S*: Geometric Multimodal Trajectory Optimization via Apex Interpolating Spiro Splines

by

Christopher W. Chang

S.B., Computer Science and Engineering, MIT (2020)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2022

Author . . . .
Department of Electrical Engineering and Computer Science
September 6, 2022

Certified by . . . .
Daniela Rus
Andrew and Erna Viterbi Professor of Computer Science
Thesis Supervisor

Accepted by . . . .
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

S*: Geometric Multimodal Trajectory Optimization via Apex Interpolating Spiro Splines

by

Christopher W. Chang

Submitted to the Department of Electrical Engineering and Computer Science
on September 6, 2022, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Non-heuristic multimodal trajectory optimization is widely considered an intractable holy grail for real-time robotic systems, with the existing state-of-the-art standing at a heuristic hierarchical approach that stacks upstream search-based or sampling-based behavior planning on top of downstream local numerical trajectory optimization. In this thesis, we present (i) the S* algorithm, a novel geometric trajectory optimization method for autonomous ground vehicles in dynamic environments that uses apex interpolating Spiro splines to optimize orders of magnitude fewer variables than numerical optimization, and (ii) an anytime best-first multimodal variant of S* using a parallel optimistic branch-and-bound on homology classes. We demonstrate a preliminary implementation of this algorithm integrated into MIT Driverless's autonomous racing stack on a full-size Roborace Devbot 2.0 racecar navigating mixed-reality obstacle courses at up to 100 mph.

Thesis Supervisor: Daniela Rus
Title: Andrew and Erna Viterbi Professor of Computer Science

Acknowledgments

This endeavor would not have been possible without the support of all the wonderful people on the MIT-PITT-RW autonomous racing team, which last year merged MIT Driverless, the Pitt Robotics and Automation Society (a.k.a. Pitt RAS), RIT Autonomous Racing, and Waterloo Autonomous Racing (a.k.a. WATORACE) into a superstar student-run team of epic proportions. I want to give special thanks to Nick Stathas for being an incredible mentor for the role of chief engineer and for all our brainstorming sessions; Andrew Saba for being an awesome, downright over-dedicated chief engineer who carries this team on his shoulders every day, often physically at the track; Brian Mao for enduring countless brainstorming and rubberducking sessions and making Las Vegas a blast; Joshua Spisak, Ben Zhang, and Prem Bharatia for literally being three of the best software engineers I know; Andy Keats for making sure no one died on track and being our resident race engineering guru; and Andrew Tresansky, Nayana Suvarna, Cindy Heredia, Rotimi Adetunji, and Jorge Castillo Lezama for holding the team together through many setbacks and somehow managing to purchase a racecar worth over a million dollars. I still get goosebumps knowing that we've raced at the Las Vegas Motor Speedway at over 140 mph, and I can't thank each and every member of the team enough for their dedication and grit.

I am also extremely grateful to my advisors, Daniela Rus, Aaron Ray, and Wilko Schwarting. Your feedback, suggestions, and challenging questions pushed me to expand the scope of my thesis to its present state.

Many thanks to the legendary Raph Levien, who graciously agreed to chat about his Spiro spline and the implications of incorporating it into the S* algorithm. I can't wait for you to see the Spiro spline in action when our newest results are published.

Words cannot express my gratitude to my parents, who have done so much to mitigate my worsening health as this thesis consumed my attention over the last two years. I love you, mom and dad.

I dedicate this thesis to my fiancée, Caitlyn Chen. After two long years, I'm finally coming home.

Contents

List of Figures

Chapter 1

Introduction

1.1 Why?

The holy grail of non-heuristic multimodal trajectory optimization belies one of the central struggles of robotic decision making in nonconvex environments — the computational tradeoff between global optimality and local optimality. The core building blocks of motion planning each excel at just one of these two objectives — search-based and sampling-based methods excel at high-level decision making, while local trajectory optimization excels at producing smooth, optimal trajectories. Today, any real-time robotic system hoping to achieve multiscale decision making uses a heuristic hierarchical approach, where an upstream behavior planner proposes a convex corridor, and a downstream numerical optimizer computes a smooth trajectory (see Chapter 2 for more).

