The Report Committee for Rishabh Saumil Thakkar
Certifies that this is the approved version of the following Report:

Hierarchical Game-Theoretic Control for Multi-Agent Autonomous Racing

APPROVED BY

SUPERVISING COMMITTEE:

Ufuk Topcu, Supervisor

David Fridovich-Keil

Hierarchical Game-Theoretic Control for Multi-Agent Autonomous Racing

by

Rishabh Saumil Thakkar

REPORT

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Computational Science, Engineering, and Mathematics

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2022

Dedicated to all of my teachers.

Acknowledgments

This project and my five years of education at The University of Texas at Austin have come together through the support of many people. First, I would like to thank my advisor, Dr. Ufuk Topcu, for his support over the last two years and for encouraging me to pursue this work. I have always loved computer science, racecars, and strategy games, but I never thought that I would have the opportunity to bring them together into the center of my graduate research. I thank Dr. David Fridovich-Keil for helping me transform my ideas into reality and for his suggestions for continually improving my work. I thank Dr. Zhe Xu for his advice to pursue publishing this project. I also thank Dr. Ilyas Iyoob who taught me how to be an independent researcher and manage large-scale projects.

I would like to especially show my appreciation for Dr. Stephen Boyles who introduced me to the world of scientific research and the vast collection of deceptively challenging problems waiting to be solved. Thank you for welcoming me into SPARTA lab as an undergraduate trying to find his way around the university. Your advice and continued support are greatly appreciated.

I am grateful for the advice of my research mentors and peers, William E. Alexander, Aryaman S. Samyal, Carlin Liao, and Omar Hasan, with whom I have had the privilege to collaborate and who have helped me bring my work

to life.

I am thankful for my old friends, Karthik Velayutham, Atulya Vaidya, Alex Spiride, Avinash Damania, Ryan Menghani, Amit Samuel, Monish Challagondla, Brian Nguyen, Mohit Patel, K. Arvind Prasad, Arihant Jain, and Divyansh Devnani, who have always given an ear for my endless rambling about the latest problem I am trying to solve. I am also thankful for my friends whom I met at UT, Aparna Krishnan, Ritvik Annam, Anusha Paul, and Dinesh Balakrishnan for their encouragement through this journey. I am especially appreciative of Mita Dixit's unwavering friendship and love.

Finally, I am grateful for my family's unconditional love and immeasurable support. My parents, Hemangini and Saumil Thakkar, have sacrificed much to provide me with the opportunity to receive this education at a prestigious university. My brother, Rishi Thakkar, has always been by my side, and I am forever grateful for his support.

Hierarchical Game-Theoretic Control for Multi-Agent Autonomous Racing

Rishabh Saumil Thakkar, M.S.C.S.E.M.
The University of Texas at Austin, 2022

Supervisor: Ufuk Topcu

We develop a hierarchical control scheme for autonomous racing with realistic safety and fairness rules. The first part constructs a discrete game approximation with simplified dynamics and rules presented as temporal logic specifications. Using the discrete representation, we use a model checking tool to synthesize an optimal strategy in the form of a sequence of target waypoints. We apply the model to several case studies of common racing scenarios, and its resulting strategies are qualitatively verified against those executed by racing experts. This formulation is used as the high-level planner in the hierarchical controller but is solved using Monte Carlo tree search (MCTS) to run in real-time.

In the next part, we integrate the high-level planner with a low-level controller to track the target waypoints. Two low-level approaches are considered: a multi-agent reinforcement learning (MARL) trained policy and a linear-quadratic Nash game (LQNG) formulation. As a result, we produce

two hierarchical controllers, MCTS-RL and MCTS-LQNG, respectively. The hierarchical agents are tested against three baselines: an end-to-end MARL controller, a MARL controller tracking the optimal racing line, and an LQNG controller tracking the optimal racing line. The controllers compete head-to-head on an oval track and a complex track, and we count the number of wins and a safety score representing the number of rule violations. Our hierarchical controllers outperform their respective baseline methods in terms of wins, but only MCTS-RL is better than its baselines in terms of safety score. The MCTS-LQNG controller has a worse safety score, but this result is due to the simplicity and conservative nature of the fixed trajectory LQNG baseline. Overall, the MCTS-RL controller outperforms all of the other controllers across both metrics and executes maneuvers resembling those seen in real-life racing.

In the final part, we extend the hierarchical controllers to team-based racing where they must consider a mixture of competitive and cooperative objectives. The formulations are generalized to consider these challenging objectives while still being required to adhere to the complex rules. We test our controllers against the previously discussed baselines in races where the agents compete in teams of two instead of head-to-head. In addition to counting the number of wins and the safety score, we introduce a third metric to measure the cooperative performance of the controllers. We allocate points based on the finishing position of each agent and aggregate them across the teams, which indicates how the team performed as a whole. The results show our hierarchical

agents outperforming their baselines in terms of wins while maintaining similar safety scores. In addition, our controllers also have a higher number of average points per race indicating that they produce greater success as a team. Finally, we observe that the MCTS-RL controller continues to outperform all of the other implemented controllers across all metrics and exhibits tactics performed by expert human drivers.

We show that hierarchical planning for game-theoretic reasoning produces competitive behavior even when challenged with complex objectives, rules, and constraints.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Autonomous driving has seen an explosion of research in academia and industry. Most of these efforts focus on studying scenarios and control in urban environments such as day-to-day driving on streets and highways. However, there is a growing interest towards incorporating autonomy in motorsports. Many advancements in commercial automobiles have originated from projects invented for use in motorsports such as traction control, energy recovery systems, and sequential gearboxes [1].

Modeling an autonomous racing game is inherently similar to an urban driving scenario. For example, they are both focused on controlling four-wheeled automobiles. The primary objective of the drivers is to reach some destination in the shortest time. In racing, however, the interactions between the players are more aggressive because there is an additional nuance in the objective to finish ahead of other players. Racing and urban driving also both involve complex rules governing the interactions amongst the players. In both contexts, these rules are primarily designed to ensure everyone's safety, but the difference between the contexts relates to the extent of the rules. Depend-

ing on the situation, we have different answers to questions such as: "how fast is everyone allowed to go?" or "what is the minimum distance required to be maintained between the cars?" All of these contrasts between racing and urban driving relate to the motivation of participating in a racing game, which is to explore the limits of performance. Therefore, studying autonomous racing is an opportunity to develop autonomous driving controllers to be high-performance, robust, and safe in challenging scenarios.

Finally, in addition to pushing the boundaries of autonomous driving algorithms, researching autonomous racing also paves way for developing simulators to train human drivers. Motorsport teams already spend millions of dollars developing highly sophisticated simulators to help their drivers to become experts at driving around a track that may not even be fully constructed [2]. If they start including simulations of opponents who follow the rules of racing but also provide the precise control of a computer, the teams' drivers can learn what kind of strategies are viable and practice specific scenarios for hours without burning a single drop of fuel.

1.2 Motivation

Most prior research in autonomous racing controllers generally relies on model predictive control methods [3, 4, 5, 6, 7, 8]. As the name suggests, the general structure of these approaches is to solve optimization problems with nonlinear dynamics constraints and objectives. This type of calculation is often limited to a short time horizon (1-3 seconds) because the problem

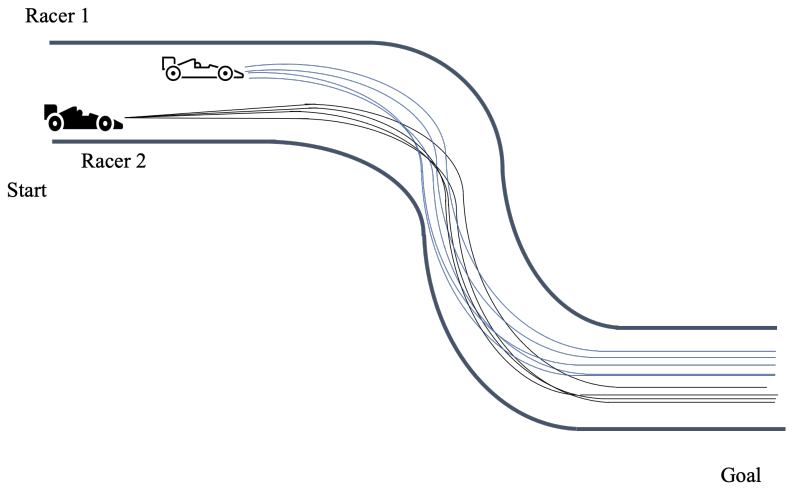

Figure 1.1: Many possibilities of trajectories to consider in racing.

must be simple or small enough to solve at rates of 20 - 50 Hz even with state-of-the-art tools.

In addition, prior work is also mainly focused on single-agent control or situations with agents who behave as stationary or dynamic obstacles rather than adversaries [3, 4, 5, 6, 7, 8, 9, 10, 11]. However, when we hear the term "racing," we naturally think of it as a multi-agent game where all participants are competing against one another. Although it is possible to introduce adversarial objectives and constraints in the optimization-based control methods, they are prone to computational limitations in terms of planning horizon or control frequency for direct low-level control [12, 13]. Consider the simple scenario in Figure 1.1. There is an infinite number of trajectories that the black car might use to overtake the white car across the next two turns on a track.

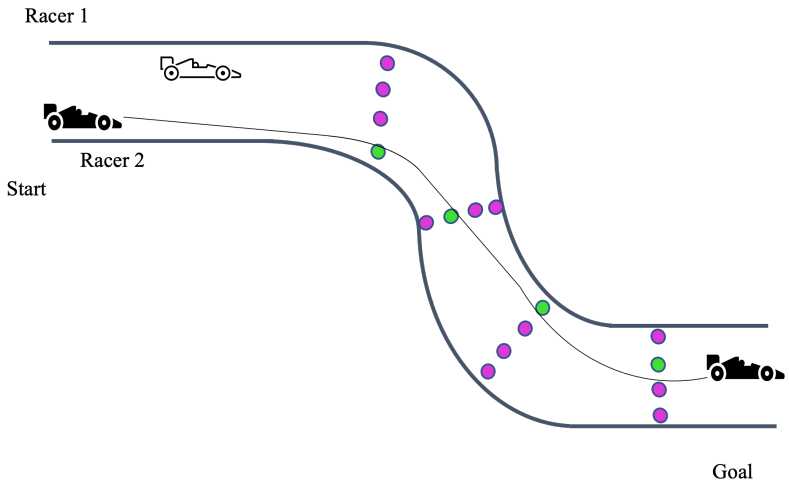

Figure 1.2: A high-level plan outlining lane choices (green) followed by a single trajectory (black).

In order to use existing methods, we must compromise by either shortening the planning horizon and/or simplifying the objectives or constraints in the model. With a short horizon, it is just impossible to plan for the medium-term to long-term strategic decisions to successfully overtake. Otherwise, if the model is simplified, we may no longer have an accurate representation of the game's rules or dynamics. With either compromise, we lose the ability to perform the long-term game-theoretic reasoning.

Therefore, practical implementation of such a complex control application suggests breaking down the system into layers of hierarchical reasoning (Figure 1.2). There are effectively four high-level decisions to make in this scenario which are located at the first turn entry, the first turn exit, the second turn entry, and the second turn exit. The domain of these decisions involves

determining which of the four lanes to be positioned at each of the locations. If there is an oracle that can determine the ideal choice of the lane for those locations such that the rules and dynamics are satisfied, highlighted in green, the problem for our precise, low-level controller becomes much simpler. It no longer needs to decide both where to be positioned and how to move between those positions; rather, it is just a matter of calculating the latter.

1.3 Contributions

This report develops a hierarchical control scheme that reasons about competitive long-term plans and adheres to the safety and fairness rules seen in real-life multi-agent racing. We provide a general formulation for a multi-agent racing game that encodes these complex rules and study the problem in three parts.

In the first part, we construct a turn-based, discrete simplification of a racing game that encapsulates the discrete nature of the safety and fairness rules from the general formulation. We outline temporal logic specifications to represent these rules and use the specifications to rule out players' choices in the game. Furthermore, we develop a turn-based mechanism to preserve the realistic flow of information that would occur in a continuous setting. Our resulting discrete game has a simple reachability objective over a finite number of steps, which is solved using a model checking tool. We evaluate our formulation by considering three case studies of common racing scenarios.

In the second part, we construct the full hierarchical control scheme for

head-to-head racing that runs in real-time. We use the discrete game formulation from the first part as the high-level planner and use Monte Carlo tree search to solve the game in real-time. The solution of the discrete formulation effectively produces a series of long-term waypoints that are safe with respect to the rules and approximately follow a Nash equilibrium. Then, we formulate a simplified version of the general racing game to track the waypoints with relaxed representations of the original rules. The simplified game is solved using two methods to produce high-resolution control inputs: training a policy using multi-agent reinforcement learning and solving a linear-quadratic Nash game approximation. Our structure yields a pair of hierarchical controllers that run in real-time and outperform baselines resembling previously studied autonomous racing methods in terms of head-to-head performance and obedience to following safety rules. Moreover, our controllers produce behavior resembling that of expert human drivers.

In the final part, we consider an extension of head-to-head racing by introducing team-based objectives, which is a major part of real-life competitions. To our knowledge, this is the first work to study a mixture of cooperative and competitive objectives for teams of players in multi-agent racing. We generalize each of our game formulations from the previous sections by updating their objectives such that they aim to improve the overall performance of a player's team in addition to its own performance while subject to the same rules and constraints. Then, we use our proposed hierarchical control structure and compare them with the baseline methods adapted to play the

updated form of the game. Our controllers continue to outperform the baselines although the game has become more complex, and they exhibit tactics resembling human experts.

Although this hierarchical control method is developed in the context of a racing game, the structure of the proposed architecture effectively reasons about optimal choices in a more general game-theoretic setting with complex constraints involving temporal logic and both continuous and discrete dynamics. As a result, we can apply this method to other competitive settings that exhibit the aforementioned properties such as financial systems, power systems, or traffic control systems.

1.4 Outline

The remainder of this report is organized as follows. Chapter 2 provides a literature review of prior research in autonomous racing. Chapter 3 introduces the general racing game formulation and outlines the discrete game abstraction of the general formulation. Chapter 4 develops the full hierarchical control scheme by utilizing the discrete formulation from the previous chapter as a high-level planner and constructing a pair of low-level formulations. Chapter 5 applies the hierarchical controller to a team-based multi-agent racing. Finally, Chapter 6 concludes this work by providing a summary and ideas for further extensions.

Chapter 2

Literature Review

There are many components in the pipeline to build a complete autonomous racing system such as perception, planning, control, and hardware integration, all of which have a rapidly growing collection of literature [14]. However, the focus of this report and literature review is limited to planning and control. Most prior work in autonomous racing control is focused on single-agent lap time optimization because multi-agent racing is an inherently more complex problem. Nevertheless, there are recent developments in multi-agent racing, but they are limited by the rules of real-life racing that are considered. They primarily only focus on basic collision avoidance. In reality, certain players bear more responsibility for collision avoidance depending on the state of the game, and there exist additional rules on lane changes to ensure safety and fairness. Finally, we also study prior works using hierarchical reasoning to solve challenging problems in several contexts of autonomous driving.

2.1 Single-Agent Racing

Single-agent racing approaches include both optimization and learning-based methods, which primarily focus on finding and tracking the optimal

racing trajectory that minimizes lap time. Hou and Wang [10] use Monte Carlo tree search to estimate where to position the car around various shaped tracks to define an optimal trajectory. Vazquez et al. [8] propose a method that computes an optimal trajectory offline and uses a model predictive control (MPC) algorithm to track the optimized trajectory online. Similarly, Stahl et al. [11] also perform calculations offline by creating a graph representation of the track to compute a target path and use spline interpolation for smooth online path generation in an environment with static obstacles. Lastly, several studies use variants of MPC depending on the dynamics model of choice and numerical optimization methods such as sequential quadratic programming or stochastic optimization algorithms [3, 4, 5, 6, 7].

In the category of learning-based approaches, Kabzan et al. [15] develop an online learning algorithm to update parameters of an MPC-based controller using feedback from applying control inputs. Remonda et al. [16] use deep reinforcement learning to train a neural network policy with vehicle telemetry data. However, their experiments indicate that the method does not generalize when extended to tracks the model is not trained on. de Bruin et al. [17] use state representation to improve the generalization of the reinforcement learning-based control for racing on unseen tracks. Finally, Weiss and Behl [18] develop a deep-learning framework that uses vision to estimate waypoints in a player's local space and track them using optimization-based control methods.

2.2 Multi-Agent Racing

In multi-agent racing studies, both optimization and learning-based control approaches are also used. Liniger and Lygeros [12] formulate and solve bimatrix games for three types of common scenarios in head-to-head racing but do not apply them to a real-time system. Li et al. [13] use mixed-integer quadratic programming formulation with realistic collision avoidance. However, they concede that this formulation struggles to run in real-time due to the computational complexity of solving integer programs. Spica et al. [19] propose a real-time control mechanism for a game with a pair of racing drones. This work provides an iterative-best response method while solving an MPC problem that approximates a local Nash equilibrium. It is eventually extended to automobile racing and multi-agent scenarios with more than two players by Wang et al. [20, 21], but they do not consider teams. He et al. [9] create a fast, real-time MPC algorithm to make safe overtakes, but their method does not consider adversarial behavior from the opposing players.

Next, we outline some of the learning-based multi-agent racing works. Schwarting et al. [22] train a policy using vision-based deep learning and self-play to predict opponent states to outperform a model that independently learns to predict opponent behaviors. Their policy produces behavior mimicking human racing drivers. Song et al. [23] use curriculum learning to train a policy that iteratively builds its knowledge from single-agent racing to overtaking and finally to collision avoidance. Again, these approaches do not consider racing rules other than simple collision avoidance.

Wurman et al. [24] develop an autonomous racing controller using deep reinforcement learning that considers the rules of racing beyond collision avoidance. Their controller outperforms expert humans while also adhering to proper racing etiquette. It is the first study to consider nuanced safety and fairness rules of racing and does so by developing a reward structure that trains a controller to understand when it is responsible for avoiding collisions, and when it can be more aggressive. They do not encode the rules directly in their model. Instead, they refer to human experts to evaluate the behavior of their trained deep learning controllers to adjust parameters that affect the aggressiveness of their controller. However, their control design is fully learning-based and doesn't involve explicit path planning or hierarchical reasoning. In addition, although this paper models more realistic racing behavior in multi-agent racing, it also still lacks consideration of cooperative objectives amongst racing teammates.

2.3 Hierarchical Control

Hierarchical reasoning is a method that has also been previously studied in various contexts for autonomous driving. Liniger [25] outlines a hierarchical racing controller that constructs a high-level planner with simplified dynamics to sample sequences of constant curvature arcs and a low-level planner to use MPC to track the arc that provided the furthest progress along the track. Fisac et al. [26] develop a two-level planning system to control an autonomous vehicle in a highway environment with aggressive human drivers. The upper-level sys-

tem produces a plan to be safe against the uncertainty of the human drivers in the system by using simplified dynamics. The lower-level planner implements the strategy determined by the upper level-planner using precise dynamics. Similarly, Moghadam and Elkaim [27] study hierarchical reasoning for sequential decision making in highway driving. They construct a high-level planner using a trained reinforcement-learning policy to determine lane-changing plans to safely pass other drivers. The lane-changing plans are shared with low-level controllers to execute those actions. Finally, Wongpiromsarn et al. [28] develop a reactive synthesis and hierarchical control approach where an urban driving agent's objective is defined by a series of temporal logic specifications. Their hierarchical control system has an upper level that selects target states to reach and a lower level that implements plans to reach those states in order to ensure the specifications are met. These papers have established the power of hierarchical reasoning in autonomous driving, but they have only applied it in a non-adversarial context. However, in the autonomous racing scenario, other participants in the system have competing objectives, which complicates how the hierarchical abstraction must be constructed.

Chapter 3

Synthesis of Verifiably Safe and Optimal Racing Maneuvers

> The first part is called
> "The Pledge." The magician
> shows you something
> ordinary... he asks you to
> inspect it to see if it is indeed
> real, unaltered, normal. But of
> course... it probably isn't.
>
> Christopher Priest, *The Prestige*

In this chapter, we develop a discrete model of an autonomous racing game. To motivate our discretized game approximation, we begin by outlining a general multi-agent racing game formulation involving rules seen in real-life racing. Then, we transform the general formulation into a discrete formulation treating the rules as temporal logic specifications, so we can synthesize strategies using model checking methods. Lastly, the discrete formulation is applied to several race scenarios where the resulting strategies are compared to those seen in real-life.

3.1 General Multi-Agent Racing Game Formulation

Let there be a set, N, of players racing over T discrete time steps in $\mathcal{T} = \{1, ..., T\}$. There is a track defined by a sequence of τ checkpoints along the center, $\{c_i\}_{i=1}^{\tau}$, whose indices are in a set $C = \{1, ..., \tau\}$. The objective for each player i is to minimize the pairwise differences in the time to reach the final checkpoint with all other players. In effect, players aim to reach the finish line with the largest time advantage with respect to all of the other players. The continuous state (including, e.g., position, speed, or tire wear) for each player, denoted as $x_t^i \in X \subseteq \mathbb{R}^n$, and control, denoted as $u_t^i \in U \subseteq \mathbb{R}^k$, are governed by known dynamics f^i. We also introduce a pair of discrete state variables $r_t^i \in C$ and $\gamma^i \in \mathcal{T}$. The index of the latest checkpoint passed by player i at time t is r_t^i, and it is computed by function $p : X \to C$. This state variable effectively represents a player's progress along the race. The time when player i reaches c_τ, i.e. the final checkpoint, is γ^i. Using these definitions, we formulate the objective (3.1.1) and core dynamics (3.1.2)-(3.1.6) of the game as follows:

$$\min_{u_0^i, ..., u_T^i} (|N| - 1)\gamma^i - \sum_{j \neq i}^{N} \gamma^j \tag{3.1.1}$$

$$x_{t+1}^j = f(x_t^j, u_t^j), \quad \forall \ t \in \mathcal{T}, j \in N \tag{3.1.2}$$

$$r_{t+1}^j = p(x_{t+1}^j, r_t^j), \quad \forall \ t \in \mathcal{T}, j \in N \tag{3.1.3}$$

$$r_1^j = 1, \quad \forall \ j \in N \tag{3.1.4}$$

$$r_T^j = \tau, \quad \forall \ j \in N \tag{3.1.5}$$

$$\gamma^j = \min\{t \,|\, r_t^i = \tau \wedge t \in \mathcal{T}\}, \quad \forall \ j \in N \tag{3.1.6}$$

In addition to the individual dynamics, we introduce constraints modeling the rules of the game. To ensure that the players stay within the bounds of the track we introduce a function, $q : X \to \mathbb{R}$, which computes a player's distance to the closest point on the center line. This distance must be limited to the width of the track w. Therefore, for all $t \in \mathcal{T}$ and $j \in N$:

$$q(x_t^j) \leq w \tag{3.1.7}$$

Next, we define the collision avoidance rules. We evaluates if player i is "behind" player j, and depending on the condition, the distance between every pair of players, computed by function $d : X \to \mathbb{R}$, is required to be at least s_1 if player i is behind another player j or s_0 otherwise. For all $t \in \mathcal{T}$, $j \in N$, and $k \in N \setminus \{j\}$ these rules are expressed by the constraint:

$$d(x_t^j, x_t^k) \geq \begin{cases} s_1 & \text{player } i \text{ behind player } j \\ s_0 & \text{otherwise} \end{cases} \tag{3.1.8}$$

Finally, players are limited in how often they may change lanes depending on the classification of part of the track they are located at. We assume that there are $\lambda \in \mathbb{Z}^+$ lanes across all parts of the track. If the player's location on the track is classified as a curve, there is no limit on lane changing. However, if the player is at a location classified as a straight, it may not change lanes more than L times for the contiguous section of the track

classified as a straight. We define a set $\mathcal{S}$ that contains all checkpoint indices where a player is located at a straight section. We also introduce a function $z : X \to \{-1, 1, 2, ..., \lambda\}$ that returns the lane ID of a player's position on the track or -1 if the player is off-track. Using these definitions, we introduce a variable l_t^j calculated by the following constraint for all $t \in \mathcal{T}$ and $j \in N$:

$$l_t^j = \begin{cases} l_{t-1}^j + 1 & \mathbb{1}_{r_t^j \in \mathcal{S}} = \mathbb{1}_{r_{t-1}^j \in \mathcal{S}} \wedge z(x_t^j) \neq z(x_{t-1}^j) \\ 0 & \text{otherwise} \end{cases} \tag{3.1.9}$$

l_t^j effectively represents a player's count of "recent" lane changes over a sequence of states located across a contiguous straight or curved section of the track. However, the variable is only required to be constrained if the player is on a straight section of the track. Therefore, the following constraint must hold for all $t \in \mathcal{T}$ and $j \in N$ and if $r_t^j \in \mathcal{S}$:

$$l_t^j \leq L \tag{3.1.10}$$

Most prior multi-agent racing formulations [20, 21, 13] do not include the complexities we introduced through defining constraints (3.1.8)-(3.1.10). They usually have a similar form regarding continuous dynamics and discrete checkpoints (3.1.2)-(3.1.6), and their rules only involve staying on track (3.1.7) and collision avoidance with a fixed distance. However, in real-life racing, there do exist these complexities both in the form of mutually understood unwritten rules and explicit safety rules [29]. As a result, we account for two of the key rules that ensure the game remains fair and safe:

1. There is a greater emphasis on and responsibility of collision avoidance for a vehicle that is following another (3.1.8).

2. The player may only switch lanes L times while on a straight section of the track (3.1.9)-(3.1.10).

The first rule ensures that a leading player can choose a trajectory without needing to consider an aggressive move that risks a rear-end collision or side collision while turning from the players that are following. This second rule ensures that the leading player may not engage in aggressive swerving or "zig-zagging" across the track which would make it impossible for a player that is following the leader to safely challenge for an overtake.

While functions may exist to evaluate these spatially and temporally dependent constraints in the formulation, their discrete nature suggests that they cannot be easily differentiated. Therefore, it is not possible to simply apply state-of-the-art optimization algorithms would not apply to produce optimal control inputs in real-time.

3.2 Discrete Game Formulation

In order to design a controller that can solve the realistic racing game formulation in the previous section, we must use an alternative to traditional optimization-based control methods. We recognize that complexities in the formulation arise from constraints over discrete properties of the players' trajectories. This characteristic of the formulation suggests that constructing a discrete abstraction of the game is an appropriate method to estimate feasible trajectories. In the following sections, we describe some abstractions and their

justifications to construct a turn-based discrete game approximation of the original formulation.

3.2.1 Abstraction of State Space

We begin by constructing the discrete abstraction of the state space from the original formulation. We do not explicitly specify any components of players' states when defining the original formulation because it is agnostic to the vehicle dynamics model being considered. However, including variables computed by constraints (3.1.3) and (3.1.9), we assume each player's state in the original game at least consists of the following five variables as they are the only ones encoded in our discrete state representation:

- position
- velocity
- number of "recent" lane changes
- tire wear
- last passed checkpoint index

In the following subsections, we describe how the aforementioned state variables are transformed to produce a discrete model of a player.

