments. Koirala and Laine [18] proposed a Monte Carlo-based gradient-free algorithm for solving multilevel Stackelberg games, though its computational complexity limits its real-time applicability.

A recent paper by Li et al. [21] builds on the principles applied in the computation of GFNE [19], and aims to solve for the feedback Stackelberg equilibrium (FSE). The approach reformulates the feedback Stackelberg equilibrium problem as a sequence of nested optimization problems, which enables the derivation of the KKT conditions and second-order sufficient conditions for local FSE. Unlike iLQGames [9], which quadraticize the cost function, Li et al. quadraticize the Lagrangian, using a primal-dual interior point (PDIP) method. This provides a more computationally efficient solution compared to the active set method used in [19], offering polynomial complexity rather than exponential complexity in

solving constrained LQ and nonlinear games.

2.2.3. Potential games

Potential games simplify multi-agent optimization by ensuring that all players' objectives align with a global potential function. Kavuncu et al. [17] introduced Potential iLQR, leveraging the efficiency of iLQR for solving constrained multi-agent trajectory optimization problems. Jia et al. [14] extended potential games to dynamic racing environments with RAPID, demonstrating strategic behaviors such as overtaking and blocking in drone racing experiments. However, the assumption of symmetric cost structures limits their applicability in asymmetric racing interactions. Despite these challenges, the authors suggest that the algorithm can be used as a fast warm-starting method in scenarios with asymmetric couplings, offering computational advantages for solving more complex problems.

Additionally, another theoretical work [30] extends potential games to a feedback information structure by formulating the necessary game conditions, further broadening their applicability in dynamic multi-agent scenarios.

2.3. Race design

In multi-agent autonomous racing research, experimental setups are designed to showcase strategic interactions between agents, such as overtaking, blocking, and collision avoidance. These setups differ in the vehicle type used (e.g., ground vehicles or aerial drones), track dimensionality (2D vs. 3D), starting conditions, speed variations, and track complexity to generate competitive racing scenarios.

For example, in ground vehicle racing, Thakkar et al. [34], ensure fairness by giving all vehicles identical dynamic properties, including top speed, acceleration, and grip, while varying their initial lane positions to create different starting conditions. To avoid bias, they alternate starting lanes between races. The experiments in [16] and [34] involve head-to-head quadrotor races on both simple oval tracks and more complex 3D tracks with challenging geometries, such as turns with varying radii and tight U-turns, which demand careful long-term planning from the agents.

In [32], Spica et al. enforce interactions by handicapping one of the robots. The faster robot starts behind the slower one, ensuring that it must engage with the slower opponent to overtake during the race. Similarly, in [14] and [23], the initial positions of the agents are randomized to ensure they start within 20-50 centimeters of each other, with the trailing agent given a slightly higher maximum speed. This setup guarantees frequent interactions between agents, promoting overtaking and strategic maneuvers during the race.

Table 2.1 presents an overview of different methods used in designing the racing problem to generate interesting behaviour. These methods involve varying collision responsibility [4, 23, 29, 33], setting different acceleration and velocity limits based on relative positioning of the players [4, 29, 32], having special cost terms for aggressiveness by penalizing large distances from opponents [14, 37, 43], and varying the information structures by playing as a follower or leader [4, 23].

Table 2.1: Overview of methods used to incentivize interesting racing maneuvers

Racing problem setup	Reference
aggressiveness term	[14, 37, 43]
acceleration and velocity constraints	[4, 29, 32]
information structure	[4, 23]
varying collision responsibility	[4, 23, 29, 33]

2.4. Discussion

The reviewed literature provides a comprehensive understanding of state-of-the-art methodologies in multi-agent autonomous racing, highlighting different approaches to interaction-aware planning. While both model predictive and game-theoretic methods have demonstrated their efficacy in competitive racing scenarios, this section discusses how our research aligns with these approaches, the choices made regarding the methodologies employed in this thesis, and the rationale behind these decisions.

Our research adopts a game-theoretic approach, leveraging Nash equilibrium-based planning to model the strategic interactions between agents. As discussed in the reviewed literature, Nash games capture mutual influences between agents in a competitive setting, making them well-suited for the autonomous racing domain. However, we do not consider Stackelberg or potential game formulations in our study. Solving for generalized Stackelberg equilibria in dynamic games is computationally expensive, making it impractical for real-time racing applications. Similarly, potential games offer computational advantages due to their alignment with global optimization techniques, but their assumption of symmetric inter-agent cost structures limits their applicability in competitive racing scenarios where asymmetric interactions are common.

Another key methodological choice in our study is the use of open-loop strategies rather than feedback strategies. While feedback-based approaches enable agents to continuously adapt their strategies based on real-time observations, they require solving complex trajectory optimization problems at each time step, making them computationally infeasible for real-time execution in high-speed racing scenarios. Although multi-agent reinforcement learning (MARL) provides a potential solution to this computational challenge by learning feedback strategies offline, it comes with its own drawbacks – namely, the significant computational resources required for training these models. Given these constraints, we exclude MARL-based methods from this study.

Furthermore, our approach incorporates a hierarchical planning framework, where a game-theoretic planner operates as a high-level controller to generate interaction-aware waypoints that guide the low-level controller in real-time execution. This design choice aligns with recent hierarchical methods that decouple long-term strategic planning from short-term control execution, allowing for a balance between computational efficiency and strategic adaptability. Additionally, our race design draws inspiration from previous studies to construct competitive and strategically engaging race scenarios. Techniques such as varying initial conditions, introducing velocity constraints, and incorporating aggressiveness terms in cost functions are key considerations in our experimental setup. These elements are essential for ensuring meaningful interactions between agents, promoting behaviors such as overtaking, blocking, and competitive positioning.

3

Game theory preliminaries

This chapter introduces the fundamental concepts of game theory relevant to multi-agent motion planning, especially in the context of autonomous racing. Game theory offers a mathematical framework to model interactions where the outcome for each participant depends on the actions of others. Taking inspiration from [8] [41], we first aim to make some distinctions between the different types of games.

Cooperative vs non-cooperative In cooperative games, agents cooperate to improve collective outcomes, while in non-cooperative games, each agent optimizes their own payoff independently. Racing is inherently a non-cooperative game since each agent (vehicle) aims to maximize its competitive success and no agent shares it's strategies with the opponents.

Static vs dynamic In static games, all agents choose their actions simultaneously for only a single instant in time, whereas for dynamic games, the game is played for multiple instants over a period of time, one other name for these is "multi-stage" games.

Finite vs infinte Games in which the number of actions available to each player is finite are called finite games, while games with an uncountable number of actions are called infinite.

Zero-sum vs general-sum In zero-sum games, one agent's gain is balanced by the loss of others. Racing is often modeled as a zero-sum game, where positions are directly competitive (i.e., overtaking improves one agent's rank while decreasing another's). However, this could also be modeled as a general-sum game, where agents have independent objectives but also shared goals, like avoiding collisions.

Constrained vs unconstrained Constrained games include constraints on player's states and actions. These constraints can only appear for a subset of the players, there's no requirement that all players should have the same constraints. Unconstrained games do not pose constraints for any players.

Pure vs mixed strategies In pure strategies, an agent always chooses a specific action in a deterministic manner, while in mixed strategies, they randomize over multiple actions. Mixed strategies are relevant when dealing with uncertainties in the opponents' behaviors. For instance, a racer might choose to probabilistically switch between aggressive and conservative driving modes to maintain unpredictability. Mixed strategies are also relevant for games which don't possess equilibria in pure strategies, for example the rock-paper-scissors game.

Trajectory games Following the definition from [8]: "A game in which N agents interact in a physical space over time. The agents, which could represent players in a race, for example, may each select of sequence and states and control inputs (i.e. a trajectory). That is, $\mathrm{P}i$'s decision variable x_i consists of

sequences of state and control variables for that player, and must satisfy private constraints which enforce, e.g., dynamic feasibility (physics), staying on road, maintaining speed limits, collision avoidance, etc."

3.1. Solution concepts

There are a wide variety of solution (or equilibrium) concepts. In this section, we introduce the most relevant one for the scope of this paper, namely, Nash equilibria. To illustrate the construction and assumptions of this concept, we turn our attention to non-cooperative, unconstrained, general-sum, static finite games.

A normal form representation is the standard way of modeling static finite games. Each player selects an action without knowing the choices of others, and the resulting payoffs are recorded in a payoff matrix. Each player has its own objective function J_i, which it aims to optimize. Formally, a static finite game in normal form consists of:

- A finite set of players $\mathcal{N} = \{1, \dots, N\}$,
- A finite action set $\mathcal{U}^i$ for each player i,
- An objective function $J^i : \mathcal{U}^1 \times \cdots \times \mathcal{U}^N \to \mathbb{R}$ that determines player i's objective.

Each player aims to select an action $\boldsymbol{u}^i \in \mathcal{U}^i$ that optimizes its individual objective J^i. In non-cooperative games, each player's objective depends not only on its own action but also on the actions of others, i.e.,

$$J^i(\boldsymbol{u}^1, \boldsymbol{u}^2, \dots, \boldsymbol{u}^N). \tag{3.1}$$

To illustrate this, we consider a bimatrix game, where two players (Player 1 and Player 2) simultaneously choose actions, and their payoffs are determined by payoff matrices. A well-known example is the prisoner's dilemma, taken from [8], which is defined by the following payoff matrices:

$$M^1 = \begin{bmatrix} 2 & 0 \\ 3 & 1 \end{bmatrix} \text{ and } M^2 = \begin{bmatrix} 2 & 3 \\ 0 & 1 \end{bmatrix} \tag{3.2}$$

where M^1 is the payoff matrix for Player 1, and M^2 is the payoff matrix for Player 2. The players' action sets consist of two possible actions: "confess" or "stay quiet". We encode these actions as vectors $\boldsymbol{u}^i$, where the first entry of $\boldsymbol{u}^i$ corresponds to the action "confess" and the second entry of $\boldsymbol{u}^i$ corresponds to the action "stay quiet".

Given that each player aims to maximize their own payoff, the objective functions are:

$$J^1(\boldsymbol{u}^1, \boldsymbol{u}^2) = {\boldsymbol{u}^1}^T M^1 \boldsymbol{u}^2, \quad J^2(\boldsymbol{u}^1, \boldsymbol{u}^2) = {\boldsymbol{u}^1}^T M^2 \boldsymbol{u}^2. \tag{3.3}$$

Nash equilibrium A Nash equilibrium occurs in a game when no agent can unilaterally improve their payoff by changing their strategy, assuming other agents keep their strategies fixed. In other words, each player's strategy is a best response to the strategies of the other players. This concept assumes that all agents make their decisions simultaneously, or without knowing the strategies of others beforehand. We formulate the given game as an optimization problem, where the strategy of player i corresponds to their choice of action $\boldsymbol{u}^i$.

$$\forall i \in \mathcal{N} \begin{cases} \boldsymbol{u}^{i*} \in \underset{\boldsymbol{u}^i}{\operatorname{argmin}} \; J^i(\boldsymbol{u}^i, \boldsymbol{u}^{\neg i*}) \\ \text{s.t.} \quad c^i(\boldsymbol{u}^i) = 0 \\ \qquad h^i(\boldsymbol{u}^i) \geq 0 \end{cases} \tag{3.4}$$

Where:

- $\boldsymbol{u}^i$ is any strategy from the set of strategies $\mathcal{U}^i$ for agent i

- $\boldsymbol{u}^{i*}$ is the optimal strategy of agent i
- $\boldsymbol{u}^{\neg i*} = (\boldsymbol{u}^{j*})_{j \neq i}$ is the strategy profile of all agents except i
- c^i and h^i represent the equality and inequality constraints for player i
- J^i represents the objective function of agent i

The N-tuple of strategies $\{\boldsymbol{u}^{i*}; i \in \mathcal{N}\}$ constitutes a Nash equilibrium if

$$J^i(\boldsymbol{u}^{i*}, \boldsymbol{u}^{\neg i*}) \leq J^i(\boldsymbol{u}^i, \boldsymbol{u}^{\neg i*}), \quad \forall \boldsymbol{u}^i \in \mathcal{U}^i \tag{3.5}$$

This inequality implies that for each agent i, given the strategies of other agents the strategy $\boldsymbol{u}^{i*}$ is the best choice. In the previous bimatrix game, the Nash equilibrium is given by the point $\boldsymbol{u}^{1*} = \boldsymbol{u}^{2*} = (1, 0)^T$, which corresponds to both prisoners confessing.

3.1.1. Generalization

In many multi-agent systems, agents often face constraints that are not only private but also shared among others, leading to the concept of a Generalized Nash Equilibrium (GNE). This differs from the standard Nash equilibrium as it includes the consideration of these coupled constraints. Mathematically, a GNE is a solution to the following problem which also satisfies Equation 3.5

$$\forall i \in \mathcal{N} \begin{cases} \boldsymbol{u}^{i*} \in \underset{\boldsymbol{u}^i}{\operatorname{argmin}} \; J^i(\boldsymbol{u}^i, \boldsymbol{u}^{\neg i*}) \\ \text{s.t.} \quad c^i(\boldsymbol{u}^i, \boldsymbol{u}^{\neg i}) = 0 \\ \qquad h^i(\boldsymbol{u}^i, \boldsymbol{u}^{\neg i}) \geq 0 \end{cases} \tag{3.6}$$

Where c^i and h^i represent coupled equality and inequality constraints, respectively. The players' strategies must jointly minimize their objectives while satisfying all relevant constraints.

In both standard and generalized Nash equilibria, multiple equilibria can exist. Non-uniqueness in equilibria arises in cases where multiple sets of strategies satisfy the equilibrium conditions. In the case of racing, non-uniqueness can manifest as different, equally optimal trajectories or strategies that satisfy the equilibrium conditions.

3.2. Information structure

The information structure of a game plays a crucial role in determining the strategies of agents. A key distinction is made between open-loop and feedback games:

Open-loop (static) information structure An open-loop information structure refers to a scenario where agents commit to their strategies at the beginning of the game, without the ability to adjust them based on the evolving state of the game. Once the game starts, players have no access to updated information about their opponent's actions or the game's progression, leading them to make decisions purely based on initial conditions. This results in the class of open-loop strategies.

Feedback information structure In contrast, a feedback information structure allows agents to adjust their strategies dynamically based on the current state and time, leading to the class of feedback strategies. These strategies ensure strong time consistency, meaning that decisions remain optimal when re-evaluated at any future time. As discussed in [1], feedback strategies are often formulated as Markovian strategies γ, where the control action $\boldsymbol{u}^i$ depends on the current state of the game $\boldsymbol{x}_t$:

$$\boldsymbol{u}^i = \gamma^i(\boldsymbol{x}_t) \tag{3.7}$$

This adaptability often leads to superior performance compared to open-loop strategies, except in cases where precommitment provides a strategic advantage.

4

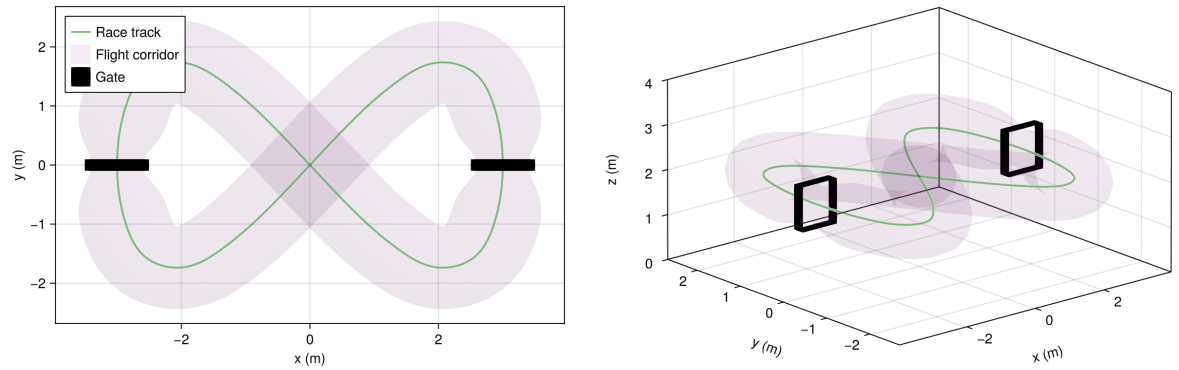
Racing Game

This chapter defines the structure of the autonomous racing game, focusing on planning and control in a non-cooperative setting where players compete for position without sharing strategies. The racing rules, inspired by real-world competitions, enforce fair overtaking, track limits, collision responsibility, and velocity constraints to prevent unintended exploits. Additionally, a referee system is introduced, ensuring compliance with race rules by monitoring player states, enforcing constraints, and detecting collisions.

4.1. Formal description

The autonomous racing game is structured as a multi-agent dynamic system, where each player aims to navigate the race track while optimizing its own control strategy under a set of predefined rules. The game environment consists of a closed-loop race track with a known layout, parameterized using a smooth periodic spline representation.

The race track includes designated checkpoints in the form of gates as shown in Figure 4.1. These gates serve as verification points that enforce adherence to the intended trajectory, preventing shortcuts or excessive deviations from the track. The track is further confined within a predefined flight corridor, which players must remain within to avoid penalties. The corridor width is set adaptively to accommodate different sections of the track, such as narrow gate regions.



(a) Top view of the lemniscate race track

(b) Perspective view of the lemniscate race track

Figure 4.1: Example of a 3D lemniscate race track. The track consists of two loops, traversed in a ”clockwise” direction on the right side and a ”counterclockwise” direction on the left. Players start at the center and must pass through two elevated gates while staying within the designated flight corridor

4.1.1. Assumptions

In the formulation of the racing problem and throughout this paper, the focus is primarily on control and planning, rather than perception. To simplify the problem the following assumptions are formulated:

- **Full knowledge of opponents' states:** each player is assumed to have complete information about the states of all other players at any given time, including their positions, velocities, and accelerations. If this were not the case, uncertainty and observational errors would need to be accounted for, significantly complicating the problem.
- **Full global knowledge of the race track:** each player knows the layout of the race track, including its current position and progress along the track. This eliminates the need for track discovery or learning during the race and allows players to focus on decision-making based on known track conditions.
- **Non-cooperative game setting:** the game is non-cooperative, meaning players do not share their strategies or intentions with each other. Each player is focused on its own objectives, and any interaction between players arises naturally from the dynamics of the game, such as competing for track position or avoiding collisions.

4.2. Racing rules

The racing rules are designed to ensure fair competition while addressing edge cases that could lead to unintended strategic exploits. Each rule is motivated by potential ambiguities and practical considerations observed in competitive autonomous racing[1].

To structure competitive interactions, players are assigned roles at the start of the race:

- The attacker starts behind and aims to overtake the defender, who starts in front.
- Roles switch once an overtake is considered valid, which occurs when the attacker moves at least 0.75 m ahead of the defender along the track.
- The winner is determined by the total time spent as the defender, promoting continuous engagement in overtaking rather than last-minute maneuvers.

This role-based structure and metric discourages passive racing strategies and ensures that players remain engaged in the competition throughout the race.

Players must follow track limits and pass through all gates to ensure adherence to the racecourse:

- Missing a gate results in disqualification.
- Players must remain within the track boundaries; exceeding this limit also leads to disqualification.

These constraints prevent players from gaining an unfair advantage by taking unintended shortcuts.

Collisions and velocities are regulated to enforce safe racing and interesting interactions:

- The attacker will have a higher velocity limit than the defender.
- The attacker is responsible for avoiding collisions.
- A collision results in the attacker's disqualification, and a win attributed to the defender.

This rule follows the precedent set by real-world autonomous racing competitions, where right-of-way is typically granted to the defender in ambiguous cases.

To prevent players from exploiting stopping or excessive speed differences to manipulate race outcomes:

- Exceeding the maximum velocity by a large margin results in disqualification.
- Exceeding the maximum velocity by a small margin for a long period results in disqualification.
- Players must maintain a minimum speed. Dropping below this for a long period results in disqualification.

[1] More specifically the Indy Autonomous Challenge Rulebook https://www.indyautonomouschallenge.com/rules

Table 4.1 provides a structured summary of the racing rules.

Table 4.1: Summary of racing rules, including role assignments, overtaking conditions, track limits, collision responsibilities, and velocity constraints.

Race Rules		
ID	**Rule**	**Description**
R1	Role assignment	At the start of a head-to-head race, players are assigned roles: the attacker (behind) and the defender (in front). Roles switch once a valid overtake has occurred.
R2	Overtaking	An overtake is considered valid when the attacker moves at least 0.75 m ahead of the defender along the track.
R3	Gate passage	Players must pass through all gates. Missing a gate results in disqualification (**by deviation violation**).
R4	Track limits	Players must not deviate more than 2.0 m off-track. Exceeding this limit results in a disqualification (**by deviation violation**).
R5	Collision responsibility	The attacker will always be responsible for collision and shall maintain a distance larger than 0.35 m. A crash results in the disqualification (**by collision violation**) of the attacker.
R6	Velocity limits	Players shall adhere to their respective speed limits depending on their roles.
R7	Hard velocity limit	Exceeding $v_{\text{max}} + 4.0$ ms^{-1} results in disqualification (**by velocity violation**).
R8	Max soft velocity duration	Exceeding $v_{\text{max}} + 0.25$ ms^{-1} for more than 5.0 s results in disqualification (**by velocity violation**).
R9	Min soft velocity duration	Players must maintain at least 0.5 ms^{-1}; flying below this speed for more than 5.0 s results in disqualification (**by velocity violation**).
R10	Race duration	The race is limited to a maximum of 5 laps.
R11	Winner determination	The winner is decided based on the total time spent as the defender (i.e., leading the race).

4.3. Referee design

The referee system maintains fair competition by enforcing race rules, monitoring player behavior, and determining race events such as the start, finish, and disqualifications. It operates based solely on observable agent states – positions, velocities, and accelerations – without access to internal control inputs. This section outlines the referee's role in computing track position, monitoring deviations, detecting collisions, enforcing velocity constraints, and interacting with other system components.

To evaluate whether agents comply with race rules, the referee must determine each agent's position along the track. This requires solving an inverse mapping problem: given an agent's arbitrary position $\boldsymbol{p}$ in 3D space and a parametrized racetrack $\boldsymbol{p}^d$, the corresponding progress variable θ along the race track must be determined. In other words, the aim to find the closest point on the track to the agent's current position.

Since the track is parameterized as a periodic spline, we formulate this as an optimization problem below, where the objective is to find θ that minimizes the squared Euclidean distance between the agent's position and the closest point on the track spline.

$$\begin{aligned} \pi(\boldsymbol{p}) = \ & \underset{\boldsymbol{\theta}}{\operatorname{argmin}} ||\boldsymbol{p} - \boldsymbol{p}^d(\theta)||^2 \\ & \text{s.t. } \theta_{\text{init}} - 1.5 \leq \theta \leq \theta_{\text{init}} + 1.5 \end{aligned} \tag{4.1}$$

This root-finding problem is efficiently solved using Brent's method, a hybrid approach combining the reliability of the bisection method with the speed of inverse quadratic interpolation methods. Brent's

method guarantees convergence while maintaining fast numerical performance. We use the implementation from Optim.jl [25], with a search window of radius 1.5 around the previous estimate θ_{init}. This constraint accelerates convergence and reduces erroneous solutions, particularly in small or high curvature tracks. For instance, given a perfectly circular track: if the agent is positioned at the center of the circle, multiple values of θ correspond to track points that are equidistant from the agent. Without a constrained search window, the optimization might return an incorrect θ. By leveraging prior knowledge of the agent's position at the start of the race, we effectively limit the search space and ensure an accurate estimate of progress along the track. A visual representation of this optimization problem, illustrating the search for the closest point on the track, is provided in Figure 4.2.