This heuristic, however, is not infallible, and nowhere is this more clear than in the domain of autonomous racing, where inaccurate estimates can make the difference between a successful overtake and a failed overtake. In particular, while the S* algorithm is fairly extensible to other domains, it was directly inspired by MIT Driverless's participation in two recent international competitions in the autonomous racing space characterized by their record-breaking speeds and dynamic environments, namely the Indy Autonomous Challenge (IAC) and Roborace.

The Indy Autonomous Challenge is the world's first head-to-head autonomous racecar competition. Each of its nine teams purchase a full-size Dallara AV-21 racecar retrofitted with sensors and controls to enable automation. We then must get our car racing fully autonomously on iconic Indy tracks, both in single-agent and double-agent scenarios, at speeds of up to 173 mph. In particular, everything from perception to state estimation to planning to controls is left up to the teams. So far, IAC has raced on banked oval tracks at the Indianapolis Motor Speedway and the Las Vegas Motor Speedway.

Roborace is the world's first mixed-reality autonomous racecar competition. In its Season Beta, each of its six teams must automate a full-size Roborace Devbot 2.0 electric racecar to navigate the Roborace Metaverse, a mixed-reality racing format reminiscent of Mario Kart where we race solo on iconic tracks while avoiding dynamically-perceived virtual obstacles and hitting dynamically-perceived virtual targets. So far, Roborace Season Beta has featured flat but more complex tracks than IAC, including the Anglesey National Circuit, the Thruxton Circuit, the Bedford Autodrome, and the Las Vegas Motor Speedway's Outside Road Course. The challenge lies almost entirely in motion planning and controls — virtual obstacles and targets are directly perceived through a V2X framework, and robust state estimation is less important than in IAC because the tracks are not surrounded by walls and every obstacle is virtual, enabling us to easily restart an attempt if GPS fails and we exit the track. However, the flipside is that motion planning for Roborace is insanely challenging. Traditional hierarchical methods struggle to consistently make the right decision in complex multimodal scenarios like this; even to a professional, it may not be obvious whether it is worth the reward to diverge from the racing line for targets, or which path is optimal in a dense obstacle course. However, the whimsical challenge of the Metaverse format belies a more universal motivation for non-heuristic multimodal trajectory optimization — the importance of designing robotic systems with robustness and certifiability at the forefront.

The S* algorithm is our holistic answer to the challenge of tractable non-heuristic multimodal trajectory optimization for autonomous ground vehicles in dynamic environments. In truth, it may be overkill for Roborace, and it is definitely overkill for the Indy Autonomous Challenge. After all, except in highly complex scenarios, it suffices to optimize each of the most promising convex corridors in parallel. Between these two competitions, only the Roborace Metaverse presents a scenario complex enough to warrant non-heuristic multimodal optimization; but even then, why invest in a novel unconventional approach that takes years to develop when the tried-and-true method of hierarchical planning can achieve a good enough solution with enough tuning?

While it'd be funny to respond with "because we can", the truth is that robustness and certifiability have never been more important in robotics. The autonomous vehicle industry has long surpassed the goal of "good enough"; and yet, until it achieves "virtually perfect", or ideally, "provably perfect", it may never earn the trust it needs to overcome regulatory hurdles. The problem is, on-board computation also needs to be lightweight and fast. Hierarchical methods excel at this, hence why most implementations will split forecasting from planning, just as they split behavior planning from motion planning, despite major advances in game theory in recent years. While extending S* to joint motion forecasting and planning is out of the scope of this thesis, S* was conceived and pursued in part because of an ambition to one day consolidate forecasting and planning into a single universal, certifiable, and tractable algorithm.

1.2 How?

At its core, the S* algorithm was born from the intuition of a racing enthusiast. What if we could achieve tractable computation by simplifying the problem of racing line optimization to the problem of apex finding? In Chapter 3, we set up the problem, and in Chapter 4, we use this intuition to develop a geometric method of anytime local path optimization in static environments using apex interpolating Spiro splines.

In Chapter 5, we extend S* to trajectory optimization by decoupling path optimization and speed optimization. Only through our use of Spiro splines in Chapter 4 are we assured that this decoupled formulation is a good approximation for true trajectory optimization.

In Chapter 6, we extend S* to dynamic environments, which presents a major challenge to the geometric formulation of S* given that the concept of apexes is traditionally applied to static environments. It also motivates us to generalize speed optimization to account for the yield maneuver, i.e. when we slow down for an obstacle instead of nudging to either side.