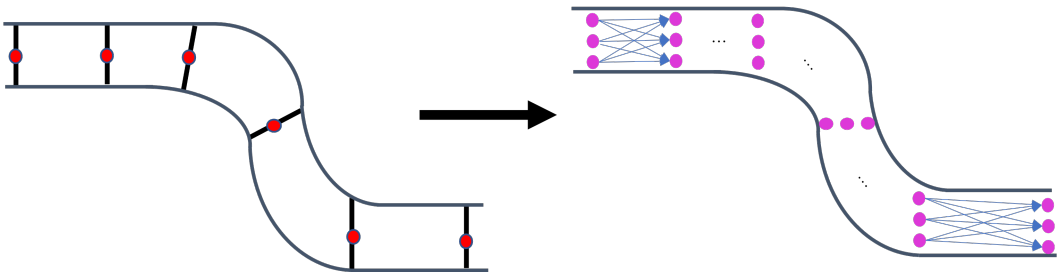

Figure 3.1: Transformation of the continuous race track with checkpoints (red) into discrete lanes at each checkpoint (purple).

Discrete Representation of Racetrack

We specify the play order so that the discrete game is played by making choices at the checkpoints that are indexed by elements of C rather than at each time-step from $\mathcal{T}$ defined in the original formulation. This transformation is natural to consider because all players must ultimately pass all of the checkpoints in order. As a result, the turns of the discrete game and players' states in the discrete game are indexed by their last passed checkpoint, and time becomes a variable in the discrete game state. Furthermore, indexing by the checkpoints also produces a natural discretization for the position state variable in the original formulation. Around each checkpoint, we select λ (which is the number of lanes) discrete locations along the line perpendicular to the direction of travel where each location evaluates to a unique lane ID on the track when passed into function $z(\cdot)$ defined in the general formulation. Therefore, we represent a player's position in discrete game formulation by its lane ID and index of the game state, i.e., the last passed checkpoint. This choice allows us to naturally encode the rules governing players' lanes and en-

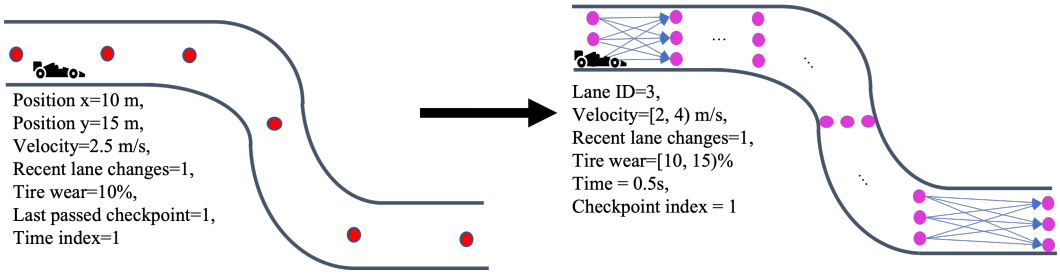

Figure 3.2: Transformation from original formulation state to discrete formulation state.

sures that every location considered in the discrete game remains within the bounds of the track. Figure 3.1 visualizes the continuous space of the track with checkpoints (in red) transformed into discrete locations associated with a unique lane ID at each checkpoint (in purple).

Discrete Representation of Player States

The remaining components of players' states are either already discrete-valued, such as the count of "recent lane changes," or they are represented in the form of discrete buckets or rounded to a finite precision. For example, instead of considering real number value for a player i's velocity from its state $x_v^i = 2.5\,\mathrm{m\,s^{-1}}$ in the original game, the discrete representation would simply be $v^i \in [2, 4)\mathrm{m\,s^{-1}}$. Figure 3.2 visualizes a continuous state for a player converted into discrete form. The overall components of a player i's discrete state are:

- lane ID a_k^i

- velocity bucket v_k^i
- number of "recent" lane changes l_k^i
- tire wear e_k^i
- time t_k^i

where k is the index of the state and the checkpoint associated with the state. We also use the notation k^i to refer to the index of the player i's checkpoint in the overall game state because the players' indices can be different as we are modeling a turn-based game.

3.2.2 Abstraction of Dynamics

Given discrete states of the players, we outline our abstraction of the dynamics, i.e. transitions between discrete states in the game. Because the discrete game is indexed by track checkpoint, we must also transform the players' control inputs. At each checkpoint, the players' choices are now the lane ID and target velocity bucket for the upcoming checkpoint. Next, we rewrite the constraints related to the racing rules in the original formulation as temporal logic specifications. We then discuss the specifics of how our state transitions are computed based on a given action and finish by describing the turn-based mechanism of the discrete formulation.

Temporal Logic Specifications of Rules

Constraints regarding the rules of racing in the original transformation are now expressed as temporal logic specifications over the discretized state transformation for some player i as follows:

$$\square\,(a_k^i \in \{1, ..., \lambda\}) \tag{3.2.1}$$

$$\square\,(\bigwedge_{j \in N \setminus i} (k^i = k^j \wedge a_k^i = a_k^j) \implies |t_k^i - t_k^j| \geq \mu)) \tag{3.2.2}$$

$$\square\,((k \in \mathcal{S}) \implies (l_k^i \leq L\ U\ k \notin \mathcal{S})) \tag{3.2.3}$$

Specification (3.2.1) models constraint (3.1.7) requiring players to stay on track by ensuring that the lane ID is within a set of positive lane IDs. Specification (3.2.2) represents constraint (3.1.8) requiring players to avoid collisions. To abstract the concept of collision avoidance in the discrete space, we require that players must always maintain a minimum time difference of μ if two players share the same lane ID at the same checkpoint. We assume that if there is a small time difference (e.g. 0.1 s) between the players at the same checkpoint and the same lane, there would be an increased risk of collision. Lastly, specification (3.2.3) refers to the constraint (3.1.10) requiring players to limit their lane-changing while on a straight. The specification ensures that the "recent" lane changes variable does not exceed L if checkpoint index k is a straight part of the track represented by the set $\mathcal{S}$ defined in the original formulation.

The overall specification for each of the players is the conjunction of the three. Given our state discretization and action set, we could use synthesis

methods to produce a controller to meet the specifications for some objective. However, the computational complexity of synthesizing a controller to meet such specifications is not manageable due to the exponential growth with respect to the number of specifications and variables [30]. Therefore, we take advantage of the fact that most of these specifications are easily verified by simply examining a player's resulting state from a given state-action pair and that specification (3.2.1) is trivially satisfied by our state and action space abstraction (players are not allowed to choose target lanes outside of $\{1, ..., \lambda\}$). When constructing the model of the game, we simply disregard any player choice that violates the specifications thereby only considering states and trajectories that are allowed.

Computing State Transitions

To check if a state-action pair satisfies the temporal logic specifications and reasonably approximates vehicle dynamics, we must have a straightforward way of computing the resulting state variables when applying some action.

Updating the lane ID and velocity states is trivial because it is exactly determined by the player's action. Similarly, updating the "recent" lane changes variable is also simple with our state space design. We directly apply the logic from the original formulation (3.1.9). It only requires evaluating whether the track type classification of the pair of checkpoints is the same and if the choice of lane ID is different from the lane ID at the initial checkpoint.

Therefore, if choosing an action implies that lane change limit L is exceeded on the straight checkpoints, the action would be disregarded in the model to satisfy specification (3.2.3).

To calculate updates for the elapsed time state, we first use the known track parameters (such as turning radius or lane width) to estimate the distance to travel between the lane in the current checkpoint c_k to the lane in the subsequent checkpoint c_{k+1}. If the track between two checkpoints is a straight, the Euclidian is used to estimate the distance to travel based on the lane width (w_l) and the distance between the location of the checkpoints $\gamma_{k,k+1}$. If the track between the two checkpoints is a curve, then we calculate a coarse estimate of the distance by averaging the radius of the turn for the player's lane at the initial checkpoint r_k and the radius of the turn for the player's target lane at the next checkpoint r_{k+1} and multiply it by the central angle of the turn θ_k. These calculations are summarized below:

$$d = \begin{cases} \sqrt{w_l^2 + \gamma_{k,k+1}^2} & \text{if } k \in \mathcal{S} \\ \frac{r_k + r_{k+1}}{2}\theta_k & \text{otherwise} \end{cases} \tag{3.2.4}$$

Given the estimated distance d, the average velocity of the bucket at the initial checkpoint $\bar{v}_k$ and the average velocity of the target bucket $\bar{v}_{k+1}$ and parameters of the vehicle such as maximum allowed velocity v_*, maximum acceleration a, and maximum braking b, we use simple equations of motion to calculate the minimum time it takes to travel the distance. Moreover, maximum allowed velocity v_* is estimated using the tire wear state at the

initial checkpoint e_k, track radius, and the parameter of a vehicle's maximum allowed lateral acceleration. We enforce that $\bar{v}_{k+1} \leq v_*$ and disregard all actions that violate this requirement because such an action would not obey the lateral acceleration limitation of the vehicle. In addition, we verify it is possible to accelerate or decelerate from v_k to v_{k+1} within the distance if $v_k \leq v_{k+1}$ or $v_k \geq v_{k+1}$, respectively. If that is not possible, then the action with target velocity v_{k+1} is also disregarded. For the remaining cases, we use the following calculation for the time update δt_k:

$$\delta t_k = \begin{cases} \frac{v_* - \bar{v}_k}{a} + \frac{v_* - \bar{v}_{k+1}}{b} + \frac{d - \frac{v_*^2 - \bar{v}_k^2}{2a} - \frac{v_*^2 - \bar{v}_{k+1}^2}{2b}}{v_*} & \text{if } v_* \geq \bar{v}_k \wedge \\ & \quad \frac{d - \frac{v_*^2 - \bar{v}_k^2}{2a} - \frac{v_*^2 - \bar{v}_{k+1}^2}{2b}}{v_*} \geq 0 \\ \frac{\bar{v}_k - v_*}{b} + \frac{v_* - \bar{v}_{k+1}}{b} + \frac{d - \frac{\bar{v}_k^2 - v_*^2}{2b} - \frac{v_*^2 - \bar{v}_{k+1}^2}{2b}}{v_*} & \text{if } v_* < \bar{v}_k \wedge \\ & \quad \frac{d - \frac{\bar{v}_k^2 - v_*^2}{2b} - \frac{v_*^2 - \bar{v}_{k+1}^2}{2b}}{v_*} \geq 0 \\ \frac{\sqrt{\frac{-2dba - b\bar{v}_k^2 - a\bar{v}_{k+1}^2}{-a-b}} - \bar{v}_k}{a} + \frac{\sqrt{\frac{-2dba - b\bar{v}_k^2 - a\bar{v}_{k+1}^2}{-a-b}} - \bar{v}_{k+1}}{b} & \text{if } \bar{v}_k \leq v_* \\ \text{action ruled out} & \text{otherwise} \end{cases} \tag{3.2.5}$$

The calculation assumes the player accelerates or brakes to reach v_* from $\bar{v}_k$, maintains that speed for as long as possible until the player must brake to hit $\bar{v}_{k+1}$ if $\bar{v}_{k+1} \neq v_*$. If there is not enough distance to perform this maneuver and $\bar{v}_k \leq v_*$, we calculate the highest velocity the player can hit given we must end at the target velocity within the specified distance. All other possible maneuvers would violate the dynamical limitations of the vehicle and are ruled out of the set of allowed actions. We also use the time

state update (3.2.5) to estimate collision avoidance and satisfy specification (3.2.2). If a player chooses a lane that a prior player has already selected for its turn and the difference in the time states for these players would be smaller than μ if the action is applied, then the action is disregarded.

Finally, in order to calculate the tire wear state update, we use different calculations for the straight or curve sections of the track. If the track between the checkpoints is a straight, we multiply a tire wear factor parameter L_{straight} associated with driving straight with the distance of the straight d. When the track between the checkpoints is a curve, we multiply the tire wear factor parameter L_{curve} associated with driving on a curve, the distance of the curve d, and an estimate for the average lateral acceleration achieved by hitting the target velocity $\bar{v}_{k+1}$ calculated using equations of circular motion. The tire wear update δe_k is calculated as follows:

$$\delta e_k = \begin{cases} dL_{\text{straight}} & \text{if } k \in \mathcal{S} \\ \frac{2dL_{\text{curve}}\bar{v}_{k+1}^2}{r_k + r_{k+1}} & \text{otherwise} \end{cases} \tag{3.2.6}$$

For both the time and tire wear states, the updates are added to the initial state and projected back into their discrete buckets or rounded to the finite precision. Note that all of the known parameters used in our calculations are standard in most vehicle dynamics models except for tire wear related parameters[31]. We emphasize this note because our high-level planner is designed to be agnostic to the underlying dynamics model. If tire wear is not modeled, one can just assume that e_k is always zero, and the remaining calculations are left unchanged or unused without impacting the discrete game

implementation.

Using these calculations, we have a way of computing the exact sequence of states given a sequence of player actions. Next, we discuss the final piece of our discrete game's dynamics by outlining the logic behind determining the order in which the players choose actions.

Turn-Based Mechanics

Although players make decisions concurrently in the original game and real-life, our discrete abstraction is modeled as a turn-based game. Because the states are indexed by the checkpoint instead of time, the game is played by evaluating the choices of all players one checkpoint at a time. The order in which the players choose their actions at each checkpoint is determined by the player who has the smallest time state at the checkpoint being processed. A lower time state value implies that a player was at the given checkpoint before other players, so it would have made its choice at that location before the others in the continuous setting. This ordering also implies that players who arrive at a checkpoint after preceding players observe the actions of those preceding players preserving the realistic flow of information. Most importantly, because the ordering forces the trailing players to choose last, we also capture the rule that the trailing players (i.e. those that are "behind" others) are responsible for collision avoidance after observing the leading players' actions. As a result, in conjunction with satisfying specification (3.2.2), we model realistic collision avoidance rules.

3.2.3 Discrete Game Objective

The final component of our discrete game is the determining the rewards and the objective. The reward function for a player i is:

$$R^i(s^1, ...s^n) = \begin{cases} (\sum_{j \in N \setminus i} s_t^j) - |N-1| s_t^i & \text{if } (\bigwedge_{m \in N} s_k^m = c_\tau) \\ 0 & \text{otherwise} \end{cases} \tag{3.2.7}$$

It is zero for all states except the one after all players have reached the final checkpoint. At the final checkpoint, player i's reward is the pairwise difference in the time state with each player after everyone has reached the goal, which resembles the original game formulation's objective (3.1.1). The objective for each player is to simply maximize this reward.

3.2.4 Summary of Formulation

Given the state space, dynamics, and objective of the game, we have effectively constructed a finite dynamic game. We write this game in a more general form as a stochastic multiplayer game (SMG) originally defined by Shapley [32] but in the form presented by Chen et al. [30] as a tuple of the following elements:

- S is the finite state space of the game, which is the set product of all players and the domains of each of the state variables.
- $A^1(\boldsymbol{s}), \ldots, A^{|N|}(\boldsymbol{s})$ are the finite action sets of each of the players dependent on the current state, which includes full knowledge of the other player' state and the track position variable.

- $F^1(\boldsymbol{s}, a^1, \boldsymbol{s}'), \ldots, F^{|N|}(\boldsymbol{s}, a^{|N|}, \boldsymbol{s}')$ are the transition functions for each player following the calculations described previously.
- $R^1(\boldsymbol{s}), \ldots, R^{|N|}(\boldsymbol{s})$ are the reward functions provided in the prior section.
- $AP = \{\text{goal}\}$ atomic propositions.
- $L : S \rightarrow 2^{AP}$ is the labeling function for the states for the set AP. The function labels all states where all players have reached the final checkpoint with $\{\text{goal}\}$, and all other states with $\varnothing$.

Once in the SMG form, we can use state-of-the-art model checking tools with a simple reachability specification, $\Diamond$goal, to synthesize optimal strategies for the players to maximize their rewards.

3.3 Case Studies of Strategic Scenarios

To evaluate if our discrete game formulation is a reasonable representation of real-life racing, we apply the model to several case studies resembling some common race scenarios for two players. The goal is to visualize and interpret the produced strategies to ensure it matches our prior understanding of what might happen in a similar scenario in real-life racing. We categorize the scenarios based on three common types of track shapes: long straight, hairpin, and chicane shown in Figure 3.3. For each of the track shapes, we explore various combinations of initial states for tire wear, velocity, and elapsed time for the players. We analyze the scenarios from the perspective of Player

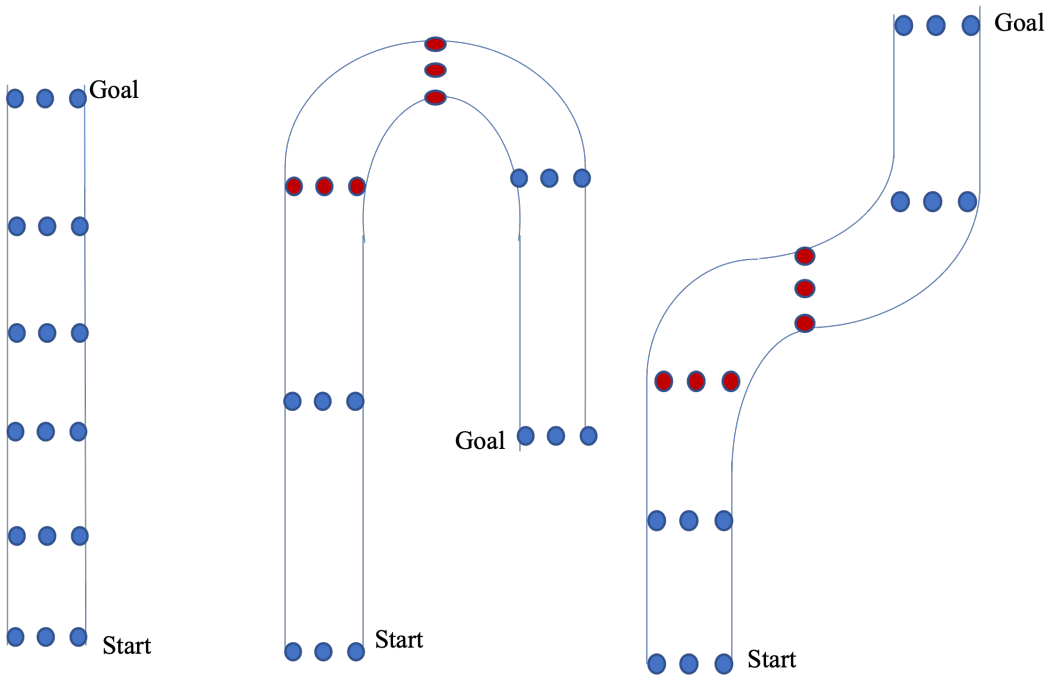

Figure 3.3: Three track types straight, hairpin, and chicane visualized from left to right. Each shape is discretized by 6 checkpoints and 3 lanes. The discrete nodes are color coded using their classification as straights (blue) or corners (red).

Vehicle Parameter	**Player 1**	**Player 2**
Maximum velocity ($\mathrm{m\,s^{-1}}$)	7	6
Maximum longitudinal acceleration ($\mathrm{m\,s^{-2}}$)	3	4
Minimum longitudinal acceleration, i.e. maximum braking force ($\mathrm{m\,s^{-2}}$)	-4	-4
Maximum lateral acceleration ($\mathrm{m\,s^{-2}}$)	5.88	6.86
Minimum lateral acceleration (force that can be sustained regardless at any tire wear state) ($\mathrm{m\,s^{-2}}$)	2.94	2.94

Table 3.1: Parameters used to generate transitions for the model tested in the case study scenarios.

1 and start it with a non-zero time state for each study. Setting a non-zero initial time state indicates that it is starting with a disadvantage and is behind Player 2. Furthermore, we focus on the initial states where Player 1 is able to overtake Player 2 or see what the best time gap it achieves.

To make the results of the studies more interesting, we also assume that the players' vehicles have different dynamical parameters. Player 1 has a higher top speed than Player 2 but has a lower acceleration than Player 2. However, Player 2 has a higher limit for maximum lateral acceleration, which is used for computing the maximum velocity allowed during turning, but it also has a higher tire wear factor. If both cars have the same parameters, the strategies of both vehicles would be similar. Furthermore, it would be near impossible for Player 1 to overtake Player 2 with perfect play. The parameters used when generating the transitions functions for each player are listed in Table 3.1.

For the results in each case, we plot the "best" time gap player 1 achieves assuming optimal strategy is implemented by both the players. If this value is positive, it means Player 1 has reached the goal position before Player 2 and vice-versa if negative. All of the models are implemented using

PRISM-games v3.0, a state-of-the-art model checking software [33]. Furthermore, our analysis for each track shape includes a visualization plotting the optimal strategy and a comparison of the resulting strategy to a similar, real-life racing scenario. This comparison allows us to verify that the strategic choices of our model synthesis are comparable to human experts. Appendix A provides detailed results for all of the experiments discussed in the following sections.

3.3.1 Long Straight

The first track shape is a simple long straight. In this case, tire wear has little to no effect on the performance of the players because the vehicles are not subject to significant lateral loads. Rather, we know from real-life races that having an advantage in the initial time gap or higher initial speed when approaching the long straight results in a favorable time gap at the end of the straight. Therefore, we experiment with various initial velocities and time gaps between the players. Player 1's initial value of the time state, i.e. time disadvantage, varies from 0.5 s to 0.7 s. However, player 1's initial speeds may be higher in some experiments yielding the possibility of an overtake.

For each of the initial states, we plot the resulting time gap assuming the players select their best action in Figure 3.4. We observe a negative correlation between initial time disadvantage and the resulting time gap and a positive correlation between initial speed and the resulting time gap. Therefore, the results confirm that the model does meet the expected results based

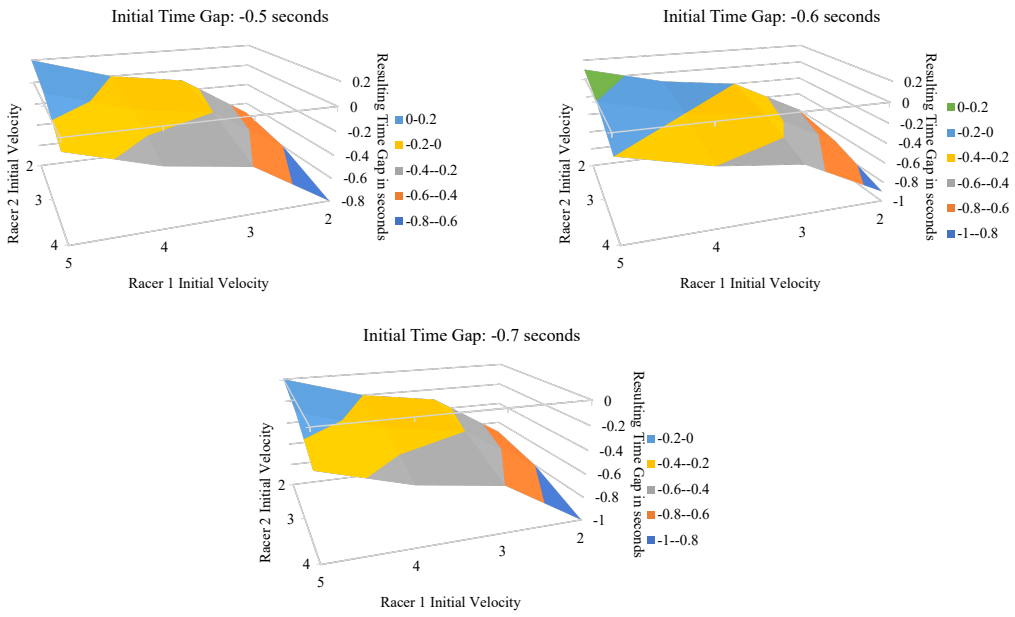

Figure 3.4: Plots for the resulting time gap by implementing both players' synthesized optimal strategy for various initial velocities and tire wear percentages in the long straight shape.

on the real-life understanding of where advantages are observed in this type of track shape. Next, we take a deeper dive in Figure 3.5 by visualizing the players' strategies in one of the experiments. We show the specific scenario where the initial time disadvantage is 0.6 s, the initial velocity of racer 1 is $5\,\mathrm{m\,s^{-1}}$ and the initial velocity of racer 2 is $2\,\mathrm{m\,s^{-1}}$. Player 1 and player 2 refer to the white car and black car, respectively. The synthesized strategy tells both cars to reach their top speeds, but it also indicates to Player 1 to switch to the center lane. Player 1 is eventually able to build a 0.1 s time gap at the goal state due to its top speed advantage. The screenshots and the plotted real-life example[1] is a straight section of the track that is about several hundred meters. While we don't know the precise states of the players in the real-life video, we still use the video clip to plot the drivers' maneuvers using the timestamps. The tactic employed by the expert driver is similar to the one synthesized by the model in our case study. The driver switches to the center lane and uses the higher top speed and initial velocity to launch past the car that was originally in front.

[1] `https://youtu.be/z0tW3wY758Q?t=57`

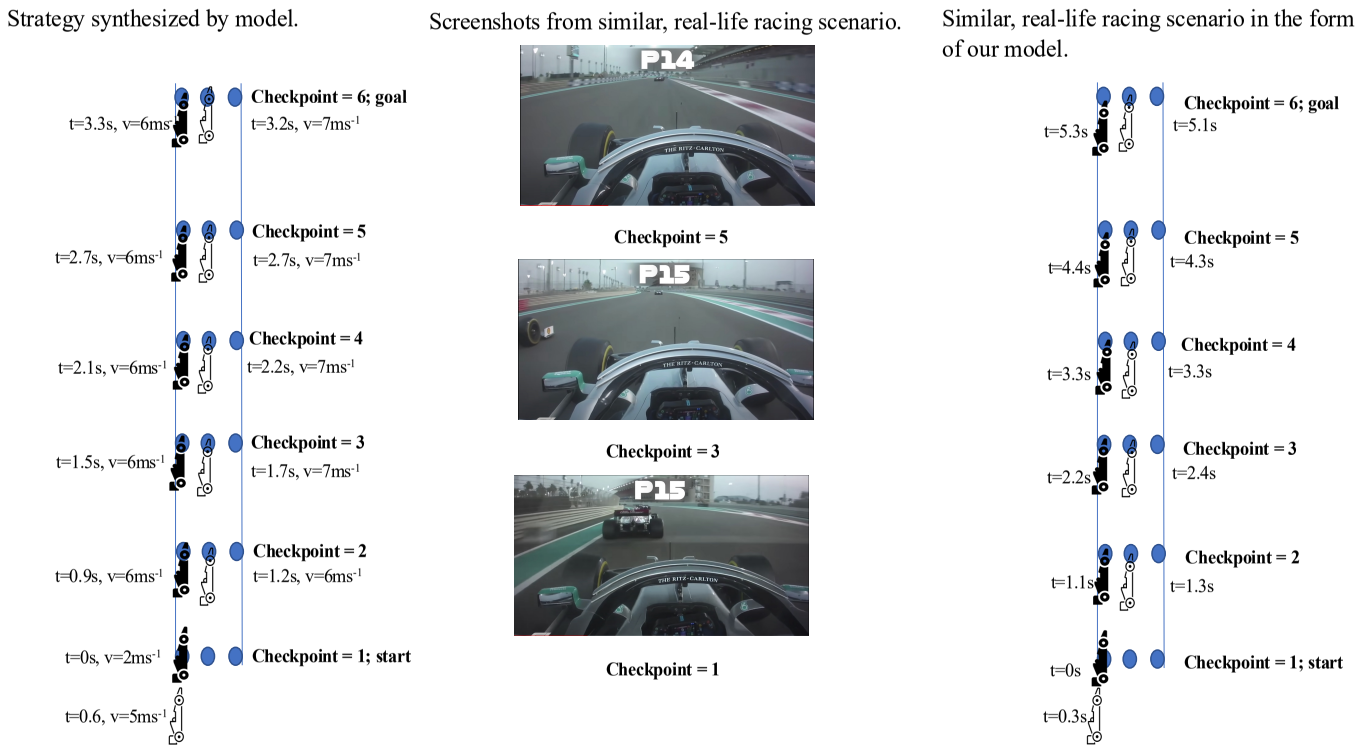

Figure 3.5: Left: Visualized strategy synthesized in our case study scenario. Middle: Screenshots from a real-life race scenario resembling our case study. Right: Extracted real race scenario represented in our model formulation.