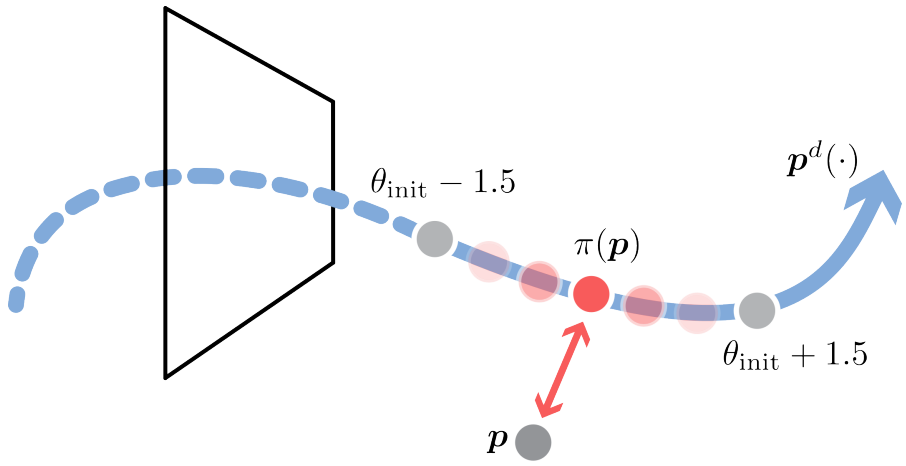


Figure 4.2: Illustration of the inverse mapping problem for determining a player's progress θ. The agent's position $\boldsymbol{p}$ is projected onto the closest point on the track spline $\boldsymbol{p}^d(\cdot)$. The corresponding progress variable is given by $\pi(\boldsymbol{p}) = \theta$

Once the agents' positions along the track are established, the referee determines their respective roles before assessing any rule violations. The roles – attacker and defender – are assigned based on the agents' relative progress along the track. Specifically, the attacker is the agent whose position is behind or within a small threshold ϵ_{ov} of the opponent's position, while the other agent assumes the defender role. This assignment follows the rule

$$\text{attacker} = \begin{cases} i & \text{if} \quad \theta^i \leq \theta^{\neg i} + \epsilon_{\text{ov}} \\ \neg i & \text{otherwise} \end{cases} \qquad \text{defender} = \begin{cases} \neg i & \text{if} \quad \theta^i \leq \theta^{\neg i} + \epsilon_{\text{ov}} \\ i & \text{otherwise} \end{cases} \tag{4.2}$$

where θ^i and $\theta^{\neg i}$ denote the progress variables of the ego player and opponent, respectively. The threshold ϵ_{ov} ensures that minor position fluctuations do not cause frequent role switching.

The referee monitors deviations from the track center line to ensure that agents remain within track limits. This is implemented via a deviation function below that defines an allowable deviation radius r_{dev} as a function of the agent's progress along the track. The deviation tolerance varies depending on proximity to the 3D Gaussians placed at gate centers $\boldsymbol{p}_{g,j}$.

$$\text{dev}(\boldsymbol{p}^d(\theta)) = r_{\text{dev}}^{\max} + (r_{\text{dev}}^{\min} - r_{\text{dev}}^{\max}) \sum_{j=0}^{M} e^{-\frac{1}{2}(\boldsymbol{p}^d(\theta) - \boldsymbol{p}_{g,j})^T \Sigma^{-1} (\boldsymbol{p}^d(\theta) - \boldsymbol{p}_{g,j})} \tag{4.3}$$

Here, Σ is the diagonal covariance matrix that indicates the width of the Gaussians in x, y, and z axes, $r_{\text{dev}}^{\max}$ represents the maximum allowable deviation in open track sections, while $r_{\text{dev}}^{\min}$ is the stricter deviation constraint enforced near gates. The allowable flight corridor, which visually represents this function, is shown in Figure 4.1.

Moreover, the referee continuously measures the euclidean distance between players, and absolute velocities of both agents to check whether they are causing any violations according to the rules presented in Table 4.1. The whole workflow of the referee is presented in Figure 4.3.

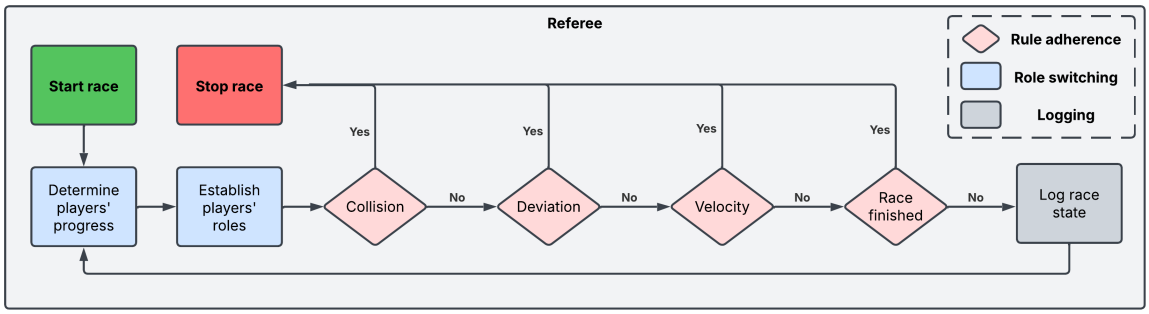


Figure 4.3: Functional diagram showing the the complete referee workflow

At the start of each race, the referee initializes the competition and continuously estimates each agent's progress along the track by solving the inverse mapping problem formulated in Equation 4.1. Once progress estimates are established, the referee assigns roles based on relative positions. With roles determined, the referee enforces race rules by monitoring compliance with track boundaries, velocity constraints, and collision responsibilities. Throughout the race, the referee logs state information and checks for completion conditions. If an agent completes five laps or violates a race rule, the competition is halted. The winner is determined either based on the total time spent as the defender or by disqualification due to rule violations, where the non-violating player is declared the winner.

5 Methodology

In this chapter, we present the methodological framework used to develop and analyze interaction-aware autonomous racing strategies. Our approach builds upon game-theoretic principles and model predictive control to enable strategic decision-making among competing agents.

We begin by defining the model components, including the track representation, gate passage formulation, and cost components inspired by Model Predictive Contouring Control (MPCC) [27]. Next, we introduce Model Predictive Game (MPG), which formulates the racing problem as a multi-agent dynamic game, followed by Model Predictive Game with Blocking (MPGB), which explicitly incorporates blocking strategies. We then describe the Baseline Method (MPC), which serves as a non-game-theoretic reference, and the Lifted Model Predictive Game (LMPG) approach, which accelerates online computation by leveraging learned trajectory candidates.

Finally, we outline the system architecture detailing the interaction between the optimizer, simulator, and referee module, which ensures asynchronous communication and real-time enforcement of racing rules.

5.1. Model components

The model components used in our approach are inspired by MPCC, which employs spline-based trajectory representations and a contouring formulation for time-optimal flight. Specifically, we adopt a periodic spline parametrization of the track and contouring weight scaling for gate passage, ensuring smooth trajectory tracking while allowing for deviations to optimize racing performance.

Race track parametrization The track is defined by a sequence of P points forming a closed loop, and a third-order spline interpolation is used to obtain an arc-length parametrization.

$$\boldsymbol{p}^d(\theta) = \begin{cases} \boldsymbol{\rho}_0(\theta_k) & \theta_0 \leq \theta_k \leq \theta_1 \\ \boldsymbol{\rho}_1(\theta_k) & \theta_1 \leq \theta_k \leq \theta_2 \\ \quad \vdots & \\ \boldsymbol{\rho}_{P-1}(\theta_k) & \theta_{P-1} \leq \theta_k \leq \theta_P \end{cases} \tag{5.1a}$$

$$\text{s.t.} \quad \boldsymbol{\rho}_i(\theta_k) = a_i + b_i\theta_k + c_i\theta_k^2 + d_i\theta_k^3, \quad i \in [0, ..., P-1] \tag{5.1b}$$

$$\boldsymbol{\rho}_0(\theta_0) = \boldsymbol{\rho}_{P-1}(\theta_P) \tag{5.1c}$$

$$\boldsymbol{\rho}_0'(\theta_0) = \boldsymbol{\rho}_{P-1}'(\theta_P) \tag{5.1d}$$

$$\boldsymbol{\rho}_0''(\theta_0) = \boldsymbol{\rho}_{P-1}''(\theta_P) \tag{5.1e}$$

where $\boldsymbol{p}^d$ is the arc length parametrized periodic spline, ρ_i are piecewise cubic splines. Equation 5.1c to Equation 5.1e represent the periodicity conditions.

Contouring terms The contouring terms are used to minimize the projected distance from the current position $\boldsymbol{p}$ to the desired position $\boldsymbol{p}^d(\theta)$ while maximizing the progress θ along the race track. These terms quantify the deviation from the reference path and are key components in optimizing the vehicle's motion. Figure 5.1 aims to help visualize these terms.

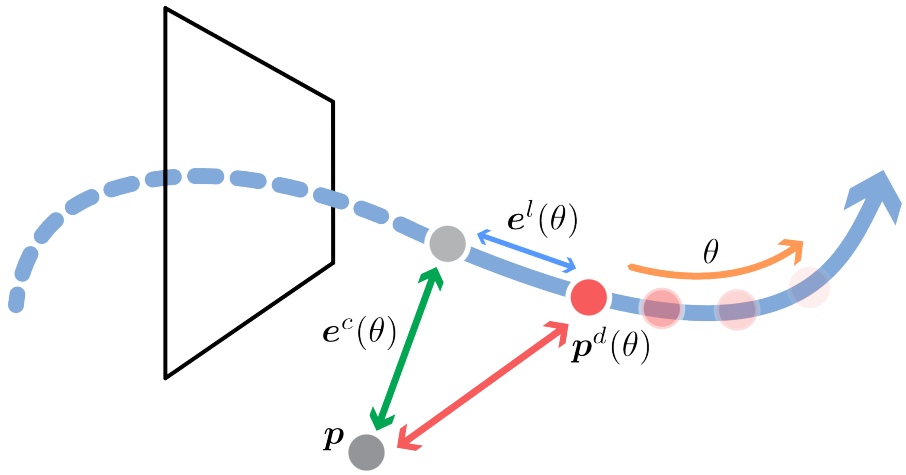


Figure 5.1: Visual representation of lag and contour terms, illustrating their roles in minimizing trajectory deviation while maximizing progress.

The contour error $\boldsymbol{e}^c(\theta)$ represents the lateral deviation from the desired trajectory, measuring how far the drone is from the optimal racing line in a perpendicular direction, while the lag error $\boldsymbol{e}^l(\theta)$ quantifies the longitudinal deviation. Since the task is to maximize the progress θ, the point on the track $\boldsymbol{p}^d(\theta)$ is acting as a moving reference the drone should track both laterally and longitudinally.

Mathematically, these errors are defined as

$$\begin{aligned} \boldsymbol{e}^l(\theta) &= e^l(\theta)\boldsymbol{t}(\theta) \\ \boldsymbol{e}^c(\theta) &= \boldsymbol{p} - \boldsymbol{p}^d(\theta) - \boldsymbol{e}^l(\theta) \end{aligned} \tag{5.2}$$

where $\boldsymbol{p}^d(\theta)$ is the desired position on the track at progress θ, $\boldsymbol{p}$ is the current position of the vehicle, and $\boldsymbol{t}(\theta)$ is the unit tangent vector to the trajectory at θ, indicating the forward direction of motion. The lag error $e^l(\theta)$ is the magnitude of the displacement along the tangent direction, and the contour error $\boldsymbol{e}^c(\theta)$ captures the lateral deviation from the reference trajectory after accounting for the longitudinal displacement.

Contour weight scaling To enforce gate passage, the contouring weight is modulated using 3D Gaussian functions centered at gate positions, similarly to how the referee deviation check was formulated in the previous chapter. This method encourages alignment with the race track while still allowing controlled deviations when beneficial. The contouring weight formulation is given in as

$$q_c(\boldsymbol{p}^d(\theta)) = q_c^{\text{nom}} + (q_c^{\text{max}} - q_c^{\text{nom}}) \sum_{j=0}^{M} e^{-\frac{1}{2}(\boldsymbol{p}^d(\theta)-\boldsymbol{p}_{g,j})^T \Sigma^{-1} (\boldsymbol{p}^d(\theta)-\boldsymbol{p}_{g,j})} \tag{5.3}$$

In this equation, $\boldsymbol{p}^d(\theta)$ represents the arc-length parameterized trajectory from Equation 5.1, while $\boldsymbol{p}_{g,j}$ denotes the position vector (x, y, z) of the j-th gate among a total of M gates. The weighting parameters q_c^{nom} and q_c^{max} control the influence of the 3D Gaussians. This allows a nominal contouring cost in regions without gates and a peak contouring penalty at gate locations. A visual representation is given in Figure 5.2.

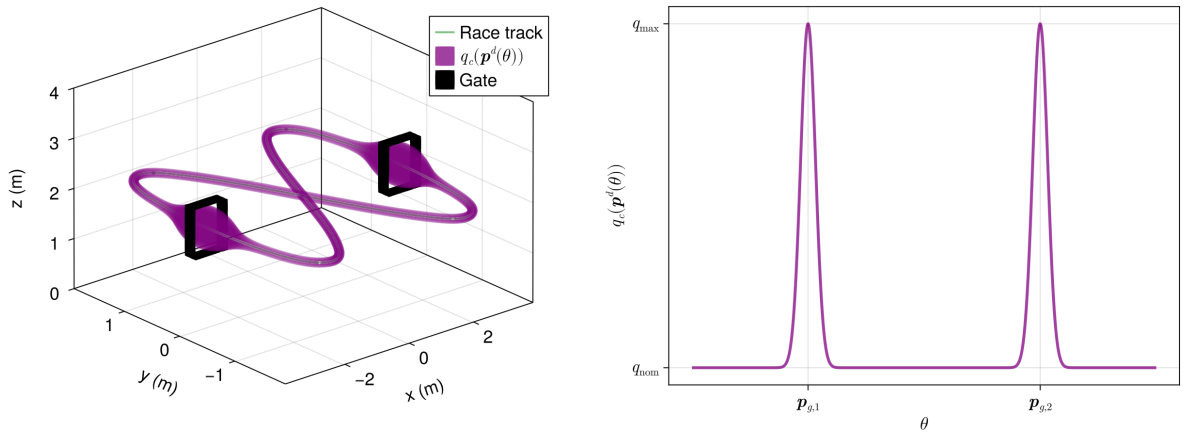


Figure 5.2: Visualization of contouring weight scaling using 3D Gaussian modulation for gate passage enforcement. The left plot illustrates the contouring weight scale in 3D along the race track, while the right plot shows it in 2D along the track length. The positions $\boldsymbol{p}_{g,1}$ and $\boldsymbol{p}_{g,2}$ indicate the locations of the gates.

Collision avoidance As outlined in the previous chapter, collision avoidance is handled by the attacking player. To enforce this, a cost term penalizing proximity beyond a predefined threshold is introduced.

$$\text{col}(\boldsymbol{p}^i, \boldsymbol{p}^{\neg i}) = \begin{cases} 0 & \text{if} \quad ||\boldsymbol{p}^i - \boldsymbol{p}^{\neg i}||^2 \geq r_{\text{col}}^2 \\ (||\boldsymbol{p}^i - \boldsymbol{p}^{\neg i}||^2 - r_{\text{col}}^2)^2 & \text{otherwise} \end{cases} \tag{5.4}$$

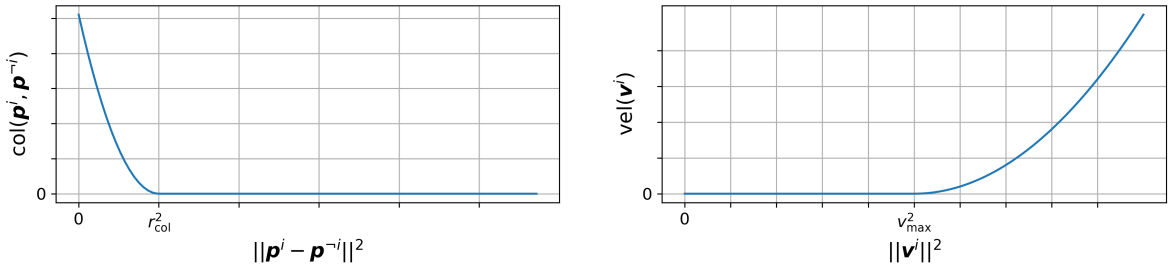
In this formulation, $\boldsymbol{p}^i$ and $\boldsymbol{p}^{\neg i}$ represent the position vectors (x, y, z) of ego and opponent, and r_{col} defines the minimum allowable separation distance. The cost function evaluates to zero when the players are sufficiently far apart, but increases quadratically if the separation falls below the threshold.

Velocity soft constraints Velocity limits are modeled as soft constraints via the cost function, distinguishing between the two roles. This formulation guarantees that each player adheres to the maximum imposed speed limits while penalizing any violations.

$$\begin{aligned} \text{vel}(\boldsymbol{v}^i) =& (v_{\max}^2 - ||\boldsymbol{v}^i||^2 - |v_{\max}^2 - ||\boldsymbol{v}^i||^2|)^2 \\ \text{s.t.} \;\; v_{\max} =& \begin{cases} v_{\text{attacker}} & \text{if} \quad \text{attacker} = i \\ v_{\text{defender}} & \text{otherwise} \end{cases} \end{aligned} \tag{5.5}$$

In this equation, $\boldsymbol{v}^i$ represents the velocity vector (x, y, z) of player i. The cost function is designed to be zero when the velocity remains within the prescribed limits and increases quadratically when the speed exceeds the threshold. The maximum velocity, $v_{\max}$, differs for attackers and defenders, enforcing role-specific constraints as described in the previous chapter.

The following Figure 5.3 aims to visualize the cost functions for collision avoidance and velocity constraints.



(a) Collision avoidance cost term from Equation 5.4

(b) Velocity soft constraints via cost function from Equation 5.5

Figure 5.3: Visualization of collision avoidance and velocity constraint cost functions

Notably, the use of soft constraints for collision avoidance, velocity limits, and gate passage ensures that the optimization problem remains feasible and computationally tractable, even at high speeds. By penalizing constraint violations rather than imposing hard constraints, the approach allows for smoother control actions and better adaptability to dynamic interactions. While a detailed discussion on the implementation and experimental setup appears later, these design choices play a key role in handling the complexities of real-time decision-making in high-speed competitive racing scenarios.

5.2. Model predictive game - MPG

We adopt a point mass dynamic model with jerk control to describe the motion of the agents. The system state includes position, velocity, and acceleration, while the state space is further extended to incorporate progress dynamics. The control input consists of jerk and a virtual input for progress acceleration, denoted as $\Delta v_\theta = \frac{dv_\theta}{dt}$, where v_θ is the speed of the progress along the track. The corresponding augmented state and input spaces are defined as

$$\boldsymbol{x}^i = [p_x \;\; p_y \;\; p_z \;\; v_x \;\; v_y \;\; v_z \;\; a_x \;\; a_y \;\; a_z \;\; \theta \;\; v_\theta]^T, \quad \boldsymbol{u}^i = [j_x \;\; j_y \;\; j_z \;\; \Delta v_\theta]^T \tag{5.6a}$$

$$\dot{\boldsymbol{x}}^i = [\dot{\boldsymbol{p}} \;\; \dot{\boldsymbol{v}} \;\; \dot{\boldsymbol{a}} \;\; \dot{\theta} \;\; \dot{v_\theta}]^T = [\boldsymbol{v} \;\; \boldsymbol{a} \;\; \boldsymbol{j} \;\; v_\theta \;\; \Delta v_\theta]^T = A\boldsymbol{x}^i + B\boldsymbol{u}^i \tag{5.6b}$$

where A and B are the state and input control matrices that follow the continuous time dynamics.

The continuous-time dynamics of each player i are expressed in Equation 5.6b, where the state derivative comprises velocity, acceleration, jerk, and the progress terms. These dynamics describe the evolution of the player's state over time and serve as the foundation for the discrete-time system.

Since the game involves multiple players, the joint state vector in Equation 5.7 is formed by concatenating the individual state vectors of all N players. This allows for a compact representation of the entire system.

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}^1 \\ \vdots \\ \boldsymbol{x}^N \end{bmatrix} \tag{5.7}$$

The game is played over a discrete-time horizon of length K, where each stage $k \in \mathcal{K} = \{0, \ldots, K-1\}$ represents a time step in the decision-making process. The discrete-time dynamics, which govern the evolution of the joint state $\boldsymbol{x}_k$ in this K-stage dynamic game, are formulated as a time-invariant nonlinear function $\boldsymbol{f}_k$:

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}_k(\boldsymbol{x}_k, \boldsymbol{u}_k^1, ..., \boldsymbol{u}_k^N) \tag{5.8}$$

This function captures how the state transitions from one time step to the next based on the control inputs of all players. The discretization process relies on the fourth-order Runge-Kutta method in allowing for stability and accuracy in the numerical integration of the continuous-time dynamics.

$$\begin{aligned}\boldsymbol{A}_k &= \mathbb{I} + \Delta t A + \frac{\Delta t^2}{2!}A^2 + \frac{\Delta t^3}{3!}A^3 + \frac{\Delta t^4}{4!}A^4 \\ \boldsymbol{B}_k &= (\mathbb{I}\Delta t + \frac{\Delta t^2}{2!}A^2 + \frac{\Delta t^3}{3!}A^3 + \frac{\Delta t^4}{4!}A^4)B\end{aligned} \tag{5.9}$$

where Δt is the discretization time step.