Finally, in Chapter 7, we develop a best-first multimodal variant of S* using a parallel optimistic branch-and-bound on homology classes. This application of multimodal optimization is particularly synergistic for our geometric formulation in contrast to a numerical formulation, as we will see.

In Chapter 8, we demonstrate a preliminary implementation of S* integrated into MIT Driverless's autonomous racing stack on the Roborace Devbot 2.0 racecar navigating a static but dynamically-perceived obstacle course at up to 100 mph. We also show preliminary results of a more recent implementation of S* planning a single trajectory through randomly generated simple dynamic scenarios. While we still have a lot more results to collect, particularly results that feature the most recent implementation of S* navigating dynamic obstacle courses, we hope these limited results highlight the potential of S*. But there is still a lot to do, as we will discuss in Chapter 9.

1.3 Outline

The remainder of the thesis is split into the following chapters:

- Chapter 2 focuses on related work.
- Chapter 3 formally describes the planning problem.
- Chapter 4 solves a limited version of the problem described in Chapter 3, namely local path optimization in static environments, using a novel geometric approach based on apex interpolating Spiro splines.
- Chapter 5 extends this solution to trajectory optimization by describing a decoupled speed optimization algorithm and incorporating it into S*.

- Chapter 6 extends this solution to dynamic environments by generalizing the geometric concepts in Chapter 4 to spacetime, and extending the speed optimization concepts in Chapter 5 to enable the yield maneuver.

- Chapter 7 extends this solution to multimodal optimization using a parallel best-first branch-and-bound on homology classes.

- Chapter 8 presents results from preliminary implementations of S^*, most notably from a race in which MIT Driverless's autonomous racing stack navigated a full-size Roborace Devbot 2.0 racecar through mixed-reality obstacle courses at up to 100 mph.

- Chapter 9 discusses future work.

Chapter 2

Related Work

In this chapter we go over existing work in literature that either attempt to solve the same problem with a different approach, or that provide an insight that directly contributed to the conception of S*.

2.1 Local Trajectory Optimization

Numerical methods for local trajectory optimization in scenes of static/dynamic obstacles and targets are well-explored in literature. Methods like direct single shooting, direct multiple shooting, and direct collocation are standard — hundreds of variables representing control inputs and/or states are numerically optimized until convergence. However, these methods struggle to deal with complex or dynamic environments under limited computation. Timed elastic bands (TEBs) improve efficiency by using artificial forces to deform a trajectory to prevent violations, but they do not account for dynamic objects in spacetime [6].

Apex-based local trajectory optimization, which the Spline Racer algorithm expands on, has nearly no precedent in literature. The one existing work that resembles our approach is [2], which iteratively improves spline paths by computing collisions with obstacles and adding new waypoints on the obstacles' boundaries to mitigate collisions. However, this work only handles static obstacles, uses cubic splines, relies on heuristics to prune unnecessary waypoints, and takes computation on the order of seconds to produce a single local solution.

2.2 Multimodal Trajectory Optimization

None of the above numerical methods extend well to multimodal optimization on their own, on account of their gradient-based formulation. Instead, the standard is to utilize a heuristic hierarchical approach, relying on an upstream search-based or sampling-based behavior planner to select a few candidate homotopies for downstream optimization. For example, [7] samples a random acyclic graph to probabilistically explore homologies, and then solves a TEB for each homology in parallel. [4] searches a Voronoi graph for promising homotopies instead.

2.3 Spiro Splines

The S* algorithm relies on Raph Levien's Spiro spline, the four-parameter cubic cousin of the Euler spiral spline. In Chapter 4, we discuss why this spline family in particular is a great fit for S* and motion planning in general. The best reference on the Spiro spline, and its relation to other splines, is Raph Levien's PhD thesis [5].

2.4 Time-Optimal Speed Profiling

The problem of generating a time-optimal speed profile for a given path and initial speed subject to acceleration constraints is solved in literature, and S* only makes minor adjustments to the general idea to suit its needs. The general idea, which is well illustrated in section 3.5 of [3], is to overlay partial speed profiles, one resulting from a forward pass and another resulting from a backward pass, and take the minimum of the two profiles as the maximum speed that ensures recursive feasibility at each step. Acceleration constraints are often formulated using a GGV performance envelope, which is maps speed to a contour of feasible longitudinal/lateral accelerations.