3.3.2 Hairpin

The second track shape is a hairpin turn, which is defined as a 180-degree turn. In reality, these types of turns generally follow long straights, but due to state-space explosion, the lead-up to the hairpin turn is limited to just two checkpoints. Furthermore, because the turns are 180 degrees, the cars do not need to travel very fast. Therefore, having better acceleration and lower tire wear results in better performance in this type of track shape, and this pattern is seen in real-life races. In terms of strategy, the player that is initially ahead ideally positions itself in the middle of the straight approaching the turn to make it challenging for the opponent to pass. Then, at the turn, the player targets to be at the innermost lane of the corner to force the opponent to take a wider turn or fall back in line. On the other hand, the player that is initially behind would like to reach the inside lane if at all possible and prevent the car that is ahead from using the inside lane. Tire wear begins to play an important role because it limits how tight of a turn is feasible for the players. Sometimes, the tire wear may be too high such that a player cannot use the innermost lane at the turn. Therefore, in the experiments for this scenario, we fix the initial time disadvantage for Player 1 at 0.5 seconds but use varying combinations of initial velocities and tire wears for the racers.

In Figure 3.6, we plot the results of our experiments. We see that player 2's higher acceleration allows it to stay ahead or only allow racer 1 to catch up but not pass despite having an initial velocity disadvantage. In other words, Player 1's time gap at the final checkpoint is always non-positive. The best

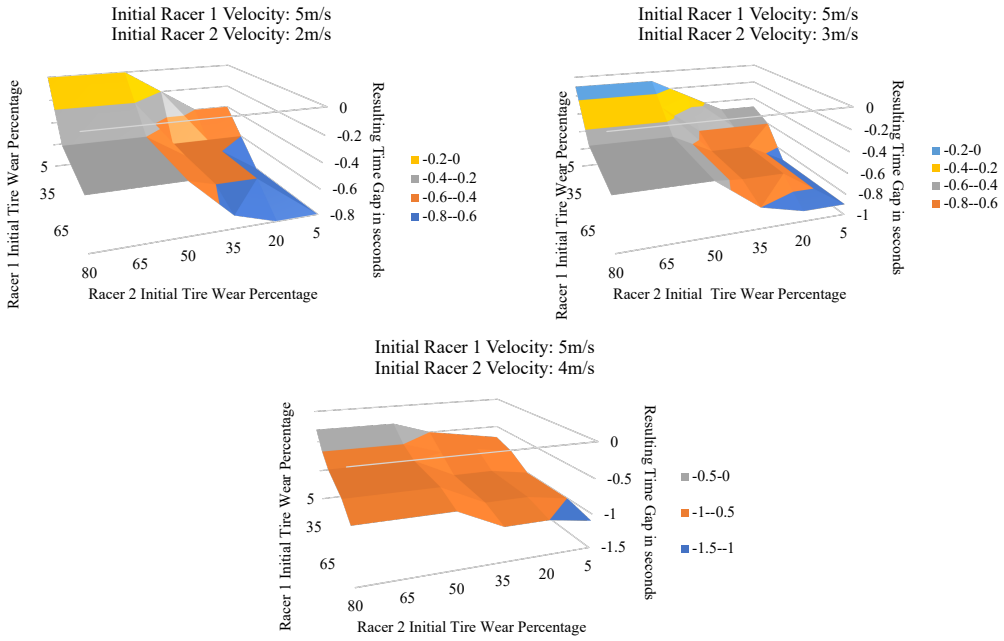

Figure 3.6: Plots for the resulting time gap by implementing both players' synthesized optimal strategy for various initial velocities and tire wear percentages in the hairpin shape.

case for Player 1 is closing the time gap to 0 at the final checkpoint, and it occurs when Player 2 is given a significant tire wear disadvantage. However, when there is a tire wear disadvantage for Player 1, Player 2 still maintains an advantage at the end despite having a lower initial velocity. These trends confirm that the model properly represents the advantages and disadvantages observed in real-life racing.

Next, we take a deeper dive in Figure 3.7 to visualize the synthesized strategy from one of our scenarios. We study the scenario where Player 1's initial velocity is 5 m/s, Player 2's initial velocity is 2 m/s, Player 1's initial tire age is 5%, and Player 2's initial tire age is 50%. The screenshots from the real-life hairpin scenario[2] have other cars in the image, but the two main cars in question are circled in their respective colors matching the model extrapolation. Player 1 and Player 2 refer to the white car and black car, respectively. In the synthesized model strategy, we see that Player 2 immediately shifts over to the innermost lane to prevent any possible plan from the innermost lane because it knows that Player 1 has both the tire age and speed advantage. However, Player 2's state and dynamics prevent it from using the tightest lane in the turn due to its aged tires. As a result, it switches to the middle lane in the corner allowing Player 1 to eventually choose the innermost lane. Upon the exiting the corner (from checkpoints 3 to 4), Player 2 again chooses the innermost line because it is still ahead by 0.1 s. This choice forces Player 1 to use the wide lane and traverse a longer distance. As a result, Player 1 is

[2] `https://youtu.be/CZdsBMhYnT4`

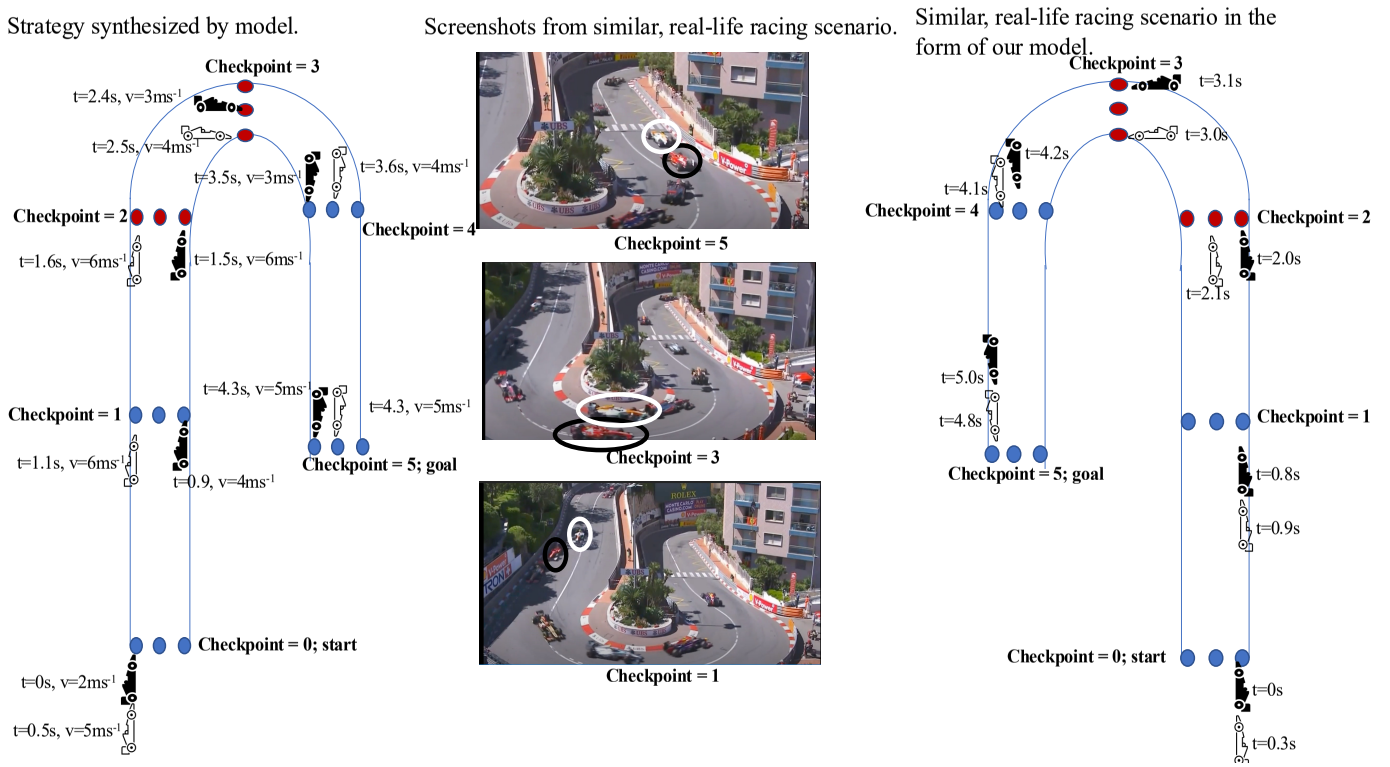

Figure 3.7: Left: Visualized strategy synthesized in our case study scenario. Middle: Screenshots from a real F1 race. Right: Extracted real race scenario represented in our model formulation.

not able to overtake Player 2, but it closes the gap to 0 seconds at the final position. This type of defensive maneuver of covering the inside lane, then going wide in the middle of the turn, and finally taking the inside on the exit of the turn is known as the "switch-back" in racing dialogue. The "switch-back" is commonly used by expert drivers because it sometimes catches the attacking driver off-guard. However, in the real-life video that we compare to, we observe a slightly different strategy being executed. Player 1 passes Player 2 because Player 2 makes a sub-optimal choice and does not cover the inside line at the hairpin. Rather, Player 2 chooses to stay wide throughout the turn allowing the fast approaching Player 1 to take the inside lane at the corner and pass Player 2 on the exit.

3.3.3 Chicane

The final track shape we study is the chicane, which is a successive pair of turns in opposite directions (left-right or right-left). The chicane puts a greater emphasis on the tire wear than the prior two shapes, and success depends on the parameters of lateral force limits of a player's vehicle. The inside of one turn is the outside of the following turn, so the racers must also have the low tire wear to maintain high speed over the shortest distance, which is to hit the inside lane of both turns. Otherwise, compromising by selecting the middle lane may allow the other player to pass on the inside of one of the turns. In a similar fashion to the hairpin case study experiments, we fix the initial time disadvantage for Player 1 at 0.5 s and use varying combinations of

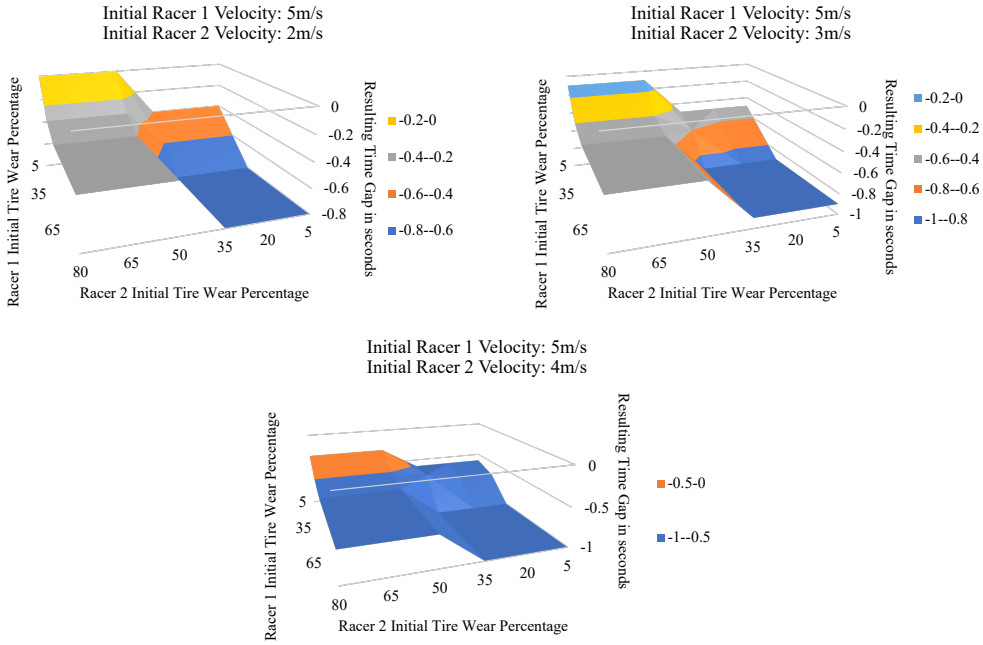

Figure 3.8: Plots for the resulting time gap by implementing both players' synthesized optimal strategy for various initial velocities and tire wear percentages in the chicane shape.

initial velocities and tire wears for these experiments.

Figure 3.8 shows a negative correlation between the final time gap between Player 1 and Player 2 and player 1's initial tire wear as expected. Furthermore, the lack of symmetry on the surface of the plot shows the increased advantage for Player 2 in some scenarios because it has a higher lateral acceleration limit. When the initial tire wear values are swapped for the racers, Player 2 maintains or builds on the initial 0.5 s time gap more often than Player 1 closes it. Again, the results indicate that model is a good representation of real-life racing because it follows the known patterns seen in real-life racing.

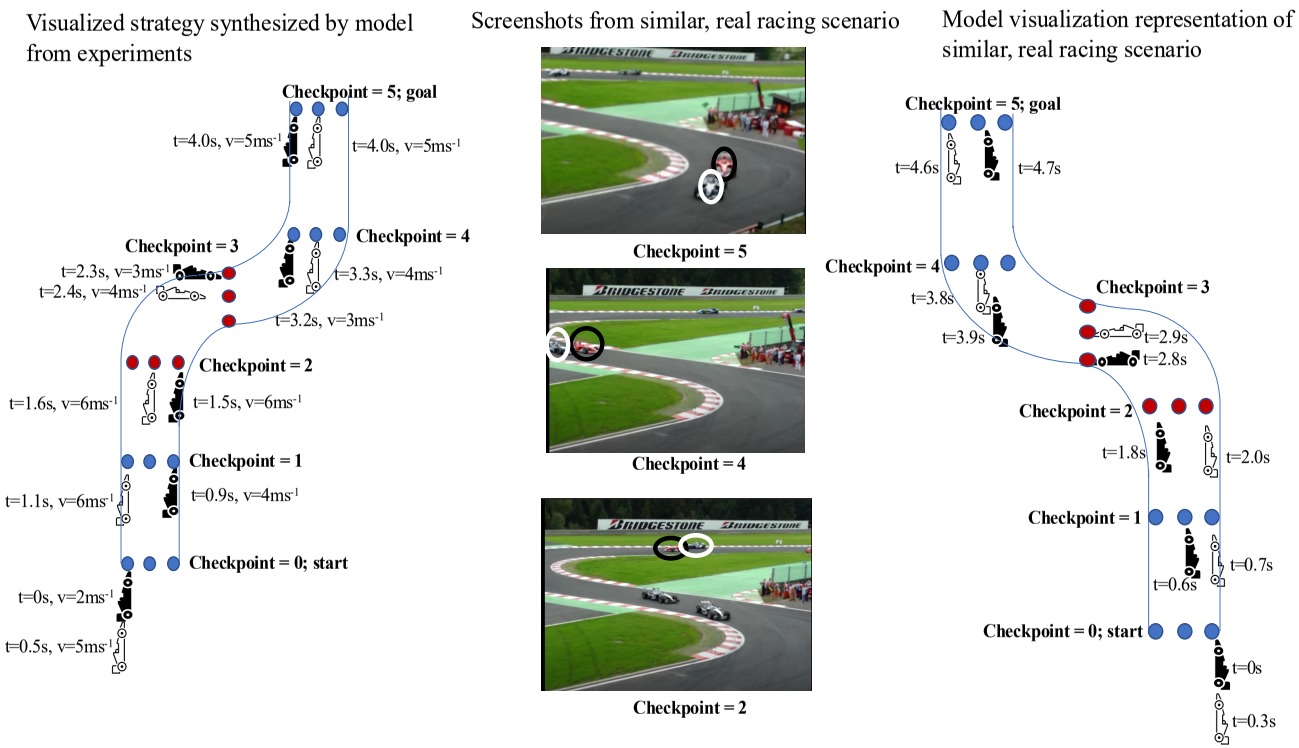

Figure 3.9: Left: Visualized strategy synthesized in our case study scenario. Middle: Screenshots from a real F1 race. Right: Extracted real race scenario represented in our model formulation.

In Figure 3.9, we visualize a synthesized strategy of one of the experiments over the chicane. We study the scenario where the initial velocity of player 1 is $5\,\mathrm{m\,s^{-1}}$, the initial velocity of player 2 is $2\,\mathrm{m\,s^{-1}}$, the initial tire age of player 1 is 5%, and the initial tire age of player 2 is 50%. The screenshots from the real-life chicane scenario[3] have the two main cars circled in their respective colors matching the model extrapolation. Player 1 and Player 2 are the white and black cars, respectively. The synthesized strategy for this scenario presents a typical defensive maneuver where the initial leader, Player 2, selects the inside line of both corners. This choice forces Player 1 to use the wider line for both corners. However, Player 1 has lower tire wear and higher initial velocity, so it improves the time gap and maintains a higher speed despite choosing the unfavorable line. As a result, the initial time difference is reduced to 0 s at the final checkpoint. In the corresponding real-life scenario, we again observe a sub-optimal move by the leading human driver (in black) resulting in him being overtaken by the trailing driver (in white). The leading driver commits to the inside line for the first turn in the chicane but stays wide on the second turn allowing the initially trailing driver to use the inside line at the second turn. As a result, the initially trailing driver has both the advantage of the speed and shorter distance to travel with the inside line of the turn and makes the pass.

Track Shape	Average Model Size	Average Time for Strategy Synthesis
Straight	4200441 states	394 s
Hairpin	2139777 states	200 s
Chicane	2970922 states	273 s

Table 3.2: Summary of model sizes and synthesis times for various track shapes.

3.4 Summary

The results from the experiments show that our model provides a good representation of the prior understanding of advantage and disadvantage patterns in racing. We observe our discrete formulation and a state-of-the-art synthesis algorithm produce maneuvers that are comparable to those executed by professional human drivers while meeting the specification. In other words, the synthesized strategies follow all of the nuanced rules of racing while achieving the goal of reaching the target state before opponents or within the shortest time gap possible. In addition, we use these models to explain how the sub-optimal decisions of the human drivers in the real life-scenarios may have been corrected. The primary limitation of solving the discrete game formulation using formal methods is that it takes about 4 minutes, on average, to produce a result with various inputs we tested. Table 3.2 summarizes the average sizes of the models and average computation time for the experiments in our case studies. Therefore, we cannot directly use the model-checking tool for real-time control of an autonomous racecar suggesting that we consider

[3] `https://youtu.be/_gWkHuOKR9w?t=6`

probabilistic methods to approximate strategies.

Chapter 4

Hierarchical Control for Head-to-Head Racing

> The second act is called
> "The Turn." The magician
> takes the ordinary something
> and makes it do something
> extraordinary.
>
> Christopher Priest, *The Prestige*

In this chapter, we construct our complete hierarchical controller. We explain how the discretized model from the previous chapter is used as the high-level planner and introduce two variants to serve as low-level planners. We finish the chapter by evaluating our hierarchical control variants against baseline methods representing prior works in head-to-head race simulations. This chapter is an extended discussion of work titled, "Hierarchical Control for Multi-Agent Autonomous Racing" co-authored with Aryaman Singh Samyal, David Fridovich-Keil, Zhe Xu, and Ufuk Topcu [34].

4.1 Hierarchical Control Design

Traditional optimization-based control methods cannot easily be utilized for the general multi-agent racing game formulated with realistic safety and fairness rules. The rules involve nonlinear constraints over both con-

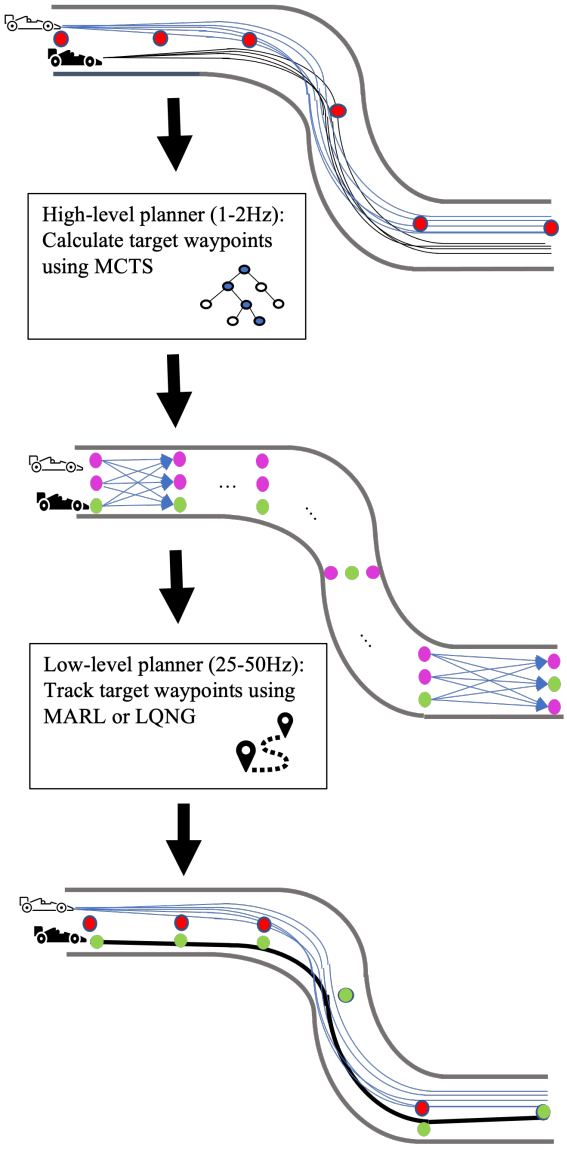

Figure 4.1: The uncountably infinite trajectories of the general game (top) discretized by the high-level planner (middle). The sequence of target waypoints calculated by the high-level planner (in green) is tracked by the low-level planner (bottom) and converges to a continuous trajectory (in black).

tinuous and discrete variables, and a mixed-integer nonlinear programming algorithm to solve the game would be unlikely to run at rates of 15 - 50 Hz for precise control. This inherent challenge encourages utilizing a method such as deep reinforcement learning or trying to solve the game using short receding horizons.

However, we propose a two-level hierarchical control design involving two parts that work to ensure the rules are followed while approximating long-term optimal choices. The high-level planner transforms the general formulation into a game with discrete states and actions where all of the discrete rules are naturally encoded. The solution provided by the high-level planner is a series of discrete states (i.e waypoints) for each player, which satisfies all of the rules. Then, the low-level planner solves a simplified version of the racing game. The simplified version has an objective that places greater emphasis on tracking a series of waypoints and smaller emphasis on the original game-theoretic objective and a reduced version of the rules. Therefore, this simplified game can be solved by an optimization method in real-time or be trained in a neural network when using a learning-based method.

This method assumes if the series of waypoints produced by the high-level planner is guaranteed to follow the rules, then the control inputs generated by the waypoint tracking low-level planner will also satisfy the rules of the original game when applied to the actual underlying system. Figure 4.1 visualizes the overall control architecture.

4.1.1 High-Level Planner

The high-level planner constructs a turn-based discrete, dynamic game that is an approximation of the general game outlined in Section 3.1. We formulate the high-level game using the model developed in Section 3.2 and repeatedly solve it using a receding horizon of 8 checkpoints ahead of the player for real-time control. However, our choice of horizon extends much further into the future than an MPC-based continuous state/action space controller can handle in real-time. For example, the distance covered by 8 checkpoints in our horizon is upwards of 80 meters while the MPC-based continuous controller only plans up to 25-30 meters ahead in [20, 21]. In the following subsections, we briefly summarize how the discrete game is constructed and how we use Monte Carlo tree search to approximately solve the game.

Discrete Game Construction

Constructing our discrete game relies primarily on forming the initial states of the players. Given the initial states, the remaining feasible states and transitions of the game arise naturally by following the dynamics in Section 3.2.2.

To select the initial state of the game, we first construct a subset of the overall set of players based on their distances from the ego player. Because the discrete game is fixed to a finite horizon of checkpoints, the players that would likely have a significant impact on an ego player's trajectory are those who are within some local vicinity of the ego player. Using the set of local players,

the initial checkpoint of the game is selected to be the one that the furthest forward player in the local vicinity has passed, and the target checkpoint is a fixed count ahead of that initial checkpoint.

The continuous components of the players' states are projected into their discrete representations as described in Section 3.2.1. Initial state values for speed, velocity, tire wear, lane ID, and "recent" lane changes are easily extracted from their continuous state. However, because some players may not have reached the selected initial checkpoint of the discrete game, their initial time state must be estimated. We assume every player has knowledge of the times at which each checkpoint was passed by each of the other players. The initial time state is set to 0 for the furthest forward player. Each of the remaining players' initial time states is set based on the time difference between the furthest forward player and the player. Figure 4.2 shows this initial state construction process applied to a 4-player situation from the perspective of Player 1. Player 1 ignores Player 3 in the discrete game because he is out of the range of vicinity. Within Player 1's vicinity, Player 2 is the furthest forward player, so its time state in the discrete game is 0. Player 1 sets its time state as 0.2 s because the last passed checkpoint of both players was checkpoint 2, and the time difference at that checkpoint was 0.2 s. The same logic applies in setting Player 4's initial time state. The remaining state variables are categorized in the discrete buckets based on the player's state at whatever checkpoint they may be at. This simplification may be coarse, but in practice, a player's vicinity range is relatively short. Therefore, the players' states do

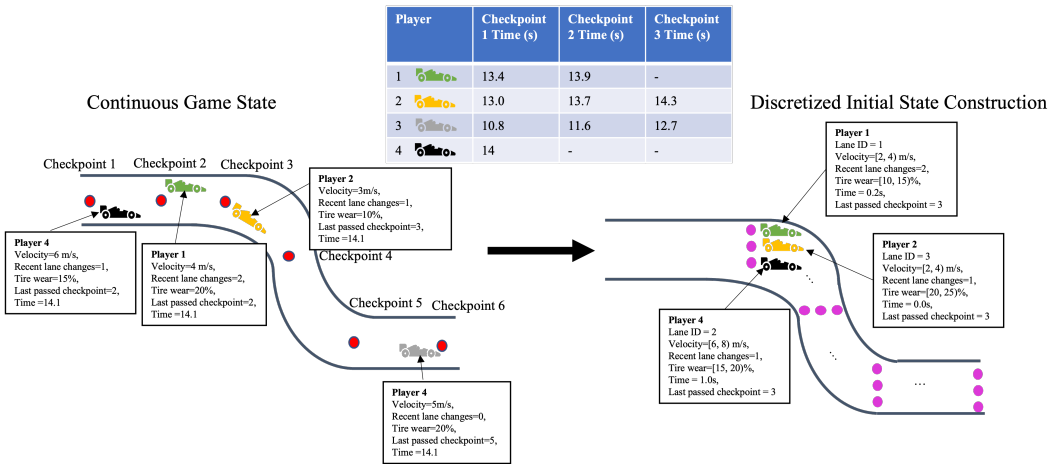

Figure 4.2: Using the checkpoint time table (top) and the continuous states (left), Player 1 constructs the initial state of the high-level discrete game (right).

not change significantly within the range.

As mentioned, given the initial states of the players, the remaining construction of the game follows directly from the other parts of the formulation. The objective is the same as described in Section 3.2.3, but we do not use model checking software/algorithms to solve the game for our hierarchical controller. Table 3.2 shows that it takes about 3-4 minutes to solve a single instance using state-of-the-art model checking software. We expect to run the high-level planner at 1-2 Hz.