At each stage k of the receding horizon plan with length K, player i incurs a stage cost that depends on both its control inputs $\boldsymbol{u}_k^i \in \boldsymbol{u}^i = \{\boldsymbol{u}_0^i, \boldsymbol{u}_1^i, ..., \boldsymbol{u}_{K-1}^i\}$ and the system state $\boldsymbol{x}_k$. The sequence of states is influenced by the control inputs of all players, which is often expressed using the notation $\neg i$ to denote all players except i. Given an initial state $\boldsymbol{x}_0$, the total cost for player i is computed as the sum of the stage costs g_k^i over the horizon, with an additional terminal cost g_K^i applied at the final step.

$$J^i(\boldsymbol{x}_0, \boldsymbol{u}^i, \boldsymbol{u}^{\neg i}) = \sum_{k=0}^{K-1} g_k^i(\boldsymbol{x}_k, \boldsymbol{u}_k^i) + g_K^i(\boldsymbol{x}_K) \tag{5.10}$$

In the context of competitive multi-agent decision-making, each agent computes a sequence of actions to maximize their performance in the race. The policy governing an agent's action selection process is referred to as its strategy. A strategy $\gamma^i(\cdot) = \{\gamma_0^i(\cdot), \gamma_1^i(\cdot), \ldots, \gamma_{K-1}^i(\cdot)\}$ of the strategy space $\Gamma^i = \{\Gamma_0^i, \Gamma_1^i, \ldots, \Gamma_{K-1}^i\}$ determines the sequence of control inputs $\boldsymbol{u}^i$ at each stage k, based on the information available to player i. In an open-loop setting, all players observe the initial state $\boldsymbol{x}_0$ and precompute a sequence of control inputs for the entire time horizon in a single act. The strategy at stage k is a constant function, with $\gamma_k^i(\cdot) \in \Gamma_k^i = \mathcal{U}^i$, meaning that players do not adjust their actions based on new observations beyond the initial state.

$$\begin{aligned}\Gamma_k^i \ni \gamma_k^i &: \mathcal{X} \to \mathcal{U}^i \\ \boldsymbol{u}_k^i &= \gamma_k^i(\boldsymbol{x}_0)\end{aligned} \tag{5.11}$$

Similarly, we define the strategy vector of player i for the receding horizon as

$$\gamma^i(\boldsymbol{x}_0) = \{\overbrace{\gamma_0^i(\boldsymbol{x}_0)}^{\boldsymbol{u}_0^i}, \overbrace{\gamma_1^i(\boldsymbol{x}_0)}^{\boldsymbol{u}_1^i}, \ldots, \overbrace{\gamma_{K-1}^i(\boldsymbol{x}_0)}^{\boldsymbol{u}_{K-1}^i}\} = \boldsymbol{u}^i \tag{5.12}$$

The N-tuple of strategies $\{\gamma^{i*}(\boldsymbol{x}_0) \in \Gamma^i; i \in \mathcal{N}\}$, directly translates into the players' input sequences $\{\boldsymbol{u}^{i*} = \gamma^{i*}(\boldsymbol{x}_0); i \in \mathcal{N}\}$, and constitutes an open-loop Nash equilibrium if

$$\forall i \in N : J^i(\boldsymbol{x}_0, \gamma^{i*}(\boldsymbol{x}_0), \gamma^{\neg i*}(\boldsymbol{x}_0)) \leq J^i(\boldsymbol{x}_0, \gamma^i(\boldsymbol{x}_0), \gamma^{\neg i*}(\boldsymbol{x}_0)) \tag{5.13}$$

In other words, no player can improve its outcome by unilaterally altering its strategy. Starting from the given initial state, $\boldsymbol{x}_0^* = \boldsymbol{x}_0$, the discrete system dynamics generate the corresponding open-loop trajectory $\{\boldsymbol{x}_{k+1}^*; k \in \{0, 1, \ldots, K-1\}\}$.

Using the previously defined model components, we construct the cost function for player i, which depends on the initial state and the strategies of both the ego player and the opponent.

$$\begin{aligned}J^i(\boldsymbol{x}_0, \overbrace{\boldsymbol{u}^i}^{\gamma^i(\boldsymbol{x}_0)}, \overbrace{\boldsymbol{u}^{\neg i}}^{\gamma^{\neg i}(\boldsymbol{x}_0)}) = &\sum_{k=0}^{K-1} ||e^l(\theta_k^i)||_{q_l}^2 + ||e^c(\theta_k^i)||_{q_c(\boldsymbol{p}^d(\theta_k^i))}^2 && \text{(5.14a)} \\ &+ \mathbf{1}\{\text{attacker} = i\} q_{\text{col}} \ \text{col}(\boldsymbol{p}_k^i, \boldsymbol{p}_k^{\neg i}) && \text{(5.14b)} \\ &+ q_{\text{vel}} \ \text{vel}(\boldsymbol{v}_k^i) && \text{(5.14c)} \\ &+ ||\Delta v_{\theta,k}^i||_{q_{\Delta v}}^2 + ||\boldsymbol{j}_k^i||_{q_j}^2 && \text{(5.14d)} \\ &+ \mu(v_{\theta,k}^{\neg i} - v_{\theta,k}^i) && \text{(5.14e)}\end{aligned}$$

Equation 5.14a consists of contouring and lag cost terms, adopted from [27], with the appropriate contour cost scaling $q_c(\boldsymbol{p}^d(\theta_k^i))$ defined in Equation 5.3. Equation 5.14b includes the collision cost term, which is active only for the attacker role. The operator $\mathbf{1}\{\cdot\}$ evaluates to 1 if the condition holds and 0 otherwise. Equation 5.14c introduces the cost component for the soft velocity constraints, as defined in Equation 5.5. Equation 5.14d comprises of regularization terms on the control inputs of player i. Finally, the term in Equation 5.14e incentivizes the ego player to maximize progress along the track by increasing its progress velocity while simultaneously minimizing the opponent's progress velocity. This addition is intended to encourage blocking and other competitive strategic behaviors, as demonstrated in the method of [37].

We now formulate the game as a coupled optimal control problem for the two-player case. At each simulation time step t, an optimal strategy γ^{i*} is computed in a receding horizon fashion, where each player selects a sequence of control inputs that minimizes its cost function while anticipating the actions of the opponent. This approach operates with a moving horizon, meaning that the initial state x_0 is updated at each re-planning phase to reflect the current system state, accounting for the dynamically changing environment and potential deviations from predicted opponent behavior.

In a moving horizon implementation, the control inputs can be warm-started using the solution from the previous planning step. This initialization reduces the number of required solver iterations and therefore allows for real-time capabilities. The game takes the form

$$\gamma^{1*}(\boldsymbol{x}_0) = \boldsymbol{u}^{1*} \in \underset{\boldsymbol{u^1}}{\operatorname{argmin}} \sum_{k=0}^{K-1} J^1(\boldsymbol{x}_0, \boldsymbol{u}_k^1, \boldsymbol{u}_k^2) \tag{5.15a}$$

$$\text{s.t.} \quad \begin{matrix} c^1(\boldsymbol{u}_k^1) = 0 \\ h^1(\boldsymbol{u}_k^1) \geq 0 \end{matrix} \tag{5.15b}$$

$$\gamma^{2*}(\boldsymbol{x}_0) = \boldsymbol{u}^{2*} \in \underset{\boldsymbol{u^2}}{\operatorname{argmin}} \sum_{k=0}^{K-1} J^2(\boldsymbol{x}_0, \boldsymbol{u}_k^1, \boldsymbol{u}_k^2) \tag{5.15c}$$

$$\text{s.t.} \quad \begin{matrix} c^2(\boldsymbol{u}_k^2) = 0 \\ h^2(\boldsymbol{u}_k^2) \geq 0 \end{matrix} \tag{5.15d}$$

Each player must satisfy its respective equality constraint c^i and inequality constraint h^i. The equality constraints ensure that the resulting sequence of states and control inputs remains dynamically feasible, meaning that the computed trajectories are physically consistent with the vehicle dynamics. The inequality constraints define upper and lower bounds on the states and control inputs, limiting factors such as maximum acceleration, jerk, and velocity along the track. By incorporating these constraints, the optimization problem accounts for the physical limitations of the quadrotors, ensuring feasible and safe trajectories during the race.

To analyze the necessary optimality conditions for this problem, we define the Lagrangian function as

$$\mathcal{L}^i(\boldsymbol{x}_0, \boldsymbol{u}^i, \boldsymbol{u}^{\neg i}, \lambda^i, \mu^i) = J^i(\boldsymbol{x}_0, \boldsymbol{u}^i, \boldsymbol{u}^{\neg i}) - {\lambda^i}^T h^i(\boldsymbol{u}^i) - {\mu^i}^T c^i(\boldsymbol{u}^i) \tag{5.16}$$

The Lagrangian incorporates the objective function J^i for player i, along with the associated equality and inequality constraints weighted by the Lagrange multipliers λ^i and μ^i, respectively. Using this formulation, we derive the coupled KKT conditions that characterize the open-loop Nash equilibrium. The system consists of three conditions as follows

$$\forall i \in [N] \begin{cases} \nabla_{\boldsymbol{u}^i} \mathcal{L}^i = 0 \\ 0 = c^i(\boldsymbol{u}^i) \\ 0 \leq h^i(\boldsymbol{u}^i) \perp \lambda^i \geq 0 \end{cases} \tag{5.17}$$

The first condition enforces stationarity, requiring that the gradient of the Lagrangian with respect to the decision variables of player i is zero, ensuring that no player can unilaterally improve their strategy.

The second condition enforces primal feasibility, ensuring that the equality constraints c^i are satisfied exactly and that the inequality constraints h^i remain non-negative. The third condition imposes dual feasibility and complementary slackness, meaning that the Lagrange multipliers associated with inequality constraints must be non-negative and that each inequality constraint is either strictly satisfied or active with a corresponding nonzero multiplier. Together, these conditions establish the first-order necessary requirements for an open-loop Nash equilibrium. For further details on necessary optimality conditions and the derivation of KKT conditions in multi-agent optimization and Nash equilibrium problems, we refer to [1].

The KKT conditions can now be reformulated as a Mixed Complementarity Problem (MCP), to explicitly capture the structure of the optimization problem where each player's strategy must satisfy stationarity, feasibility, and complementarity conditions.

$$\text{Given } F : \mathbb{R}^d \mapsto \mathbb{R}^d; \overline{\mathbf{w}}, \underline{\mathbf{w}} \in R^d; \text{find } \mathbf{w} \in \mathbb{R}^d \text{ s.t.} \tag{5.18a}$$

$$\text{if } \mathrm{w}_j = \underline{\mathrm{w}_j} \text{ then } F_j(\mathbf{w}) \geq 0 \tag{5.18b}$$

$$\text{if } \underline{\mathrm{w}_j} < \mathrm{w}_j < \overline{\mathrm{w}_j} \text{ then } F_j(\mathbf{w}) = 0 \tag{5.18c}$$

$$\text{if } \mathrm{w}_j = \overline{\mathrm{w}_j} \text{ then } F_j(\mathbf{w}) \leq 0 \tag{5.18d}$$

The MCP framework defines a function $F : \mathbb{R}^d \mapsto \mathbb{R}^d$ along with lower and upper bounds $\underline{\mathbf{w}}, \overline{\mathbf{w}}$. The solution $\mathbf{w}$ must satisfy three possible conditions at each element j: Equation 5.18b if $\mathbf{w}_j$ is at its lower bound, the corresponding function value must be non-negative; Equation 5.18c if $\mathbf{w}_j$ lies strictly between the bounds, the function value must be exactly zero; and Equation 5.18d if $\mathbf{w}_j$ is at its upper bound, the function value must be non-positive.

Mapping this to the problem presented in Equation 5.15, the decision vector $\mathbf{w}$ consists of the players' decision variables $\boldsymbol{u}^1, \boldsymbol{u}^2$, the Lagrange multipliers for inequality constraints λ^1, λ^2, and the Lagrange multipliers for equality constraints μ^1, μ^2. The function $F(\mathbf{w})$ encodes the first-order optimality conditions, including the stationarity conditions for each player's strategy, as well as the feasibility constraints. The choice of infinite upper and lower bounds for certain variables ensures that they are treated as unconstrained, meaning they fall under the second case where $F_j(\mathbf{w}) = 0$. This applies to the strategy variables and equality constraint multipliers, while the inequality constraint multipliers remain non-negative, enforcing complementarity. The values are given as

$$\begin{aligned} \mathbf{w} &= [\boldsymbol{u}^1, \boldsymbol{u}^2, \lambda^1, \lambda^2, \mu^1, \mu^2]^T, \qquad F(\mathbf{w}) = [\nabla_{\boldsymbol{u}^1}\mathcal{L}^1, \nabla_{\boldsymbol{u}^2}\mathcal{L}^2, h^1, h^2, c^1, c^2] \\ \underline{\mathbf{w}} &= [-\infty, -\infty, 0, 0, -\infty, -\infty]^T, \quad \overline{\mathbf{w}} = [\infty, \infty, \infty, \infty, \infty, \infty]^T \end{aligned} \tag{5.19}$$

To compute a local NE for this problem, we leverage MCPTrajectoryGameSolver.jl[1] [24]. This toolchain is specifically designed for formulating dynamic trajectory games by transcribing the equilibrium conditions into a mixed complementarity problem. The resulting problem is then solved using PATH.jl[2], a widely used solver for MCPs.

5.3. Blocking method - MPGB

The cost function in Equation 5.14, consisted of a term that includes the opponent's progress velocity to incentivize blocking behavior. However, when expanding the stationarity condition of the Lagrangian in the equation below, we observe that this term does not influence the ego player's decision. Since the opponent's strategy is fixed from the ego player's perspective in the open-loop setting, the progress velocity term acts as a constant and merely shifts the overall cost landscape without altering the optimal solution. As a result, no actual blocking behavior emerges, and the Nash equilibrium remains unchanged.

[1] https://github.com/JuliaGameTheoreticPlanning/MCPTrajectoryGameSolver.jl, accessed: March 2025
[2] https://github.com/chkwon/PATHSolver.jl, accessed: March 2025

$$
\begin{aligned}
\nabla_{\boldsymbol{u}^i}\mathcal{L}^i &= \nabla_{\boldsymbol{u}^i}J^i(\boldsymbol{x}_0,\boldsymbol{u}^i,\boldsymbol{u}^{\neg i}) - \nabla_{\boldsymbol{u}^i}\left(\lambda^{i^T}h^i(\boldsymbol{u}^i)+\mu^{i^T}c^i(\boldsymbol{u}^i)\right)\\
&= \nabla_{\boldsymbol{u}^i}\left(||e^l(\theta_k^i)||^2_{q_l} + ||e^c(\theta_k^i)||^2_{q_c(p^d(\theta_k^i))} + \mathbf{1}\{\text{attacker}=i\}q_{\text{col}}\ \text{col}(\boldsymbol{p}_k^i,\boldsymbol{p}_k^{\neg i})\right)\\
&+ \nabla_{\boldsymbol{u}^i}\left(q_{\text{vel}}\ \text{vel}(\boldsymbol{v}_k^i) + ||\Delta v_{\theta,k}^i||^2_{q_{\Delta v}} + ||\boldsymbol{j}_k^i||^2_{q_j}\right) + \cancelto{0}{\nabla_{\boldsymbol{u}^i}\mu v_{\theta,k}^{\neg i}} - \nabla_{\boldsymbol{u}^i}\mu v_{\theta,k}^i\\
&- \nabla_{\boldsymbol{u}^i}\left(\lambda^{1^T}h^i(\boldsymbol{u}^i)+\mu^{i^T}c^i(\boldsymbol{u}^i)\right)
\end{aligned}
\tag{5.20}
$$

To explicitly introduce blocking strategies, we modify the previous cost functional of by adding an additional blocking term.

$$
\begin{aligned}
J^i(\boldsymbol{x}_0,\boldsymbol{u}_k^i,\boldsymbol{u}_k^{\neg i}) = \ & ||e^l(\theta_k^i)||^2_{q_l} + ||e^c(\theta_k^i)||^2_{q_c(\boldsymbol{p}^d(\theta_k^i))} && \text{(5.21a)}\\
& + \mathbf{1}\{\text{attacker}=i\}q_{\text{col}}\ \text{col}(\boldsymbol{p}_k^i,\boldsymbol{p}_k^{\neg i}) && \text{(5.21b)}\\
& - \mathbf{1}\{\text{attacker}\neq i\}q_{\text{block}}\ \text{col}(\boldsymbol{p}_k^i,\boldsymbol{p}_k^{\neg i}) && \text{(5.21c)}\\
& + q_{\text{vel}}\ \text{vel}(\boldsymbol{v}_k^i) && \text{(5.21d)}\\
& + ||\Delta v_{\theta,k}^i||^2_{q_{\Delta v}} + ||\boldsymbol{j}_k^i||^2_{q_j} && \text{(5.21e)}\\
& - \mu v_{\theta,k}^i && \text{(5.21f)}
\end{aligned}
$$

The term from Equation 5.21c encourages the ego player to minimize the distance to the opponent when in the defender role. Unlike the previous formulation, this new term actively depends on the ego player's decision variables, meaning it does not cancel out in the KKT conditions. Specifically, when the ego player is not the attacker, the cost function now includes a negative collision penalty, incentivizing the defender to approach the opponent and obstruct their movement.

This modification ensures that blocking is an emergent strategic behavior rather than an incidental outcome of the optimization. By structuring the cost function in this way, we allow players to dynamically switch between aggressive overtaking and defensive blocking, depending on their assigned role. This adjustment enables the model to capture competitive interactions more effectively, leading to more realistic and tactically rich racing strategies.

5.4. Baseline method - MPC

The baseline method, based on model predictive approaches, formulates the trajectory planning problem as a single-player optimal control problem. In this formulation, the opponent is treated as a constant velocity obstacle, meaning that its future positions are predicted using a simple kinematic model rather than being actively optimized as in the game-theoretic approach.

Unlike the previous methods, there is no strategic coupling between players, meaning the ego player's strategy does not directly depend on the opponent's decisions. The cost function for the ego player is given by:

$$
\begin{aligned}
J^1(\boldsymbol{x}_0^1,\boldsymbol{u}_k^1,\boldsymbol{p}_k^2) = \ & ||e^l(\theta_k^1)||^2_{q_l} + ||e^c(\theta_k^1)||^2_{q_c(\boldsymbol{p}^d(\theta_k^1))} && \text{(5.22a)}\\
& + \mathbf{1}\{\text{attacker}=1\}q_{\text{col}}\ \text{col}(\boldsymbol{p}_k^1,\boldsymbol{p}_k^2) && \text{(5.22b)}\\
& + q_{\text{vel}}\ \text{vel}(\boldsymbol{v}_k^1) && \text{(5.22c)}\\
& + ||\Delta v_{\theta,k}^1||^2_{q_{\Delta v}} + ||\boldsymbol{j}_k^1||^2_{q_j} && \text{(5.22d)}\\
& - \mu v_{\theta,k}^1 && \text{(5.22e)}
\end{aligned}
$$

The optimal control problem for the single player is given in the equation below. At each time step, the ego player calculates an optimal sequence of control inputs while treating the opponent's motion as a known quantity. The opponent's position $\boldsymbol{p}_k^2$ is derived by assuming it moves at a constant velocity,

following a simple kinematic rule. The same constraints applied in previous game-theoretic methods are retained here.

$$\begin{aligned} \gamma^{1*}(\boldsymbol{x}_0) = \boldsymbol{u}^{1*} \in \underset{\boldsymbol{u}^1}{\operatorname{argmin}} \ & \sum_{k=0}^{K-1} J^1(\boldsymbol{x}_0^1, \boldsymbol{u}_k^1, \boldsymbol{p}_k^2) \\ & \boldsymbol{p}_k^2 = \boldsymbol{v}_0^2 k \Delta t \\ \text{s.t.} \quad & c^1(\boldsymbol{u}_k^1) = 0 \\ & h^1(\boldsymbol{u}_k^1) \geq 0 \end{aligned} \tag{5.23}$$

5.5. Lifted method - LMPG

The lifted method is based on the work by Peters et al. [26] and reformulates the problem by having each player optimize a distribution over multiple trajectory candidates rather than a single deterministic strategy. This is formulated as a game where players select both the trajectory candidates and their respective mixing weights, enabling them to sample from an optimized distribution during execution.

$$\min_{q^1, \tau_1^1, \ldots, \tau_m^1} \quad \mathop{\mathbb{E}}_{\substack{\tau^1 \sim \mathcal{T}^1 \\ \tau^2 \sim \mathcal{T}^2}} [J^1(\boldsymbol{x}_0, \tau^1, \tau^2)] \qquad \min_{q^2, \tau_1^2, \ldots, \tau_m^2} \quad \mathop{\mathbb{E}}_{\substack{\tau^1 \sim \mathcal{T}^1 \\ \tau^2 \sim \mathcal{T}^2}} [J^2(\boldsymbol{x}_0, \tau^1, \tau^2)] \tag{5.24}$$

$$\mathcal{T}^i := \operatorname{Cat}(\ \underbrace{\{\tau_1^i, \ldots, \tau_m^i\}}_{\text{trajectory candidates}}\ ,\ \overbrace{q^i}^{\text{mixing weights}}\ ) \tag{5.25}$$

where a single trajectory candidate is equal to a strategy $\tau_j^i = \gamma^i(\boldsymbol{x}_0)$ for $j \in \{1, \ldots, m\}$ possible candidates, and $\mathcal{T}^i$ represents the categorical distribution from which player i samples to minimize its expected cost.

While this formulation admits mixed strategies, it drastically increases problem size. To mitigate this, the method incorporates a computational framework that learns competitive trajectory candidates to accelerate online decision-making.

The key idea is to shift part of the computational load to an offline phase by training a neural network-based reference generator for each player to solve the game in Equation 5.24. This generator, denoted as $\pi^i(\boldsymbol{x}_0^1, \boldsymbol{x}_0^2)$, maps the initial states of both players $(\boldsymbol{x}_0^1, \boldsymbol{x}_0^2)$ to a set of reference trajectories ξ^i. These reference trajectories serve as guiding solutions for an embedded differentiable trajectory optimizer, which refines them into dynamically feasible trajectories τ^i. The reference generator is typically implemented as a multi-layer perceptron (MLP) trained on a dataset of prior game configurations.

The overall pipeline of lifted games is illustrated in Figure 5.4. Each player generates a set of m trajectory candidates, forming a bimatrix game where a NE in mixed strategies is computed.

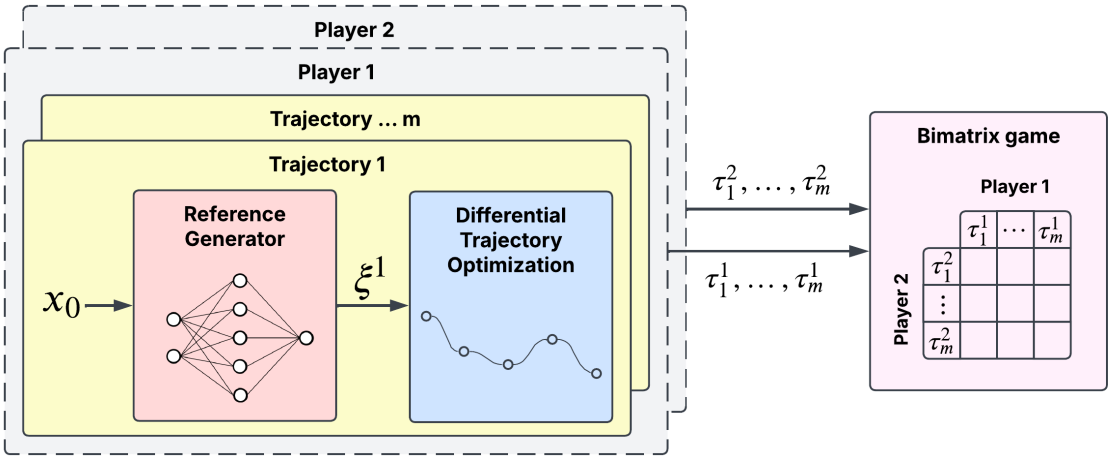


Figure 5.4: Pipeline of lifted games. Each player generates multiple trajectory candidates, and a bimatrix game solver finds a Nash Equilibrium over these choices

To effectively learn competitive trajectories, the reference generator must be designed to capture key aspects of racing dynamics. This is achieved by structuring its input representation in a way that provides relevant spatial and dynamic information to the network. The design of this input configuration is illustrated in Figure 5.5.