2.5 Racing Theory

We take this opportunity to highlight a book by Adam Brouillard, an avid racer and specialist in the physics of racing line optimization, titled *The Perfect Corner: A Driver's Step-By-Step Guide to Finding Their Own Optimal Line Through the Physics of Racing (The Science of Speed)* [1]. His insights on the intuitive nature of the apex solidified our conviction that the Spiro spline was the ideal fit for S^*.

Chapter 3

Problem Formulation

In this chapter, we define the model of our planning problem.

3.1 Definitions

We denote *ego* as the agent for which we wish to plan a trajectory.

We represent ego's *3D configuration* as $q = (p, R, \kappa_y, \kappa_z)$, including position $p \in \mathbb{R}^3$, orientation $R \in SO(3)$, signed lateral curvature $\kappa_y \in \mathbb{R}$ (positive denotes curve left), and signed vertical curvature $\kappa_z \in \mathbb{R}$ (positive denotes curve up).

We also define ego's associated *2D configuration* $\tilde{q} = (\tilde{p}, \psi, \tilde{\kappa})$ as the topdown projection of q, including 2D position $\tilde{p} \in \mathbb{R}^2$, heading $\psi \in SO(2)$, and signed topdown curvature $\tilde{\kappa} \in \mathbb{R}$ (positive denotes curve counterclockwise). By querying road geometry, we can easily convert between q and $\tilde{q}$. As we will discuss, path optimization and collision handling are better handled in 2D than 3D, motivating this conversion.

We will sometimes refer to a 3D point/vector (x, y, z) in the *ego frame* $F_{\text{ego}} = (\mathbf{i}_{\text{ego}}, \mathbf{j}_{\text{ego}}, \mathbf{k}_{\text{ego}})$. This frame's origin is p and its axes align with R. In particular, $\mathbf{i}_{\text{ego}}$ points ahead of ego, $\mathbf{j}_{\text{ego}}$ points left of ego, and $\mathbf{k}_{\text{ego}}$ points up from ego. Unless explicitly stated, we will instead imply the *world frame* $F_{\text{world}} = (\mathbf{i}_{\text{world}}, \mathbf{j}_{\text{world}}, \mathbf{k}_{\text{world}})$.

We represent ego's *path* π as an arclength-parameterized sequence

$$\pi(s) = q, \quad 0 \leq s \leq S(\pi)$$

where $S(\pi)$ is the length of π.

We represent ego's *trajectory* τ as a time-parameterized sequence

$$\tau(t) = (q,\, v,\, a),\ \ 0 \leq t \leq T(\tau)$$

where $T(\tau)$ is the traversal time of τ, $v \in \mathbb{R}_{\geq 0}$ is speed, and $a = (a_x,\, a_y,\, a_z) \in \mathbb{R}^3$ is acceleration expressed in F_{ego} and split into contributions from gravity, road contact, and aerodynamics:

$$a = a_{\text{grav}} + a_{\text{road}} + a_{\text{aero}}$$

We encapsulate most of the determination of a at a given $(q,\, v)$ with an *acceleration model* $\mathcal{A}$. We can determine all but $a_{\text{road,x}}$ from $\mathcal{A}$, easily accounting for inclined roads, drag, downforce, and wind. We then explicitly control $a_{\text{road,x}}$ in order to speed up or slow down ego. $\mathcal{A}$ may also encapsulate constraints on $a_{\text{road,x}}$ such as throttle limits.

We define the *friction coefficient* μ associated with a_{road} as:

$$\mu(a_{\text{road}}) = \frac{\sqrt{a_{\text{road},x}^2 + a_{\text{road},y}^2}}{a_{\text{road},z}}$$

and we define $\mu(\tau)$ as the maximum such friction coefficient across all t.

We also define the *effort* η associated with a_{road}, best conceptualized as a signed friction coefficient, as:

$$\eta(a_{\text{road}}) = \begin{cases} \mu(a_{\text{road}}) & \text{if } a_{\text{road},x} > 0 \\ -\mu(a_{\text{road}}) & \text{otherwise} \end{cases}$$

We represent an *obstacle* o as the 2D topdown projection of its collision region parameterized by time, $o(t)$. Unless explicitly stated to be *static* obstacles, we will assume all obstacles are *dynamic*, i.e. $o(t)$ is not necessarily invariant with respect to t. We will denote the set of all obstacles as $\mathcal{O}$.