Monte Carlo Tree Search

Although the discrete game is much simpler than the original formulation, its state space still grows exponentially as the number of actions and

players increases. Therefore, we cannot rely on synthesizing strategies using model checking methods for real-time control. To produce a reasonable strategy for the discrete game in real-time, we use the Monte Carlo tree search (MCTS) algorithm to approximate the solution to our game [35]. Our implementation of the MCTS algorithm is the same as the original with one exception: instead of sampling the feasible action set uniformly in the simulation step, we use a heuristic to bias our sampling towards those actions deemed more likely to be realistic or optimal. Recall that the actions available at each state are pairs of lane ID and velocity in a set A. We sort the action set using the following characteristics of each action in the following priority:

1. Ascending by the elapsed time calculation (3.2.5)
2. Descending by target velocity of the action
3. Ascending by the absolute value of the difference between target lane ID and current lane ID
4. Ascending by the difference between target lane ID and the lane ID of the optimal racing line

Once the actions are sorted, we randomly sample an index to select an action from the sorted set using the following calculation:

$$\text{action index} = [\min(|x|, |A|)], \quad x \sim \mathcal{N}(0, \frac{|A|}{6}) \tag{4.1.1}$$

Though the reward function is the same as the discrete formulation, we normalize it between 0 and 1 when calculating the upper confidence bound scoring in the MCTS algorithm. The solution produced by applying MCTS to our discrete game is a series of waypoints in the form of target lane IDs (which are mapped back to locations on track) and the target velocities at each of the locations. These waypoints are generated for all players considered in the discrete game and are selected assuming everyone selects their optimal choice. The top and middle of Figure 4.1 visualize the original, continuous formulation expanded into a discrete space of lanes (in purple) and highlight target waypoints that might be calculated by MCTS (in green).

4.1.2 Low-Level Planner

Low-Level Simplified Game Formulation

The low-level planner is responsible for producing the control inputs, so it must operate at even higher frequencies than the high-level planner. Because we have a long-term plan in the form of target waypoints from the high-level planner, we formulate a reduced version of the original game for our low-level planner. The low-level game is formulated over a shorter horizon compared to the original game of just δ steps in $\hat{\mathcal{T}} = \{1, ..., \delta\}$. We assume that the low-level planner for player i has received k waypoints, $\psi^i_{r^i_1}, ..., \psi^i_{r^i_1+k}$, from the high-level planner. Lastly, we also refer to player i's last passed checkpoint as r^i_*.

The low-level objective involves two components. The first is to max-

imize the difference between its own checkpoint index and the opponents' checkpoint indices at the end of δ steps. The second is to minimize the tracking error, η_y^i, of every passed waypoint ψ_y^i. The former component influences the player to pass as many checkpoints as possible, which effectively pushes the player to reach c_τ, the final checkpoint, as quickly as possible. The latter influences the player to be close to the high-level planner's target waypoints when passing each of the checkpoints. The objective also includes a multiplier α that balances the emphasis of the two parts. The objective of player i is written as follows:

$$\min_{u_1^i,\ldots,u_\delta^i} (\sum_{j\neq i}^{N} r_\delta^j - (|N|-1)r_\delta^i) + \alpha \sum_{c=r_1^i}^{r_1^i+k} \eta_c^i \tag{4.1.2}$$

The players' continuous state dynamics, calculations for each checkpoint, and constraints on staying within track bounds (4.1.3)-(4.1.6) are effectively the same as the original formulation. For all players $j \in N$, the following must hold:

$$x_{t+1}^j = f(x_t^j, u_t^j), \quad \forall\ t \in \hat{\mathcal{T}} \tag{4.1.3}$$

$$r_{t+1}^j = p(x_{t+1}^j, r_t^j), \quad \forall\ t \in \hat{\mathcal{T}} \tag{4.1.4}$$

$$r_1^j = r_*^j \tag{4.1.5}$$

$$q(x_t^m) \leq w, \quad \forall\ t \in \hat{\mathcal{T}} \tag{4.1.6}$$

The collision avoidance rules are simplified to just maintaining a minimum separation s_0 as the high-level planner would have already considered

the nuances of rear-end collision avoidance responsibilities outlined in (3.1.8). As a result, we require the following constraint to hold for all $t \in \hat{\mathcal{T}}$, $j \in N$, and $k \in N \setminus \{j\}$:

$$d(x_t^j, x_t^k) \geq s_0 \tag{4.1.7}$$

Finally, we define the dynamics of the waypoint error, η_y^i, introduced in the objective. It is equivalent to the accumulated tracking error of each target waypoint that player i has passed using a function $h : X \times X \to \mathbb{R}$ that measures the distance. If a player has not passed a waypoint, then the error variable indexed by that waypoint is set to 0. Its dynamics are expressed by the following constraint:

$$\eta_y^i = \begin{cases} \sum_t^{\hat{\mathcal{T}}} h(x_t^i, \psi_c^i) & \text{if } \exists\ r_t^i \geq y \\ 0 & \text{otherwise} \end{cases} \qquad \forall\ y \in \{r_1^i, ..., r_1^i + k\} \tag{4.1.8}$$

This simplified formulation (4.1.2)-(4.1.8) is similar to the general formulation (3.1.1)-(3.1.10). However, the constraints introduced by the complex fairness and safety rules are dropped since they are considered by the high-level planner. The middle and bottom of Figure 4.1 show how the target waypoints from the high-level planner (in green) are approximately tracked by a low-level planner to produce a continuous trajectory (in black).

We consider two computational methods to solve this low-level formulation. The first method develops a reward and an observation structure to represent this simplified formulation for a multi-agent reinforcement learning

(MARL) algorithm to train a policy that serves as a controller. The second method further simplifies the low-level formulation into a linear-quadratic Nash game (LQNG) to compute the control inputs. In the following subsections, we describe each of these low-level control methods.

Multi-Agent Reinforcement Learning Controller

Designing the MARL-based controller primarily involves shaping a reward structure that models the low-level formulation. While we provide a high-level description of the reward and penalty behaviors below, Appendix B includes specific details about the reward functions and when they are applied. In general, the RL agent is rewarded for the following behaviors that would improve the objective function from the low-level formulation (4.1.2):

- Passing a checkpoint with an additional reward for being closer to the target lane and velocity.
- Minimizing the time between passing two checkpoints.
- Passing as many checkpoints in the limited time.

On the other hand, the agent is penalized for actions that would violate the constraints:

- Swerving too frequently on straights (3.1.10).
- Going off track or hitting a wall (4.1.6).

- Colliding with other players (4.1.7) with additional penalty if the agent is responsible for avoidance (3.1.8).

The rewards capture our low-level formulation objective (4.1.2) to pass as many checkpoints as possible while closely hitting the lane and velocity targets (4.1.8). The penalties capture the on-track (4.1.6) and collision avoidance (4.1.7) constraints. Note that the penalties reintroduce the original safety and fairness from the original general game that were simplified away from the low-level formulation (3.1.8) and (3.1.10). Because these rules are inherently met by satisfying the objective of reaching the high-level planner's waypoints, their penalties have the weights set to have a lower impact than other components of the reward structure. We also justify incorporating the original form of these penalties because learning-based methods provide the freedom to easily encode these rules. Moreover, we are robustifying the low-level agent against the possibility that the ego player might be forced to deviate far away from the high-level plan in extreme circumstances.

The observation structure of the MARL controller captures the assumption of perfect information we have in all of our game formulations. Therefore, the observation structure consists of the following elements:

- Perfect state information of one's own state and all other player states:
  - Normalized velocity
  - Lane ID

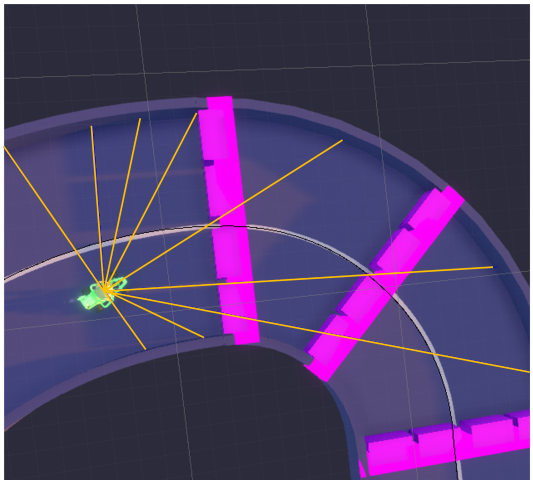

Figure 4.3: A bird's eye view of the LIDAR measurements captured as part of the MARL controller's observation structure.

  - Proportion of max allowed lane changes used
  - Proportion of tire wear
  - Proportion of checkpoints passed
  - For opponent states only:
    * Position in the local coordinate frame
    * Distance to opponent

- 9 Local LIDAR distance observations spaced over a 180° field of view centered in the direction that the player is facing. Figure 4.3 displays the LIDAR rays captured by the MARL agents in our implementation.
- Locations and target velocities of upcoming k waypoints, $\psi^i_{r^i_1}, ..., \psi^i_{r^i_1+k}$, in the local coordinate frame.

Linear-Quadratic Nash Game Controller

Our second low-level approach solves an LQNG using the coupled Riccati equations [36]. This method involves further simplifying the low-level formulation into a structure with a quadratic objective and linear dynamics. The continuous state is simplified to just four variables: x position, y position, v velocity, and θ heading. The control inputs u_t^i are also explicitly broken into acceleration, a_t^i, and yaw-rate, e_t^i. The planning horizon is reduced to $\bar{\delta}$ where $\bar{\delta} \ll \delta < T$. To construct our quadratic objective for player i, we break it into three components. The first is to minimize the squared distance to the upcoming target waypoint from the high-level planner $\bar{\psi}^i$ calculated by the following function of some weight parameters $\rho_1, \rho_2,$ and ρ_3:

$$\upsilon^i(\bar{\psi}^i, \rho_1, \rho_2, \rho_3) = \sum_{t=1}^{\bar{\delta}} (\rho_1((x_t^i - \bar{\psi}_x^i)^2 + (y_t^i - \bar{\psi}_y^i)^2) + \rho_2(v_t^i - \bar{\psi}_v^i)^2 + \rho_3(\theta_t^i - \bar{\psi}_\theta^i)^2) \quad (4.1.9)$$

The second component is to maximize each of the other player's squared distances from the location of their estimated target waypoints $\bar{\psi}^j$ to effectively hinder their progress. This component is calculated by the following function of the waypoint estimated target waypoint $\bar{\psi}^j$ and a weight parameter ρ:

$$\phi^i(\bar{\psi}^j, \rho) = \sum_{t=1}^{\bar{\delta}} \rho((x_t^j - \bar{\psi}_x^j)^2 + (y_t^j - \bar{\psi}_y^j)^2) \quad (4.1.10)$$

We drop all of the constraints with the exception of collision avoidance, and it is incorporated as the third component and penalty term in the objective

where the distance to all other players should be maximized. This term is calculated by the following function of the opponent's position (x_t^j, y_t^j) and a weight parameter ρ:

$$\chi^i(x_t^j, y_t^j, \rho) = \sum_{t=1}^{\bar{\delta}} \rho((x_t^j - x_t^i)^2 + (y_t^j - y_t^i)^2) \tag{4.1.11}$$

The final quadratic objective for a player i on team μ aggregates (4.1.9)-(4.1.11) using weight multipliers (ρ_i) to place varying emphasis on the components as follows:

$$\begin{aligned}\min_{a_1^i, e_1^i, \ldots, a_{\bar{\delta}}^i, e_{\bar{\delta}}^i} \upsilon^i(\rho_1, \rho_2, \rho_3) + \sum_{j \in \{\mu \setminus \{i\}\}} (\phi^i(\bar{\psi^j}, \rho_4)) \\ - \sum_{j \in \{N \setminus \mu\}} (\phi^i(\bar{\psi^j}, \rho_5)) - \sum_{j \in \{N \setminus \{i\}\}} (\chi^i(x_t^j, y_t^j, \rho_6))\end{aligned} \tag{4.1.12}$$

Finally, the dynamics are time invariant and linearized around initial state $(x_{t_0}, y_{t_0}, v_{t_0}, \theta_{t_0})$ for all players $j \in N$:

$$\begin{aligned}\begin{bmatrix} x_{t+1}^j \\ y_{t+1}^j \\ v_{t+1}^j \\ \theta_{t+1}^j \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cos(\theta_{t_0}^j)\Delta t & -v_{t_0}^j \sin(\theta_{t_0}^j)\Delta t \\ 0 & 1 & \sin(\theta_{t_0}^j)\Delta t & v_{t_0}^j \cos(\theta_{t_0}^j)\Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t^j \\ y_t^j \\ v_t^j \\ \theta_t^m \end{bmatrix} \\ + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} a_t^j \\ e_t^j \end{bmatrix}\end{aligned} \tag{4.1.13}$$

We use linear time-invariant dynamics for the purpose of simplicity. It is possible to estimate future states by interpolating the high-level plan to produce time-varying dynamics. However, situations may arise where a player

is not near the interpolated trajectory that would require the interpolation calculations to be robust to such situations. In addition, the low-level planner runs at high frequencies, so we assume the dynamics should hold at the short horizon for which the calculations are performed.

4.1.3 Summary of Control Structure

Algorithm 1 outlines the main control loop of the controller. The control loop is executed at a frequency greater than or equal to that of the low-level planner. Because the rate of the high-level plan is lower than the rate of the main control loop, we just construct the initial game states and run the MCTS calculations on a background thread, asynchronously updating the target waypoints after the calculations are complete. For the low-level calculations, parallelism is not necessary because they are fast enough to meet the deadline of the main control loop.

The high-level planner is paired with each of the two low-level planners to produce two hierarchical controllers. We refer to our two hierarchical variants as MCTS-RL and MCTS-LQNG.

4.2 Experimental Setup

4.2.1 Baseline Agents

To measure the importance of our design innovations, we also consider three baseline controllers that resemble common planning and control methods developed in prior works.

Algorithm 1 Control loop for hierarchical controller for a player i.

$x^1, \ldots, x^{|N|} \leftarrow$ Current states of all players
$f_h \leftarrow$ High-level control frequency
$f_l \leftarrow$ Low-level control frequency
$t \leftarrow$ Elapsed time
if $r^i \neq c_\tau$ **then** ▷ Only run until we reach final checkpoint.
  if $t \mod 1/f_l = 0$ **then** ▷ Low-level control loop
    if MARL Planner **then**
      $o \leftarrow \textsc{CollectObservations}(x^1, \ldots, x^{|N|})$
      $u_t^i \leftarrow \textsc{ComputeActions}(o)$
    else if LQNG Planner **then**
      $\bar{\psi}^1, \ldots, \bar{\psi}^{|N|} \leftarrow \psi^i_{r_1^i}, \ldots, \psi^{|N|}_{r_1^{|N|}}$
        ▷ Latest estimate of upcoming waypoint of each player.
      $u_t^i \leftarrow \textsc{SolveLQNG}(\bar{\psi}^1, \ldots, \bar{\psi}^{|N|})$
    end if
  end if
  if $t \mod 1/f_h = 0$ **then** ▷ High-level control loop
    $\tilde{N} \leftarrow \textsc{NearbyPlayers}^i(x^1, \ldots, x^{|N|})$
    $\tilde{x^1}, \ldots, \tilde{x^{|\tilde{N}|}} \leftarrow \textsc{DiscreteProjection}(\tilde{N})$
    $\psi^i_{r_1^i}, ..., \psi^i_{r_1^i+k} \leftarrow \text{MCTS}(\tilde{x^1}, \ldots, \tilde{x^{|\tilde{N}|}}, \text{time limit} = 1/f_h)$
      ▷ Run MCTS in background thread and save target waypoints.
  end if
end if

End-to-End Multi-Agent Reinforcement Learning

The end-to-end MARL controller referred to as "E2E," represents the pure learning-based methods such as those developed by Wurman et al. and Schwarting et al. [24, 22]. This controller has a similar reward/penalty structure as our low-level controller, but its observation structure is slightly different. Instead of observing the sequence of upcoming states as calculated by a high-level planner, E2E only receives the subsequence of locations from $\{c_i\}_{i=1}^{\tau}$ that denote the checkpoints at the center of the track near the agent. As a result, it is fully up to its neural networks to learn how to plan strategic and safe moves.

Fixed Trajectory Linear-Quadratic Nash Game

The fixed trajectory LQNG controller, referred to as "Fixed-LQNG," uses the same LQNG low-level planner as our hierarchical variant, but it tracks a fixed trajectory around the track instead of using a dynamic high-level planner such as our discrete game. This fixed trajectory is a racing line that is computed offline for a specific track using its geometry and parameters of the vehicle as seen in prior works [8, 11]. Furthermore, in the prior works, the method was only applied to single agent racing scenarios, whereas we use the game-theoretic LQNG controller and apply it to multi-agent racing.

Fixed Trajectory Multi-Agent Reinforcement Learning

The fixed trajectory MARL controller, referred to as "Fixed-RL," is a learning-based counterpart to Fixed-LQNG. Control inputs are computed using a deep RL policy trained to track the precomputed target waypoints that are fixed before the race.

4.2.2 Controller Implementation

Our controllers are implemented[1] in the Unity Game Engine. Screenshots of the simulation environment are shown in Figure 4.4. We extend the Karting Microgame template provided by Unity [37] to model our vehicles and racing environments.

The kart physics from the template utilizes Unity's built-in physics engine by simply updating the kart's velocity and yaw rate at each control step using the acceleration and steering angle as inputs. However, to incorporate cornering limitations and tire wear modeling we modify the calculations. Tire wear proportion e is modeled as an exponential decay curve that is a function of the accumulated angular velocity endured by the kart. This model captures the concept of losing grip as the tire is subjected to increased lateral loads.

The velocity update of the kart is clamped between 0 and $\min(v_*, v_{\max})$ where $v_{\max}$ is the kart's maximum allowed top speed from the fixed parameters of the kart, and v_* is the maximum allowed velocity based on the kart's evolved

[1] `https://www.github.com/ribsthakkar/HierarchicalKarting`

state and turning radius. The calculation for v_* is used in Section 3.2.2 of the discrete game formulation to approximate feasibility of certain actions depending on the shape of the track. It is calculated by first linearly interpolating the minimum lateral acceleration $a_{\max}$ and maximum lateral acceleration $a_{\max}$ by the proportion of tire wear e to compute the maximum allowed lateral acceleration for the kart a_*. Then a_* is used with turning radius of the vehicle r and standard formulas of circular motion to compute v_*. The calculations are summarized by the following pair of equations:

$$a_* = a_{\max} - (a_{\max} - a_{\min})e \tag{4.2.1}$$

$$v_* = \sqrt{a_* r} \tag{4.2.2}$$

Multi-agent support is also added to the template to race the various autonomous controllers against each other or human players. The high-level planners run at 1 Hz, and low-level planners run at 13 -50 Hz. $\bar{\delta}$, the planning horizon for the LQNG planner, is set to 0.06 s for all controllers using it. The implementation of the learning-based agents utilizes a library called Unity ML-Agents [38]. All of the learning-based control agents are trained using proximal policy optimization and self-play implementations from the library. They are also only trained on two sizes of oval-shaped tracks with the same number of training steps.

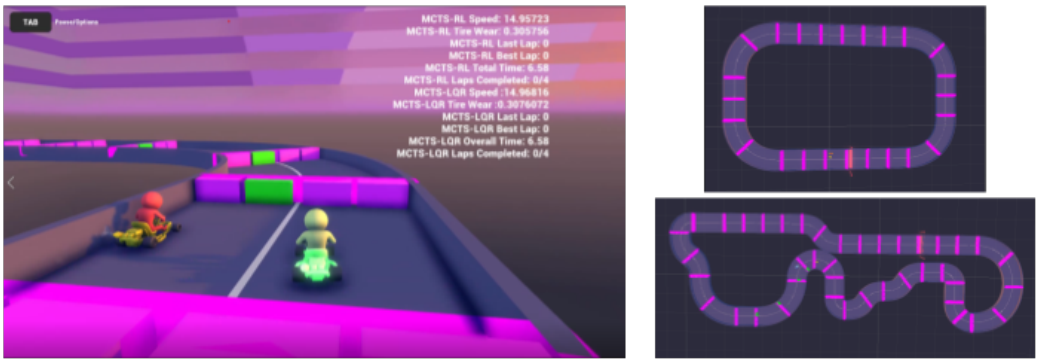

Figure 4.4: Kart racing environment from a racer's perspective (left), a bird's eye view of the oval track (right-top), and the complex track (right-bottom) in the Unity environment. The purple boxes visualize the lanes across checkpoints along the track, and the highlighted green boxes show planned waypoints determined by the hierarchical controllers.

4.3 Head-to-Head Racing Results

Our experiments include head-to-head racing on a basic oval track (which the learning-based agents were trained on) and a more complex track shown in Figure 4.4. Specifically, the complex track involves challenging track geometry with turns whose radii change along the curves, tight U-turns, and turns in both directions. To be successful, the optimal racing strategy requires some understanding of the shape of the track along a sequence of multiple turns. Every pair of controllers competes head-to-head in 50 races on both tracks. The dynamical parameters (cornering limitations, top speed, acceleration, braking, etc.) of each player's vehicle are identical, and the players start every race at the same initial checkpoint with the same initial tire wear of 20%. The only difference in their initial states is the lane in which they start.

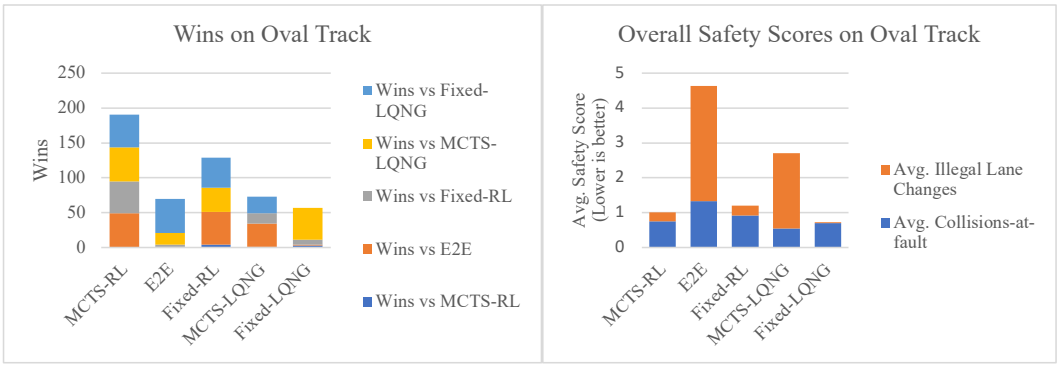

Figure 4.5: Results from head-to-head racing on the oval track.

To maintain fairness with respect to starting closer to the optimal racing line, we alternate the starting lanes between each race for the players.

Our experiments primarily seek to identify the importance of hierarchical game-theoretic reasoning and the strength of MCTS as a high-level planner for racing games. We count the number of wins against each opponent, average collisions-at-fault per race, average illegal lane changes per race, and a safety score (a sum of the prior two metrics) for the controllers. Appendix C provides a complete breakdown of all of the head-to-head results for every pair of agents. We also provide a video[2] demonstrating our controllers in action.

Based on the plots in Figure 4.5 and Figure 4.6 and Table 4.1, we conclude the following key points:

1. The proposed hierarchical variants outperformed their respective baselines.

[2]https://www.youtube.com/playlist?list=PLEkfZ4KJSCcG4yGWD7K5ENGXW5nBPYiF1

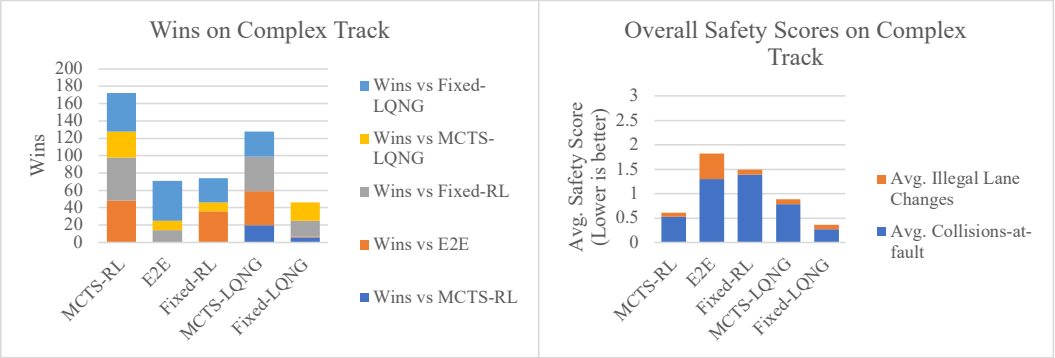

Figure 4.6: Results from head-to-head racing on the complex track.

Aggregate Results (400 Total Races)	**Wins**	**Average Safety Score**
MCTS-RL	363	0.805
E2E	141	3.23
Fixed-RL	203	1.35
MCTS-LQR	201	1.80
Fixed-LQR	83	0.542

Table 4.1: Aggregated head-to-head racing results across both tracks.

The results amongst MCTS-RL, Fixed-RL, and E2E show the effectiveness of our hierarchical structure. While all three of the MARL-based agents were only trained on the oval track, the MCTS-RL agent won the most head-to-head races across both tracks compared to the two baselines despite being trained with the same environmental setup. Furthermore, MCTS-RL's safety score is significantly better than both E2E's and Fixed-RL's safety scores. Comparing the baselines against each other, Fixed-RL also has more wins than E2E aggregated across both tracks. This result indicates that some type of hierarchical structure is favorable. It suggests that a straightforward task

of trajectory tracking is easier to learn and more effectively generalized for a deep neural network than having to learn both strategic planning and respect for the safety and fairness rules. Specifically, while learning collision avoidance was reasonably captured by all of the MARL-based agents, E2E struggled with learning the illegal lane changes rule. On the other hand, the hierarchically structured agents did not have to focus on learning such a complex rule because they rely on a provided trajectory that is assumed to follow the lane-changing rules. However, while a hierarchical structure helps with safety, it is not enough to win the races. The controller must consider multiple trajectories to be able to overtake and defend against opponents as MCTS-RL does, unlike Fixed-RL.

Next, we compare MCTS-LQNG and Fixed-LQNG. Although MCTS-LQNG has a worse overall safety score, it has 25% more wins when aggregated over both tracks. Furthermore, the main difference in their safety score is attributed to the fact that MCTS-LQNG considers multiple trajectories whereas Fixed-LQNG only follows a fixed trajectory that does not involve changing lanes often. Therefore, it is rare for Fixed-LQNG to break the lane-changing rule. When just comparing their collision-at-fault counts, they are similar. Although on the oval track, Fixed-LQNG has a similar number of overall wins, when the racetrack is more complicated, Fixed-LQNG quickly becomes inferior. The oval track has just one main racing line, but there are many reasonable racing lines in the complex track that must be considered to be competitive. MCTS-LQNG accounts for these trajectories by using the

high-level MCTS planner and is, therefore, more successful in its races against MARL-based agents on the complex track with quadruple the number of wins against them compared to the Fixed-LQNG agent. MCTS-LQNG considered trajectories that could result in overtakes when opponents made mistakes from any part of the track. On the other hand, Fixed-LQNG was forced to rely on opponents making mistakes that were not along its fixed trajectory to make overtakes.

2. MARL is more successful and robust than LQNG as a low-level planner.

Overall, the MARL-based agents outperformed their LQNG-based counterparts in terms of both key metrics, wins, and safety scores. However, this result is likely due to the heuristic simplifications in our LQNG design. Vehicle dynamics are only linearized around the initial state of each agent and are time-invariant, meaning the linear approximation is only valid for a very short time horizon. Therefore, the LQNG-based agents could only rely on braking/acceleration instead of yaw-rate to avoid collisions. As a result, the weights in the objective of the LQNG formulation are set conservatively to emphasize avoiding collisions. This setup also implies that LQNG-based agents often concede in close battles and thereby lose races because of the high cost in the planning objective of driving near another player even if there is no collision.