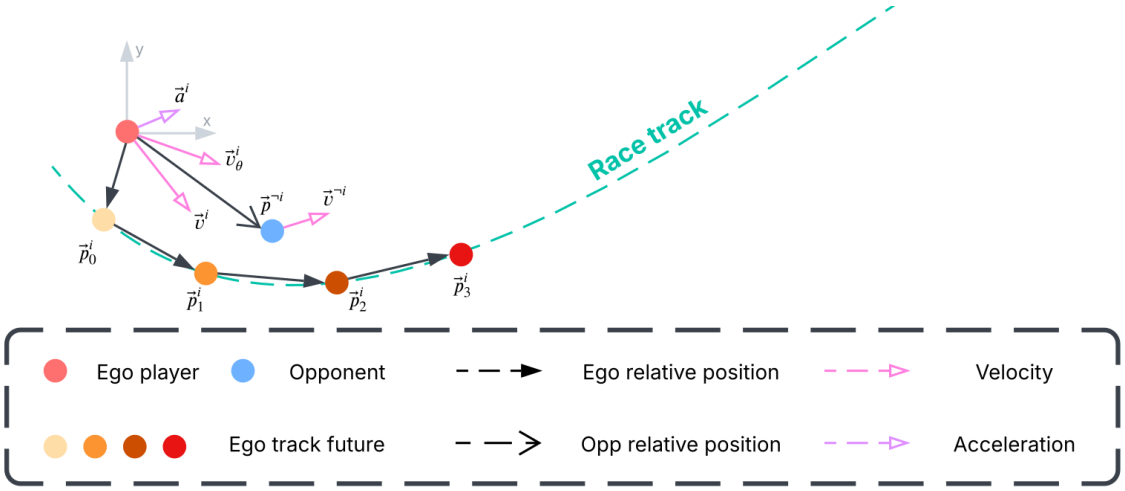


Figure 5.5: Neural network input configuration for reference generation in lifted games (LMPG)

The input to the network consists of the following components:

- Positional References: The current position of the ego player, denoted as p_0^i, is projected onto the track. Future positions, p_1^i to p_3^i, are defined at equal intervals along the track. These positional references are designed to help the network learn the structure of the track by providing information about upcoming turns, bends, and trajectory changes. By incorporating future points, the network can anticipate the track layout and adjust its strategy accordingly. Importantly, each position is defined relative to the previous one, forming a shifted coordinate system where
  - p_0^i is relative to the ego player's current off-track position.
  - p_1^i is measured from p_0^i.
  - p_2^i is measured from p_1^i, and so on.

- Opponent Information: The relative position of the opponent, $\boldsymbol{p}^{\neg i}$, is measured from the ego player's off-track position. This provides the network with information about the opponent's spatial configuration, relevant for avoiding collisions.
- Velocity Terms: The velocity of both the ego player and the opponent are included as $\boldsymbol{v}^i$ and $\boldsymbol{v}^{\neg i}$, respectively.
- Progress Velocity: The longitudinal velocity along the track, denoted as v_θ^i, provides insight into the player's motion along the racing trajectory.
- Attacker Variable: An indicator specifying whether the ego player is in an attacking or defending role, allowing the network to learn strategic variations based on the competitive context.

The complete input layer used for reference generation is mathematically defined in Equation 5.26:

$$\boldsymbol{\pi}^i_{\text{input}} = [\boldsymbol{p}^i_0, \boldsymbol{p}^i_1, \boldsymbol{p}^i_2, \boldsymbol{p}^i_3, \boldsymbol{p}^{\neg i}, \boldsymbol{v}^i, \boldsymbol{v}^{\neg i}, \boldsymbol{a}^i, v^i_\theta, \text{attacker}]^T \tag{5.26}$$

To ensure the lifted game framework effectively learns competitive trajectory generation, we design a structured training pipeline that enables the model to refine its strategy through self-play and against predefined opponents. We use Weights & Biases (WANDB)[3] to monitor whether the running cost is decreasing for each player, ensuring that training progresses as expected.

The training procedure consists of the following steps:

1. Dataset collection: We gather a dataset of observed game states by simulating races between the previously defined Baseline method (MPC) and Model Predictive Game (MPG).
2. Input normalization: We compute Z-score normalization for all input features based on the collected dataset. The mean and standard deviation of each feature from Equation 5.26 are calculated to ensure consistent scaling during training.
3. Positional constraints: We impose upper and lower bounds on the player's positional states based on the race track extremities. This ensures that players do not stray too far from the track, improving the realism and relevance of the learned strategies.
4. Disabling referee rule enforcement: In the early training phase, referee rule enforcement is disabled. Keeping it on during initial training leads to frequent race resets, which can slow down generalization to the whole track. Allowing unconstrained gameplay at first helps the model explore a broader range of scenarios before stricter rules are introduced.
5. Self-play training (Part 1): The model is trained on-policy for $50,000$ steps playing against itself to learn competitive and adaptive trajectory strategies without external interference.
6. Training against a defined opponent (Part 2): After self-play, the model is trained for an additional $50,000$ steps against one of the predefined methods (such as MPC or MPG) with a smaller learning rate. At this stage, referee rule enforcement is re-enabled to ensure that the learned strategies adhere to racing rules while still being competitive.

Once training is complete, learning is completely disabled, and the trained reference generators operate in inference mode. To maintain consistency, input normalization is applied using the mean and standard deviation computed during training, ensuring that test-time inputs are processed in the same scale as those seen during training. To evaluate the model under realistic racing conditions, referee rule enforcement is re-enabled, meaning that players must adhere to racing rules such as staying on track and avoiding collisions. However, positional constraints are lifted, this ensures that the learned policy is not overly restricted by artificial constraints imposed during training but still operates within the expected rules of the game.

5.6. System architecture

The methods presented so far include Model Predictive Game (MPG), Model Predictive Game with Blocking (MPGB), Baseline method (MPC) and Lifted Model Predictive Game (LMPG). These methods

[3] https://wandb.ai/, accessed: March 2025

are integrated into a system architecture designed for real-time interaction between the optimizer, a flight simulator, and a referee module.

Figure 5.6 provides an overview of the interaction between different components in our architecture. The optimizer module, which represents any of the defined methods, communicates with a flight simulator via websockets. The flight simulator executes the point mass strategies, while the referee ensures rule enforcement and race progression.

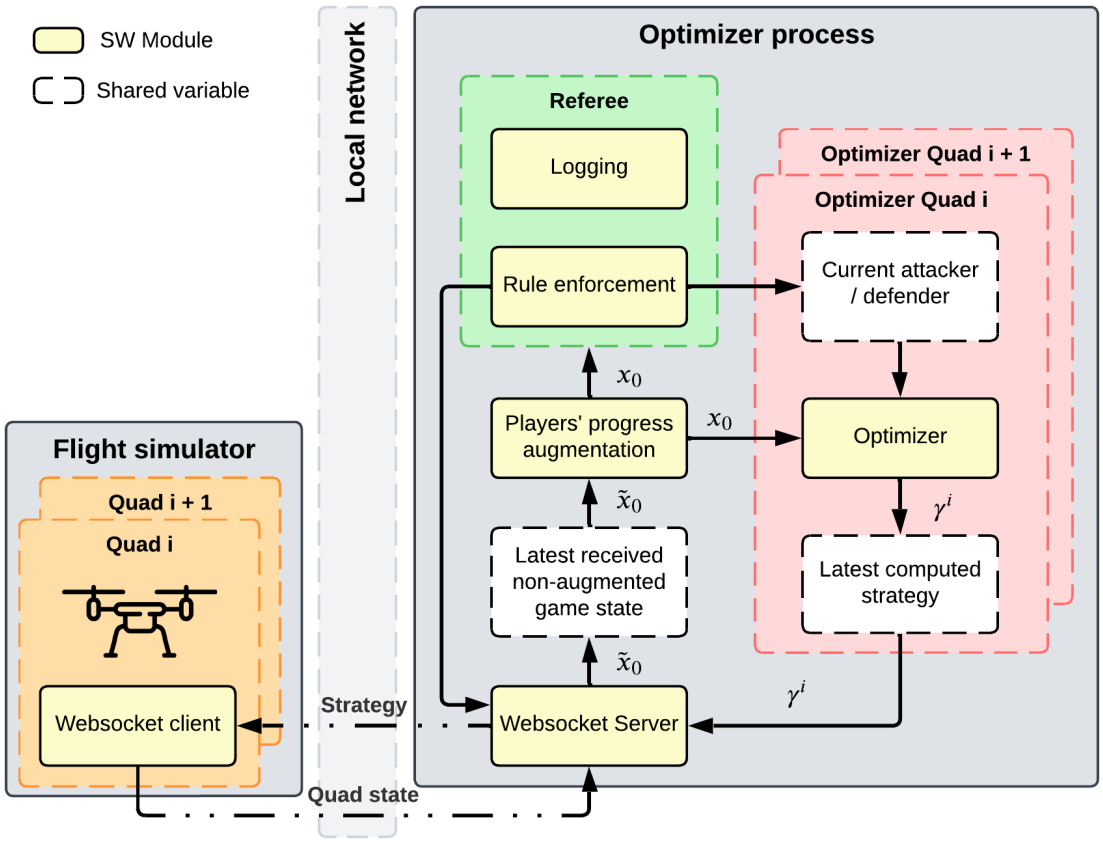


Figure 5.6: Overview of the system architecture, illustrating the interaction between the optimizer, WebSocket server, flight simulator, and referee module.

The optimizer process initializes a WebSocket server, which facilitates communication with the players inside the flight simulator. Each time a new quadrotor state is received from one of the clients, the non-augmented game state $\tilde{x}_0$ is updated and stored in a shared variable. The flight simulator is unaware of the contouring terms, therefore we define $\tilde{x}^i$ as

$$\tilde{x}^i = [p_x \;\; p_y \;\; p_z \;\; v_x \;\; v_y \;\; v_z \;\; a_x \;\; a_y \;\; a_z]^T \tag{5.27}$$

In a separate thread, the latest received non-augmented game state $\tilde{x}_0$ is augmented with the players' progress θ by solving the problem formulated in Equation 4.1. The remaining state component v_θ is found by projecting the velocity vector v onto the tangent $t(\theta)$ unit vector. These two components for each player are augmented to $\tilde{x}_0$, therefore resulting in x_0. The referee module processes this augmented game state, enforces race rules, and determines whether the race should continue or be terminated. If the race needs to be terminated, the referee communicates with the WebSocket server, which in turn signals the flight simulator to stop. Additionally, the referee assigns player roles and logs the game state.

On separate processes, each player's optimizer utilizes the latest available augmented game state and solves an optimization problem in the form of Equation 5.15, then stores and communicates back the latest strategy $\gamma^{i*}(\boldsymbol{x}_0)$. Inside the flight simulator, each element of the received strategy is time-stamped using the simulator time t_0 at which the quad state was communicated. The strategy becomes $\gamma^i(\boldsymbol{x}_0) = \{\gamma^i_{t_0}(\boldsymbol{x}_0), \gamma^i_{t_0+\Delta t}(\boldsymbol{x}_0), \ldots, \gamma^i_{t_0+\Delta t(K-1)}(\boldsymbol{x}_0)\}$. The flight simulator then executes the strategy while accounting for any delays between the request time t_0 and current time t.

It is important to note that the optimizers, WebSocket server, and referee operate asynchronously, each relying on the most recently stored shared variables.

In the following chapters, we analyze the performance of the proposed methods using the point mass dynamics simulator. However, in Chapter 8, we transition to a high-fidelity flight simulator, where a low-level nonlinear model predictive control (NMPC) controller is introduced to track the point mass strategies in a hierarchical manner.

6

Experimental Setup

This chapter presents the experimental setup used to evaluate the performance of various autonomous racing strategies in a competitive, head-to-head racing scenario. The goal is to benchmark the proposed methods under controlled conditions, ensuring reproducibility and fairness. We implement a structured tournament format where each method competes against others in a round-robin style, with multiple variations in environmental configurations to assess robustness and adaptability.

6.1. Simulation environment

All implementations utilize PATH as the low-level mixed complementarity problem (MCP) solver, ensuring consistent numerical optimization across all tested methods. The evaluated methods include:

- MPC (Model Predictive Control)
- MPG (Model Predictive Game)
- MPGB (Model Predictive Game with Blocking)
- LMPG (Lifted Model Predictive Game)

The simulation supports both synchronous and asynchronous execution modes, which define how agents process information and make decisions. These modes are described as follows:

Synchronous mode In *synchronous execution*, all agents make decisions at the same discrete timestep, ensuring that they have access to the same updated state information before computing their respective strategies. The agents must then wait for each other to complete their computations before executing their strategies simultaneously.

Asynchronous mode In contrast, *asynchronous execution* allows agents to observe the environment, compute and execute strategies at their own independent rates, without waiting for other agents. This results in a setting that better captures real-time decision-making and response dynamics.

Additionally, large-scale testing is conducted across three different speed configurations, summarized in Table 6.1. These configurations define the maximum speeds for both the attacker and defender roles, ensuring a range of difficulty levels.

Table 6.1: Maximum speed configurations for the defender and attacker roles

Configuration	v_{defender} **(ms^{-1})**	v_{attacker} **(ms^{-1})**
Low Speed	1.0	2.0
Medium Speed	2.0	3.0
High Speed	4.0	5.0

6.1.1. Race tracks

The competition takes place on four distinct race tracks, each designed to introduce unique challenges in terms of layout complexity, gate configurations, and required maneuvering skills. The tracks are parametrized using periodic functions, specifically sine and cosine, to create smooth and dynamic racing paths. The parameters and mathematical formulations of the tracks are given in Appendix A. The race tracks shown in Figure 6.1 are constructed within the dimensions of the real-world flight cage, facilitating potential physical testing of the methods.

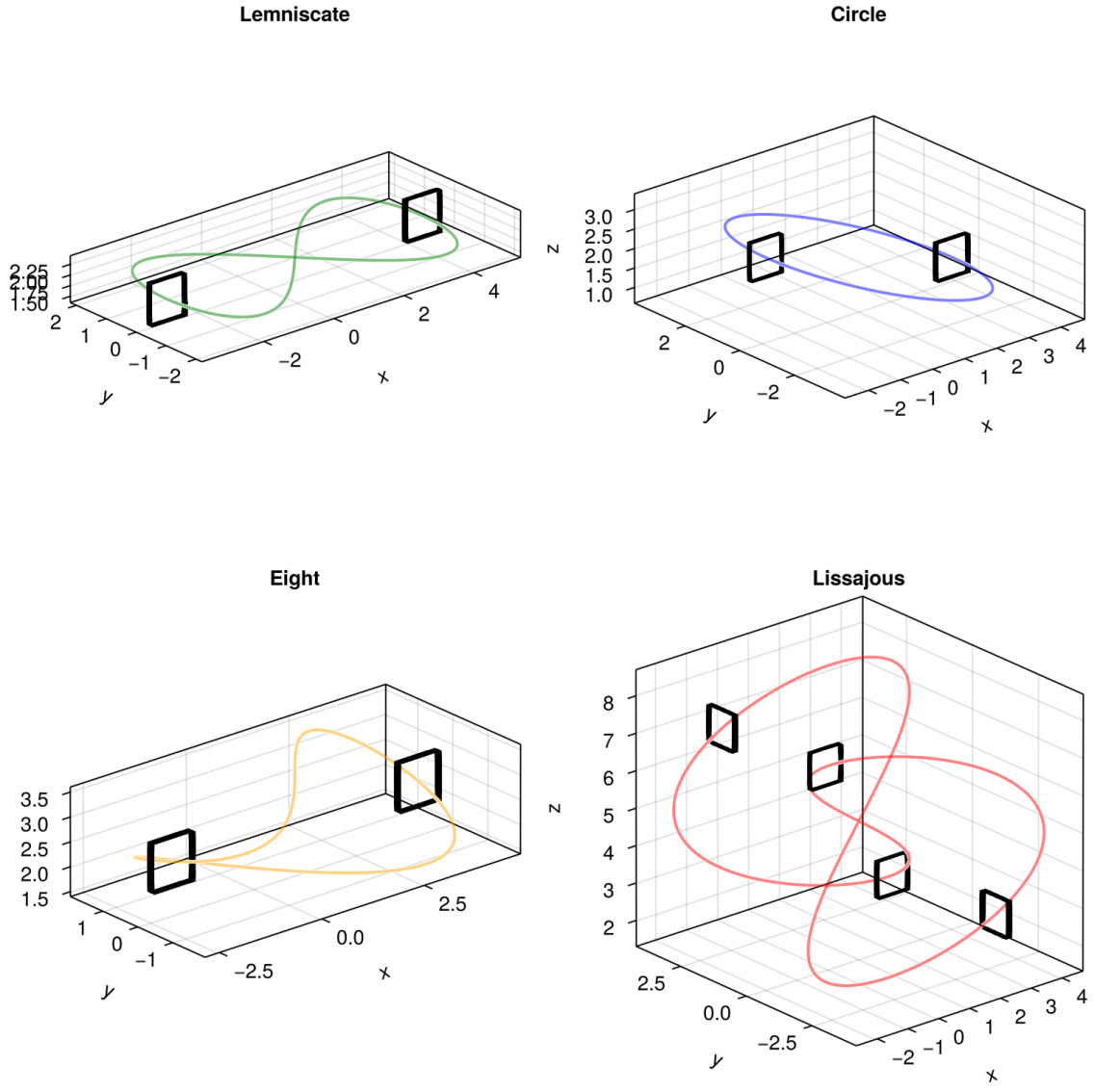


Figure 6.1: All four race tracks varying in size, complexity, length and gate configuration

6.1.2. Starting conditions

To avoid bias from fixed starting locations, we employ uniform sampling around predefined starting positions. These positions are set at 1 and 2.5 meters along the track length behind the start line and the sampling is performed within a spherical region of 0.15 meters in radius. Figure 6.2 shows uniform sampling for 10 initial conditions on the lemniscate track.

Moreover, each set of sampled initial positions is used for two races: in the first race, *Method 1* starts as the defender and *Method 2* as the attacker; in the second race, their roles are reversed, with *Method 2* defending and *Method 1* attacking. This setup ensures a fair and comprehensive evaluation of performance where both methods experience the same initial conditions.

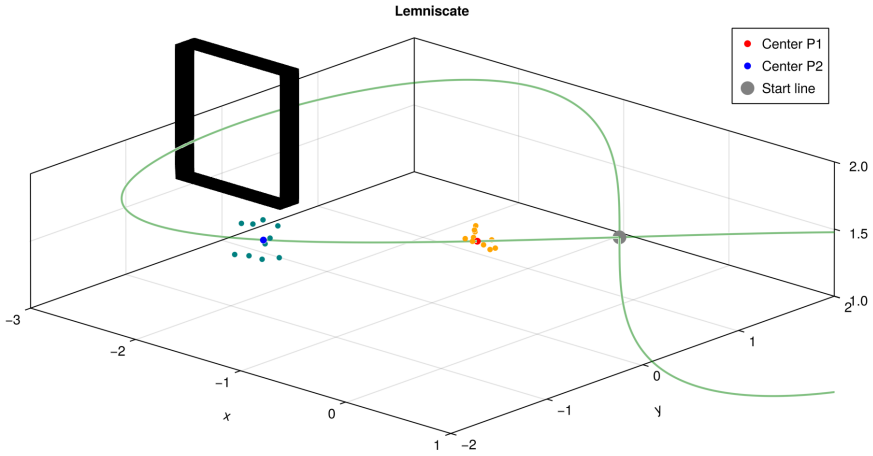


Figure 6.2: Uniform sampling of the initial starting positions for each race

6.1.3. Parameters

The referee parameters and experimental conditions are adjusted to reflect real-world limitations of the testing area and hardware. Collision thresholds, deviation limits, and velocity violation tolerances are set based on physical feasibility, and rules described in Table 4.1.

To promote strategic and competitive behavior, the shared cost function parameters are tuned to balance safety and aggressive racing behavior, particularly in asynchronous mode where decision-making is decentralized.

All methods share identical referee, environment and cost parameters unless explicitly stated otherwise. The complete list of parameters used for the large scale simulation is summarized in Table 6.2.

Table 6.2: Experimental setup racing parameters

Referee Parameters		**Solver Parameters**		**Cost Parameters**	
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Collision limit	$d_{ij} \leq 0.35$ m	Gate radius	0.875 m	q_l	3.0
Hard speed limit	$v \geq v_{\max} + 4.0$	Horizon	15	$q_c^{\max}$	3.0
Soft speed limit	$v \geq v_{\max} + 0.25$	Time step	50 ms	q_c^{nom}	1.5
Min speed limit	$v < 0.5$	Mass	0.6017 kg	$q_{\Delta v}, q_j$	0.001
Soft violation time	5.0 s	$a_{\max}$	10 ms^{-2}	q_{col}	1.5
Deviation limit	$d_{i,\text{track}} \geq 2.0$ m	$j_{\max}$	100 ms^{-3}	r_{col}	1.0
Overtake threshold ϵ_{ov}	0.75 m			q_{vel}	0.75
Max laps	5			μ	1.5
Defender $v_{\max}$	$1, 2, 4$ ms^{-1}			q_{block}	0.5
Attacker $v_{\max}$	$2, 3, 5$ ms^{-1}				

6.2. Evaluation metrics

To ensure a fair and structured assessment of the proposed autonomous racing methods, we employ a set of evaluation metrics that account for both competitive performance and rule adherence. These metrics are categorized into performance metrics, which determine racing success, and auxiliary metrics, which provide additional insights into method characteristics.

6.2.1. Performance metrics

Performance metrics measure which method performs best in direct competition. The win rate is a primary indicator, capturing how often a method secures victory based on the rules outlined in Table 4.1. Wins are further classified to distinguish between strategic superiority and rule enforcement:

- *Clean wins* - The race concludes after the maximum number of laps without violations. The winner is determined based on the time spent in front (i.e. defender role).
- *Collision wins* - The opponent triggers a collision violation by getting too close, awarding victory to the ego player.
- *Deviation wins* - The opponent strays too far from the track or fails to pass through a gate, resulting in a win for the ego player.
- *Velocity wins* - The opponent exceeds the speed limits assigned to their role, leading to a win for the ego player.

This classification highlights whether a method favors safer or more aggressive strategies.

6.2.2. Auxiliary metrics

These metrics provide a deeper understanding of method efficiency, rule compliance, and competitiveness:

- *Solve time* - The computational time required for the methods to compute a receeding horizon plan.
- *Number of violations* - The frequency of violations on each separate track separated by collisions, track deviations, or speed limit breaches.
- *Number of overtakes* - The total count of successful overtaking maneuvers.

6.3. Large scale simulation

To ensure fair comparisons, each method pair was tested across four different tracks, with ten distinct sampled starting conditions per track. Each sampled starting condition was used for two races – one with the original roles and one with swapped roles – to guarantee that both methods experienced the same initial conditions.

Overall, the large-scale experimental study includes 4 tracks, 10 starting configurations per track, and 2 races per starting configuration, resulting in a total of 80 races of 5 laps each per method-vs-method pair per speed configuration. This setup provides a comprehensive evaluation framework, ensuring that the tested strategies are assessed under diverse and realistic racing conditions.

7

Results

With the experimental setup and conditions established in the previous chapter, this chapter presents the results and discussion of the large-scale empirical study on head-to-head racing simulations conducted in an all-vs-all tournament format. The results are organized based on the execution mode – synchronous or asynchronous – and further categorized by speed configurations, as described in section 6.1. This structured analysis allows us to examine the interplay between different control strategies under varying conditions and highlight their strengths and weaknesses.