We define *safety distance* $\delta(\tau,\, o)$ as the minimum signed distance between $\tau(t)$

and $o(t)$ across all t (positive denotes non-collision), which we compute in 2D. We also define $\delta(\tau, \mathcal{O}) = \min_{o \in \mathcal{O}} \delta(\tau, o)$. We assume for collision handling purposes that ego is a point; this approach can easily handle circles by dilating obstacles. While we posit that it should be simple to extend to other shapes (e.g. trucks), this is outside the scope of this thesis.

We consider the homology class $h(\pi, \tilde{p}) \in \{ \text{left}, \text{right} \}$ of a path π with respect to an adjacent 2D point $\tilde{p}$ to be left if π lies left of $\tilde{p}$, and right if π lies right of $\tilde{p}$.

We formulate ego's *homology constraints* $\mathcal{H}$ with respect to $\mathcal{O}$ as:

$$\mathcal{H}(o) \subseteq \{ \text{left}, \text{yield}, \text{right} \}, \quad \forall\, o \in \mathcal{O}$$

If ego does not maintain a sufficient safety distance from o, then:

- If left $\in \mathcal{H}(o)$, then ego may nudge left to increase its safety distance.
- If right $\in \mathcal{H}(o)$, then ego may nudge right to increase its safety distance.
- If yield $\in \mathcal{H}(o)$, then ego may slow down while maintaining its nominal path to increase its safety distance.

Additionally, even if $|\mathcal{H}(o)| > 1$, only one such action may be taken. We also require $\mathcal{H}(o) \notin \{ \emptyset, \{ \text{left}, \text{right} \} \}$ because it simplifies our algorithm and there are no motivating use cases.

We will sometimes refer to a 2D point (s, d) in the *Frenet frame* $\mathcal{F}(\pi)$ of a path π, i.e. the point d units to the left of $\pi(s)$. A static obstacle o can intuitively be mapped to a set of points (s, d) in $\mathcal{F}(\pi)$.

We extend this concept to refer to a spacetime point (t, d) in the "Frenet frame" $\mathcal{F}(\tau)$ of a trajectory τ, i.e. the 2D point d units to the left of $\tau(t)$ at time t. A dynamic obstacle o can also intuitively be mapped to a set of points (t, d) in $\mathcal{F}(\tau)$.

We define a *corridor* $\mathcal{C}$ with respect to the Frenet frame $\mathcal{F}(\pi)$ of a path π as an arclength-parameterized sequence

$$\mathcal{C}(s) = (d_{\min}, d_{\max}), \quad 0 \leq s \leq S(\pi)$$

and similarly, we define a corridor $\mathcal{C}$ with respect to the Frenet frame $\mathcal{F}(\tau)$ of a trajectory τ as a time-parameterized sequence

$$\mathcal{C}(t) = (d_{\min},\, d_{\max}),\;\; 0 \le t \le T(\tau)$$

A corridor is considered *degenerate* if $d_{\min} > d_{\max}$ anywhere.

We define the *free corridor* between $\mathcal{O}_{\text{left}}$ and $\mathcal{O}_{\text{right}}$ with respect to the Frenet frame of a path or trajectory and subject to safety distance δ as the maximal corridor on the left of $\mathcal{O}_{\text{left}}$, on the right of $\mathcal{O}_{\text{right}}$, and maintaining safety distance δ from $\mathcal{O}_{\text{left}} \cup \mathcal{O}_{\text{right}}$. When this free corridor is degenerate, it implies that it is unreasonable to expect ego to respect these constraints.

3.2 "Optimization" Formulation

We wish to compute, in anytime best-first fashion, a library $\mathcal{T}$ of homologically distinct trajectories where each trajectory $\tau \in \mathcal{T}$ starts from initial configuration $\tilde{q}_i$ and initial speed v_i, ends at terminal configuration $\tilde{q}_f$, and never exceeds maximum effort $\eta_{\max}$. Except in extraordinary circumstances, we also reject any trajectories for which $\delta(\tau,\, \mathcal{O}) < 0$ or $\mu(\tau) > \mu_{\max}$, where $\mu_{\max}$ is the maximum feasible friction coefficient.