Due to these properties, most of their races were decided over the first few turns of the race. If the LQNG-based agent found itself ahead after the

first several turns, then it usually built a considerable lead, especially Fixed-LQNG. Its opponents could not close this lead because the LQNG controller is very effective at tracking a trajectory with no obstacles. However, more often, the LQNG-based agent would fall behind or collide with an opponent at the start of the races causing it to lose speed. Once it fell behind, it could catch up to its opponents, but it could not successfully overtake unless the opponents made significant mistakes allowing it to pass with a considerable safety margin. As a result, the LQNG-based agents generally resulted in fewer wins compared to their RL-based counterparts. If the dynamics were linearized over the state at each timestep t instead of just the initial state at t_0, we could use a larger time horizon and expect better performance for the LQNG-based agent.

While Fixed-LQNG has a better safety score than Fixed-RL, MCTS-RL has a significantly better safety score than MCTS-LQNG. Just in terms of collision avoidance, both RL-based agents have worse numbers because the LQNG-based agents are tuned to be conservative. However, MCTS-LQNG significantly increased illegal lane changes per race compared to MCTS-RL while Fixed-LQNG has slightly fewer illegal lane changes per race compared to Fixed-RL. As discussed previously, the fixed trajectory agents do not consider alternative racing lines, so they rarely break the lane-changing limit rule in the first place. In the MCTS case, the high-level planner runs in parallel with the low-level and at a lower frequency. As a result, the calculated high-level plan uses slightly out-of-date information and does not account that the low-level controllers have already made choices that might contradict the initial steps

in the plan. This mismatch causes the LQNG-based controller to more often break the lane-changing rules by swerving across the track to immediately follow the high-level plan when it is updated. The MCTS-RL is more robust to this situation because they have those safety rules encoded in their reward structures, albeit with smaller weights. They do not track the waypoints exactly and learn to smooth the trajectory produced by the high-level plan and the live situation in the game.

3. MCTS-RL outperforms all other implemented controllers.

Aggregating the results from both tracks, MCTS-RL recorded a win rate of 83% of the 400 head-to-head and the second-best safety score, only behind the conservatively tuned Fixed-LQNG agent. It combined the advantage of having a high-level planner that evaluates long-term plans and a low-level planner that is robust to the possibility that the high-level plans may be out of date. For example, Figure 4.8 demonstrates how the high-level planner provided a long-term strategy, guiding the agent to give up an advantage at present for a greater advantage in the future when overtaking. The RL-based low-level planner approximately follows the high-level strategy in case stochasticity of the MCTS algorithm yields a waypoint that seems out of place (e.g., the checkpoint between $t = 3$ and $t = 4$ in Figure 4.8). Furthermore, MCTS-RL is also successful at executing defensive maneuvers as seen in Figure 4.7 due to those same properties of long-term planning and low-level robustness. Both of these tactics resemble strategies of expert human drivers in real head-to-head racing.

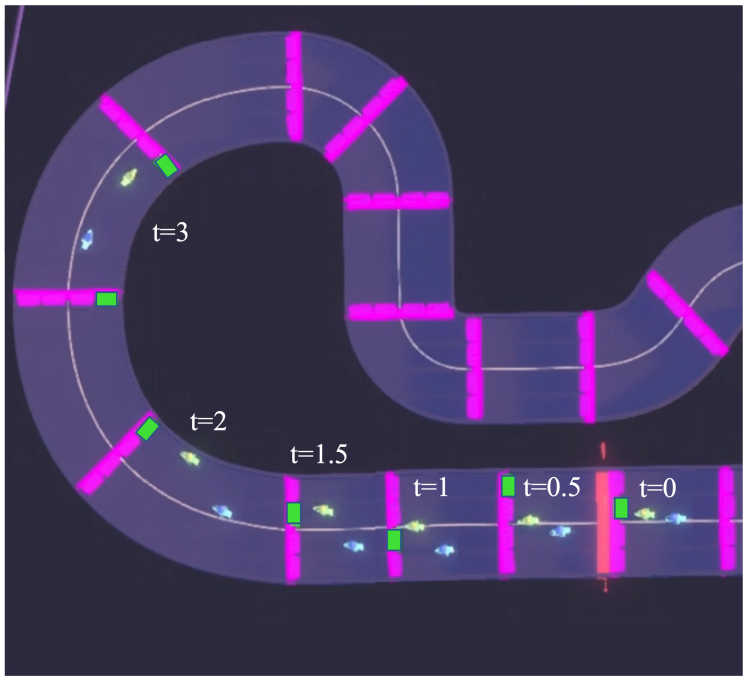

Figure 4.7: A defensive maneuver executed by the MCTS-RL agent (green) against the E2E agent (blue) on the complex track. Before reaching the turn, the MCTS planner calculates to switch lanes to the left first ($t = 0$ to $t = 1$) and then move to the right for the inside of the turn. This motion forces the E2E agent to make an evading move to avoid collision and take an even wider turn, thus increasing the overall gap at the end. The green boxes at each checkpoint highlight the long-term plan calculated by the MCTS planner for this tactic.

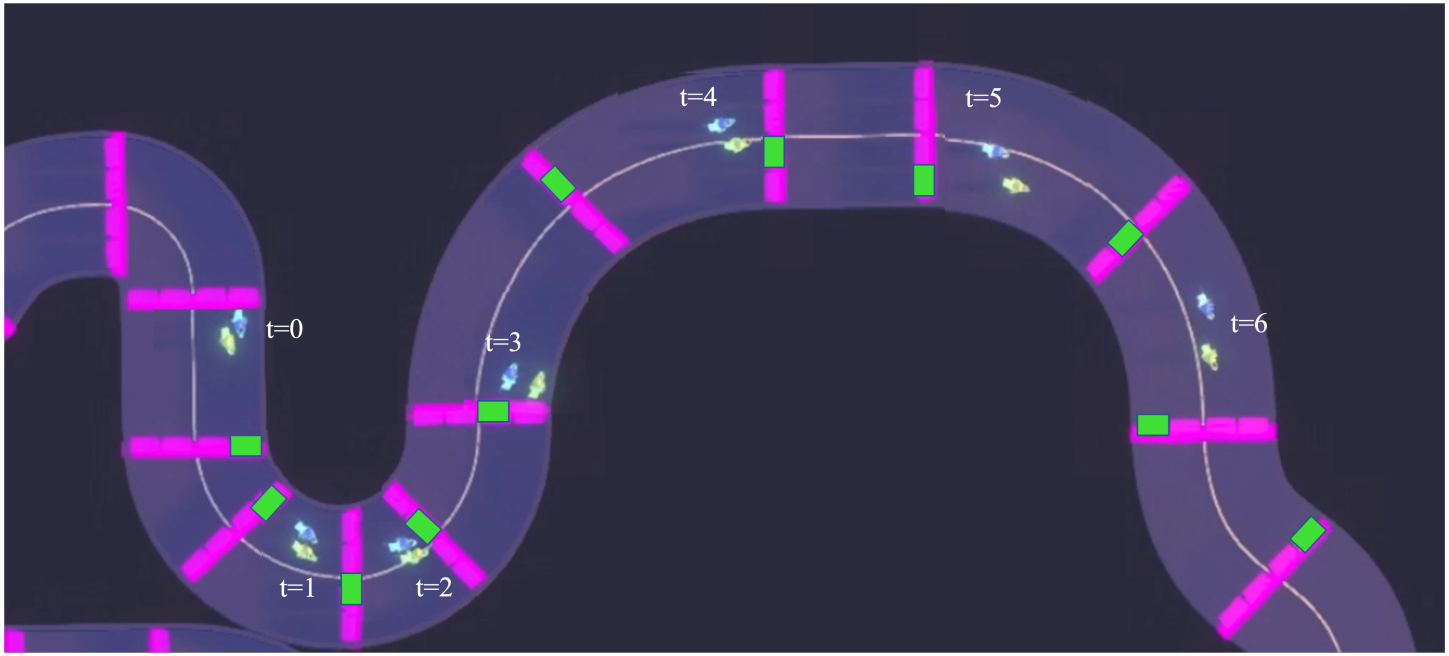

Figure 4.8: An overtaking maneuver executed by the MCTS-RL agent (green) against the E2E agent (blue) on the complex track. Notice how, from $t = 0$ to $t = 2$, the MCTS-RL agent gives up a slight advantage and takes a wider racing line on the first turn. However, the exit of the wide racing line of the first turn places the MCTS-RL agent at the inside of the next two turns where it can gain an even greater advantage when passing the E2E agent from $t = 3$ to $t = 6$. The green boxes at each checkpoint also highlight the long-term plan calculated by the MCTS planner for this tactic.

Chapter 5

Hierarchical Control for Team-Based Multi-Agent Racing

> But you wouldn't clap yet.
> Because making something
> disappear isn't enough; you have
> to bring it back. That's why
> every magic trick has a third act,
> the hardest part, the part we call
> "The Prestige."
>
> Christopher Priest, *The Prestige*

The final part of this report introduces another important facet of real-life racing into consideration: teamwork. Most professional racing series involve two competitions that run concurrently. There is an individual competition amongst the drivers, but there is also a competition over the performance of the racing teams based on the finishing positions of their respective drivers. Therefore, drivers are required to race with a mix of cooperative and competitive objectives in mind. We start by providing a motivating example of a team-based racing scenario where there exist several reasonable strategies, which may not be obvious to choose from. Next, we update the formulations discussed in the prior two chapters to generalize them to a team-based racing scenario. We conclude by evaluating our hierarchical control variants

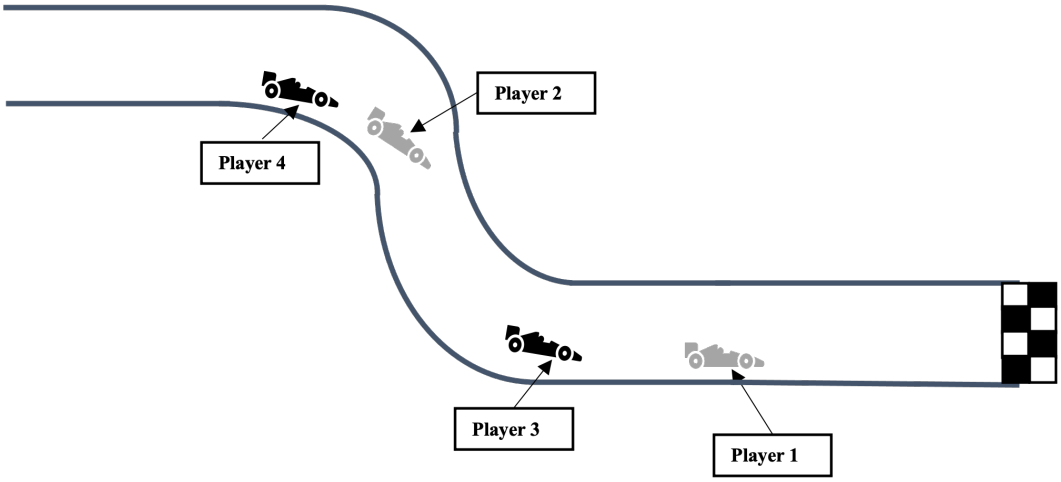

Figure 5.1: Player 3's strategy is unclear. Should it try to pass Player 1, slow down on purpose to try help Player 4 pass Player 3, or maintain its position?

against the baselines introduced in the previous chapter in simulated races of four players with two on each team. This chapter is an extended discussion of work titled, "Hierarchical Control for Cooperative Teams in Competitive Autonomous Racing" co-authored with Aryaman Singh Samyal, David Fridovich-Keil, Zhe Xu, and Ufuk Topcu [39].

5.1 Motivating Example

Consider the example in Figure 5.1. Player 1 and Player 2 are on one team, and Player 3 and Player 4 are on another team. Player 1 is clearly first and almost at the finish line, so it is unlikely that Player 3, who is second, can catch it before the finish line. On the other hand, Player 4 is last, but he is close to Player 2 in third. Player 3 now has three high-level choices to consider:

1. Try to overtake Player 1 before the finish line.

2. Maintain its position to the finish line.

3. Purposely slow down with the risk of being passed by Player 2 but also improve the chances of Player 4 overtaking Player 2 by slowing down Player 2.

If all players were racing independently, choice 1 would likely be the most reasonable because that is only possibility of any payoff. However, because there is an incentive to finish higher overall as a team, Player 3 must consider the payoffs from all three choices. The payoffs and the risks associated with these choices are not necessarily obvious to evaluate. The implications of the choices are not immediately observed, and it is usually challenging to switch from one choice to the next. For example, committing to choice 3 means that Player 3 cannot realistically change its mind and switch to choice 1 if it realizes the risk is too high.

We have discussed in prior chapters how the original formulation, before the introduction of team-based objectives, is already a challenging problem to tackle. Adding these complexities only makes the problem more difficult. Therefore, we propose that using our hierarchical control structure is even more vital in these scenarios as it allows players to evaluate the long-term implications of complex strategies while still adhering to the rules of the game. To study the impact of our method performs in these scenarios, we begin by generalizing all of our racing game formulations to the team-based setting.

5.2 Formulation Updates for Team-based Racing

To adapt each of our formulations for team-based racing, we introduce some additional notation to indicate how the teams are organized and describe how to update the objective function to reflect both competitive and cooperative goals. The constraints in the game remain the same as players still must follow the safety and fairness rules regarding lane changes, collision avoidance, and track constraints.

5.2.1 General Racing Game Formulation

We introduce a set M consisting of mutually exclusive subsets of players in N. Each of the sets in M represents a team of players who have a share the objective of also improving the teams overall finishing positions. We introduce a multiplier $\zeta \in [0, 1]$ to balance a player's emphasis on its team objective vs. its personal objective. For a player i on team μ, the updated objective of the general formulation is as follows:

$$\min_{u_0^i,\ldots,u_T^i} \gamma^i + \zeta(\sum_{j\in\mu\setminus i} \gamma^k) - \frac{(1+\zeta(|\mu|-1))\sum_{j\in N\setminus\mu}\gamma^j}{|N|-|\mu|} \tag{5.2.1}$$

If $\zeta = 0$, then the objective closely resembles the head-to-head case but is different in that the player is only minimizing its pairwise differences in finishing time with all other players, not on its team. If $\zeta = 1$, the player places equal emphasis on its own finishing time with that of all other members of its team. Lastly, this objective function does generalize to the head-to-head racing scenario studied previously because each player's team set would contain 1 just

one element, i.e. $|\mu| = 1$, and the value of ζ would be irrelevant.

5.2.2 High-Level Discrete Game Formulation

The updated discrete game objective has a similar form to the updated objective of the general game formulation because both of have a similar structure in their original form. We reuse the same symbols introduced in the prior section and chapters. The updated reward function for player i in the discrete game on team μ and a team performance multiplier is:

$$R^i(s^1, ...s^n) = \begin{cases} \frac{(1+\zeta(|\mu|-1))\sum_{j\in N\setminus\mu} s_t^j}{|N|-|\mu|} - s_t^i - \zeta(\sum_{j\in\mu\setminus i} s_t^k) & \text{if } (\bigwedge_{p\in N} s_k^p = c_\tau) \\ 0 & \text{otherwise} \end{cases} \tag{5.2.2}$$

The players aim to maximize the difference between a scaled score of all other opponent's finishing times and their teammates' and own finishing times. Similar to the updated objective described in the previous section η balances the cooperative and competitive objectives.

5.2.3 Low-Level Simplified Game Formulation

Even though we do not directly solve the low-level formulation introduced in the previous chapter, we provide an updated objective function here for completeness. In the original low-level formulation, the objective is to maximize the checkpoint difference at the end of the planning horizon. We use the similar trick of multiplying the performance of teammates by a factor ζ and normalizing the scores based on the number of opponents and teammates.

The updated objective for player i on team μ playing the low-level game is:

$$\min_{u_1^i,\ldots,u_\delta^i}\Big(\frac{((1+\zeta(|m|-1))\sum_{j\in N\setminus\mu}^{N} r_\delta^j}{|N|-|m|} - r_\delta^i - \zeta\sum_{j\in\mu\setminus i} r_\delta^j\Big) + \alpha\sum_{c=r_1^i}^{r_1^i+k}\eta_c^i \qquad (5.2.3)$$

Again, there are two parts to the low-level objective. The first part incentivizes players to pass as many checkpoints as possible as a team. The second part incentivizes the individual player to hit its target waypoints as closely as possible. The second part remains unchanged from the original formulation, but the first part considers the team's overall count of passing the checkpoints within the fixed time horizon.

We do not directly solve the low-level simplified formulation, but rather integrate into a MARL-based model and further simplify it into an LQNG representation. Therefore, in the following subsections, we discuss how the implementation of these controllers is updated to represent the updated objective of the low-level game.

Multi-Agent Reinforcement Learning Controller

We make two changes to our MARL model to reflect the changes in the updated low-level formulation. First, we update the reward structure to incorporate the overall team-performance calculations. We introduce a shared reward for all agents of a team using the order in which the checkpoints are passed resembling the points that are awarded to teams based on their driver's finishing positions in real-life racing. Using this shared reward mechanism connects to the second change to the MARL model, which is changing the

learning algorithm used to train the control policy. We use an algorithm called posthumous counterfactual credit assignment) developed by the creators of Unity ML-Agents library [40]. Their algorithm is an extension of the state-of-the-art counterfactual multi-agent policy gradients algorithm developed by Foerster et al. [41]. In their extension, the scientists at Unity modify how team rewards impact the policy after an agent might have reached the end of its life while the other players in the system may still be active. In our case, the end of an agent's life refers to finishing the race.

Linear-Quadratic Nash Game Controller

Updating the LQNG-based controller does not require any explicit changes to the LQNG formulation. In the original LQNG formulation, we introduced a series of multipliers ρ that is multiplied by other players' distances to their upcoming waypoints in the second component of the objective (4.1.10). We ensure the signs of those multipliers for the players on the ego player's team are negative. By doing so, the player will no longer have an objective to hinder those players' progress towards their checkpoints; instead, it will be rewarded for selecting trajectories that aid its teammates in reaching their target waypoints.

5.3 Team-based Racing Results

Our experiments include 2v2 team racing on a basic oval track (which the learning-based agents were trained on) and a more complex track (which

they were not trained on) shown in Figure 4.4. Each team is composed of two players both using one of the five types of implemented controllers, MCTS-RL, MCTS-LQNG, E2E, Fixed-LQNG, and Fixed-RL, constructing five teams. Every pair of teams competes head-to-head in 48 races on both tracks. The dynamical parameters (cornering limitations, top speed, acceleration, braking, etc.) of each player's vehicle are identical. The only difference in their initial states is the lane in which they start and the initial checkpoint. Two of the players start 10 m in front of the other pair resembling the starting grid of real-life racing. To maintain fairness with respect to starting closer to the optimal racing line or ahead of others, we rotate through each of the six unique ways to place players at the four possible starting positions.

Our experiments seek to reinforce the importance of hierarchical game-theoretic reasoning and study its scalability to challenging problems with strategies requiring coordination in addition to competition when long-term planning. We are also interested in observing maneuvers where teammates use tactical positioning to help pass or defend against the opposing team, commonly seen in expert human racing. We assign points to each of the four finishing positions, $[10, 7.5, 6, 4]$ and 0 for not finishing the race. The points are summed at the end of the race for each team. The number of points is important because it measures the overall performance of the team rather than the individual ability of a specific controller. Nevertheless, for each team, we still count the number of wins (i.e. 1st place finishes), average collisions-at-fault per race, average illegal lane changes per race, and a safety score (a sum of

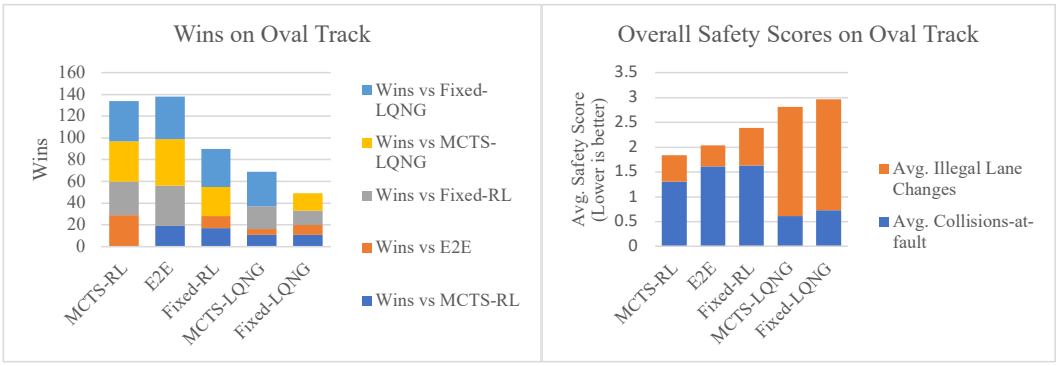

Figure 5.2: Results from head-to-head racing on the oval track.

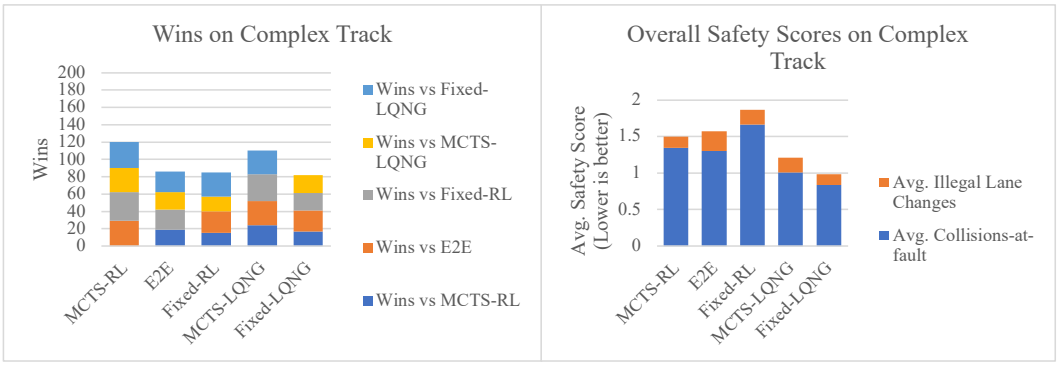

Figure 5.3: Results from head-to-head racing on the complex track.

the prior two metrics) to obtain a holistic view of the implemented controllers. Our analysis focuses on three key metrics: average points per race, wins, and safety score. Appendix D provides tables with a complete breakdown of the results for every pair of teams. We also provide a video[1] demonstrating them in action.

Based on the plots in Figure 5.2 and Figure 5.3 and Table 5.1, we

[1] https://www.youtube.com/playlist?list=PLEkfZ4KJSCcG4yGWD7K5ENGXW5nBPYiF1

Aggregate Results (384 Total Races)	**Wins**	**Average Points Per Race**	**Average Safety Score**
MCTS-RL	254	14.93	1.61
E2E	224	13.20	1.67
Fixed-RL	175	13.51	2.14
MCTS-LQR	175	12.89	1.91
Fixed-LQR	132	12.56	1.78

Table 5.1: Aggregated team-based racing results across both tracks.

conclude the following key points:

1. The proposed hierarchical controllers scale to the challenge of team-based racing by continuing to outperform their respective baselines.

The results amongst MCTS-RL, Fixed-RL, and E2E continue to show the effectiveness of our hierarchical structure. Again, all of the MARL-based agents were trained only on the oval track, but MCTS-RL continues to lead in all of the key metrics. While MCTS-RL has more wins overall, the difference in the number of wins is not as high as in the head-to-head case. However, the essential metric of interest in this study is average points per race, which evaluates team-based performance. MCTS-RL maintains a considerable difference in terms of average points per race compared to the baselines. The higher points per race imply that even if MCTS-RL is not able to finish 1st, it collaborates more effectively to produce better results as a team.

Next, comparing just the baselines, we notice that Fixed-RL is worse in terms of wins and safety score compared to E2E. Recall that the Fixed-RL

controller simply follows a fixed optimal racing line. While such a strategy might be successful in the head-to-head case where there is only one opponent to consider, in the team racing scenario, it is imperative for players to consider alternative racing lines especially if one's teammate is already following a specific line. As a result, Fixed-RL often had collisions with its own teammate as both players competed over the same space. In those situations, one or both of the Fixed-RL teammates sometimes lost a position. However, once they were separated far enough after recovering from the collision, the agents on the Fixed-RL team could both drive fast enough to at least maintain their new positions or independently overtake their opponents, which is reflected in its higher points-per-race score compared to E2E. This pattern implies that hierarchical reasoning is important to be quick, but is not necessarily enough. To be the most successful, game-theoretic hierarchical reasoning, e.g. MCTS, should be used to allow teammates to predict each other's plans and work together effectively.

Finally,, we compare MCTS-LQNG and Fixed-LQNG. Both LQNG agents have similar safety scores. However, MCTS-LQNG still has 33% more wins and a better points-per-race metric overall. Again like the head-to-head case, the main drawback with the fixed trajectory tracking agents is that they do not consider alternative racing lines. While in the head-to-head case considering alternative lines might not be as important, it becomes considerably more vital to success in multi-agent multi-team racing.

2. MARL continues to perform better than LQNG as a low-level

planner but, the difference in the metrics is smaller in team-based racing compared to head-to-head racing.

Across both tracks, the MARL-based agents are still generally better than the LQNG-based agents in terms of our key metrics. However, the difference in their performance is narrowed compared to the head-to-head case. For example, in the complex track, both the LQNG-based agents have better safety scores than their MARL-based counterparts, but in the oval track, the scores are significantly worse due to the number of illegal lane changes by the LQNG-based agents. As discussed in the analysis of the head-to-head racing results in Chapter 4, the LQNG-based controllers are conservatively tuned for collision avoidance due to their short horizons. Therefore, they continue to maintain fewer collisions-at-fault, but this setup also forces them to change lanes more often to avoid collisions, especially given that there are more players in the game. Furthermore, it also results in the LQNG-based agents often conceding in close battles and thereby losing races because of the high cost in the planning objective of driving near another player even if there is no collision. Despite that, MCTS-RL has just 45% more wins in the team-based experiments compared to the 80% more wins it has against MCTS-LQNG in the head-to-head experiments. For the fixed trajectory agents, this gap drops from 250% to 33%. Nevertheless, when we consider our primary metric evaluating team-based performance, points-per-race, both MARL-based variants are better than the LQNG-based variants, but the gap between Fixed-RL and Fixed-LQNG is smaller than the gap between MCTS-RL and MCTS-LQNG.

Overall, all of the metrics are still in favor of using the MARL-based agents because they are generally more robust to nuances of the many possibilities of situations that arise. On the other hand, our LQNG formulation has a mixture of concave and convex components in the objective function, is only linearized around the initial state, and uses short horizons, so our cost surface is sometimes unreliable degrading the resulting behavior as also discussed in the results of the head-to-head experiments.

3. MCTS-RL outperforms all other implemented controllers exhibiting teamwork tactics resembling real-life experts.

The MCTS-RL team records a win rate of over 66% of the 384 races it participated in across both tracks, the best overall safety score, and the highest points per race. The MCTS high-level planners provided the agents with a series of waypoints allowing them to make decisions in complex tactical situations where there is a mix of both competitive and cooperative objectives. The MARL-based low-level planner provided robustness to adapt to the true scenario that plays out. Although the players do not communicate or explicitly coordinate strategies, they still produce cooperative behaviors that improve their overall performance as a team.

We also observe our controllers execute plans that resemble those performed by expert human drivers. For example, Figure 5.4 demonstrates how the two high-level planners of each MCTS-RL agent developed a strategy to perform a pincer-like maneuver to overtake an opponent. Both agents from

the MCTS-RL team approached the opponent from either side of the opponent at the same time. The opponent could only defend one of the agents on the MCTS-RL team allowing the other agent on the team to pass. In addition, MCTS-RL is also successful at executing strategic maneuvers as seen in Figure 5.5 where an agent, who is ahead, momentarily slows down and blocks an opponent behind to allow for its teammate to pass the opponent. Both of these tactics resemble strategies of expert human drivers in real head-to-head racing.