The key findings from this chapter include

1. MPG consistently maintains a competitive advantage over MPC in synchronous mode. This is evident across different speed configurations where MPG executes strategic overtakes and maintains a dominant racing position.
2. Induced delays and decentralized play reduces racing performance, particularly affecting MPG at higher speeds, which suffers from increased computational overhead.
3. By accelerating MPG via learning, we are able to achieve solve times comparable to MPC while maintaining its competitive edge in both synchronous and asynchronous modes.
4. The introduction of an extra blocking cost term enables strategic behaviors such as preventing overtakes by actively obstructing opponents.
5. The effectiveness of the proposed planners is demonstrated through high-fidelity simulation, where the combination of hierarchical planning and a nonlinear low-level controller results in successful overtaking maneuvers.

This chapter is structured to first present results under ideal conditions, where all agents operate synchronously, evaluating performance across different speed configurations and analyzing how the methods compare when all players have unlimited time to compute a solution. Next, the asynchronous mode is examined, investigating the effects of communication delays and solve-time differences on performance, as well as assessing how execution constraints influence competitive balance and overtaking dynamics. The chapter then explores extensions to the game-theoretic methods designed to enhance real-world racing performance, including the learning-based method (LMPG) that improves MPG's solve time and the integration of a blocking cost function (MPGB) to influence competitive interactions and defensive strategies. Finally, the proposed methods are validated in a hierarchical control setup, where a nonlinear MPC serves as the low-level controller within a high-fidelity simulation environment. This validation compares planned trajectories against actual drone performance, demonstrating the effectiveness of the approaches in executing successful overtaking maneuvers.

7.1. Synchronous racing

Synchronous racing provides insight into competitive performance under ideal conditions, where all agents operate with perfect state awareness and make decisions simultaneously. This means that

all agents compute their strategies at the same timestep and wait for each other to complete their computations before proceeding with executing the strategies.

7.1.1. Low speed

The low-speed configuration was selected to replicate the experimental conditions presented in [32], where both robots were constrained to a maximum velocity of either 0.3 ms^{-1} or 0.6 ms^{-1} depending on their initial starting positions. Maintaining the same ratio of maximum velocities between the attacker and defender, the results in Figure 7.1 show that MPG wins 100% of the races against MPC, regardless of its starting position.

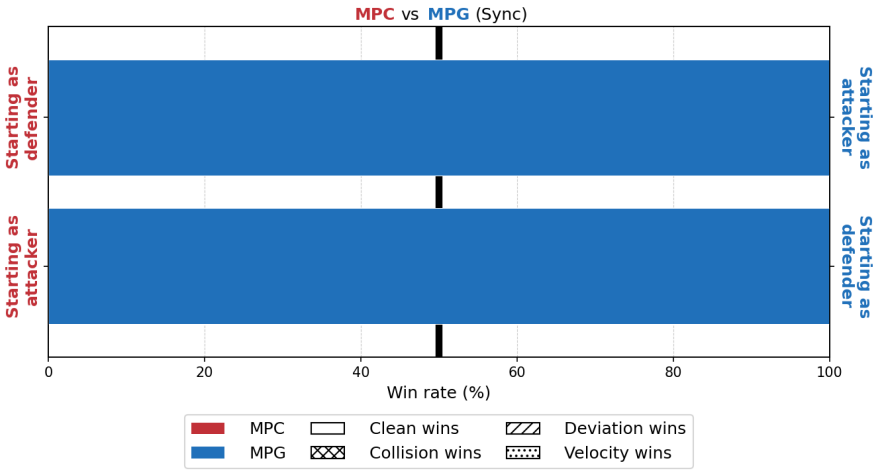


Figure 7.1: Performance results of MPC vs MPG at the **low speed** configuration in **synchronous** mode

The figure illustrates the win rate of both methods (MPC and MPG) across two different starting positions. Each horizontal bar represents a different case:

- The **top bar** corresponds to the scenario where MPC starts as the defender and MPG starts as the attacker.
- The **bottom bar** represents the reverse situation, where MPC starts as the attacker and MPG starts as the defender.

In both cases, MPG consistently secures victory, regardless of whether it begins ahead or behind its opponent. This dominance is visually evident as the bars are entirely filled with MPG's win color, indicating that MPC does not achieve any wins in these scenarios. Moreover, the breakdown of win types reveals that all of MPG's victories are categorized as clean wins, meaning that the opponent has not committed any violations. Additional details and a more granular performance analysis across all four race tracks are provided in Table 7.1.

Table 7.1: Breakdown of performance results of MPC vs MPG at **low speed** configuration in **synchronous** mode

Race track	**Wins**		**Violations**			**Overtakes**
	Start as attacker	Start as defender	Collision	Velocity	Deviation	Count
lemniscate	0 / 10	0 / 10	0 / 0	0 / 0	0 / 0	0 / 10
circle	0 / 10	0 / 10	0 / 0	0 / 0	0 / 0	0 / 10
lissajous	0 / 10	0 / 10	0 / 0	0 / 0	0 / 0	0 / 10
eight	0 / 10	0 / 10	0 / 0	0 / 0	0 / 0	0 / 10
all tracks	**0 (0) / 40 (40)**	**0 (0) / 40 (40)**	**0 / 0**	**0 / 0**	**0 / 0**	**0 / 40**

The table presents the results for all tracked metrics across both players in a structured manner for

clarity. Each cell reports values for Player 1 / Player 2. In the last row, the number of clean wins out of total wins as attacker and defender is recorded in parentheses. The results confirm that MPG successfully overtakes MPC in every race when starting as the attacker, while MPC is never able to overtake MPG. This is a significant finding given that both methods share the same cost function and racing parameters, highlighting the superior decision-making of MPG in strategic racing scenarios.

7.1.2. Medium speed

Figure 7.2 presents the performance comparison between MPC and MPG at medium speeds in synchronous mode. Similar to the low-speed case, MPC fails to secure any victories as an attacker, while MPG continues to win consistently across all race tracks. The performance of MPG remains superior, demonstrating the effectiveness of game-theoretic reasoning in strategic positioning and overtaking maneuvers.

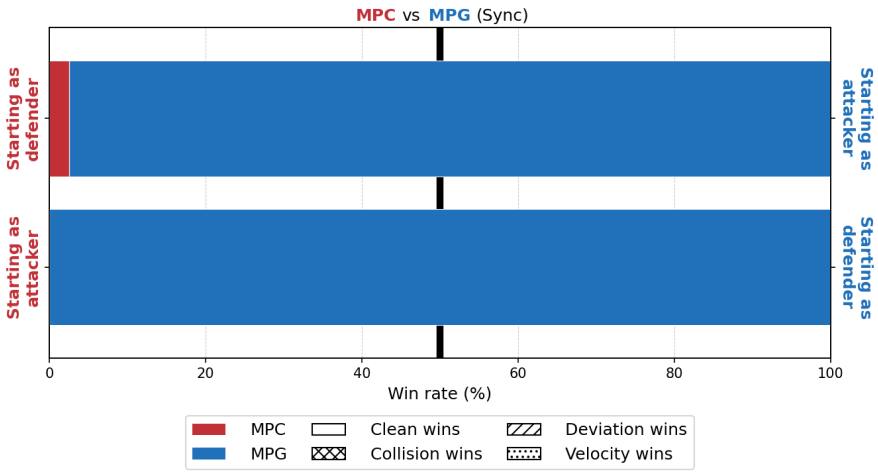


Figure 7.2: Performance results of MPC vs MPG at the **medium speed** configuration in **synchronous** mode

At medium speeds, the overtaking process becomes more gradual, particularly in tracks with complex curvature, and short gate to gate sections (lemniscate and eight tracks). The breakdown in Table 7.2 reveals that overtakes by MPG occur in every track. Notably, on the eight track, the plot indicates a singular instance where MPC won by time spent as a defender, and the breakdown reveals MPC successfully overtook MPG once in one of the 20 races. This anomaly can be attributed to the open-loop prediction error inherent in MPC's constant velocity model, which temporarily misrepresents the opponent's trajectory, allowing for an opportunistic overtake.

Table 7.2: Breakdown of performance results of MPC vs MPG at **medium speed** configuration in **synchronous** mode

Race track	**Wins**		**Violations**			**Overtakes**
	Start as attacker	Start as defender	Collision	Velocity	Deviation	Count
lemniscate	0 / 10	0 / 10	0 / 0	0 / 0	0 / 0	0 / 10
circle	0 / 10	0 / 10	0 / 0	0 / 0	0 / 0	0 / 10
lissajous	0 / 10	0 / 10	0 / 0	0 / 0	0 / 0	0 / 10
eight	0 / 9	1 / 10	0 / 0	0 / 0	0 / 0	1 / 11
all tracks	**0 (0) / 39 (39)**	**1 (1) / 40 (40)**	**0 / 0**	**0 / 0**	**0 / 0**	**1 / 41**

Despite this isolated occurrence, the overall results are consistent with the low-speed configuration. MPG retains its advantage due to its strategic decision-making, while MPC struggles to anticipate dynamic interactions effectively. These findings highlight the limitations of MPC's predictive assumptions. Nevertheless, both methods remain collision-free under the current collision avoidance settings, confirming the safety of the racing policies in this speed regime.

7.1.3. High speed

Increasing the maximum velocities further to match the high-speed configuration exposes critical weaknesses in the MPC baseline, primarily due to its reliance on the constant velocity assumption for opponent modeling. As shown in Figure 7.3, MPC consistently fails when starting as the attacker, resulting in frequent collisions leading to solely collision wins attributed for the opponent. This outcome suggests that at high speeds, the opponent's trajectory prediction is no longer accurately represented by the constant velocity assumption, leading to severe misjudgments in close-proximity racing scenarios.

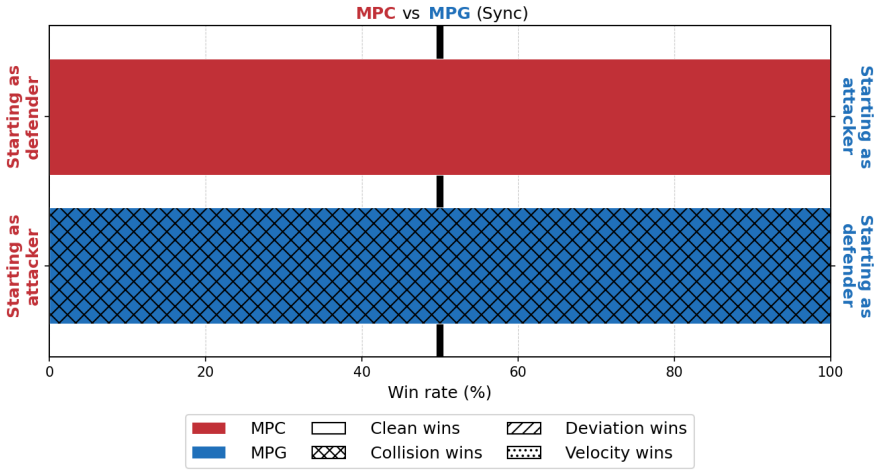
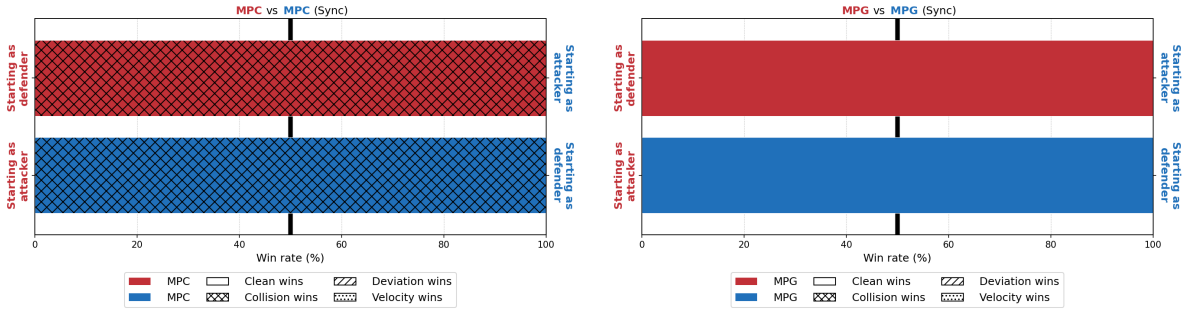


Figure 7.3: Performance results of MPC vs MPG at the **high speed** configuration in **synchronous** mode, MPC and MPG share the same collision weights

A deeper analysis is provided by examining self-play scenarios in Figure 7.4. The MPC vs. MPC race further confirms that MPC's opponent model becomes unreliable at high speeds, as both agents frequently crash when competing against each other. In contrast, MPG vs. MPG races remain stable, indicating that MPG inherently incorporates strategic collision avoidance. These results suggest that MPC, in its current form, requires additional tuning to remain viable under high-speed conditions.



(a) Performance of MPC vs MPC

(b) Performance of MPG vs MPG

Figure 7.4: Comparison of MPC and MPG playing against themselves at the **high speed** configuration in **synchronous** mode, where MPC and MPG share the same collision weights

To address this issue, the collision penalty weights for MPC were systematically increased until crashes were eliminated in self-play. After applying these adjustments, the modified MPC was tested against MPG, yielding the results shown in Table 7.3. Unlike previous configurations, both methods now exhibit equivalent performance, winning only when starting as defenders and never securing victories as attackers. Furthermore, neither method successfully overtakes the other, as reflected in the absence of recorded overtakes. This stagnation suggests that, at high speeds, overtaking opportunities are constrained by track design and the high cost of deviating from the optimal racing line.

Table 7.3: Breakdown of performance results of MPC vs MPG at **high speed** configuration in **synchronous** mode, after inflating the collision penalties for MPC

Race track	Wins		Violations			Overtakes
	Start as attacker	Start as defender	Collision	Velocity	Deviation	Count
lemniscate	0 / 0	10 / 10	0 / 0	0 / 0	0 / 0	0 / 0
circle	0 / 0	10 / 10	0 / 0	0 / 0	0 / 0	0 / 0
lissajous	0 / 0	10 / 10	0 / 0	0 / 0	0 / 0	0 / 0
eight	0 / 0	10 / 10	0 / 0	0 / 0	0 / 0	0 / 0
all tracks	**0 (0) / 0 (0)**	**40 (40) / 40 (40)**	**0 / 0**	**0 / 0**	**0 / 0**	**0 / 0**

To validate this hypothesis, two separate ablation studies were conducted: (i) halving the contouring weights to reduce the penalty for lateral deviations from the track centerline, and (ii) scaling all race tracks by a factor of two to provide more space for overtaking maneuvers. The number of wins and overtakes under these conditions is summarized in Table 7.4. The results reveal that halving the contouring weights significantly increases overtakes in the lissajous track, where its pre-existing wide straight sections and gentle curvature provide feasible overtaking zones. However, simply increasing track size does not yield additional overtakes, indicating that track width alone is insufficient to facilitate strategic overtakes without also reducing the constraint of lateral positioning.

Table 7.4: Comparison on wins and number of overtakes of MPC vs MPG at **high speed** configuration in **synchronous** mode, for varying track size and lowered contour weights separately

Race track	Wins as attacker		Wins as defender		Overtakes	
	Low contour	Scaled track	Low contour	Scaled track	Low contour	Scaled track
lemniscate	0 / 0	0 / 0	10 / 10	10 / 10	0 / 0	0 / 0
circle	0 / 0	0 / 0	10 / 10	10 / 10	0 / 0	0 / 0
lissajous	1 / 1	0 / 0	9 / 9	10 / 10	28 / 29	0 / 0
eight	0 / 0	0 / 0	10 / 10	10 / 10	0 / 0	0 / 0
all tracks	**1 / 1**	**0 / 0**	**39 / 39**	**40 / 40**	**28 / 29**	**0 / 0**

Combining both ablations—reducing contour weights and enlarging track size—restores MPG's dominance, as shown in Table 7.5. The results indicate that under these conditions, MPG consistently wins, securing clean victories with a significantly higher number of overtakes from both methods compared to the previous scenarios. This reinforces the conclusion that high-speed racing is fundamentally constrained by track design and lateral maneuverability, and that MPG's advantage is most evident when overtaking opportunities are structurally present.

Table 7.5: Breakdown of performance results of MPC vs MPG at **high speed** configuration in **synchronous** mode, with increased track size and lowered contour penalty at the same time

Race track	Wins		Violations			Overtakes
	Start as attacker	Start as defender	Collision	Velocity	Deviation	Count
lemniscate	0 / 9	1 / 10	0 / 0	0 / 0	2 / 0	60 / 61
circle	0 / 10	0 / 10	0 / 0	0 / 0	0 / 0	0 / 10
lissajous	0 / 10	0 / 10	0 / 0	0 / 0	8 / 0	138 / 143
eight	0 / 10	0 / 10	0 / 0	0 / 0	0 / 0	43 / 50
all tracks	**0 (0) / 39 (34)**	**1 (1) / 40 (35)**	**0 / 0**	**0 / 0**	**10 / 0**	**241 / 264**

These findings highlight the necessity of incorporating adaptable opponent models in MPC to maintain competitiveness at high speeds. While cost tuning can mitigate crashes, the lack of strategic reasoning

in overtaking situations remains a fundamental limitation. In contrast, MPG continues to demonstrate superior adaptability, making it a more reliable approach for high-speed competitive racing scenarios under synchronous conditions.

7.2. Asynchronous racing

In contrast to the synchronous mode, asynchronous execution allows agents to process and compute strategies independently. This means that agents no longer wait for each other to complete their computations; instead, they solve and communicate their strategies asynchronously. Furthermore, both methods utilize the latest state updates of each player to compute a strategy. Because of the asynchronous nature of this approach and the communication delays it introduces, the initial states observed by the optimizers no longer fully represent the true state of the game.

7.2.1. Low speed

At low speeds, the transition from synchronous to asynchronous execution introduces new competitive dynamics, but the overall trend remains consistent – MPG continues to outperform MPC. As seen in Figure 7.5, MPG still wins the majority of races, indicating that the core advantages of its game-theoretic approach transfer effectively to asynchronous conditions at low speeds. However, a key difference from the synchronous case is the increased number of victories attributed to MPC, primarily due to deviation and velocity violations committed by MPG.

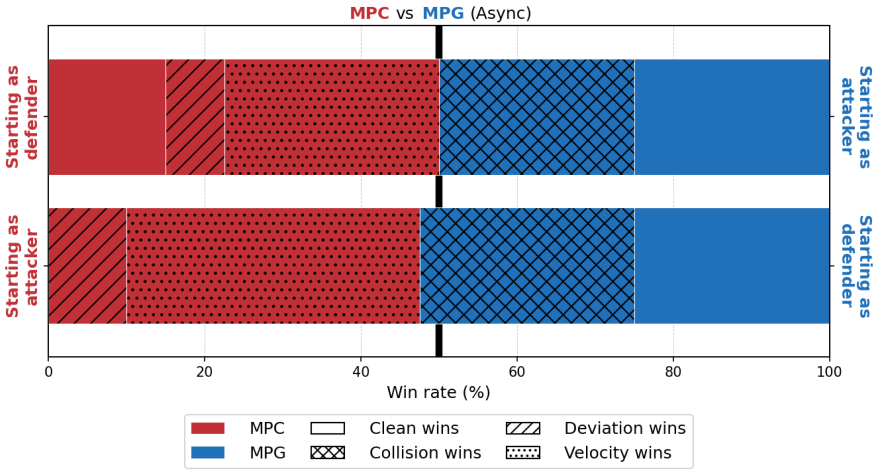


Figure 7.5: Performance results of MPC vs MPG at the **low speed** configuration in **asynchronous** mode

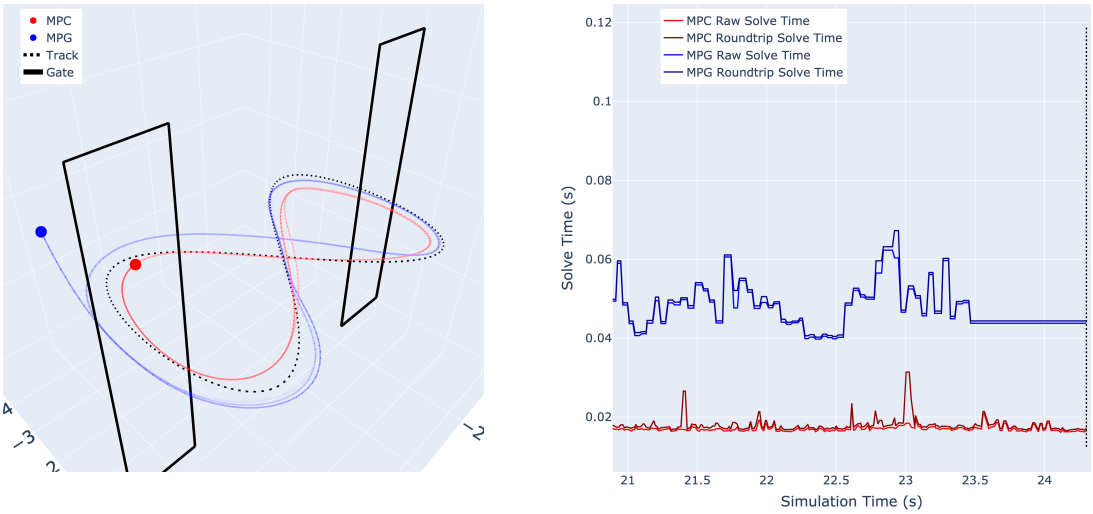
The breakdown in Table 7.6 confirms that these violations play a crucial role in shifting race outcomes.

Table 7.6: Breakdown of performance results of MPC vs MPG at **low speed** configuration in **asynchronous** mode

Race track	Wins		Violations			Overtakes
	Start as attacker	Start as defender	Collision	Velocity	Deviation	Count
lemniscate	10 / 1	9 / 0	1 / 0	0 / 15	0 / 4	31 / 37
circle	8 / 0	10 / 2	2 / 0	0 / 12	0 / 2	90 / 96
lissajous	1 / 9	1 / 9	18 / 0	0 / 1	0 / 1	81 / 90
eight	0 / 10	0 / 10	0 / 0	0 / 0	0 / 0	97 / 100
all tracks	**19 (0) / 20 (10)**	**20 (6) / 21 (10)**	**21 / 0**	**0 / 28**	**0 / 7**	**299 / 323**

A deeper analysis of these violations reveals that MPG's computational delays are a primary contributing factor. Figure 7.6 illustrates a representative failure case on the lemniscate track. In Figure 7.6a, MPG drifts off the track and incurs a velocity violation, leading to a loss. The corresponding solve time

analysis in Figure 7.6b reveals a critical issue: MPG experiences a prolonged computational delay, during which it fails to generate a new control solution. This is evident in the flat solve time profile, where for approximately one second, MPG executes its last computed trajectory without updating its strategy. As a result, it is unable to correct its course, ultimately violating track constraints.