While we do not claim to minimize a particular objective function, we instead claim to heuristically balance the following conflicting objectives through our specialized approach:

- minimize $T(\tau)$
- penalize $\delta(\tau,\, \mathcal{O}) < \delta_{\text{safe}}$
- penalize $\mu(\tau) > \mu_{\text{safe}}$
- minimize jerk

where $\delta_{\text{safe}} \ge 0$ is the ideal safety distance and $\mu_{\text{safe}} \le \mu_{\max}$ is the ideal friction coefficient. For the purposes of optimization, we will assume that $\delta = \delta_{\text{safe}}$ and $\mu = \mu_{\text{safe}}$ are equally safe, while $\delta = 0$ and $\mu = \mu_{\max}$ are equally dangerous.

Chapter 4

Geometric Local Path Optimization in Static Environments

We begin the discussion on S* by considering a simple operational design domain (ODD) with many tractable solutions in literature — an autonomous ground vehicle (such as a racecar) navigating a static environment with a single desired homology class with respect to obstacles.

Numerical local path optimization, where we assume ego drives as fast as possible along the path, is largely a sufficient and tractable approach to this problem, failing only when a safe path does not exist. And when those failures arise, most implementations would opt to emergency stop.

However, while numerical optimization is tractable in this circumstance, we can greatly reduce the number of optimization variables by formulating the problem as a geometric optimization over apex interpolating Spiro splines. While quantitative proof of computational superiority is outside the scope of this thesis for lack of time, our limited results give us reason to believe that the efficiency of this method may enable S* to uniquely offer a tractable solution to multimodal trajectory optimization in dynamic environments, as we will cover in later chapters.

4.1 Assumptions

In this chapter, we add the following assumptions to the original problem formulation presented in Chapter 3:

- $\mathcal{H}(o) \in \{\, \{\, \text{left} \,\}, \{\, \text{right} \,\} \,\}, \ \ \forall\, o \in \mathcal{O}$
- o is static, i.e. $o(t)$ is invariant with respect to t, $\ \forall\, o \in \mathcal{O}$
- $\mu_{\text{safe}} = \infty$, i.e. there is no penalty for $\mu > \mu_{\text{safe}}$

Also, we compute a single path π instead of a trajectory library $\mathcal{T}$. We wish to generally minimize the traversal time and jerk of the associated time-optimal trajectory τ to π, but we will not directly compute τ.

4.2 Apex Interpolating Spiro Splines?

We introduce this concept with a key inspiration from the world of racing, as S* was conceived in response to the unique challenges of autonomous racing. Professional racecar drivers surely don't perform expensive numerical optimizations in their head; they instead focus on key pieces of information, and let muscle memory fill in the rest. One key concept is the *apex* — any point on a trajectory that just barely maintains the desired safety distance from the nearest obstacle. If we can represent trajectories using apexes instead of dense controls, we can eliminate orders of magnitude of variables from our optimization.

Assuming $\delta_{\text{safe}} = 0$ for simplicity, we arrive at our key insight: any trajectory that optimally avoids a set of obstacles can be dually represented as the optimal trajectory interpolating some sequence of apexes lying on the boundaries of those obstacles. (See Figure 4-1 for a visual example.) As a corollary, an apex sequence is a sufficient representation for any optimal trajectory.

We formally define an *apex sequence* α as a sequence

$$\alpha(i) = (\tilde{p},\, h), \ \ i = 1 \ldots |\alpha|$$

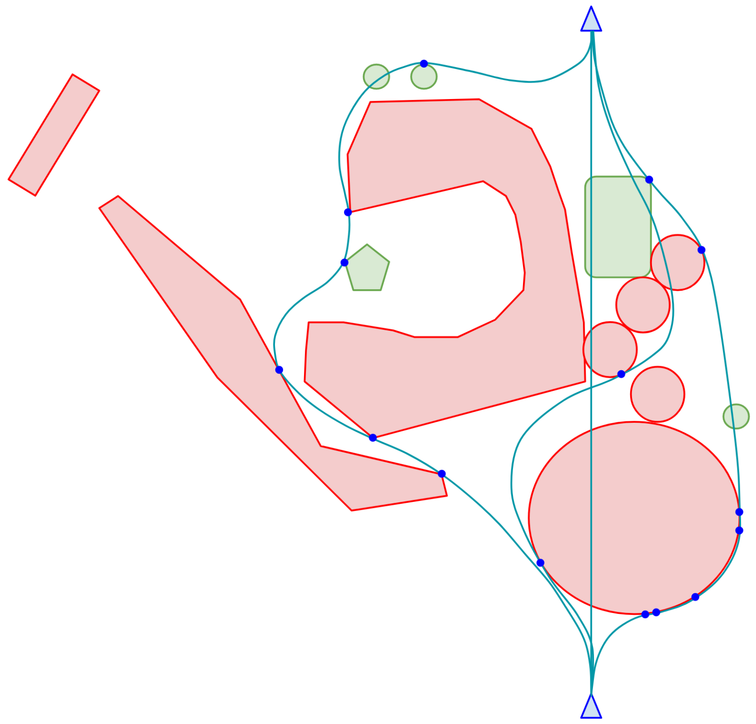
Figure 4-1: Any trajectory that optimally avoids a set of obstacles can be dually represented as the optimal trajectory interpolating some sequence of apexes lying on the boundaries of those obstacles. This visual example contains obstacles (in red) and targets (in green), alluding to an analogous statement for targets; however, that is out of scope for this thesis.