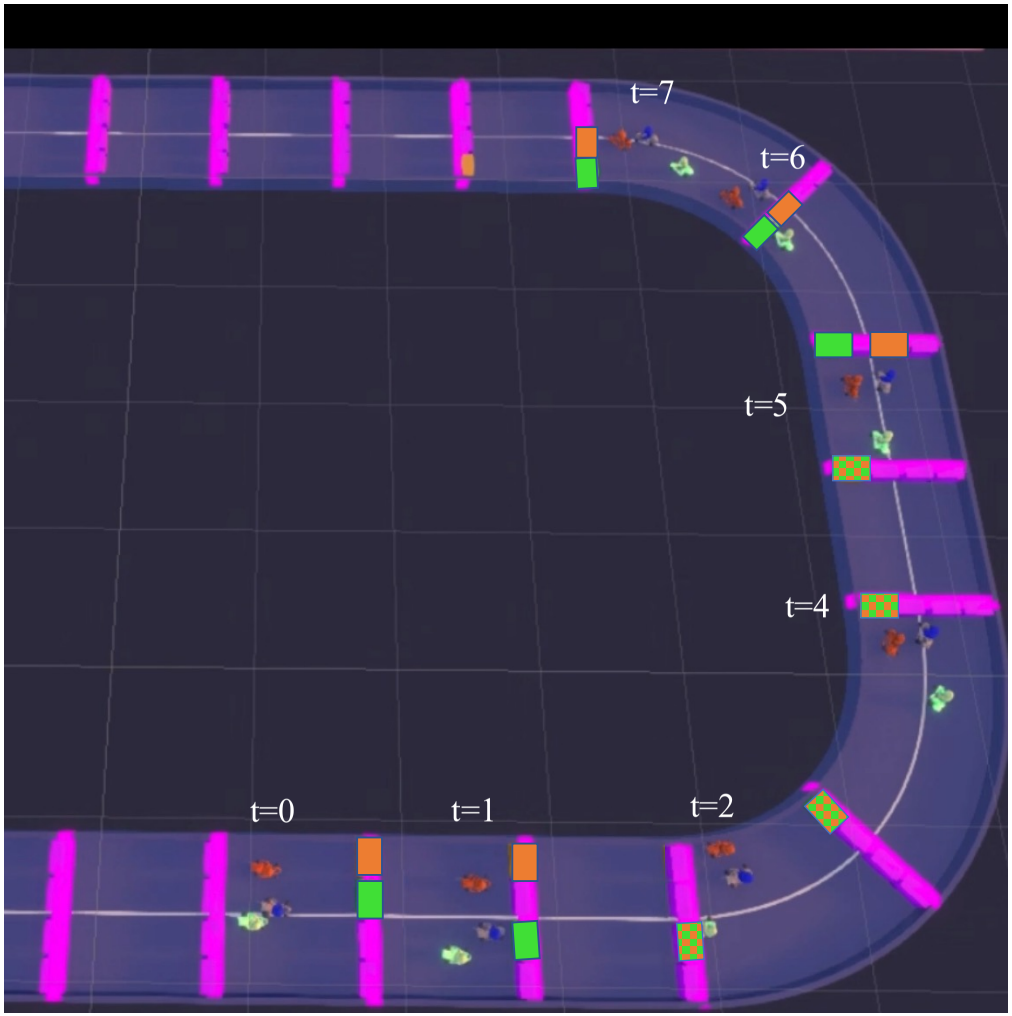

Figure 5.4: An overtaking maneuver executed a team MCTS-RL agents (green and orange) against the Fixed-RL agent (blue) on the oval track. From $t = 0$ to $t = 1$, the MCTS-RL agents split and attack the Fixed-RL agent from both sides. The Fixed-RL agent attempts to defend the green MCTS-RL agent on its right allowing the orange MCTS-RL agent to overtake on its left from $t = 2$ to $t = 6$. The green and orange boxes along each checkpoint highlight the long-term plans calculated by the MCTS planners of each of the MCTS-RL agents, respectively. The checkered boxes indicate a shared checkpoint in their plans.

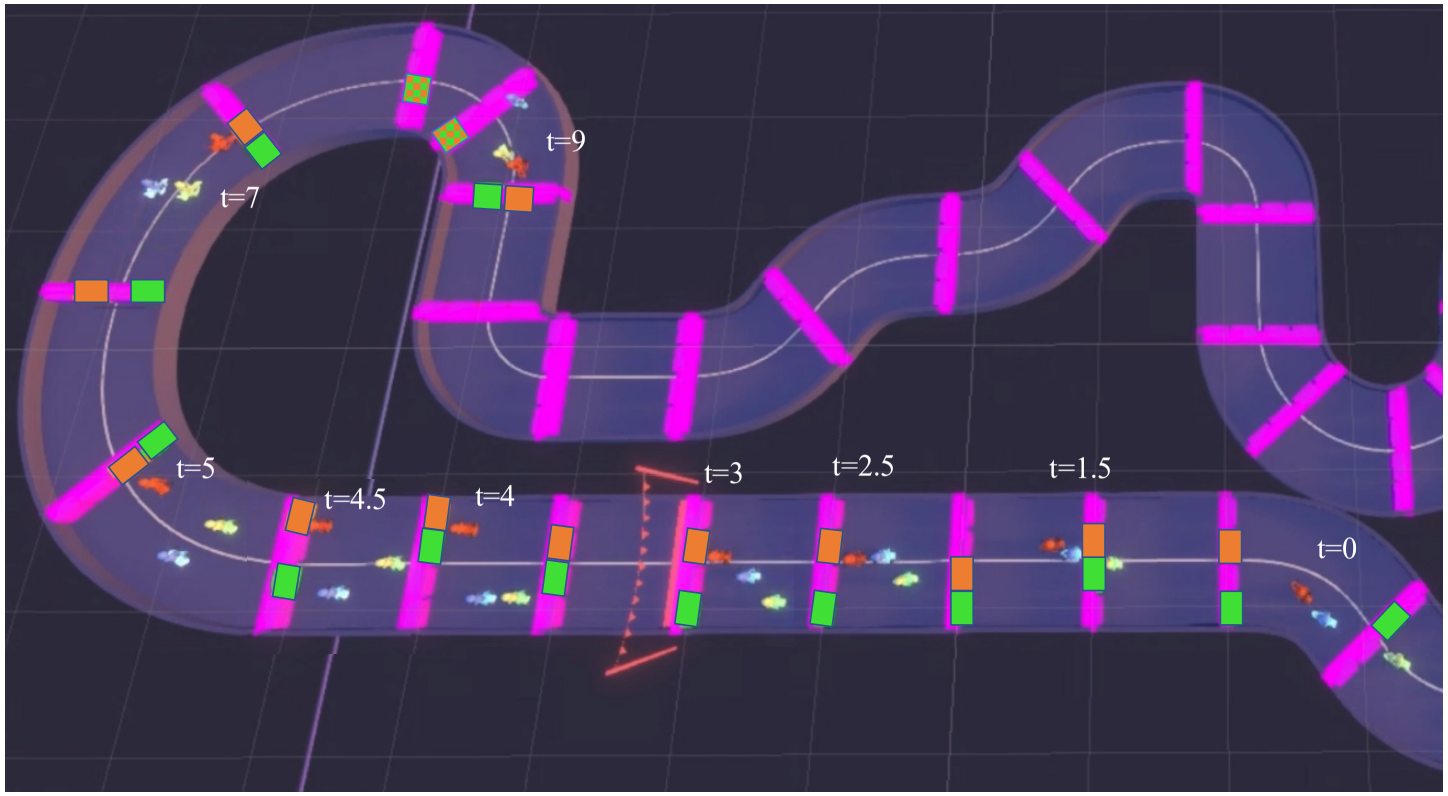

Figure 5.5: A tactical maneuver executed by the MCTS-RL team (green and orange) against an E2E agent (blue) on the complex track. Before reaching the turn, the green MCTS-RL's high-level planner calculates a trajectory suggesting to switch lanes to the left for the first part of the upcoming straight and to block the E2E agent forcing it to slow down and evade further to the left of the track ($t = 0$ to $t = 3$). This blocking move allows the orange MCTS-RL to plan to take advantage of the opponent's disruption and quickly switch to the right for the inside line of the upcoming turn ($t = 3$ to $t = 5$). Eventually, the orange MCTS-RL agent completes the overtake by $t = 9$. The green and orange boxes along each checkpoint highlight the long-term plan calculated by MCTS planners of each of the MCTS-RL agents, respectively. The checkered boxes indicate a shared checkpoint in their plans.

Chapter 6

Conclusion

Autonomous racing serves as a compelling application to develop control algorithms because it involves scenarios with complex rules and objectives. We construct a hierarchical game-theoretic controller for autonomous racing by breaking the challenging problem into two simplified levels. The high-level component encodes the complex rules, which often include constraints over discrete parameters and variables, by constructing a discrete abstraction of the game. Using a discrete representation allows us to consider maneuvers deeper in the future and simplifies the problem for our low-level controller, which just needs to follow the provided long-term plan. We show that hierarchical game-theoretic reasoning produces behavior that is not only competitive but also safer with respect to the rules.

Abstractly, hierarchical reasoning represents how we, as humans, make long-term decisions when taking on a challenging task. It is impossible to consider all of the details and implications for every situation along the way to complete the task. Therefore, we break the problem into a sequence of discrete checkpoints that we consider reaching along the way to achieving our final goal.

6.1 Summary of Contributions

This report introduces the following key contributions:

1. We develop a model to transform a complex head-to-head racing game with realistic safety and fairness rules into a simplified discrete game with temporal logic specifications. This representation is solved by model checking tools to produce strategies that resemble those performed by human experts.

2. We use our discrete game model as a high-level planner and combine it with a pair of low-level controllers using multi-agent reinforcement learning and linear-quadratic Nash game methods to produce a pair of hierarchical controllers. The discrete model is solved in real-time using Monte Carlo tree search to estimate a strategic plan that is tracked by the low-level controllers. Our hierarchical controllers outperform other baseline controllers resembling prior works in autonomous racing research. The hierarchical structure enables them to produce and execute plans that are optimal in the long-term and mirror those seen in real-life racing.

3. We extend our hierarchical control design to a team-based racing game where players have a mix of competitive and cooperative objectives. To our knowledge, this report is the first work to study this problem. We show that our hierarchical controller scales to the even more complex problem and outperforms the baseline methods adapted to the team-

based racing game. Again, our model produces behavior that mimics strategies executed by expert human drivers.

6.2 Future Extensions

Future extensions of this work should introduce additional high-level and low-level planners. Examples of additional low-level controllers include time-varying linear-quadratic approximations or iterative best response methods. With a larger collection of control options, one might investigate policy-switching hierarchical controllers where we switch between various high and low-level controllers depending on the state of the game. For example, we noticed in our experiments that the LQNG-based controllers are reliable in situations where there are no nearby opponents avoiding any risk of issues arising from using a black-box MARL-based controller. Furthermore, if there are no other players nearby, then the high-level control can also switch to following the fixed, optimal line as all other racing trajectories would be sub-optimal. Deciding when to make such switches would possibly require an additional layer of hierarchy.

Lastly, our hierarchical control design can be extended to other multi-agent systems applications where there exist complex rules such as energy grid systems or air traffic control. Constructing a discrete high-level game allows for natural encoding of the complex constraints, often involving discrete components, to find an approximate solution that can warm start a more precise low-level planner.

Appendices

Appendix A

Discrete strategy synthesis case studies

We provide detailed results of the case study scenarios in Chapter 3. Using the initial state of the game and initial time gap (difference between player 2's time state and player 1's time state), the resulting time gap between at the final state of the game is evaluated by applying the the optimal choice for both players for each step in the game.

A.1 Long Straight Results

Init. Time Gap (s)	P1 Init. Tire Wear (%)	P1 Init. Velocity ($m\,s^{-1}$)	P2 Init. Tire Wear (%)	P2 Init. Velocity ($m\,s^{-1}$)	Number of States	Model Checking Time (s)	Resulting Time Gap (s)
-0.5	0	2	0	2	2761897	270.227	-0.5
-0.5	0	3	0	2	4108602	409.806	-0.1
-0.5	0	4	0	2	4397368	425.665	0
-0.5	0	5	0	2	5670812	522.411	0.2
-0.5	0	2	0	3	2786465	272.384	-0.6
-0.5	0	3	0	3	4238686	390.534	-0.2
-0.5	0	4	0	3	4511669	447.354	-0.1
-0.5	0	5	0	3	5495638	510.543	0.1
-0.5	0	2	0	4	2925857	273.566	-0.8
-0.5	0	3	0	4	4241126	390.457	-0.4
-0.5	0	4	0	4	4584613	420.162	-0.3

Init. Time Gap (s)	P1 Init. Tire Wear (%)	P1 Init. Velocity ($m\,s^{-1}$)	P2 Init. Tire Wear (%)	P2 Init. Velocity ($m\,s^{-1}$)	Number of States	Model Checking Time (s)	Resulting Time Gap (s)
-0.5	0	5	0	4	5609633	527.874	-0.1
-0.6	0	2	0	2	2692135	261.436	-0.6
-0.6	0	3	0	2	4104572	404.029	-0.2
-0.6	0	4	0	2	4346639	420.909	-0.1
-0.6	0	5	0	2	5303420	488.12	0.1
-0.6	0	2	0	3	2814772	277.193	-0.7
-0.6	0	3	0	3	4120148	377.282	-0.3
-0.6	0	4	0	3	4462046	410.307	-0.2
-0.6	0	5	0	3	5488421	508.794	0
-0.6	0	2	0	4	2935622	270.973	-0.9
-0.6	0	3	0	4	4122155	379.988	-0.5
-0.6	0	4	0	4	4398427	401.358	-0.4
-0.6	0	5	0	4	5467741	502.284	-0.2
-0.7	0	2	0	2	2719391	265.059	-0.7
-0.7	0	3	0	2	3984071	368.115	-0.3
-0.7	0	4	0	2	4324281	420.799	-0.2
-0.7	0	5	0	2	5299284	490.169	0
-0.7	0	2	0	3	2834115	273.737	-0.8
-0.7	0	3	0	3	4042128	376.66	-0.4
-0.7	0	4	0	3	4316010	398.653	-0.3
-0.7	0	5	0	3	5403780	500.127	-0.1
-0.7	0	2	0	4	2993862	277.193	-1
-0.7	0	3	0	4	3991167	370.752	-0.6
-0.7	0	4	0	4	4333817	396.432	-0.5
-0.7	0	5	0	4	5385497	495.624	-0.3

Table A.1: Full results from various scenarios on long straight track shape.

A.2 Hairpin Results

Init. Time Gap (s)	P1 Init. Tire Wear (%)	P1 Init. Velocity ($m\,s^{-1}$)	P2 Init. Tire Wear (%)	P2 Init. Velocity ($m\,s^{-1}$)	Number of States	Model Checking Time (s)	Resulting Time Gap (s)
-0.5	5	3	5	2	3794001	351.878	-0.7
-0.5	5	3	20	2	3141156	291.433	-0.6
-0.5	5	3	35	2	2512129	231.642	-0.6
-0.5	5	3	50	2	1907399	177.964	-0.3
-0.5	5	3	65	2	1907399	182.887	-0.3
-0.5	5	3	80	2	1907399	177.179	-0.3
-0.5	20	3	5	2	2858591	267.035	-0.8
-0.5	20	3	20	2	2395431	224.431	-0.7
-0.5	20	3	35	2	1867182	174.599	-0.7
-0.5	20	3	50	2	1434663	135.326	-0.5
-0.5	20	3	65	2	1434663	134.966	-0.5
-0.5	20	3	80	2	1434663	137.088	-0.5
-0.5	35	3	5	2	2173880	201.674	-1
-0.5	35	3	20	2	1833345	173.639	-0.9
-0.5	35	3	35	2	1457679	143.535	-0.9
-0.5	35	3	50	2	1088218	102.991	-0.6
-0.5	35	3	65	2	1088218	108.499	-0.6
-0.5	35	3	80	2	1088218	104.456	-0.6
-0.5	50	3	5	2	2173880	206.347	-1
-0.5	50	3	20	2	1833345	172.807	-0.9
-0.5	50	3	35	2	1457679	137.752	-0.9
-0.5	50	3	50	2	1088218	104.838	-0.6
-0.5	50	3	65	2	1088218	109.867	-0.6
-0.5	50	3	80	2	1088218	105.998	-0.6
-0.5	65	3	5	2	2173880	210.852	-1
-0.5	65	3	20	2	1833345	173.679	-0.9
-0.5	65	3	35	2	1457679	140.87	-0.9
-0.5	65	3	50	2	1088218	107.72	-0.6
-0.5	65	3	65	2	1088218	108.935	-0.6
-0.5	65	3	80	2	1088218	108.444	-0.6
-0.5	80	3	5	2	1549890	144.371	-1.1

Init. Time Gap (s)	P1 Init. Tire Wear (%)	P1 Init. Velocity ($m\,s^{-1}$)	P2 Init. Tire Wear (%)	P2 Init. Velocity ($m\,s^{-1}$)	Number of States	Model Checking Time (s)	Resulting Time Gap (s)
-0.5	80	3	20	2	1271042	120.269	-1
-0.5	80	3	35	2	1030446	102.899	-1
-0.5	80	3	50	2	768291	75.702	-0.7
-0.5	80	3	65	2	768291	77.273	-0.7
-0.5	80	3	80	2	768291	74.747	-0.7
-0.5	5	4	5	2	4069848	377.145	-0.6
-0.5	5	4	20	2	3368802	319.511	-0.6
-0.5	5	4	35	2	2695228	253.776	-0.5
-0.5	5	4	50	2	2054603	195.012	-0.2
-0.5	5	4	65	2	2054603	194.944	-0.2
-0.5	5	4	80	2	2054603	196.54	-0.2
-0.5	20	4	5	2	3109243	289.701	-0.7
-0.5	20	4	20	2	2603325	239.309	-0.6
-0.5	20	4	35	2	2027441	184.504	-0.6
-0.5	20	4	50	2	1562657	144.628	-0.4
-0.5	20	4	65	2	1562657	152.474	-0.4
-0.5	20	4	80	2	1562657	143.382	-0.4
-0.5	35	4	5	2	2368144	216.028	-0.9
-0.5	35	4	20	2	1991638	188.28	-0.8
-0.5	35	4	35	2	1579479	149.729	-0.8
-0.5	35	4	50	2	1178868	113.749	-0.6
-0.5	35	4	65	2	1178868	109.56	-0.6
-0.5	35	4	80	2	1178868	114.086	-0.6
-0.5	50	4	5	2	2368144	217.55	-0.9
-0.5	50	4	20	2	1991638	187.033	-0.8
-0.5	50	4	35	2	1579479	146.49	-0.8
-0.5	50	4	50	2	1178868	112.35	-0.6
-0.5	50	4	65	2	1178868	110.593	-0.6
-0.5	50	4	80	2	1178868	109.98	-0.6
-0.5	65	4	5	2	2368144	220.751	-0.9
-0.5	65	4	20	2	1991638	180.861	-0.8

Init. Time Gap (s)	P1 Init. Tire Wear (%)	P1 Init. Velocity ($m\,s^{-1}$)	P2 Init. Tire Wear (%)	P2 Init. Velocity ($m\,s^{-1}$)	Number of States	Model Checking Time (s)	Resulting Time Gap (s)
-0.5	65	4	35	2	1579479	147.347	-0.8
-0.5	65	4	50	2	1178868	111.351	-0.6
-0.5	65	4	65	2	1178868	116.529	-0.6
-0.5	65	4	80	2	1178868	117.269	-0.6
-0.5	80	4	5	2	1777023	165.283	-1
-0.5	80	4	20	2	1458112	134.186	-0.9
-0.5	80	4	35	2	1175170	106.861	-0.9
-0.5	80	4	50	2	876723	84.267	-0.6
-0.5	80	4	65	2	876723	85.004	-0.6
-0.5	80	4	80	2	876723	84.644	-0.6
-0.5	5	5	5	2	5610932	514.826	-0.4
-0.5	5	5	20	2	4671776	426.798	-0.4
-0.5	5	5	35	2	3738924	344.946	-0.2
-0.5	5	5	50	2	2861385	267.304	0
-0.5	5	5	65	2	2861385	270.906	0
-0.5	5	5	80	2	2861385	268.945	0
-0.5	20	5	5	2	4289949	394.331	-0.6
-0.5	20	5	20	2	3594481	328.241	-0.6
-0.5	20	5	35	2	2793112	260.036	-0.6
-0.5	20	5	50	2	2166807	204.455	-0.2
-0.5	20	5	65	2	2166807	202.069	-0.2
-0.5	20	5	80	2	2166807	202.201	-0.2
-0.5	35	5	5	2	3282582	304.46	-0.8
-0.5	35	5	20	2	2759586	256.233	-0.6
-0.5	35	5	35	2	2184778	204.511	-0.6
-0.5	35	5	50	2	1647031	151.925	-0.4
-0.5	35	5	65	2	1647031	153.862	-0.4
-0.5	35	5	80	2	1647031	153.616	-0.4
-0.5	50	5	5	2	3282582	305.687	-0.8
-0.5	50	5	20	2	2759586	263.184	-0.6
-0.5	50	5	35	2	2184778	206.944	-0.6

Init. Time Gap (s)	P1 Init. Tire Wear (%)	P1 Init. Velocity ($m\,s^{-1}$)	P2 Init. Tire Wear (%)	P2 Init. Velocity ($m\,s^{-1}$)	Number of States	Model Checking Time (s)	Resulting Time Gap (s)
-0.5	50	5	50	2	1647031	160.554	-0.4
-0.5	50	5	65	2	1647031	151.711	-0.4
-0.5	50	5	80	2	1647031	154.597	-0.4
-0.5	65	5	5	2	3282582	303.201	-0.8
-0.5	65	5	20	2	2759586	255.844	-0.6
-0.5	65	5	35	2	2184778	201.729	-0.6
-0.5	65	5	50	2	1647031	152.503	-0.4
-0.5	65	5	65	2	1647031	151.415	-0.4
-0.5	65	5	80	2	1647031	153.39	-0.4
-0.5	80	5	5	2	2560829	233.018	-0.8
-0.5	80	5	20	2	2088240	191.984	-0.8
-0.5	80	5	35	2	1681079	154.435	-0.7
-0.5	80	5	50	2	1267901	124.913	-0.4
-0.5	80	5	65	2	1267901	123.082	-0.4
-0.5	80	5	80	2	1267901	117.81	-0.4
-0.5	5	3	5	3	4099690	382.264	-0.7
-0.5	5	3	20	3	3425238	320.18	-0.6
-0.5	5	3	35	3	2741180	257.346	-0.6
-0.5	5	3	50	3	2088091	194.835	-0.3
-0.5	5	3	65	3	2088091	195.004	-0.3
-0.5	5	3	80	3	2088091	193.84	-0.3
-0.5	20	3	5	3	3114550	296.06	-0.8
-0.5	20	3	20	3	2630251	243.163	-0.7
-0.5	20	3	35	3	2056641	193.335	-0.7
-0.5	20	3	50	3	1583616	148.608	-0.5
-0.5	20	3	65	3	1583616	146.07	-0.5
-0.5	20	3	80	3	1583616	149.5	-0.5
-0.5	35	3	5	3	2385773	222.595	-1
-0.5	35	3	20	3	2019984	191.636	-0.9
-0.5	35	3	35	3	1609197	150.738	-0.9
-0.5	35	3	50	3	1211615	114.716	-0.6

Init. Time Gap (s)	P1 Init. Tire Wear (%)	P1 Init. Velocity ($m\,s^{-1}$)	P2 Init. Tire Wear (%)	P2 Init. Velocity ($m\,s^{-1}$)	Number of States	Model Checking Time (s)	Resulting Time Gap (s)
-0.5	35	3	65	3	1211615	120.203	-0.6
-0.5	35	3	80	3	1211615	114.435	-0.6
-0.5	50	3	5	3	2385773	221.101	-1
-0.5	50	3	20	3	2019984	188.711	-0.9
-0.5	50	3	35	3	1609197	159.794	-0.9
-0.5	50	3	50	3	1211615	120.74	-0.6
-0.5	50	3	65	3	1211615	119.18	-0.6
-0.5	50	3	80	3	1211615	116.637	-0.6
-0.5	65	3	5	3	2385773	226.38	-1
-0.5	65	3	20	3	2019984	189.5	-0.9
-0.5	65	3	35	3	1609197	148.684	-0.9
-0.5	65	3	50	3	1211615	117.885	-0.6
-0.5	65	3	65	3	1211615	115.185	-0.6
-0.5	65	3	80	3	1211615	118.457	-0.6
-0.5	80	3	5	3	1712005	162.245	-1.1
-0.5	80	3	20	3	1426611	133.813	-1
-0.5	80	3	35	3	1155964	112.392	-1
-0.5	80	3	50	3	871876	85.115	-0.7
-0.5	80	3	65	3	871876	83.637	-0.7
-0.5	80	3	80	3	871876	86.823	-0.7
-0.5	5	4	5	3	4384857	414.657	-0.7
-0.5	5	4	20	3	3664780	338.243	-0.6
-0.5	5	4	35	3	2929364	271.778	-0.6
-0.5	5	4	50	3	2240504	210.19	-0.3
-0.5	5	4	65	3	2240504	214.271	-0.3
-0.5	5	4	80	3	2240504	213.587	-0.3
-0.5	20	4	5	3	3367269	314.187	-0.8
-0.5	20	4	20	3	2840671	261.828	-0.7
-0.5	20	4	35	3	2219885	207.484	-0.7
-0.5	20	4	50	3	1716891	161.739	-0.5
-0.5	20	4	65	3	1716891	159.256	-0.5

Init. Time Gap (s)	P1 Init. Tire Wear (%)	P1 Init. Velocity ($m\,s^{-1}$)	P2 Init. Tire Wear (%)	P2 Init. Velocity ($m\,s^{-1}$)	Number of States	Model Checking Time (s)	Resulting Time Gap (s)
-0.5	20	4	80	3	1716891	171.087	-0.5
-0.5	35	4	5	3	2574905	243.183	-1
-0.5	35	4	20	3	2181626	207.487	-0.9
-0.5	35	4	35	3	1734510	159.047	-0.9
-0.5	35	4	50	3	1307886	123.223	-0.6
-0.5	35	4	65	3	1307886	122.465	-0.6
-0.5	35	4	80	3	1307886	122.326	-0.6
-0.5	50	4	5	3	2574905	238.74	-1
-0.5	50	4	20	3	2181626	202.23	-0.9
-0.5	50	4	35	3	1734510	159.714	-0.9
-0.5	50	4	50	3	1307886	123.826	-0.6
-0.5	50	4	65	3	1307886	125.642	-0.6
-0.5	50	4	80	3	1307886	122.487	-0.6
-0.5	65	4	5	3	2574905	245.099	-1
-0.5	65	4	20	3	2181626	205.792	-0.9
-0.5	65	4	35	3	1734510	164.343	-0.9
-0.5	65	4	50	3	1307886	124.024	-0.6
-0.5	65	4	65	3	1307886	123.863	-0.6
-0.5	65	4	80	3	1307886	122.813	-0.6
-0.5	80	4	5	3	1951330	184.555	-1.1
-0.5	80	4	20	3	1617205	160.432	-1
-0.5	80	4	35	3	1306191	128.559	-1
-0.5	80	4	50	3	982468	94.137	-0.7
-0.5	80	4	65	3	982468	96.913	-0.7
-0.5	80	4	80	3	982468	93.627	-0.7
-0.5	5	5	5	3	5311745	498.145	-0.5
-0.5	5	5	20	3	4442361	423.49	-0.5
-0.5	5	5	35	3	3561394	329.365	-0.3
-0.5	5	5	50	3	2762536	264.018	-0.1
-0.5	5	5	65	3	2762536	256.838	-0.1
-0.5	5	5	80	3	2762536	260.32	-0.1