(a) Trajectories of MPC vs MPG

(b) Solve time of MPC vs MPG

Figure 7.6: Velocity and deviation violation as a consequence of the long solve times of MPG during an overtake maneuver

This computational bottleneck becomes especially problematic during overtaking events, where the optimization complexity increases. The solve time distributions in Figure 7.7 highlight this discrepancy – MPC achieves a median solve time of 17ms, whereas MPG's median solve time is significantly higher at 47ms. This translates to an average reaction-time advantage of 30ms for MPC, allowing it to make decisions more frequently and adapt to the evolving race dynamics more effectively. More critically, MPG exhibits outliers with solve times reaching up to 2 seconds, whereas the highest observed solve time for MPC is 0.5 seconds. These observations are based on a sample size of $15,000$ data points, collected across all MPC vs. MPG races.

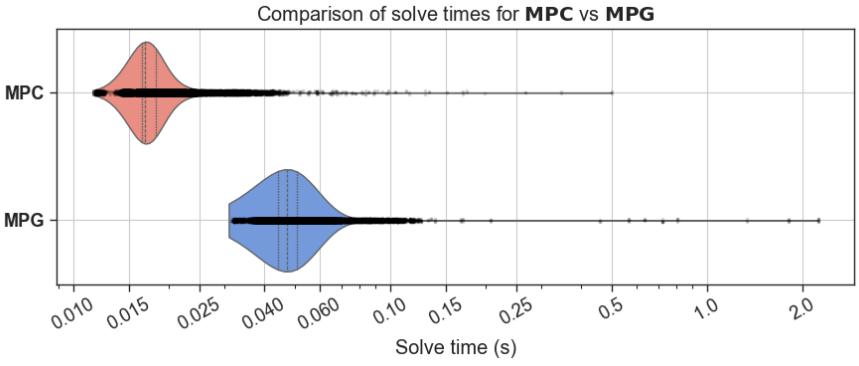


Figure 7.7: Distribution of solve times for MPC and MPG, highlighting the difference in median solve times and outliers

Another apparent feature of asynchronous execution is the emergence of overtaking opportunities for MPC. Unlike in the synchronous case, where MPC was unable to overtake MPG, the asynchronous setup introduces state update mismatches that create new openings. Because state communication occurs independently for each agent, MPC can sometimes compute its trajectory based on outdated

opponent state information. This, combined with its faster reaction time, enables MPC to detect and benefit on overtaking opportunities that were unavailable in synchronous execution. The effect is clearly visible in the overtaking statistics from Table 7.6, where MPC achieves a substantial number of overtakes.

7.2.2. Medium speed

At medium speeds, asynchronous execution intensifies the impact of computational delays on racing dynamics, significantly influencing competitive outcomes. Figure 7.8 illustrates the performance comparison between MPC and MPG, revealing a stark contrast to the synchronous case. Unlike in the low-speed asynchronous scenario, MPC now consistently secures more victories due to a combination of clean wins and opponent violations.

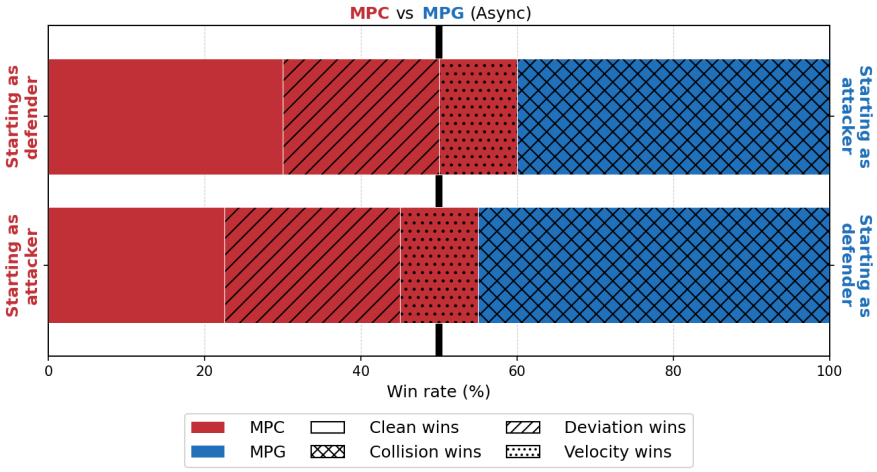


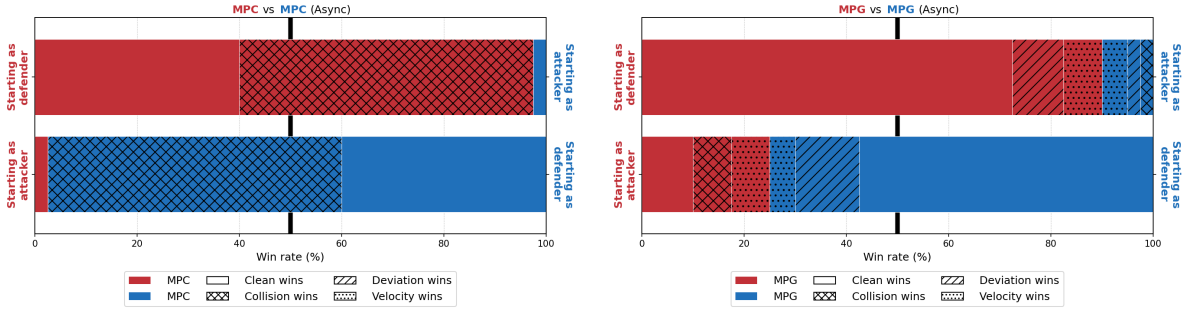
Figure 7.8: Performance results of MPC vs MPG at the **medium speed** configuration in **asynchronous** mode

The breakdown in Table 7.7 confirms that MPG suffers frequent velocity and deviation violations, primarily caused by excessive solve times that prevent timely replanning.

Table 7.7: Breakdown of performance results of MPC vs MPG at **medium speed** configuration in **asynchronous** mode

Race track	Wins		Violations			Overtakes
	Start as attacker	Start as defender	Collision	Velocity	Deviation	Count
lemniscate	8 / 2	8 / 2	4 / 0	0 / 0	0 / 16	38 / 32
circle	10 / 0	10 / 0	0 / 0	0 / 6	0 / 1	74 / 77
lissajous	3 / 5	5 / 7	12 / 0	0 / 2	0 / 0	141 / 149
eight	1 / 9	1 / 9	18 / 0	0 / 0	0 / 0	37 / 45
all tracks	**22 (9) / 16 (0)**	**24 (12) / 18 (0)**	**34 / 0**	**0 / 8**	**0 / 17**	**290 / 303**

A key observation is that MPG still wins a substantial number of races through collision-based outcomes, suggesting that while asynchronous effects degrade its performance, its game-theoretic framework continues to enforce perfect collision avoidance against MPC. However, a closer look at the self-play comparisons in Figure 7.9 provides additional insight. The MPC vs. MPC matchup exhibits a high number of collision violations, particularly on the lemniscate and eight tracks, where sharp turns necessitate rapid adjustments. This observation suggests that MPC's reliance on a constant velocity opponent model becomes even more problematic under asynchronous conditions. In contrast, MPG vs. MPG races remain safer, highlighting the benefit of its strategic opponent reasoning.



(a) Performance of MPC vs MPC

(b) Performance of MPG vs MPG

Figure 7.9: Comparison of MPC and MPG playing against themselves at the **medium speed** configuration in **asynchronous** mode

Examining the underlying cause of the high collision rates in MPC vs. MPC races from Table 7.8, we observe that in tracks with high-curvature sections, such as the lemniscate and eight tracks, MPC frequently mispredicts the opponent's future trajectory. Instead of anticipating a sharp turn, it assumes a straight-line continuation, leading to direct collisions.

Table 7.8: Breakdown of performance results of MPC vs MPC at **medium speed** configuration in **asynchronous** mode

Race track	Wins		Violations			Overtakes
	Start as attacker	Start as defender	Collision	Velocity	Deviation	Count
lemniscate	0 / 0	10 / 10	10 / 10	0 / 0	0 / 0	0 / 0
circle	0 / 0	10 / 10	0 / 0	0 / 0	0 / 0	102 / 100
lissajous	1 / 1	9 / 9	3 / 3	0 / 0	0 / 0	257 / 253
eight	0 / 0	10 / 10	10 / 10	0 / 0	0 / 0	0 / 0
all tracks	**1 (1) / 1 (1)**	**39 (16) / 39 (16)**	**23 / 23**	**0 / 0**	**0 / 0**	**359 / 353**

This phenomenon, absent in the synchronous case, confirms that the combined effects of asynchronous state updates and open-loop constant velocity assumptions degrade MPC's predictive reliability. Thus, while asynchronous conditions improve MPC's relative competitiveness against MPG, they also introduce fundamental weaknesses in its trajectory planning under dynamic race conditions.

7.2.3. High speed

At high speeds, the negative impact of asynchronous effects becomes even more pronounced. Figure 7.10 presents the results for the high-speed configuration under asynchronous conditions, using the standard track size and contour weights, and incorporating inflated collision penalties for MPC to mitigate the crashes observed in synchronous mode. Unlike the synchronous case, where neither method could reliably overtake the other at high speeds, due to the constraints imposed by high contouring costs and limited track space, the asynchronous configuration allows MPC to exploit communication and computational delays in MPG's strategy updates.

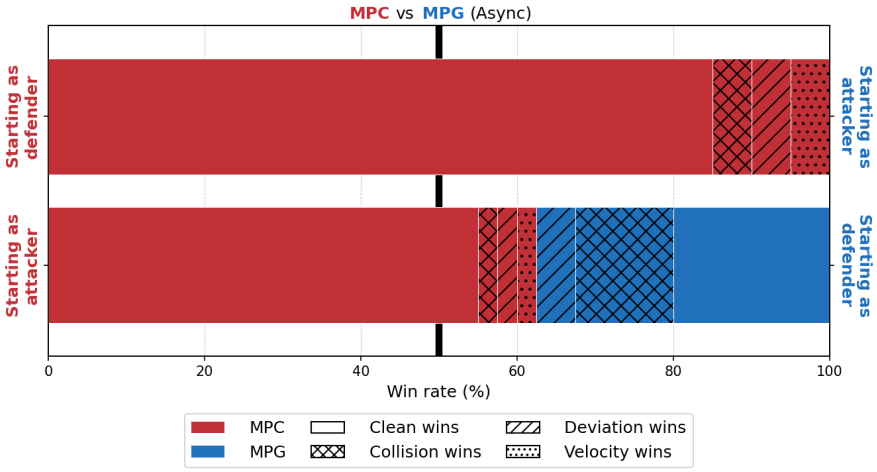


Figure 7.10: Performance results of MPC vs MPG at the **high speed** configuration in **asynchronous** mode, after inflating the collision penalties for MPC

This is evident in the overtaking counts in Table 7.9, which show that MPC frequently overtakes MPG across all race tracks, while MPG fails to act on similar opportunities. Despite the presence of asynchronous effects, MPG's behavior is closer to its synchronous counterpart.

Table 7.9: Breakdown of performance results of MPC vs MPG at **high speed** configuration in **asynchronous** mode, after inflating the collision penalties for MPC

Race track	Wins		Violations			Overtakes
	Start as attacker	Start as defender	Collision	Velocity	Deviation	Count
lemniscate	0 / 0	10 / 10	2 / 1	0 / 0	0 / 0	3 / 0
circle	10 / 0	10 / 0	0 / 0	0 / 3	0 / 0	19 / 18
lissajous	10 / 0	10 / 0	0 / 2	0 / 0	0 / 2	68 / 60
eight	5 / 0	10 / 5	3 / 0	0 / 0	2 / 1	5 / 0
all tracks	**25 (22) / 0 (0)**	**40 (34) / 15 (8)**	**5 / 3**	**0 / 3**	**2 / 3**	**95 / 78**

Overall, these findings highlight a critical limitation of game-theoretic planning in high-speed asynchronous settings. While MPG remains superior in idealized synchronous conditions, its computational overhead becomes a severe bottleneck when real-time adaptability is required. In contrast, MPC, despite its simplistic opponent modeling, benefits from its lower computational demands, enabling it to exploit asynchronous update delays and secure victories through faster reaction times. These results underscore the necessity of optimizing solve times for game-theoretic methods to ensure their viability in real-time, high-speed racing scenarios.

7.3. Extensions

In the previous section, results demonstrated that high-speed racing is fundamentally constrained by track design and lateral maneuverability. Since scaled-up track dimensions are impractical in real-world scenarios, the following extensions to the game planners were tested on standard track sizes at medium speeds. This configuration represents the fastest setting that still produces interesting racing behaviors while remaining feasible for real-world implementation on hardware.

7.3.1. Accelerating game planners

In this extension, we evaluate how accelerating game planners impact the competitive dynamics between different planning methods. We compare the performance of MPC, MPG, and LMPG under both

synchronous and asynchronous execution modes on the lemniscate track at medium speeds. Lifted game planners are designed to mitigate the long computation times observed with MPG, thereby reducing the delays that hinder real-time racing. As shown in the solve time analysis of Figure 7.11, LMPG now achieves a median solve time of 15ms, surpassing even MPC, while the maximum solve times are reduced to only 40ms. This suggests that accelerating the planning process levels the playing field in terms of reaction time.

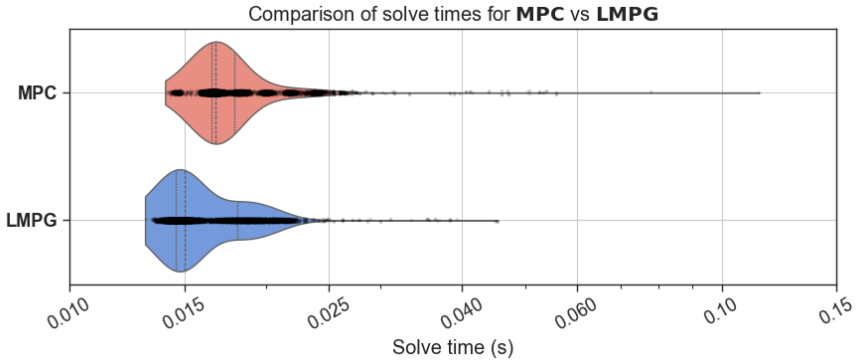
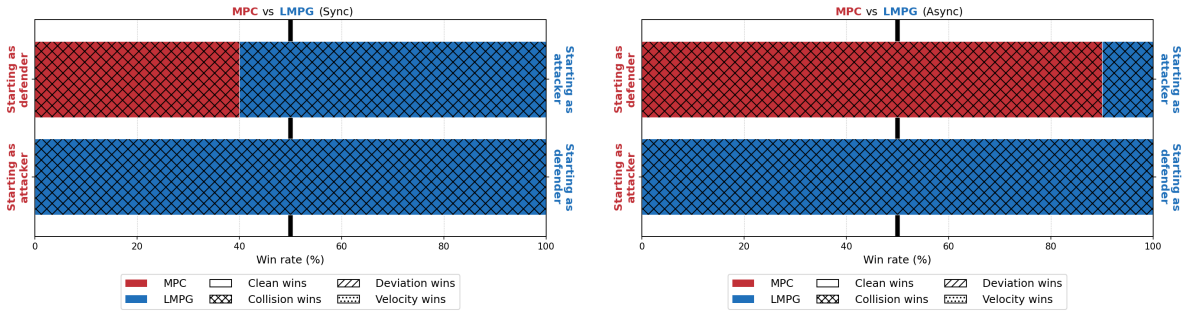


Figure 7.11: Distribution of solve times for MPC and LMPG, highlighting the difference in median solve times and outliers

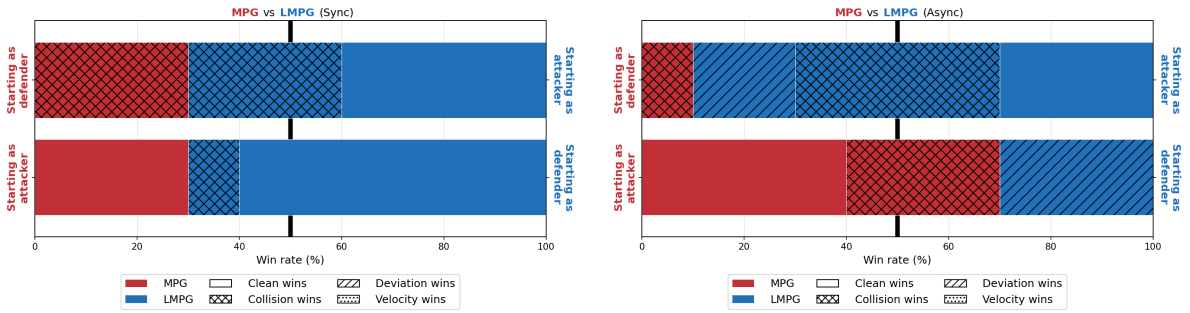
The experimental results show that LMPG consistently outperforms both MPC and MPG in terms of clean wins, primarily because it was trained through self-play and fine-tuned specifically against MPG. Figure 7.12 and Figure 7.13b provide a detailed comparison of performance across synchronous and asynchronous execution modes. Notably, LMPG maintains strong performance even under asynchronous conditions, where previously, game-theoretic planners were at a disadvantage.



(a) Synchronous performance

(b) Asynchronous performance

Figure 7.12: Performance results of MPC vs LMPG at the **medium speed** configuration on the lemniscate track



(a) Synchronous performance

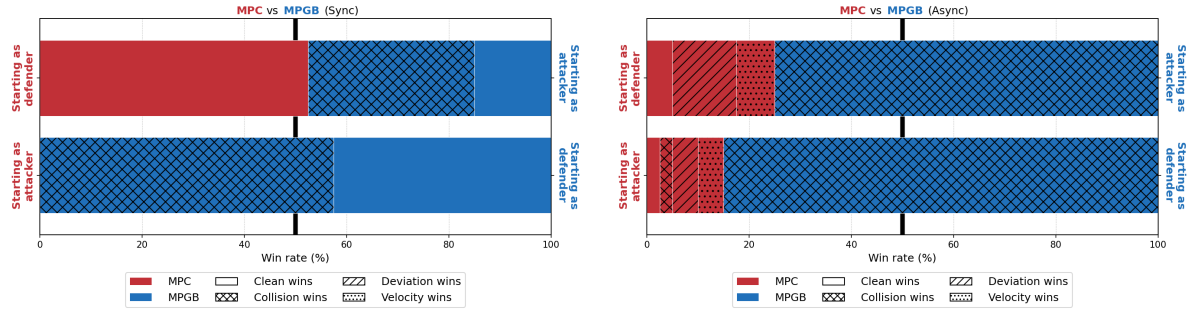
(b) Asynchronous performance

Figure 7.13: Performance results of MPG vs LMPG at the **medium speed** configuration on the lemniscate track

However, the increased performance of LMPG comes at a cost. When competing against MPC, all wins observed are based on collision violations. This phenomenon occurs because the trained solver has learned to execute maneuvers – such as sudden deceleration, that cannot be accurately anticipated by MPC's simplistic constant velocity model. A second reason behind the large number of collisions can be attributed to the limited training with the MPG method. Although similar behaviors are observed when LMPG faces MPG, the effect is less pronounced, likely due to MPG's better reasoning model of the opponent. In essence, accelerating game planners provide a clear competitive edge by reducing solve times, while still maintaining strategic decision-making advantage.

7.3.2. Strategic blocking

In this extension, an additional blocking cost term is incorporated into MPG to encourage defensive maneuvers. This term induces behaviors where an agent intentionally decelerates and alters its heading to obstruct an opponent's overtaking maneuver. When playing as the defender, the blocking-enabled planner (referred to as MPGB) actively positions itself to force collisions as the attacker attempts to pass. For instance, in the race against MPC, as illustrated in Figure 7.14, MPGB's defensive maneuvers result in a high number of collisions committed by MPC, highlighting the strategic advantage provided by blocking.

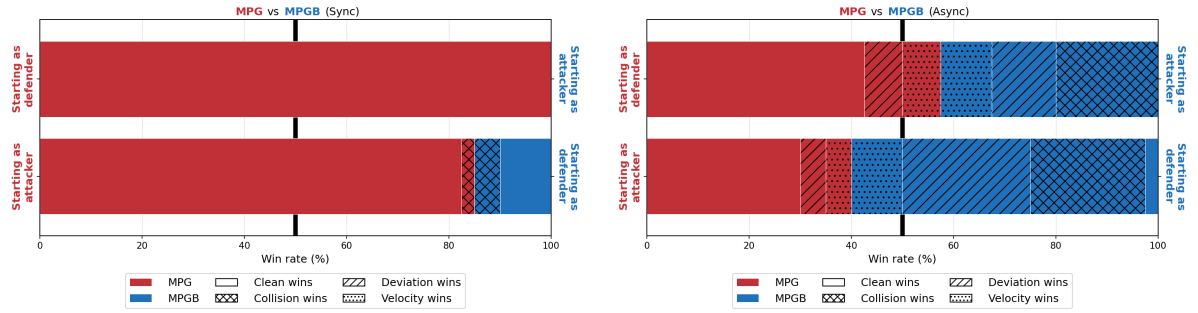


(a) Synchronous performance

(b) Asynchronous performance

Figure 7.14: Performance results of MPC vs MPGB at the **medium speed** configuration

Figure 7.15 shows the results of MPG vs MPGB for both synchronous and asynchronous cases. In synchronous races against MPG, however, MPGB's ability to force a collision is less consistent. Despite MPG's lack of explicit strategic knowledge regarding opponent blocking, its more sophisticated reasoning model allows it to evade collisions even when being blocked, and successfully overtake its opponent.



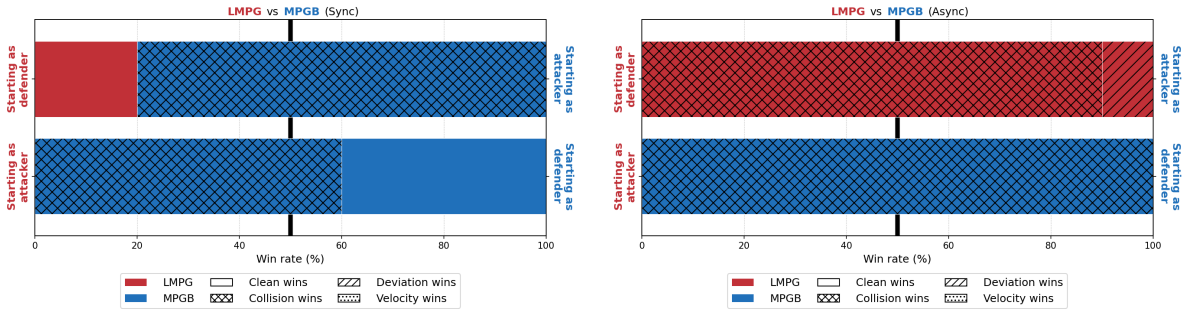
(a) Synchronous performance

(b) Asynchronous performance

Figure 7.15: Performance results of MPG vs MPGB at the **medium speed** configuration

In Figure 7.16, LMPG has not been explicitly trained against blocking strategies. The blocking method consistently wins by triggering collisions, as the opponent encounters states outside its training distribution. In asynchronous mode, both methods suffer from asynchronous effects discussed in the previous

sections.



(a) Synchronous performance

(b) Asynchronous performance

Figure 7.16: Performance results of LMPG vs MPGB at the **medium speed** configuration on the lemniscate track

Overall, the strategic blocking extension reveals that while the addition of a blocking cost can successfully disrupt overtaking attempts, the effectiveness of the maneuver depends critically on both the execution mode and the sophistication of the opponent's reasoning model.