where $\tilde{p} \in \mathbb{R}^2$ is the apex's 2D position and $h \in \{ \text{left}, \text{right} \}$ is the apex's homology with respect to the nearest obstacle.

Returning to the limited ODD of this chapter, we posit that by alternating between optimizing an apex sequence α (initialized to the empty sequence) and interpolating a spline path from $\tilde{q}_i$ to $\tilde{q}_f$ via α, we will converge to an apex interpolating spline path π that minimizes $\delta(\pi, \mathcal{O})$. If our choice of spline family is prudent enough, we will also generally minimize the traversal time and jerk of the associated time-optimal trajectory in an efficient manner.

4.3 Spiro Splines?

To select a spline family, we first delineate the desired properties of our path.

In order for a path to be feasible when ego's initial speed is sufficiently low:

1. It must be G^2-continuous.

2. It must respect initial and terminal curvature constraints, κ_i and κ_f respectively.

Furthermore, from studying telemetry of professional racecar drivers, we propose the following criteria to be good indicators that a path resembles an optimal racing line:

3. It should be exceptionally smooth, at least G^3-continuous.

4. It should generally feature low curvature and low variation in curvature.

One might naively propose the commonly-used cubic spline, since it exhibits C^2-continuity and minimizes the integral of the square of the second derivative. Indeed, earlier iterations of S* represented paths as two cubic splines, parameterizing the x and y coordinates respectively, with a distance heuristic to assign t values to waypoints in order to approximate G^2-continuity. However, C^2-continuity is not G^2-continuity, and thus curvature is discontinuous at waypoints; and similarly, minimizing the second derivative is not equivalent to minimizing curvature. The remaining criteria are also entirely unmet.

The Euler spiral spline, also commonly used in motion planning for autonomous ground vehicles, exhibits G^2-continuity by constraining $\kappa(s)$ to be a piecewise linear function. It also efficiently approximates the Minimum Energy Curve (MEC), which minimizes $\int \kappa^2 \, ds$. However, it fails the remaining criteria. In fact, any two-parameter spline, including the cubic spline and the Euler spiral spline, must violate criterion 2.

Thus, we turn to Raph Levien's Spiro spline [5], the four-parameter cubic cousin of the Euler spiral spline. Owing to its four-parameter formulation, we can use it to satisfy criterion 2. Furthermore, it efficiently achieves many remarkable properties:

1. *G^4-continuity:* By constraining $\kappa(s)$ to be a piecewise cubic function, it achieves exceptional smoothness.

2. *Approximation of the Minimum Variation Curve:* This spline is an efficient high-fidelity small-angle approximation of the Minimum Variation Curve (MVC), which minimizes $\int (\frac{d\kappa}{ds})^2 \, ds$ and generally features low curvature.

3. *Extensionality:* Adding an additional waypoint on the curve of this spline will preserve its shape. Any spline that represents the optimal path through a sequence of points must have this property, because otherwise we would violate the Triangle Inequality.

4. *Roundness:* Interpolating points on a circle results in a circle. This feature is fairly unique to the MVC and the Spiro spline, and emulates racing at the limits of traction.

Further elaboration on these three splines, as well as an efficient implementation of the Spiro spline, can be found in Raph Levien's PhD thesis [5].

So far, Algorithm 1 summarizes our approach — starting from the empty apex sequence (line 3), we alternate between optimizing a Spiro spline path π (line 5) and improving our apex sequence α (line 6) until we converge. As long as there exists an efficient method to iteratively improve α based on the previously computed π, and as long as that method converges, then we should be able to solve this ODD.