Init. Time Gap (s)	P1 Init. Tire Wear (%)	P1 Init. Velocity ($m\,s^{-1}$)	P2 Init. Tire Wear (%)	P2 Init. Velocity ($m\,s^{-1}$)	Number of States	Model Checking Time (s)	Resulting Time Gap (s)
-0.5	20	5	5	3	4100377	373.551	-0.6
-0.5	20	5	20	3	3453094	318.543	-0.6
-0.5	20	5	35	3	2702216	256.248	-0.6
-0.5	20	5	50	3	2118565	196.591	-0.3
-0.5	20	5	65	3	2118565	197.333	-0.3
-0.5	20	5	80	3	2118565	197.305	-0.3
-0.5	35	5	5	3	3156744	294.27	-0.9
-0.5	35	5	20	3	2670529	248.756	-0.7
-0.5	35	5	35	3	2126034	197.857	-0.7
-0.5	35	5	50	3	1626866	153.356	-0.5
-0.5	35	5	65	3	1626866	153.78	-0.5
-0.5	35	5	80	3	1626866	153.577	-0.5
-0.5	50	5	5	3	3156744	293.085	-0.9
-0.5	50	5	20	3	2670529	248.247	-0.7
-0.5	50	5	35	3	2126034	193.677	-0.7
-0.5	50	5	50	3	1626866	149.107	-0.5
-0.5	50	5	65	3	1626866	149.946	-0.5
-0.5	50	5	80	3	1626866	161.043	-0.5
-0.5	65	5	5	3	3156744	296.287	-0.9
-0.5	65	5	20	3	2670529	253.719	-0.7
-0.5	65	5	35	3	2126034	192.786	-0.7
-0.5	65	5	50	3	1626866	157.871	-0.5
-0.5	65	5	65	3	1626866	154.514	-0.5
-0.5	65	5	80	3	1626866	147.846	-0.5
-0.5	80	5	5	3	2489134	228.675	-0.9
-0.5	80	5	20	3	2056212	188.57	-0.9
-0.5	80	5	35	3	1657030	149.446	-0.8
-0.5	80	5	50	3	1266829	124.925	-0.5
-0.5	80	5	65	3	1266829	116.487	-0.5
-0.5	80	5	80	3	1266829	118.381	-0.5
-0.5	5	3	5	4	4677624	431.885	-0.9

Init. Time Gap (s)	P1 Init. Tire Wear (%)	P1 Init. Velocity ($m\,s^{-1}$)	P2 Init. Tire Wear (%)	P2 Init. Velocity ($m\,s^{-1}$)	Number of States	Model Checking Time (s)	Resulting Time Gap (s)
-0.5	5	3	20	4	3959483	365.223	-0.8
-0.5	5	3	35	4	3208019	295.681	-0.8
-0.5	5	3	50	4	2450810	223.724	-0.5
-0.5	5	3	65	4	2450810	224.109	-0.5
-0.5	5	3	80	4	2450810	223.891	-0.5
-0.5	20	3	5	4	3568461	334.401	-1
-0.5	20	3	20	4	3047042	279.512	-0.9
-0.5	20	3	35	4	2426644	227.114	-0.9
-0.5	20	3	50	4	1872298	177.09	-0.6
-0.5	20	3	65	4	1872298	176.106	-0.6
-0.5	20	3	80	4	1872298	175.575	-0.6
-0.5	35	3	5	4	2750279	257.371	-1.2
-0.5	35	3	20	4	2354860	213.571	-1.1
-0.5	35	3	35	4	1905655	178.231	-1.1
-0.5	35	3	50	4	1444700	135.944	-0.8
-0.5	35	3	65	4	1444700	136.38	-0.8
-0.5	35	3	80	4	1444700	132.661	-0.8
-0.5	50	3	5	4	2750279	256.915	-1.2
-0.5	50	3	20	4	2354860	212.593	-1.1
-0.5	50	3	35	4	1905655	178.667	-1.1
-0.5	50	3	50	4	1444700	140.703	-0.8
-0.5	50	3	65	4	1444700	139.041	-0.8
-0.5	50	3	80	4	1444700	140.366	-0.8
-0.5	65	3	5	4	2750279	250.82	-1.2
-0.5	65	3	20	4	2354860	213.565	-1.1
-0.5	65	3	35	4	1905655	172.076	-1.1
-0.5	65	3	50	4	1444700	134.585	-0.8
-0.5	65	3	65	4	1444700	138.803	-0.8
-0.5	65	3	80	4	1444700	134.088	-0.8
-0.5	80	3	5	4	1980965	183.692	-1.3
-0.5	80	3	20	4	1674147	152.14	-1.2

Init. Time Gap (s)	P1 Init. Tire Wear (%)	P1 Init. Velocity ($m\,s^{-1}$)	P2 Init. Tire Wear (%)	P2 Init. Velocity ($m\,s^{-1}$)	Number of States	Model Checking Time (s)	Resulting Time Gap (s)
-0.5	80	3	35	4	1373862	130.391	-1.2
-0.5	80	3	50	4	1043411	97.425	-0.9
-0.5	80	3	65	4	1043411	103.058	-0.9
-0.5	80	3	80	4	1043411	96.466	-0.9
-0.5	5	4	5	4	4996171	466.932	-0.8
-0.5	5	4	20	4	4228863	398.412	-0.7
-0.5	5	4	35	4	3422739	320.195	-0.7
-0.5	5	4	50	4	2623802	239.298	-0.4
-0.5	5	4	65	4	2623802	241.263	-0.4
-0.5	5	4	80	4	2623802	240.645	-0.4
-0.5	20	4	5	4	3864899	363.29	-0.9
-0.5	20	4	20	4	3295640	296.939	-0.8
-0.5	20	4	35	4	2622056	246.2	-0.8
-0.5	20	4	50	4	2030255	190.251	-0.5
-0.5	20	4	65	4	2030255	190.76	-0.5
-0.5	20	4	80	4	2030255	187.727	-0.5
-0.5	35	4	5	4	2981783	272.35	-1.1
-0.5	35	4	20	4	2548531	239.96	-1
-0.5	35	4	35	4	2057124	187.648	-1
-0.5	35	4	50	4	1563657	142.512	-0.7
-0.5	35	4	65	4	1563657	146.442	-0.7
-0.5	35	4	80	4	1563657	151.435	-0.7
-0.5	50	4	5	4	2981783	273.712	-1.1
-0.5	50	4	20	4	2548531	234.841	-1
-0.5	50	4	35	4	2057124	193.369	-1
-0.5	50	4	50	4	1563657	143.972	-0.7
-0.5	50	4	65	4	1563657	147.311	-0.7
-0.5	50	4	80	4	1563657	150.139	-0.7
-0.5	65	4	5	4	2981783	284.06	-1.1
-0.5	65	4	20	4	2548531	236.346	-1
-0.5	65	4	35	4	2057124	191.408	-1

Init. Time Gap (s)	P1 Init. Tire Wear (%)	P1 Init. Velocity ($m\,s^{-1}$)	P2 Init. Tire Wear (%)	P2 Init. Velocity ($m\,s^{-1}$)	Number of States	Model Checking Time (s)	Resulting Time Gap (s)
-0.5	65	4	50	4	1563657	143.925	-0.7
-0.5	65	4	65	4	1563657	145.476	-0.7
-0.5	65	4	80	4	1563657	148.008	-0.7
-0.5	80	4	5	4	2272326	205.848	-1.2
-0.5	80	4	20	4	1917668	180.471	-1.1
-0.5	80	4	35	4	1565998	145.782	-1.1
-0.5	80	4	50	4	1187591	112.108	-0.8
-0.5	80	4	65	4	1187591	109.478	-0.8
-0.5	80	4	80	4	1187591	113.885	-0.8
-0.5	5	5	5	4	5971170	560.816	-0.7
-0.5	5	5	20	4	5051715	464.379	-0.6
-0.5	5	5	35	4	4105072	373.197	-0.5
-0.5	5	5	50	4	3183061	298.969	-0.3
-0.5	5	5	65	4	3183061	290.705	-0.3
-0.5	5	5	80	4	3183061	303.141	-0.3
-0.5	20	5	5	4	4625709	425.875	-0.8
-0.5	20	5	20	4	3948021	359.061	-0.7
-0.5	20	5	35	4	3143461	286.715	-0.7
-0.5	20	5	50	4	2462178	233.437	-0.5
-0.5	20	5	65	4	2462178	232.307	-0.5
-0.5	20	5	80	4	2462178	224.449	-0.5
-0.5	35	5	5	4	3593992	328.318	-1
-0.5	35	5	20	4	3073478	291.285	-0.9
-0.5	35	5	35	4	2482274	229.454	-0.9
-0.5	35	5	50	4	1909889	173.918	-0.6
-0.5	35	5	65	4	1909889	181.91	-0.6
-0.5	35	5	80	4	1909889	179.103	-0.6
-0.5	50	5	5	4	3593992	329.377	-1
-0.5	50	5	20	4	3073478	278.981	-0.9
-0.5	50	5	35	4	2482274	227.536	-0.9
-0.5	50	5	50	4	1909889	182.273	-0.6

Init. Time Gap (s)	P1 Init. Tire Wear (%)	P1 Init. Velocity ($m\,s^{-1}$)	P2 Init. Tire Wear (%)	P2 Init. Velocity ($m\,s^{-1}$)	Number of States	Model Checking Time (s)	Resulting Time Gap (s)
-0.5	50	5	65	4	1909889	177.492	-0.6
-0.5	50	5	80	4	1909889	180.577	-0.6
-0.5	65	5	5	4	3593992	330.741	-1
-0.5	65	5	20	4	3073478	288.071	-0.9
-0.5	65	5	35	4	2482274	235.112	-0.9
-0.5	65	5	50	4	1909889	177.97	-0.6
-0.5	65	5	65	4	1909889	175.48	-0.6
-0.5	65	5	80	4	1909889	182.594	-0.6
-0.5	80	5	5	4	2845287	266.218	-1.1
-0.5	80	5	20	4	2397447	224.663	-1
-0.5	80	5	35	4	1955698	179.35	-1
-0.5	80	5	50	4	1503812	146.323	-0.7
-0.5	80	5	65	4	1503812	144.873	-0.7
-0.5	80	5	80	4	1503812	144.71	-0.7

Table A.2: Full results from various scenarios on hairpin track shape.

A.3 Chicane Results

Init. Time Gap (s)	P1 Init. Tire Wear (%)	P1 Init. Velocity ($m\,s^{-1}$)	P2 Init. Tire Wear (%)	P2 Init. Velocity ($m\,s^{-1}$)	Number of States	Model Checking Time (s)	Resulting Time Gap (s)
-0.5	5	3	5	2	4494767	421.646	-0.6
-0.5	5	3	20	2	4152356	392.277	-0.6
-0.5	5	3	35	2	3582704	333.799	-0.6
-0.5	5	3	50	2	2358600	222.101	-0.3
-0.5	5	3	65	2	2358600	220.983	-0.3
-0.5	5	3	80	2	2358600	221.656	-0.3
-0.5	20	3	5	2	4043935	370.493	-0.7
-0.5	20	3	20	2	3741542	346.941	-0.7
-0.5	20	3	35	2	3197890	295.96	-0.6
-0.5	20	3	50	2	2115272	195.138	-0.5
-0.5	20	3	65	2	2115272	195.87	-0.5
-0.5	20	3	80	2	2115272	200.086	-0.5
-0.5	35	3	5	2	3127904	293.003	-1
-0.5	35	3	20	2	2889304	263.834	-1
-0.5	35	3	35	2	2501318	228.597	-1
-0.5	35	3	50	2	1611303	148.343	-0.6
-0.5	35	3	65	2	1611303	150.156	-0.6
-0.5	35	3	80	2	1611303	149.246	-0.6
-0.5	50	3	5	2	3127904	290.443	-1
-0.5	50	3	20	2	2889304	267.108	-1
-0.5	50	3	35	2	2501318	232.567	-1
-0.5	50	3	50	2	1611303	148.397	-0.6
-0.5	50	3	65	2	1611303	152.789	-0.6
-0.5	50	3	80	2	1611303	156.35	-0.6
-0.5	65	3	5	2	3127904	290.672	-1
-0.5	65	3	20	2	2889304	268.092	-1
-0.5	65	3	35	2	2501318	233.808	-1
-0.5	65	3	50	2	1611303	149.533	-0.6
-0.5	65	3	65	2	1611303	149.492	-0.6

Init. Time Gap (s)	P1 Init. Tire Wear (%)	P1 Init. Velocity ($m\,s^{-1}$)	P2 Init. Tire Wear (%)	P2 Init. Velocity ($m\,s^{-1}$)	Number of States	Model Checking Time (s)	Resulting Time Gap (s)
-0.5	65	3	80	2	1611303	148.641	-0.6
-0.5	80	3	5	2	2571339	233.803	-1
-0.5	80	3	20	2	2364690	216.3	-1
-0.5	80	3	35	2	2069243	189.62	-1
-0.5	80	3	50	2	1317463	122.18	-0.6
-0.5	80	3	65	2	1317463	123.725	-0.6
-0.5	80	3	80	2	1317463	122.481	-0.6
-0.5	5	4	5	2	4687023	442.266	-0.6
-0.5	5	4	20	2	4334688	400.2	-0.6
-0.5	5	4	35	2	3736728	347.541	-0.6
-0.5	5	4	50	2	2468637	227.247	-0.2
-0.5	5	4	65	2	2468637	225.632	-0.2
-0.5	5	4	80	2	2468637	227.764	-0.2
-0.5	20	4	5	2	4238140	398.07	-0.7
-0.5	20	4	20	2	3925913	361.465	-0.7
-0.5	20	4	35	2	3347237	307.771	-0.6
-0.5	20	4	50	2	2224114	201.054	-0.4
-0.5	20	4	65	2	2224114	209.928	-0.4
-0.5	20	4	80	2	2224114	203.416	-0.4
-0.5	35	4	5	2	3299810	296.791	-0.9
-0.5	35	4	20	2	3052540	282.471	-0.9
-0.5	35	4	35	2	2641994	244.151	-0.9
-0.5	35	4	50	2	1703901	160.151	-0.6
-0.5	35	4	65	2	1703901	154.7	-0.6
-0.5	35	4	80	2	1703901	157.406	-0.6
-0.5	50	4	5	2	3299810	304.921	-0.9
-0.5	50	4	20	2	3052540	277.074	-0.9
-0.5	50	4	35	2	2641994	245.791	-0.9
-0.5	50	4	50	2	1703901	156.037	-0.6
-0.5	50	4	65	2	1703901	153.546	-0.6
-0.5	50	4	80	2	1703901	165.344	-0.6

Init. Time Gap (s)	P1 Init. Tire Wear (%)	P1 Init. Velocity ($m\,s^{-1}$)	P2 Init. Tire Wear (%)	P2 Init. Velocity ($m\,s^{-1}$)	Number of States	Model Checking Time (s)	Resulting Time Gap (s)
-0.5	65	4	5	2	3299810	296.644	-0.9
-0.5	65	4	20	2	3052540	286.097	-0.9
-0.5	65	4	35	2	2641994	238.657	-0.9
-0.5	65	4	50	2	1703901	159.45	-0.6
-0.5	65	4	65	2	1703901	157.562	-0.6
-0.5	65	4	80	2	1703901	155.338	-0.6
-0.5	80	4	5	2	2724280	246.126	-0.9
-0.5	80	4	20	2	2508166	229.547	-0.9
-0.5	80	4	35	2	2193154	198.148	-0.9
-0.5	80	4	50	2	1399171	129.269	-0.6
-0.5	80	4	65	2	1399171	126.988	-0.6
-0.5	80	4	80	2	1399171	126.149	-0.6
-0.5	5	5	5	2	6276401	574.332	-0.4
-0.5	5	5	20	2	5751609	531.585	-0.4
-0.5	5	5	35	2	4947545	453.213	-0.4
-0.5	5	5	50	2	3283880	306.243	0
-0.5	5	5	65	2	3283880	301.2	0
-0.5	5	5	80	2	3283880	304.994	0
-0.5	20	5	5	2	5736619	520.613	-0.6
-0.5	20	5	20	2	5266964	479.911	-0.6
-0.5	20	5	35	2	4489146	406.578	-0.6
-0.5	20	5	50	2	2997097	277.301	-0.3
-0.5	20	5	65	2	2997097	277.86	-0.3
-0.5	20	5	80	2	2997097	275.17	-0.3
-0.5	35	5	5	2	4482863	411.124	-0.8
-0.5	35	5	20	2	4103675	381.925	-0.8
-0.5	35	5	35	2	3547935	325.17	-0.8
-0.5	35	5	50	2	2312861	211.965	-0.4
-0.5	35	5	65	2	2312861	213.456	-0.4
-0.5	35	5	80	2	2312861	216.595	-0.4
-0.5	50	5	5	2	4482863	408.968	-0.8

Init. Time Gap (s)	P1 Init. Tire Wear (%)	P1 Init. Velocity ($m\,s^{-1}$)	P2 Init. Tire Wear (%)	P2 Init. Velocity ($m\,s^{-1}$)	Number of States	Model Checking Time (s)	Resulting Time Gap (s)
-0.5	50	5	20	2	4103675	376.197	-0.8
-0.5	50	5	35	2	3547935	321.694	-0.8
-0.5	50	5	50	2	2312861	209.558	-0.4
-0.5	50	5	65	2	2312861	210.328	-0.4
-0.5	50	5	80	2	2312861	213.427	-0.4
-0.5	65	5	5	2	4482863	398.875	-0.8
-0.5	65	5	20	2	4103675	367.687	-0.8
-0.5	65	5	35	2	3547935	328.329	-0.8
-0.5	65	5	50	2	2312861	217.464	-0.4
-0.5	65	5	65	2	2312861	209.472	-0.4
-0.5	65	5	80	2	2312861	212.578	-0.4
-0.5	80	5	5	2	3743486	338.432	-0.8
-0.5	80	5	20	2	3403529	311.919	-0.8
-0.5	80	5	35	2	2973027	265.065	-0.8
-0.5	80	5	50	2	1918831	176.552	-0.4
-0.5	80	5	65	2	1918831	176.399	-0.4
-0.5	80	5	80	2	1918831	177.655	-0.4
-0.5	5	3	5	3	4776594	444.134	-0.6
-0.5	5	3	20	3	4412766	414.924	-0.6
-0.5	5	3	35	3	3817249	355.011	-0.6
-0.5	5	3	50	3	2526710	249.198	-0.3
-0.5	5	3	65	3	2526710	234.416	-0.3
-0.5	5	3	80	3	2526710	241.069	-0.3
-0.5	20	3	5	3	4303321	393.384	-0.7
-0.5	20	3	20	3	3980747	366.931	-0.7
-0.5	20	3	35	3	3419935	313.527	-0.6
-0.5	20	3	50	3	2269834	209.787	-0.5
-0.5	20	3	65	3	2269834	209.148	-0.5
-0.5	20	3	80	3	2269834	208.642	-0.5
-0.5	35	3	5	3	3351408	310.041	-1
-0.5	35	3	20	3	3096369	280.836	-1

Init. Time Gap (s)	P1 Init. Tire Wear (%)	P1 Init. Velocity ($m\,s^{-1}$)	P2 Init. Tire Wear (%)	P2 Init. Velocity ($m\,s^{-1}$)	Number of States	Model Checking Time (s)	Resulting Time Gap (s)
-0.5	35	3	35	3	2693539	249.315	-1
-0.5	35	3	50	3	1745509	163.303	-0.6
-0.5	35	3	65	3	1745509	160.529	-0.6
-0.5	35	3	80	3	1745509	162.01	-0.6
-0.5	50	3	5	3	3351408	305.977	-1
-0.5	50	3	20	3	3096369	284.057	-1
-0.5	50	3	35	3	2693539	244.918	-1
-0.5	50	3	50	3	1745509	163.151	-0.6
-0.5	50	3	65	3	1745509	159.59	-0.6
-0.5	50	3	80	3	1745509	170.325	-0.6
-0.5	65	3	5	3	3351408	306.709	-1
-0.5	65	3	20	3	3096369	285.618	-1
-0.5	65	3	35	3	2693539	245.584	-1
-0.5	65	3	50	3	1745509	162.285	-0.6
-0.5	65	3	65	3	1745509	161.158	-0.6
-0.5	65	3	80	3	1745509	163.187	-0.6
-0.5	80	3	5	3	2761130	257.294	-1
-0.5	80	3	20	3	2540811	231.852	-1
-0.5	80	3	35	3	2233438	207.779	-1
-0.5	80	3	50	3	1431887	130.862	-0.6
-0.5	80	3	65	3	1431887	131.451	-0.6
-0.5	80	3	80	3	1431887	132.546	-0.6
-0.5	5	4	5	3	4965607	463.585	-0.6
-0.5	5	4	20	3	4593892	423.282	-0.6
-0.5	5	4	35	3	3971392	367.178	-0.6
-0.5	5	4	50	3	2634169	247.57	-0.3
-0.5	5	4	65	3	2634169	243.717	-0.3
-0.5	5	4	80	3	2634169	244.782	-0.3
-0.5	20	4	5	3	4495829	411.932	-0.7
-0.5	20	4	20	3	4165441	379.404	-0.7
-0.5	20	4	35	3	3570023	329.596	-0.6

Init. Time Gap (s)	P1 Init. Tire Wear (%)	P1 Init. Velocity ($m\,s^{-1}$)	P2 Init. Tire Wear (%)	P2 Init. Velocity ($m\,s^{-1}$)	Number of States	Model Checking Time (s)	Resulting Time Gap (s)
-0.5	20	4	50	3	2378618	218.38	-0.5
-0.5	20	4	65	3	2378618	220.076	-0.5
-0.5	20	4	80	3	2378618	218.253	-0.5
-0.5	35	4	5	3	3517355	327.911	-1
-0.5	35	4	20	3	3254191	302.703	-1
-0.5	35	4	35	3	2824881	257.172	-1
-0.5	35	4	50	3	1837535	167.714	-0.6
-0.5	35	4	65	3	1837535	168.194	-0.6
-0.5	35	4	80	3	1837535	169.371	-0.6
-0.5	50	4	5	3	3517355	322.153	-1
-0.5	50	4	20	3	3254191	299.268	-1
-0.5	50	4	35	3	2824881	256.233	-1
-0.5	50	4	50	3	1837535	168.755	-0.6
-0.5	50	4	65	3	1837535	167.039	-0.6
-0.5	50	4	80	3	1837535	169.48	-0.6
-0.5	65	4	5	3	3517355	317.691	-1
-0.5	65	4	20	3	3254191	299.447	-1
-0.5	65	4	35	3	2824881	259.387	-1
-0.5	65	4	50	3	1837535	171.392	-0.6
-0.5	65	4	65	3	1837535	168.264	-0.6
-0.5	65	4	80	3	1837535	168.87	-0.6
-0.5	80	4	5	3	2910852	267.863	-1
-0.5	80	4	20	3	2681306	244.525	-1
-0.5	80	4	35	3	2350413	217.3	-1
-0.5	80	4	50	3	1514442	137.9	-0.6
-0.5	80	4	65	3	1514442	137.117	-0.6
-0.5	80	4	80	3	1514442	141.418	-0.6
-0.5	5	5	5	3	5781200	524.449	-0.5
-0.5	5	5	20	3	5350787	488.242	-0.5
-0.5	5	5	35	3	4622607	420.511	-0.5
-0.5	5	5	50	3	3095240	284.619	-0.1

Init. Time Gap (s)	P1 Init. Tire Wear (%)	P1 Init. Velocity ($m\,s^{-1}$)	P2 Init. Tire Wear (%)	P2 Init. Velocity ($m\,s^{-1}$)	Number of States	Model Checking Time (s)	Resulting Time Gap (s)
-0.5	5	5	65	3	3095240	281.092	-0.1
-0.5	5	5	80	3	3095240	284.568	-0.1
-0.5	20	5	5	3	5278888	480.185	-0.7
-0.5	20	5	20	3	4893417	443.204	-0.7
-0.5	20	5	35	3	4190889	375.466	-0.6
-0.5	20	5	50	3	2819952	260.361	-0.4
-0.5	20	5	65	3	2819952	260.521	-0.4
-0.5	20	5	80	3	2819952	262.866	-0.4
-0.5	35	5	5	3	4179275	379.482	-0.9
-0.5	35	5	20	3	3869158	351.454	-0.9
-0.5	35	5	35	3	3361889	300.799	-0.9
-0.5	35	5	50	3	2207410	201.09	-0.5
-0.5	35	5	65	3	2207410	200.701	-0.5
-0.5	35	5	80	3	2207410	203.339	-0.5
-0.5	50	5	5	3	4179275	380.781	-0.9
-0.5	50	5	20	3	3869158	349.787	-0.9
-0.5	50	5	35	3	3361889	312.621	-0.9
-0.5	50	5	50	3	2207410	199.595	-0.5
-0.5	50	5	65	3	2207410	198.381	-0.5
-0.5	50	5	80	3	2207410	198.252	-0.5
-0.5	65	5	5	3	4179275	382.821	-0.9
-0.5	65	5	20	3	3869158	346.019	-0.9
-0.5	65	5	35	3	3361889	301.582	-0.9
-0.5	65	5	50	3	2207410	199.208	-0.5
-0.5	65	5	65	3	2207410	197.638	-0.5
-0.5	65	5	80	3	2207410	202.154	-0.5
-0.5	80	5	5	3	3478907	308.97	-0.9
-0.5	80	5	20	3	3206931	287.756	-0.9
-0.5	80	5	35	3	2812140	248.871	-0.9
-0.5	80	5	50	3	1829676	164.093	-0.5
-0.5	80	5	65	3	1829676	163.771	-0.5

Init. Time Gap (s)	P1 Init. Tire Wear (%)	P1 Init. Velocity ($m\,s^{-1}$)	P2 Init. Tire Wear (%)	P2 Init. Velocity ($m\,s^{-1}$)	Number of States	Model Checking Time (s)	Resulting Time Gap (s)
-0.5	80	5	80	3	1829676	167.645	-0.5
-0.5	5	3	5	4	5306033	499.774	-0.8
-0.5	5	3	20	4	4902069	448.029	-0.8
-0.5	5	3	35	4	4284765	401.174	-0.8
-0.5	5	3	50	4	2860286	270.719	-0.4
-0.5	5	3	65	4	2860286	265.555	-0.4
-0.5	5	3	80	4	2860286	263.293	-0.4
-0.5	20	3	5	4	4780593	443.255	-0.8
-0.5	20	3	20	4	4421749	411.181	-0.8
-0.5	20	3	35	4	3844102	354.178	-0.8
-0.5	20	3	50	4	2571801	239.487	-0.5
-0.5	20	3	65	4	2571801	234.493	-0.5
-0.5	20	3	80	4	2571801	237.27	-0.5
-0.5	35	3	5	4	3759164	339.771	-1.2
-0.5	35	3	20	4	3473524	316.303	-1.2
-0.5	35	3	35	4	3049685	281.21	-1.2
-0.5	35	3	50	4	2003002	186.83	-0.8
-0.5	35	3	65	4	2003002	184.201	-0.8
-0.5	35	3	80	4	2003002	180.761	-0.8
-0.5	50	3	5	4	3759164	345.899	-1.2
-0.5	50	3	20	4	3473524	313.868	-1.2
-0.5	50	3	35	4	3049685	284.321	-1.2
-0.5	50	3	50	4	2003002	180.719	-0.8
-0.5	50	3	65	4	2003002	182.346	-0.8
-0.5	50	3	80	4	2003002	184.434	-0.8
-0.5	65	3	5	4	3759164	343.681	-1.2
-0.5	65	3	20	4	3473524	320.866	-1.2
-0.5	65	3	35	4	3049685	276.224	-1.2
-0.5	65	3	50	4	2003002	185.811	-0.8
-0.5	65	3	65	4	2003002	185.142	-0.8
-0.5	65	3	80	4	2003002	183.102	-0.8