7.4. High fidelity simulation

While the large-scale experiments presented thus far have relied on simplified point-mass dynamics inside the flight simulator, the ultimate goal is to ensure that the game-theoretic planners can be reliably deployed in real-world scenarios. To this end, a high fidelity simulation was conducted in which the developed planners function as the high-level decision maker, coupled with a nonlinear MPC (NMPC) as the lower-level controller that handles the full drone dynamics. This architecture is presented in the following Chapter 8.

In the high fidelity simulation, we evaluated head-to-head races between MPC and MPG on the lemniscate track under medium-speed conditions, in asynchronous mode. As shown in Figure 7.17, one scenario highlights MPC successfully overtaking MPG, whereas Figure 7.18 depicts the reverse scenario with MPG overtaking MPC. The race dynamics are more intricate in the high fidelity setup. Both planners exchange overtaking maneuvers throughout the race, ultimately resulting in each method achieving successful overtakes.

Notably, compared to the earlier experiments using a point-mass model (such as those presented in Figure 7.6a), the high fidelity simulation exhibits larger deviations from the track. These deviations are attributable to the complex nonlinearities of full drone dynamics, which introduce additional challenges not captured by the simpler model. Despite these challenges, the results validate that the game-theoretic planners, when integrated with a robust low-level NMPC controller, can generate competitive and reliable high-level strategies that translate effectively into physical overtaking maneuvers. This integration thus demonstrates the practical viability of the proposed hierarchical control approach and system architecture for real-world drone racing.

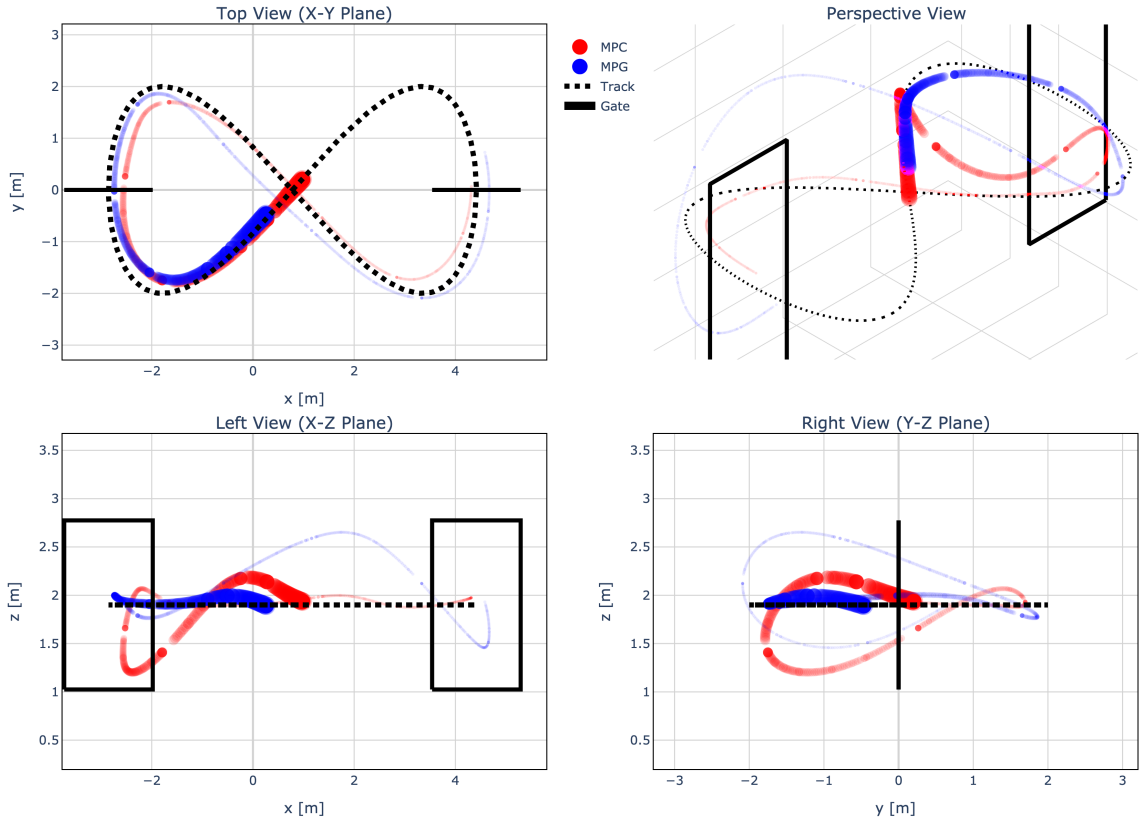


Figure 7.17: High fidelity simulation of MPC overtaking MPG at the **medium speed** configuration on the lemniscate track

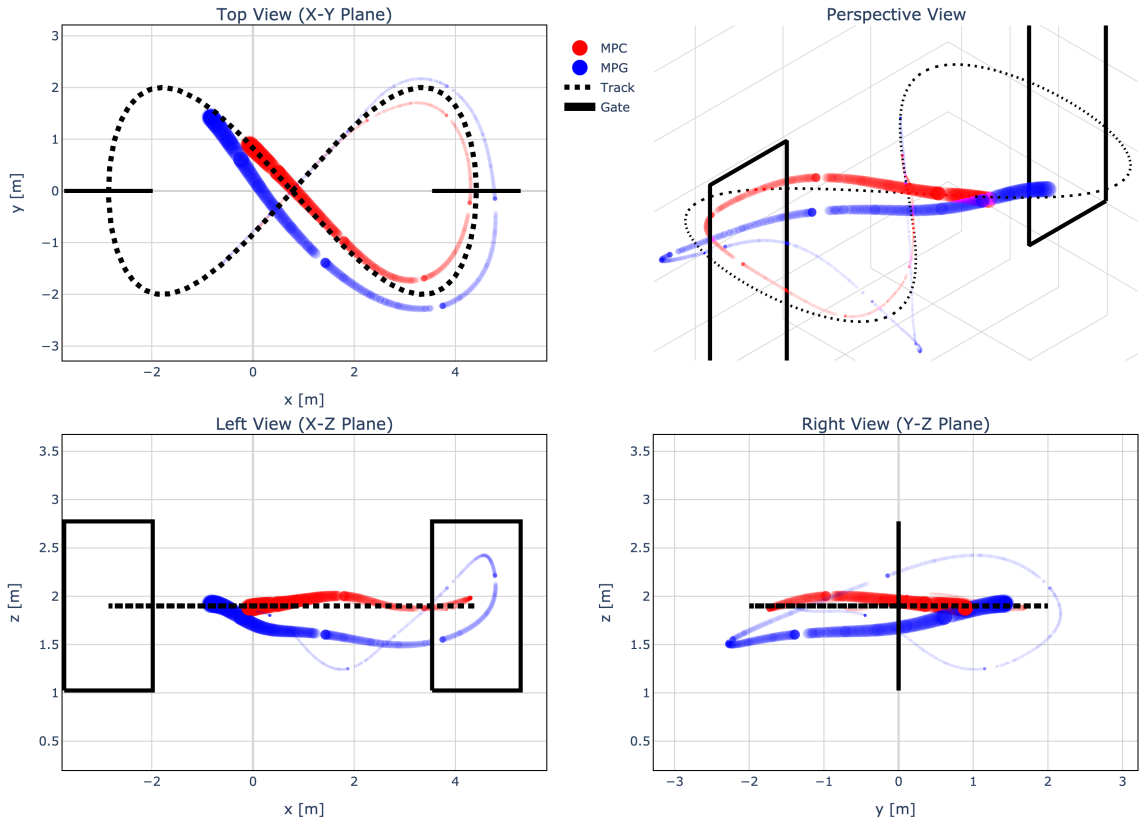


Figure 7.18: High fidelity simulation of MPG overtaking MPC at the **medium speed** configuration on the lemniscate track

8

Real-world experiments

The results obtained in simulation provide valuable insights into the effectiveness of different control strategies for multi-agent racing. However, real-world experiments are essential to validate these findings under practical conditions. In this chapter, we present a series of hardware experiments conducted with quadrotors to evaluate how well the proposed methods transfer to real-world scenarios.

8.1. Hardware setup

The system architecture for real-life racing is illustrated in Figure 8.1. It closely follows the setup used in point mass simulations, with the key difference being the use of real hardware instead of a flight simulator. Additionally, the architecture supports the high fidelity RotorS [10] flight simulator as an alternative to the custom-built point mass simulator, allowing for more realistic quadrotor dynamics in simulation-based experiments.

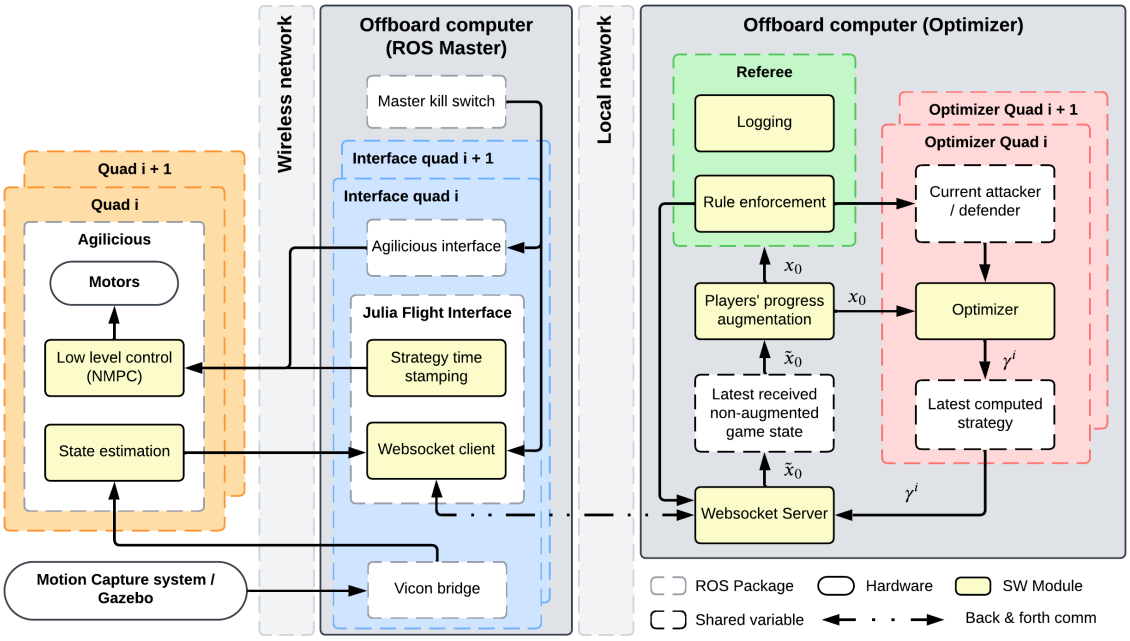


Figure 8.1: Illustration of the system architecture used for real-world autonomous drone racing experiments.

An offboard computer runs the optimizers for each drone and communicates with another computer

running a custom-built ROS node called Julia Flight Interface. This ROS node interacts with the optimizer server via WebSockets. To minimize communication delays, these two offboard computers are connected through a wired local network.

The decision to separate the optimizers from the ROS master computer is motivated by the need to reduce the computational load on the optimizer processes. Specifically, this prevents memory exhaustion on the optimizer computer and avoids overloading the CPU with additional tasks.

The ROS master serves as the central control unit, forwarding strategies to the drones via WiFi and interfacing with them through Agilicious [7]. Agilicious is a software framework designed for agile quadrotor flight and includes an implementation of a nonlinear model predictive controller (NMPC), which we use as a low-level controller.

The Julia Flight Interface, running on the ROS master, timestamps the strategies received from the optimizer and transmits the corresponding point mass plan to the NMPC controller. Each drone runs an Agilicious pilot, which includes the NMPC controller and performs state estimation using onboard IMU data combined with real-time position and attitude from a motion capture system. Additionally, each drone is connected to its respective Julia Flight Interface, ensuring continuous communication with the optimizer server.

The hardware experiments were conducted at the Autonomous Multi-Robots Lab at Delft University of Technology using a custom-built micro aerial vehicle (MAV) called Falcon. The MAV is equipped with a Raspberry Pi 5 onboard computer and relies on a VICON motion capture system for localization, which tracks reflective markers attached to the drones.

The experimental setup, corresponding to the system architecture, is depicted in Figure 8.2.

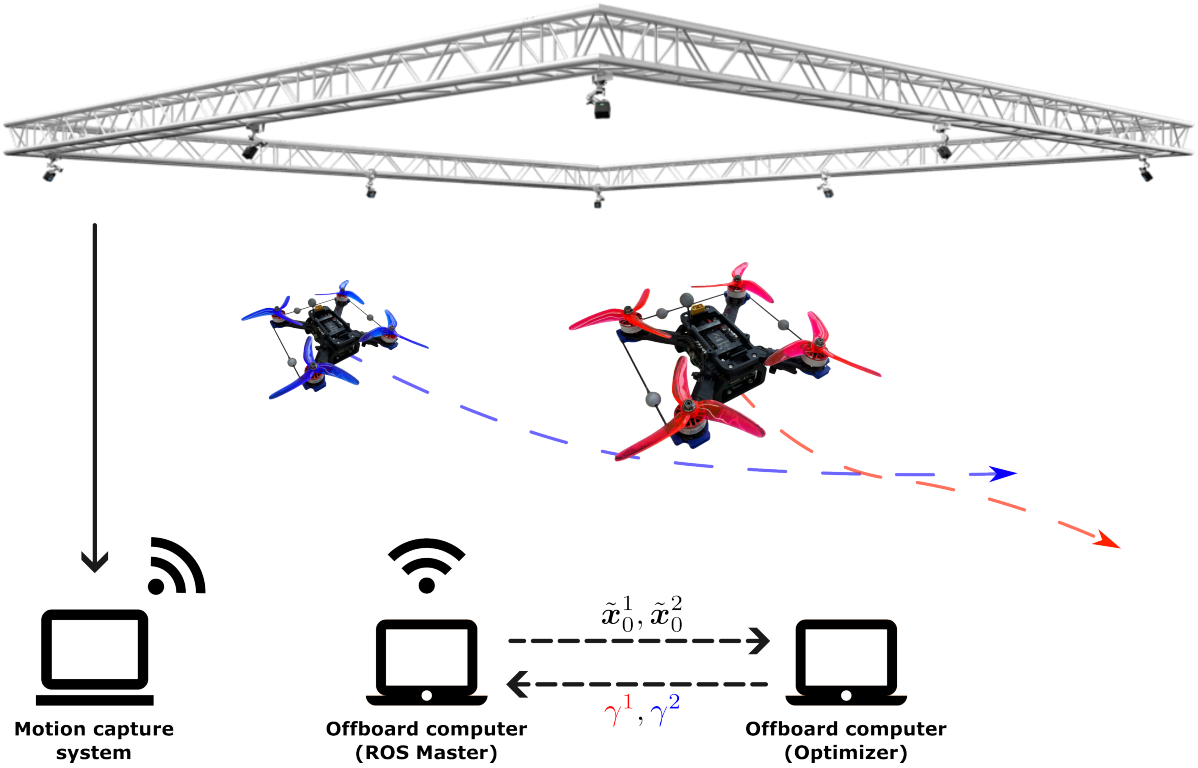


Figure 8.2: Overview of the experimental setup used for real-world quadrotor racing. The drones' position and attitude are captured by VICON and the two offboard computers exchange current states $\tilde{x}_0^i$ and strategies γ^i, to be executed by the quads.

8.2. Qualitative results

Before conducting live experiments, we first validate the system's behavior in the RotorS simulator, which is supported by Agilicious. This step helps identify potential issues and refine the racing strategies before deploying them on real hardware.

During validation in RotorS, we frequently observed that races ended in a "collision win." As discussed in the previous chapter, MPC often leads to collisions on the lemniscate track due to inaccurate predictions of opponent behavior, particularly in high-curvature regions. However, these effects were observed to be even more prominent in the high fidelity simulations.

To ensure the safety of both the quadrotors and human operators, we increased the collision avoidance radius used for MPC, as specified in Table 6.2. Specifically, we set $r_{\text{col}} = 1.25$ for the medium-speed configuration and $r_{\text{col}} = 1.5$ for the high-speed configuration.

Additionally, simulation testing revealed that LMPG was not sufficiently safe for real-world deployment. This suggests that LMPG requires further training and tuning to generalize effectively to nonlinear dynamics, particularly when coupled with the low-level nonlinear controller. As a result, the following experiments focus exclusively on races between MPC and MPG.

We conducted two speed configurations for the MPC vs. MPG comparison. The drones raced for five laps on the lemniscate track, navigating through designated gates.

Medium speed Figure 8.3 and Figure 8.4 illustrate MPG overtaking MPC. Throughout the race, both methods continue to overtake each other repeatedly. At this speed configuration, both methods are competitive however, MPC manages have a slightly faster overtakes, and wins the race by time spent as defender.

High speed Figure 8.5 and Figure 8.6 demonstrate MPC overtaking MPG in the high-speed configuration. Once MPC takes the lead, MPG is unable to regain its position and ultimately loses the race.

These findings align with the results from the previous chapter. At both medium and high speeds, MPC outperforms MPG due to its superior reaction time, which stems from a lower solve time.

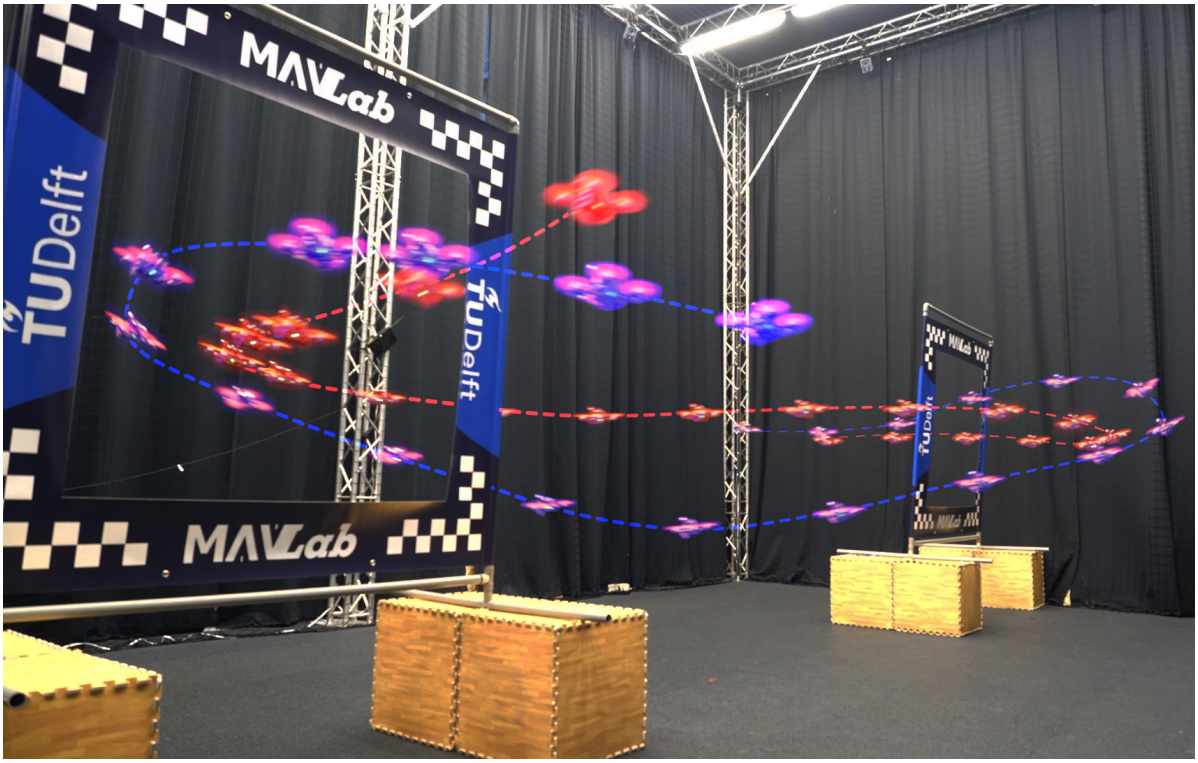


Figure 8.3: Real flight validation of MPG overtaking MPC at the **medium speed** configuration on the lemniscate track (Perspective view)

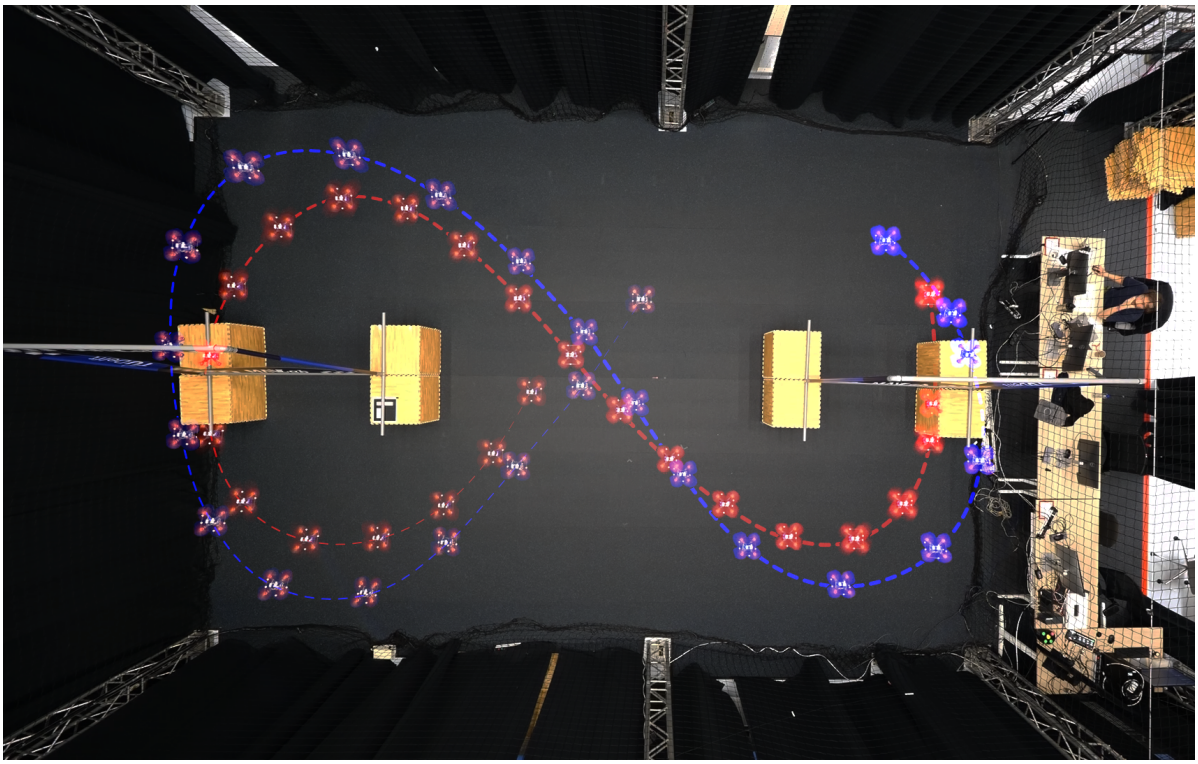


Figure 8.4: Real flight validation of MPG overtaking MPC at the **medium speed** configuration on the lemniscate track (Top view)