Init. Time Gap (s)	P1 Init. Tire Wear (%)	P1 Init. Velocity ($m\,s^{-1}$)	P2 Init. Tire Wear (%)	P2 Init. Velocity ($m\,s^{-1}$)	Number of States	Model Checking Time (s)	Resulting Time Gap (s)
-0.5	80	3	5	4	3108683	285.209	-1.2
-0.5	80	3	20	4	2862930	264.108	-1.2
-0.5	80	3	35	4	2535548	232.889	-1.2
-0.5	80	3	50	4	1650577	148.987	-0.8
-0.5	80	3	65	4	1650577	150.844	-0.8
-0.5	80	3	80	4	1650577	157.429	-0.8
-0.5	5	4	5	4	5523618	523.524	-0.7
-0.5	5	4	20	4	5107945	481.595	-0.7
-0.5	5	4	35	4	4461952	418.05	-0.7
-0.5	5	4	50	4	2984204	281.523	-0.3
-0.5	5	4	65	4	2984204	286.854	-0.3
-0.5	5	4	80	4	2984204	282.717	-0.3
-0.5	20	4	5	4	5005544	469.664	-0.7
-0.5	20	4	20	4	4634291	426.729	-0.7
-0.5	20	4	35	4	4025799	375.495	-0.7
-0.5	20	4	50	4	2699241	246.41	-0.5
-0.5	20	4	65	4	2699241	248.931	-0.5
-0.5	20	4	80	4	2699241	251.886	-0.5
-0.5	35	4	5	4	3956561	364.593	-1.1
-0.5	35	4	20	4	3659740	328.929	-1.1
-0.5	35	4	35	4	3211489	298.848	-1.1
-0.5	35	4	50	4	2112664	197.362	-0.7
-0.5	35	4	65	4	2112664	195.438	-0.7
-0.5	35	4	80	4	2112664	192.546	-0.7
-0.5	50	4	5	4	3956561	355.693	-1.1
-0.5	50	4	20	4	3659740	331.617	-1.1
-0.5	50	4	35	4	3211489	296.185	-1.1
-0.5	50	4	50	4	2112664	193.239	-0.7
-0.5	50	4	65	4	2112664	195.651	-0.7
-0.5	50	4	80	4	2112664	194.406	-0.7
-0.5	65	4	5	4	3956561	371.708	-1.1

Init. Time Gap (s)	P1 Init. Tire Wear (%)	P1 Init. Velocity ($m\,s^{-1}$)	P2 Init. Tire Wear (%)	P2 Init. Velocity ($m\,s^{-1}$)	Number of States	Model Checking Time (s)	Resulting Time Gap (s)
-0.5	65	4	20	4	3659740	327.871	-1.1
-0.5	65	4	35	4	3211489	296.145	-1.1
-0.5	65	4	50	4	2112664	195.417	-0.7
-0.5	65	4	65	4	2112664	193.753	-0.7
-0.5	65	4	80	4	2112664	191.452	-0.7
-0.5	80	4	5	4	3285910	291.749	-1.1
-0.5	80	4	20	4	3028615	277.054	-1.1
-0.5	80	4	35	4	2679937	239.227	-1.1
-0.5	80	4	50	4	1748778	161.574	-0.7
-0.5	80	4	65	4	1748778	156.976	-0.7
-0.5	80	4	80	4	1748778	159.5	-0.7
-0.5	5	5	5	4	6368218	592.852	-0.6
-0.5	5	5	20	4	5895250	542.323	-0.6
-0.5	5	5	35	4	5137407	472.901	-0.6
-0.5	5	5	50	4	3467897	322.692	-0.3
-0.5	5	5	65	4	3467897	323.651	-0.3
-0.5	5	5	80	4	3467897	325.16	-0.3
-0.5	20	5	5	4	5829614	528.635	-0.7
-0.5	20	5	20	4	5403528	488.272	-0.7
-0.5	20	5	35	4	4677587	438.81	-0.6
-0.5	20	5	50	4	3169036	299.219	-0.5
-0.5	20	5	65	4	3169036	286.419	-0.5
-0.5	20	5	80	4	3169036	299.044	-0.5
-0.5	35	5	5	4	4650482	423.13	-1
-0.5	35	5	20	4	4305908	397.485	-1
-0.5	35	5	35	4	3770740	335.839	-1
-0.5	35	5	50	4	2508516	231.22	-0.6
-0.5	35	5	65	4	2508516	233.943	-0.6
-0.5	35	5	80	4	2508516	227.15	-0.6
-0.5	50	5	5	4	4650482	421.59	-1
-0.5	50	5	20	4	4305908	399.592	-1

Init. Time Gap (s)	P1 Init. Tire Wear (%)	P1 Init. Velocity ($m\,s^{-1}$)	P2 Init. Tire Wear (%)	P2 Init. Velocity ($m\,s^{-1}$)	Number of States	Model Checking Time (s)	Resulting Time Gap (s)
-0.5	50	5	35	4	3770740	342.126	-1
-0.5	50	5	50	4	2508516	228.009	-0.6
-0.5	50	5	65	4	2508516	232.867	-0.6
-0.5	50	5	80	4	2508516	227.12	-0.6
-0.5	65	5	5	4	4650482	418.952	-1
-0.5	65	5	20	4	4305908	388.771	-1
-0.5	65	5	35	4	3770740	352.436	-1
-0.5	65	5	50	4	2508516	237.662	-0.6
-0.5	65	5	65	4	2508516	229.092	-0.6
-0.5	65	5	80	4	2508516	234.287	-0.6
-0.5	80	5	5	4	3886473	360.954	-1
-0.5	80	5	20	4	3584761	320.623	-1
-0.5	80	5	35	4	3167322	283.388	-1
-0.5	80	5	50	4	2090374	193.859	-0.6
-0.5	80	5	65	4	2090374	192.254	-0.6
-0.5	80	5	80	4	2090374	192.195	-0.6

Table A.3: Full results from various scenarios on chicane track shape.

Appendix B

Multi-Agent Reinforcement Learning Controller Reward Structure Details

We outline the specifics of the reward and penalty calculations in detail for our Multi-Agent Reinforcement Learning (MARL) low-level controller. Recall that the MARL-based agents observations include perfect state information for all players (including (x, y) position, v velocity, lane ID a, "recent" lane change count e, and last passed checkpoint r) and 9 LIDAR rays, whose distances we refer to as $I_1, \ldots, I_9$. Furthermore, we also assume players know the overall time elapsed in the game t, and the maximum time horizon T. We list functions $R(\cdot)$ that evaluate the rewards or penalties based on one or more weight parameters denoted by ω_i.

We list functions $R(\cdot)$ that evaluate the rewards or penalties based on one or more weight parameters denoted by ω_i. Our rewards and penalties are categorized into two types:

- For every time step in the environment, we provide the following rewards and penalties:
  - A reward for driving fast. The reward that scales based on the

driving close to the top speed of the kart.

$$R_{\text{speed}}(\omega_1) = \omega_1 \frac{v}{v_{\max}}$$

– A reward for moving towards the next checkpoint $r*$. We use the three-dimensional velocity vector of the agent and take the dot product with the vector between the agent's position and the next checkpoint position.

$$R_{\text{direction}}(\omega_1) = \omega_1(\langle v_x, v_y \rangle \cdot \langle r_x^* - x, r_y^* - y \rangle)$$

– A penalty for exceeding the lane changing limit. We use an indicator function to determine if the player is in the straight region of the track $\mathcal{S}$ and whether the lane changing limit L is exceeded.

$$R_{\text{swerve}}(\omega_1) = -\omega_1 \mathbb{1}_{(x,y)\in\mathcal{S}\wedge e>L}$$

– A penalty for being within h meters of the wall. We use an indicator function $\mathbb{1}_{I_j<h\wedge I_j\text{hit wall}}$ that determines if he LIDAR reading is below h and if whether the LIDAR bounced off a player or a wall.

$$R_{\text{wall-hit}}(\omega_1) = -\sum_{j=1}^{9} \omega_1 \mathbb{1}_{I_j<h\wedge I_j\text{hit wall}}$$

– A penalty for being within h meters of another player. Using a similar indicator function from above, if any LIDAR ray in that set hits another player within a distance h, then the original player is penalized for being in collision. In addition, we assume we have

a set Θ, which contains the indices of the LIDAR rays that point towards the front of the kart. There is an additional penalty if the LIDAR rays come from the subset Θ as that indicates some form of rear-end collision where the player would be at fault.

$$R_{\text{player-hit}}(\omega_1, \omega_2) = -\sum_{j=1}^{9}(\omega_1 \mathbb{1}_{I_j<h \wedge I_j \text{hit player}} + \omega_2 \mathbb{1}_{j\in\Theta})$$

- When a player passes a checkpoint with index r', we provide the following rewards and penalties:
  - A reward to teach the policy to pass as many checkpoints as possible before other players. The reward is scaled based on the order in which the checkpoint is reached. This reward is also added (with a different weight parameter) to a shared reward value used by the posthumous credit assignment algorithm to incentivize cooperative behavior.

$$R_{\text{checkpoint base}}(\omega_1) = \begin{cases} \omega_1 & \text{if first} \\ 0.75\omega_1 & \text{if second} \\ 0.6\omega_1 & \text{if third} \\ 0.4\omega_1 & \text{if fourth} \end{cases}$$

  - A reward based on the remaining time in the game to incentivize minimizing time between checkpoints. This reward is also added (with a different weight parameter) to a shared reward value used by the posthumous credit assignment algorithm to incentivize cooperative behavior.

$$R_{\text{checkpoint time}}(\omega_1) = \omega_1 \frac{T-t}{T}$$

– A reward for being closer to the target lane a' and velocity v' for the passed checkpoint.

$$R_{\text{checkpoint target}}(\omega_1, \omega_2) = \frac{\omega_1}{1.3^{|a-a'|\sqrt{(a'_x-x)^2+(a'_y-y)^2}}} + \frac{\omega_2}{1.1^{|v-v'|}}$$

– A penalty for driving in reverse. We use an indicator function to determine if checkpoint index r' is less than or equal to r implying the player passed either the same checkpoint or an earlier checkpoint.

$$R_{\text{checkpoint reverse}}(\omega_1) = -\omega_1 \mathbb{1}_{r' \leq r}$$

Appendix C

Results from head-to-head racing experiments

The following tables present all of the metrics collected for the head-to-head racing experiments from Chapter 4. All of the metrics are discussed in Chapter 4 except for DNFs and win margins. DNFs refers the number of races that an agent did not finish. Win margins refers to the number of seconds by which a player finished ahead of the other.

Agent 1	Agent 2	A1 Wins	A2 Wins	A1 DNFs	A2 DNFs	A1 Win Margins (s)	A2 Win Margins (s)	A1 Collisions-At-Fault	A2 Collisions-At-Fault	A1 Illegal Lane Changes	A2 Illegal Lane Changes
MCTS-RL	E2E	49	1	0	0	2.775	0.180	0.320	2.840	0.420	3.500
	Fixed-RL	46	4	0	0	0.609	0.190	1.400	1.180	0.240	0.380
	MCTS-LQR	49	1	0	0	2.160	0.360	0.840	0.580	0.200	1.800
	Fixed-LQR	47	3	0	0	0.638	2.227	0.420	0.840	0.160	0.000
E2E	Fixed-RL	3	47	0	0	0.113	2.588	1.820	0.920	4.060	0.340
	MCTS-LQR	17	33	0	0	0.546	2.684	0.540	0.500	3.540	2.700
	Fixed-LQR	49	1	0	0	0.719	2.960	0.120	0.940	2.120	0.100
Fixed-RL	MCTS-LQR	35	15	0	0	1.469	0.696	1.080	0.460	0.220	1.700
	Fixed-LQR	43	7	0	0	0.587	3.877	0.480	0.740	0.200	0.020
MCTS-LQR	Fixed-LQR	24	26	0	0	0.679	4.382	0.620	0.240	2.480	0.000

Table C.1: Results from head-to-head racing on the oval track.

Agent 1	Agent 2	A1 Wins	A2 Wins	A1 DNFs	A2 DNFs	A1 Win Margins (s)	A2 Win Margins (s)	A1 Collisions-At-Fault	A2 Collisions-At-Fault	A1 Illegal Lane Changes	A2 Illegal Lane Changes
MCTS-RL	E2E	48	0	2	2	4.651	0.000	0.540	1.060	0.080	1.180
	Fixed-RL	50	0	0	0	3.078	0.000	0.480	1.100	0.100	0.000
	MCTS-LQR	30	20	5	4	1.208	3.584	0.720	0.940	0.080	0.140
	Fixed-LQR	44	6	4	0	0.750	3.060	0.360	0.60	0.080	0.000
E2E	Fixed-RL	14	36	0	0	0.426	1.659	2.740	2.260	0.260	0.120
	MCTS-LQR	11	39	4	5	0.763	5.221	0.780	0.860	0.660	0.120
	Fixed-LQR	46	1	3	4	0.572	6.320	0.620	0.000	0.000	0.360
Fixed-RL	MCTS-LQR	10	40	5	6	1.120	5.629	1.180	0.880	0.180	0.060
	Fixed-LQR	28	18	11	4	0.630	8.027	1.020	0.260	0.120	0.000
MCTS-LQR	Fixed-LQR	29	21	1	0	1.440	2.091	0.460	0.240	0.100	0.000

Table C.2: Results from head-to-head racing on the complex track.

Appendix D

Results from team-based racing experiments

The following tables present all of the metrics collected for the team-based racing experiments from Chapter 5. All of the metrics are discussed in Chapter 5 except for DNFs and win margins. DNFs refers the number of races that of a specific team did not finish. Win margins refers to the number of seconds by which the overall winner finishes ahead of the first finishing player of the opposing team.

Agent 1	Agent 2	A1 Wins	A2 Wins	A1 DNFs	A2 DNFs	A1 Win Margins (s)	A2 Win Margins (s)	A1 Collisions-At-Fault	A2 Collisions-At-Fault	A1 Illegal Lane Changes	A2 Illegal Lane Changes	A1 Points Per Race	A2 Points Per Race
MCTS-RL	E2E	29	19	0	1	2.461	1.943	1.323	1.312	0.427	-0.521	15.344	12.073
	Fixed-RL	31	17	0	0	2.933	2.096	1.365	1.594	0.615	0.760	15.240	12.260
	MCTS-LQR	37	11	0	0	5.139	2.589	1.729	0.646	0.656	2.177	16.000	11.500
	Fixed-LQR	37	11	0	0	2.529	4.484	0.573	0.667	0.417	2.083	15.677	11.823
E2E	Fixed-RL	37	11	0	0	2.373	2.627	1.760	1.927	0.417	0.688	14.917	12.583
	MCTS-LQR	43	5	2	0	4.175	2.160	2.042	0.844	0.427	1.990	16.385	10.945
	Fixed-LQR	39	9	0	0	2.752	5.749	0.406	0.687	0.312	1.656	16.177	11.323
Fixed-RL	MCTS-LQR	27	21	1	1	4.368	4.793	2.427	0.615	0.967	2.396	14.542	12.792
	Fixed-LQR	35	13	0	0	3.764	6.892	0.979	0.635	0.604	2.177	15.438	12.063
MCTS-LQR	Fixed-LQR	32	16	0	0	3.094	8.013	0.323	0.521	2.234	3.021	14.708	12.792

Table D.1: Results from head-to-head racing on the oval track.

Agent 1	Agent 2	A1 Wins	A2 Wins	A1 DNFs	A2 DNFs	A1 Win Margins (s)	A2 Win Margins (s)	A1 Collisions-At-Fault	A2 Collisions-At-Fault	A1 Illegal Lane Changes	A2 Illegal Lane Changes	A1 Points Per Race	A2 Points Per Race
MCTS-RL	E2E	29	19	0	1.000	2.461	1.943	1.323	1.313	0.427	-0.521	15.344	12.073
	Fixed-RL	33	15	11	3	5.021	5.393	1.906	1.734	0.072	0.260	13.385	12.750
	MCTS-LQR	28	20	6	6	4.77	5.979	1	0.885	0.094	0.240	13.625	12.792
	Fixed-LQR	30	18	3	0	3.583	5.692	0.885	0.500	0.031	0.146	14.802	12.448
E2E	Fixed-RL	23	25	21	9	5.254	5.818	1.656	2.167	0.146	0.188	11.229	13.406
	MCTS-LQR	20	28	21	12	4.775	6.684	1.135	0.979	0.094	0.219	11.250	12.934
	Fixed-LQR	24	24	21	0	4.233	7.783	1.031	0.667	0.302	0.156	11.479	14.104
Fixed-RL	MCTS-LQR	17	31	3	3	3.54	6.567	1.438	0.938	0.198	0.219	13.281	13.677
	Fixed-LQR	28	20	9	3	2.594	7.295	0.990	0.854	0.167	0.188	13.844	12.531
MCTS-LQR	Fixed-LQR	27	21	4	0	3.645	5.810	0.469	0.177	0.135	0.115	13.771	13.396

Table D.2: Results from head-to-head racing on the complex track.

Bibliography

[1] Stephen Edelstein. The technologies your car inherited from race cars. https://www.digitaltrends.com/cars/racing-tech-in-your-current-car/, 2019. Accessed: 02-17-2022.

[2] Peter Nygaard. How simulators took drivers to portimao before the portuguese grand prix. https://www.formula1.com/en/latest/article.how-simulators-took-drivers-to-portimao-before-the-portuguese-grand-prix.1dtmSRoL9IJQvqwVOodYdd.html, oct 2020. Accessed: 02-17-2022.

[3] Alexander Liniger, Alexander Domahidi, and Manfred Morari. Optimization-based autonomous racing of 1:43 scale RC cars. *Optimal Control Applications and Methods*, 36(5):628–647, July 2014. doi: 10.1002/oca.2123. URL https://doi.org/10.1002/oca.2123.

[4] Jeffery R. Anderson, Beshah Ayalew, and T. Weiskircher. Modeling a professional driver in ultra-high performance maneuvers with a hybrid cost MPC. In *2016 American Control Conference (ACC)*. IEEE, July 2016. doi: 10.1109/acc.2016.7525209. URL https://doi.org/10.1109/acc.2016.7525209.

[5] Dvij Kalaria, Parv Maheshwari, Animesh Jha, Arnesh Issar, De-

bashish Chakravarty, Sohel Anwar, and Andres Tovar. Local nmpc on global optimised path for autonomous racing. In *2021 International Conference on Robotics and Automation (ICRA 2021) - Workshop Opportunities and Challenges With Autonomous Racing*. IEEE, 2021. URL https://linklab-uva.github.io/icra-autonomous-racing/contributed_papers/paper8.pdf.

[6] Daniel Kloeser, Tobias Schoels, Tommaso Sartor, Andrea Zanelli, Gianluca Prison, and Moritz Diehl. NMPC for racing using a singularity-free path-parametric model with obstacle avoidance. *IFAC-PapersOnLine*, 53 (2):14324–14329, 2020. doi: 10.1016/j.ifacol.2020.12.1376. URL https://doi.org/10.1016/j.ifacol.2020.12.1376.

[7] Matthew O'Kelly, Hongrui Zheng, Achin Jain, Joseph Auckley, Kim Luong, and Rahul Mangharam. TUNERCAR: A superoptimization toolchain for autonomous racing. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2020. doi: 10.1109/icra40945.2020.9197080. URL https://doi.org/10.1109/icra40945.2020.9197080.

[8] Jose L. Vazquez, Marius Bruhlmeier, Alexander Liniger, Alisa Rupenyan, and John Lygeros. Optimization-based hierarchical motion planning for autonomous racing. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, October 2020.

doi: 10.1109/iros45743.2020.9341731. URL https://doi.org/10.1109/iros45743.2020.9341731.

[9] Suiyi He, Jun Zeng, and Koushil Sreenath. Autonomous racing with multiple vehicles using a parallelized optimization with safety guarantee using control barrier functions, 2022. URL https://arxiv.org/abs/2112.06435.

[10] Jia-Hao Hou and Tsaipei Wang. The development of a simulated car racing controller based on monte-carlo tree search. In *2016 Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, pages 104–109, 2016. doi: 10.1109/TAAI.2016.7880111.

[11] Tim Stahl, Alexander Wischnewski, Johannes Betz, and Markus Lienkamp. Multilayer graph-based trajectory planning for race vehicles in dynamic scenarios. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, October 2019. doi: 10.1109/itsc.2019.8917032. URL https://doi.org/10.1109/itsc.2019.8917032.

[12] Alexander Liniger and John Lygeros. A noncooperative game approach to autonomous racing. *IEEE Transactions on Control Systems Technology*, 28(3):884–897, May 2020. doi: 10.1109/tcst.2019.2895282. URL https://doi.org/10.1109/tcst.2019.2895282.

[13] Nan Li, Eric Goubault, Laurent Pautet, and Sylvie Putot. Autonomous racecar control in head-to-head competition using mixed-integer quadratic

programming. In *2021 International Conference on Robotics and Automation (ICRA 2021) - Workshop Opportunities and Challenges With Autonomous Racing*. IEEE, 2021. URL https://linklab-uva.github.io/icra-autonomous-racing/contributed_papers/paper2.pdf.

[14] Johannes Betz, Hongrui Zheng, Alexander Liniger, Ugo Rosolia, Phillip Karle, Madhur Behl, Venkat Krovi, and Rahul Mangharam. Autonomous vehicles on the edge: A survey on autonomous vehicle racing, 2022.

[15] Juraj Kabzan, Lukas Hewing, Alexander Liniger, and Melanie N. Zeilinger. Learning-based model predictive control for autonomous racing. *IEEE Robotics and Automation Letters*, 4(4):3363–3370, October 2019. doi: 10.1109/lra.2019.2926677. URL https://doi.org/10.1109/lra.2019.2926677.

[16] Adrian Remonda, Sarah Krebs, Eduardo E. Veas, Granit Luzhnica, and Roman Kern. Formula RL: deep reinforcement learning for autonomous racing using telemetry data. *CoRR*, abs/2104.11106, 2021. URL https://arxiv.org/abs/2104.11106.

[17] Tim de Bruin, Jens Kober, Karl Tuyls, and Robert Babuska. Integrating state representation learning into deep reinforcement learning. *IEEE Robotics and Automation Letters*, 3(3):1394–1401, July 2018. doi: 10.1109/lra.2018.2800101. URL https://doi.org/10.1109/lra.2018.2800101.

[18] Trent Weiss and Madhur Behl. Deepracing: Parameterized trajectories for autonomous racing, 2020.

[19] Riccardo Spica, Eric Cristofalo, Zijian Wang, Eduardo Montijano, and Mac Schwager. A real-time game theoretic planner for autonomous two-player drone racing. *IEEE Transactions on Robotics*, 36(5):1389–1403, 2020.

[20] Mingyu Wang, Zijian Wang, John Talbot, J. Christian Gerdes, and Mac Schwager. Game theoretic planning for self-driving cars in competitive scenarios. In Antonio Bicchi, Hadas Kress-Gazit, and Seth Hutchinson, editors, *Robotics: Science and Systems XV, University of Freiburg, Freiburg im Breisgau, Germany, June 22-26, 2019*, 2019. doi: 10.15607/RSS.2019.XV.048. URL `https://doi.org/10.15607/RSS.2019.XV.048`.

[21] Mingyu Wang, Zijian Wang, John Talbot, J. Christian Gerdes, and Mac Schwager. Game-theoretic planning for self-driving cars in multivehicle competitive scenarios. *IEEE Transactions on Robotics*, pages 1–13, 2021. doi: 10.1109/tro.2020.3047521. URL `https://doi.org/10.1109/tro.2020.3047521`.

[22] Wilko Schwarting, Tim Seyde, Igor Gilitschenski, Lucas Liebenwein, Ryan Sander, Sertac Karaman, and Daniela Rus. Deep latent competition: Learning to race using visual control policies in latent space, 2021.

[23] Yunlong Song, HaoChih Lin, Elia Kaufmann, Peter Duerr, and Davide Scaramuzza. Autonomous overtaking in gran turismo sport using curriculum reinforcement learning, 2021.

[24] Peter R Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, et al. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228, 2022.

[25] Alexander Liniger. *Path Planning and Control for Autonomous Racing*. PhD thesis, ETH Zürich, 2018.

[26] Jaime F Fisac, Eli Bronstein, Elis Stefansson, Dorsa Sadigh, S Shankar Sastry, and Anca D Dragan. Hierarchical game-theoretic planning for autonomous vehicles. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9590–9596. IEEE, 2019.

[27] Majid Moghadam and Gabriel Hugh Elkaim. A hierarchical architecture for sequential decision-making in autonomous driving using deep reinforcement learning. *arXiv preprint arXiv:1906.08464*, 2019.

[28] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M Murray. Receding horizon temporal logic planning. *IEEE Transactions on Automatic Control*, 57(11):2817–2830, 2012.

[29] Tom Martin. The guide to road racing, part 8: Passing etiquette. https://www.windingroad.com/articles/blogs/the-road-racers-guide-to-passing-etiquette/, 2020. Accessed: 02-17-2022.

[30] T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis. Automatic verification of competitive stochastic systems. *Formal Methods in System Design*, 43(1):61–92, 2013.

[31] Rajesh Rajamani. *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[32] L. S. Shapley. Stochastic games*. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100, 1953. doi: 10.1073/pnas.39.10.1095. URL https://www.pnas.org/doi/abs/10.1073/pnas.39.10.1095.

[33] M. Kwiatkowska, G. Norman, D. Parker, and G. Santos. PRISM-games 3.0: Stochastic game verification with concurrency, equilibria and time. In *Proc. 32nd International Conference on Computer Aided Verification (CAV'20)*, volume 12225 of *LNCS*, pages 475–487. Springer, 2020.

[34] Rishabh Saumil Thakkar, Aryaman Singh Samyal, David Fridovich-Keil, Zhe Xu, and Ufuk Topcu. Hierarchical control for multi-agent autonomous racing. *arXiv preprint arXiv:2202.12861*, 2022.

[35] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[36] Tamer Başar and Geert Jan Olsder. *Dynamic noncooperative game theory*. SIAM, 1998.

[37] Unity Technologies. Unity technologies karting microgame template. `https://assetstore.unity.com/packages/templates/karting-microgame-150956`, 2021. Accessed: 10-2021.

[38] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, et al. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2018.

[39] Rishabh Saumil Thakkar, Aryaman Singh Samyal, David Fridovich-Keil, Zhe Xu, and Ufuk Topcu. Hierarchical control for cooperative teams in competitive autonomous racing. *arXiv preprint arXiv:2204.13070*, 2022.

[40] Andrew Cohen, Ervin Teng, Vincent-Pierre Berges, Ruo-Ping Dong, Hunter Henry, Marwan Mattar, Alexander Zook, and Sujoy Ganguly. On the use and misuse of absorbing states in multi-agent reinforcement learning. *arXiv preprint arXiv:2111.05992*, 2021.

[41] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.