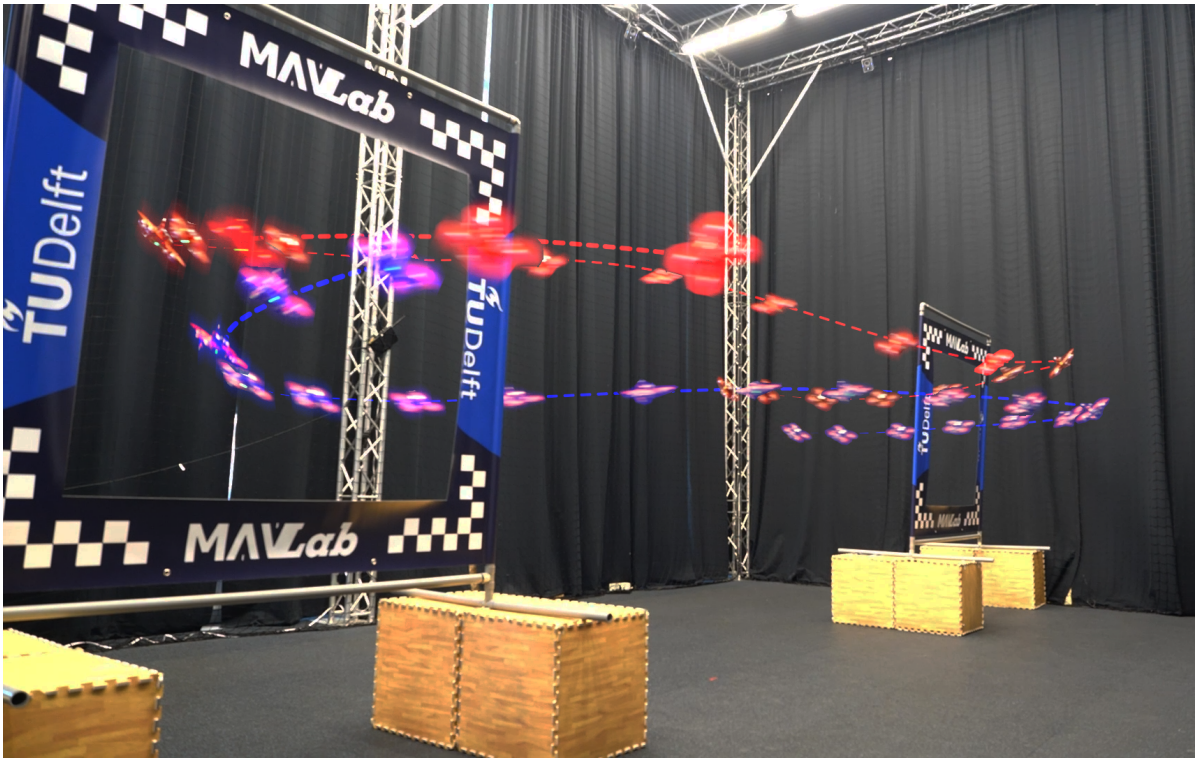


Figure 8.5: Real flight validation of MPC overtaking MPG at the **high speed** configuration on the lemniscate track (Perspective view)

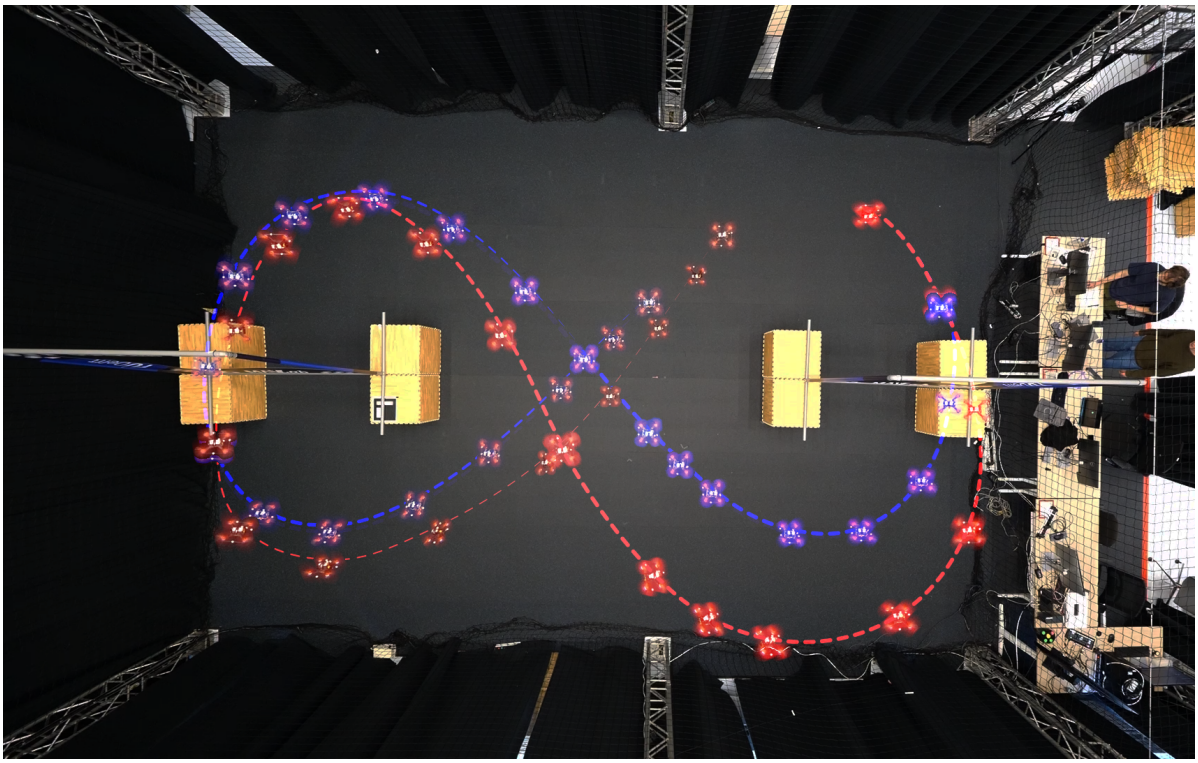


Figure 8.6: Real flight validation of MPC overtaking MPG at the **high speed** configuration on the lemniscate track (Top view)

9

Conclusion

This thesis has explored the application of game-theoretic and model predictive planners in autonomous drone racing, focusing on competitive behaviors such as overtaking, blocking, and strategic decision-making. The distinction between the two lies in the way agents anticipate and respond to the actions of their opponents. Model predictive planners optimize an agent's trajectory by predicting opponent behavior as external factors, while game-theoretic planners explicitly model multi-agent interactions as strategic decisions using equilibrium concepts.

The research evaluated performance in both synchronous and asynchronous execution modes to assess the methods under idealized and real-world conditions. In synchronous racing, all agents make decisions at the same discrete time step and wait for each other to compute a solution before executing their strategies, ensuring they have access to a fully updated and consistent state of the environment. This eliminates timing discrepancies, allowing for a controlled analysis of strategic decision-making. In contrast, asynchronous racing allows agents to make decisions at their own independent rates, meaning they do not wait for one another before computing or executing strategies. This introduces real-time decision-making challenges, where factors such as solve time variability, delayed opponent information, and communication latency can affect performance. Testing across both modes was crucial to assessing the robustness and adaptability of the proposed methods, securing their feasibility in practical racing scenarios where perfect synchronization cannot be assumed.

The key findings of this research reveal the strengths and limitations of different planning approaches. In synchronous racing, Model Predictive Game (MPG) consistently outperformed MPC across all speed configurations, securing clean wins by leveraging its ability to model opponents' strategies and anticipate their actions. However, in asynchronous execution, MPG's performance suffered due to increased solve times, leading to violations of track constraints and losses to MPC, which benefited from its lower computational demands. The introduction of Lifted Model Predictive Game (LMPG) demonstrated that learning-based acceleration could mitigate some of these challenges. By achieving solve times comparable to MPC while preserving the ability to account for opponent strategies, LMPG consistently outperformed both MPG and MPC, achieving the highest win rate across all tested conditions. Additionally, the incorporation of a blocking cost term into MPG (resulting in MPGB) enabled effective defensive maneuvers against MPC, though its success depended on the opponent's reasoning capabilities. High-fidelity simulations and real drone flights further validated the practical viability of the proposed game-theoretic planners when integrated with a nonlinear MPC (NMPC) as the low-level controller, showing that the hierarchical control approach could generate competitive and reliable strategies even under complex drone dynamics. These simulations also highlighted the limitations of MPC, particularly on tracks with high curvature, where its simplistic constant velocity model of the opponent led to frequent collisions and safety concerns. This underscores the importance of more sophisticated opponent modeling for safe and effective racing.

This research highlights the potential of game-theoretic methods for autonomous drone racing, particularly in synchronous scenarios where strategic reasoning provides a competitive edge. However, the

challenges posed by asynchronous execution and computational overhead underscore the need for further optimization and innovation, in order to fly at higher speeds.

To facilitate future work and address these challenges, this thesis introduced a set of automated tools designed to simplify the transcription of complex 3D drone racing scenarios into both model predictive control (MPC) and model predictive game (MPG) frameworks. These tools are modular and flexible, enabling researchers to easily modify race tracks, adjust gate positions, employ diverse cost formulations, and systematically generate detailed race statistics. Consequently, these tools offer a versatile foundation that supports further investigations into both model predictive and game-theoretical planning methods, streamlining experimental setup and fostering reproducibility for future research in multi-agent autonomous drone racing.

9.1. Future work

Several immediate improvements can enhance the current approach. Finding an optimal set of weights for more competitive behavior at high speeds remains a challenge, as manual tuning is time-consuming; an automated hyperparameter search could streamline this process. Learning based MPG could be refined by training against MPC and MPGB to enhance safety and reliability, while incorporating a replay buffer would help mitigate catastrophic forgetting and improve learning efficiency. To extend planning horizons without excessive computation, a non-uniform time discretization could be introduced, thus avoiding scenarios where prolonged solve times cause the MPG to exhaust its available strategy waypoints, and drift off the track. Alternatively, instead of relying solely on learning for solve time acceleration, delays could be explicitly modeled in the game formulation, for example, as a disturbance player. Another way to manage long solve times is to explore anytime solvers, such as interior point solvers [36], which can provide approximate solutions when stopped early. Finally, a large-scale experimental study incorporating a nonlinear low-level controller would offer a more realistic evaluation of all methods.

Another promising direction for improvement is the introduction of a middle-level controller that acts as an intermediary between the high-level planner and the low-level controller. Instead of directly executing the high-level strategies, this middle layer would focus on reference tracking while ensuring that critical constraints – such as track deviations, collision avoidance, and velocity limits – are strictly enforced. By embedding these as hard constraints within the middle-level controller, the number of violations committed by all methods could be significantly reduced. At the same time, this approach would preserve the high-level game theoretic planner's strategic decisions, preventing excessive deviations from the intended trajectory. This structured control hierarchy could provide a balance between computational efficiency and robust constraint enforcement, making it particularly useful for asynchronous execution, where real-time adjustments are necessary.

A more ambitious direction for future work is the development of feedback strategies and their learning-based approximations. Unlike the current open-loop implementation in MPG, feedback strategies can generate more competitive strategies [29] by capturing indirect interactions, without requiring additional cost terms, as was necessary for MPGB. However, solving for feedback solutions is significantly more computationally expensive[9][19]. The only viable approach to reducing computational costs is through learning-based methods, such as multi-agent reinforcement learning (MARL) [16][38] or imitation learning[40], which can approximate feedback strategies efficiently while preserving strategic adaptability. This would enable game-theoretic planners to remain competitive in real-time, asynchronous racing scenarios.

Another potential improvement is to accelerate model predictive games through learning, while incorporating nonlinear dynamics within the inner loop. The current implementation of LMPG relies on simplified dynamics, which may not fully capture the complexities of real-world drone racing. By integrating a nonlinear MPC as the low-level controller during training, the learned policies would better account for aerodynamic effects, motor constraints, and other real-world nonlinearities. This would lead to more accurate high-level strategies, improving both predictive accuracy and real-world applicability without necessarily increasing computational overhead during execution.

References

[1] Tamer Başar and Geert Jan Olsder. *Dynamic Noncooperative Game Theory, 2nd Edition*. Society for Industrial and Applied Mathematics, 1998. DOI: 10.1137/1.9781611971132. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611971132. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611971132.

[2] Leonard Bauersfeld and Davide Scaramuzza. "Range, endurance, and optimal speed estimates for multicopters". In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 2953–2960.

[3] Johannes Betz et al. "Autonomous vehicles on the edge: A survey on autonomous vehicle racing". In: *IEEE Open Journal of Intelligent Transportation Systems* 3 (2022), pp. 458–488.

[4] Andrew Cinar and Forrest Laine. *Does bilevel optimization result in more competitive racing behavior?* 2025. arXiv: 2402.09548 [cs.GT]. URL: https://arxiv.org/abs/2402.09548.

[5] Benjamin Evans, Herman A Engelbrecht, and Hendrik W Jordaan. "Learning the subsystem of local planning for autonomous racing". In: *2021 20th International Conference on Advanced Robotics (ICAR)*. IEEE. 2021, pp. 601–606.

[6] Jaime F Fisac et al. "Hierarchical game-theoretic planning for autonomous vehicles". In: *2019 International conference on robotics and automation (ICRA)*. IEEE. 2019, pp. 9590–9596.

[7] Philipp Foehn et al. "Agilicious: Open-source and open-hardware agile quadrotor for vision-based flight". In: *Science Robotics* 7.67 (June 2022). ISSN: 2470-9476. DOI: 10.1126/scirobotics.abl6259. URL: http://dx.doi.org/10.1126/scirobotics.abl6259.

[8] David Fridovich-Keil. *Smooth Game Theory*. 2024. URL: https://clearoboticslab.github.io/documents/smooth_game_theory.pdf.

[9] David Fridovich-Keil et al. "Efficient iterative linear-quadratic approximations for nonlinear multi-player general-sum differential games". In: *2020 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2020, pp. 1475–1481.

[10] Fadri Furrer et al. "Robot Operating System (ROS): The Complete Reference (Volume 1)". In: ed. by Anis Koubaa. Cham: Springer International Publishing, 2016. Chap. RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625. ISBN: 978-3-319-26054-9. DOI: 10.1007/978-3-319-26054-9_23. URL: http://dx.doi.org/10.1007/978-3-319-26054-9_23.

[11] Drew Hanover et al. "Autonomous Drone Racing: A Survey". In: *IEEE Transactions on Robotics* 40 (2024), pp. 3044–3067. DOI: 10.1109/TRO.2024.3400838.

[12] Suiyi He, Jun Zeng, and Koushil Sreenath. "Autonomous racing with multiple vehicles using a parallelized optimization with safety guarantee using control barrier functions". In: *2022 International conference on robotics and automation (ICRA)*. IEEE. 2022, pp. 3444–3451.

[13] Haimin Hu et al. "Who Plays First? Optimizing the Order of Play in Stackelberg Games with Many Robots". In: *Proceedings of Robotics: Science and Systems*. Delft, Netherlands, 2024. DOI: 10.15607/RSS.2024.XX.116.

[14] Yixuan Jia, Maulik Bhatt, and Negar Mehr. "Rapid: Autonomous multi-agent racing using constrained potential dynamic games". In: *2023 European Control Conference (ECC)*. IEEE. 2023, pp. 1–8.

[15] Dvij Kalaria et al. "Local nmpc on global optimised path for autonomous racing". In: *arXiv preprint arXiv:2109.07105* (2021).

[16] Yu Kang et al. "Autonomous multi-drone racing method based on deep reinforcement learning". In: *Science China Information Sciences* 67.8 (2024), p. 180203.

[17] Talha Kavuncu, Ayberk Yaraneri, and Negar Mehr. "Potential ilqr: A potential-minimizing controller for planning multi-agent interactive trajectories". In: *arXiv preprint arXiv:2107.04926* (2021).

[18] Pravesh Koirala and Forrest Laine. “Monte carlo optimization for solving multilevel stackelberg games”. In: *arXiv preprint arXiv:2312.03282* (2023).

[19] Forrest Laine et al. “The computation of approximate generalized feedback nash equilibria”. In: *SIAM Journal on Optimization* 33.1 (2023), pp. 294–318.

[20] Simon Le Cleac’h, Mac Schwager, and Zachary Manchester. “Algames: a fast augmented lagrangian solver for constrained dynamic games”. In: *Autonomous Robots* 46.1 (2022), pp. 201–215.

[21] Jingqi Li et al. “The Computation of Approximate Feedback Stackelberg Equilibria in Multiplayer Nonlinear Constrained Dynamic Games”. In: *SIAM Journal on Optimization* 34.4 (2024), pp. 3723–3749.

[22] Alexander Liniger, Alexander Domahidi, and Manfred Morari. “Optimization-based autonomous racing of 1: 43 scale RC cars”. In: *Optimal Control Applications and Methods* 36.5 (2015), pp. 628–647.

[23] Alexander Liniger and John Lygeros. “A noncooperative game approach to autonomous racing”. In: *IEEE Transactions on Control Systems Technology* 28.3 (2019), pp. 884–897.

[24] Xinjie Liu, Lasse Peters, and Javier Alonso-Mora. “Learning to Play Trajectory Games Against Opponents With Unknown Objectives”. In: *IEEE Robotics and Automation Letters* 8.7 (2023), pp. 4139–4146. DOI: 10.1109/LRA.2023.3280809.

[25] Patrick Kofod Mogensen and Asbjørn Nilsen Riseth. “Optim: A mathematical optimization package for Julia”. In: *Journal of Open Source Software* 3.24 (2018), p. 615. DOI: 10.21105/joss.00615.

[26] L. Peters et al. “Learning Mixed Strategies in Trajectory Games”. In: *Proceedings Robotics: Science and System XVIII*. Ed. by Kris Hauser, Dylan Shell, and Shoudong Huang. Robotics: Science and Systems. Robotics Science and Systems (RSS), 2022. DOI: 10.15607/RSS.2022.XVIII.051.

[27] Angel Romero et al. “Model predictive contouring control for time-optimal quadrotor flight”. In: *IEEE Transactions on Robotics* 38.6 (2022), pp. 3340–3356.

[28] Matthias Rowold et al. “Efficient spatiotemporal graph search for local trajectory planning on oval race tracks”. In: *Actuators*. Vol. 11. 11. MDPI. 2022, p. 319.

[29] Matthias Rowold et al. *Open-Loop and Feedback Nash Trajectories for Competitive Racing with iLQGames*. 2024. arXiv: 2402.01918 [cs.RO]. URL: https://arxiv.org/abs/2402.01918.

[30] Maria Luisa Scarpa and Thulasi Mylvaganam. “Open-loop and feedback LQ potential differential games for Multi-Agent Systems”. In: *2023 62nd IEEE Conference on Decision and Control (CDC)*. IEEE. 2023, pp. 6283–6288.

[31] Yunlong Song et al. “Autonomous drone racing with deep reinforcement learning”. In: *2021 IEEE International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 1205–1212.

[32] Riccardo Spica et al. “A real-time game theoretic planner for autonomous two-player drone racing”. In: *IEEE Transactions on Robotics* 36.5 (2020), pp. 1389–1403.

[33] Rishabh Saumil Thakkar et al. “Hierarchical control for cooperative teams in competitive autonomous racing”. In: *IEEE Transactions on Intelligent Vehicles* 9.5 (2024), pp. 4845–4860.

[34] Rishabh Saumil Thakkar et al. “Hierarchical control for head-to-head autonomous racing”. In: *Field Robotics* 4 (2024), pp. 46–69.

[35] José L Vázquez et al. “Optimization-based hierarchical motion planning for autonomous racing”. In: *2020 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2020, pp. 2397–2403.

[36] Andreas Wächter and Lorenz T Biegler. “On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming”. In: *Mathematical programming* 106 (2006), pp. 25–57.

[37] Zijian Wang, Tim Taubner, and Mac Schwager. “Multi-agent sensitivity enhanced iterative best response: A real-time game theoretic planner for drone racing in 3D environments”. In: *Robotics and Autonomous Systems* 125 (2020), p. 103410.

[38] Peter Werner et al. “Dynamic multi-team racing: Competitive driving on 1/10-th scale vehicles via learning in simulation”. In: *7th Annual Conference on Robot Learning*. 2023.

[39] Grady Williams et al. “Autonomous racing with autorally vehicles and differential games”. In: *arXiv preprint arXiv:1707.04540* (2017).

[40] Jiaxu Xing et al. “Bootstrapping reinforcement learning with imitation for vision-based agile flight”. In: *arXiv preprint arXiv:2403.12203* (2024).

[41] Alessandro Zanardi et al. *Game Theoretical Motion Planning. Tutorial ICRA 2021*. ETH Zurich, 2021. DOI: 10.3929/ethz-b-000507914.

[42] Edward L. Zhu and Francesco Borrelli. “A Sequential Quadratic Programming Approach to the Solution of Open-Loop Generalized Nash Equilibria”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. 2023, pp. 3211–3217. DOI: 10.1109/ICRA48891.2023.10160799.

[43] Edward L. Zhu and Francesco Borrelli. *A Sequential Quadratic Programming Approach to the Solution of Open-Loop Generalized Nash Equilibria for Autonomous Racing*. 2024. arXiv: 2404.00186 [cs.RO]. URL: https://arxiv.org/abs/2404.00186.

A

Appendix

This appendix presents the parametrizations and mathematical formulations for the race tracks used in the experiments. The parameters for each track are summarized in Table A.1. All tracks share a common rotation and translation operation defined by:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R_z(\alpha_z)\, R_y(\alpha_y)\, R_x(\alpha_x) \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix}$$

where R_x, R_y, R_z denote rotations around the x-, y-, and z-axes by angles $\alpha_x, \alpha_y, \alpha_z$ respectively, and (x_c, y_c, z_c) is the offset from the origin.

Lemniscate:
$$\begin{cases} x(t) = w_x \sin(t), \\ y(t) = w_y \sin(t)\cos(t), \\ z(t) = 0 \end{cases}$$

Circle:
$$\begin{cases} x(t) = w_x \sin(t), \\ y(t) = w_y \cos(t), \\ z(t) = 0 \end{cases}$$

Eight:
$$\begin{cases} x(t) = w_x \cos(t), \\ y(t) = w_y \sin(t)\cos(t), \\ z(t) = w_z \sin(t) \end{cases}$$

Lissajous:
$$\begin{cases} x(t) = w_x \sin(a_x t), \\ y(t) = w_y \sin(a_y t), \\ z(t) = w_z \sin(a_z t) \end{cases}$$

$$t \in [0, 2\pi]$$

Table A.1: Track Parameters for Experimental Setup

Track	α_x	α_y	α_z	w_x	w_y	w_z	x_c	y_c	z_c	a_x	a_y	a_z
Lemniscate	0.0	0.0	0.0	3.635	4.0	–	0.78	0.0	1.9	–	–	–
Circle	0.0	15.0	0.0	3.0	3.5	–	1.0	0.0	2.5	–	–	–
Eight	0.0	0.0	0.0	3.0	3.5	1.0	1.0	0.0	2.5	–	–	–
Lissajous	0.0	0.0	0.0	3.0	3.5	3.0	1.0	0.0	5.0	3.0	1.0	2.0