Algorithm 1 The S* Algorithm (local, static environments, path only, high level)

```
1: $\mathcal{O}_{\text{left}} \leftarrow \{\, o \in \mathcal{O} \mid \mathcal{H}(o) = \{\text{ left }\}\,\}$
2: $\mathcal{O}_{\text{right}} \leftarrow \{\, o \in \mathcal{O} \mid \mathcal{H}(o) = \{\text{ right }\}\,\}$
3: $\alpha \leftarrow$ empty apex sequence
4: repeat
5:     $\pi \leftarrow$ Spiro spline path from $\tilde{q}_i$ to $\tilde{q}_f$ via $\alpha$
6:     $\alpha \leftarrow$ improve $\alpha \mid \pi,\ \mathcal{O}_{\text{left}},\ \mathcal{O}_{\text{right}},\ \delta_{\text{safe}}$
7: until $\Delta\alpha \approx 0$ or search is aborted
8: return $\pi$
```

4.4 Geometric Apex Optimization via "Inflation"

At first, we consider an intuitive geometric method, which we will call *inflation*, that updates α independently of the previous α. This naive approach is incorporated in Algorithm 2.

The concept of inflation is best explained visually — see Figure 4-2, Figure 4-3, and Figure 4-4 for a great introductory example. Intuitively, if a trajectory collides with

Algorithm 2 The S* Algorithm (local, static environments, path only, naive)

```
1: $\mathcal{O}_{\text{left}} \leftarrow \{\, o \in \mathcal{O} \mid \mathcal{H}(o) = \{\,\text{left}\,\}\,\}$
2: $\mathcal{O}_{\text{right}} \leftarrow \{\, o \in \mathcal{O} \mid \mathcal{H}(o) = \{\,\text{right}\,\}\,\}$
3: $\alpha \leftarrow$ empty apex sequence
4: **repeat**
5:     $\pi \leftarrow$ Spiro spline path from $\tilde{q}_i$ to $\tilde{q}_f$ via $\alpha$
6:     $\mathcal{C} \leftarrow$ free corridor between $\mathcal{O}_{\text{left}}$ and $\mathcal{O}_{\text{right}}$ wrt $\mathcal{F}(\pi) \mid \delta_{\text{safe}}$
7:     $\alpha \leftarrow$ inflate $\pi \mid \mathcal{C}$
8: **until** $\Delta\alpha \approx 0$ **or** search is aborted
9: **return** $\pi$
```

an obstacle, a sensible candidate for a new apex would be the leftmost or rightmost point of the obstacle with respect to the trajectory. In fact, this concept works for dynamic obstacles too, as we will explore further in Chapter 6. However, if we add every local extremum as an apex, then it can result in overconstrained trajectories — see Figure 8-4 for an example. In general, we seek to utilize as few apexes as possible in order to avoid obstacles. Thus, for now, we limit ourselves to apex sequences that alternate between left and right apexes, and when given the choice between two consecutive left or right apexes, we select the more extreme one.

In order to facilitate this, we first compute the free corridor $\mathcal{C}$ between $\mathcal{O}_{\text{left}}$ and $\mathcal{O}_{\text{right}}$ with respect to $\mathcal{F}(\pi)$ and subject to ideal safety distance δ_{safe}, as defined in Chapter 3, on line 6. Now, we only consider adding apexes whenever $d_{\min} > 0$ or $d_{\max} < 0$; furthermore, we only consider alternating extrema of $d_{\min}$ and $d_{\max}$. We then improve our apex sequence α through the aforementioned process of inflation on line 7.

One important edge case is when it is infeasible to maintain the ideal safety distance δ_{safe} because $\mathcal{H}$ requires that ego drive between two obstacles that are not sufficiently far apart. In this case, we find that $\mathcal{C}$ is degenerate, and our solution is fairly simple — widen $\mathcal{C}$ until it is no longer degenerate, i.e. decrease $d_{\min}$ and increase $d_{\max}$ everywhere until $d_{\min} \leq d_{\max}$ everywhere. In practice, this is a computationally efficient local approximation of simply reducing the required safety distance until it is feasible, and it converges to the same solution.

So far, this approach succeeds whenever the optimal path can be represented

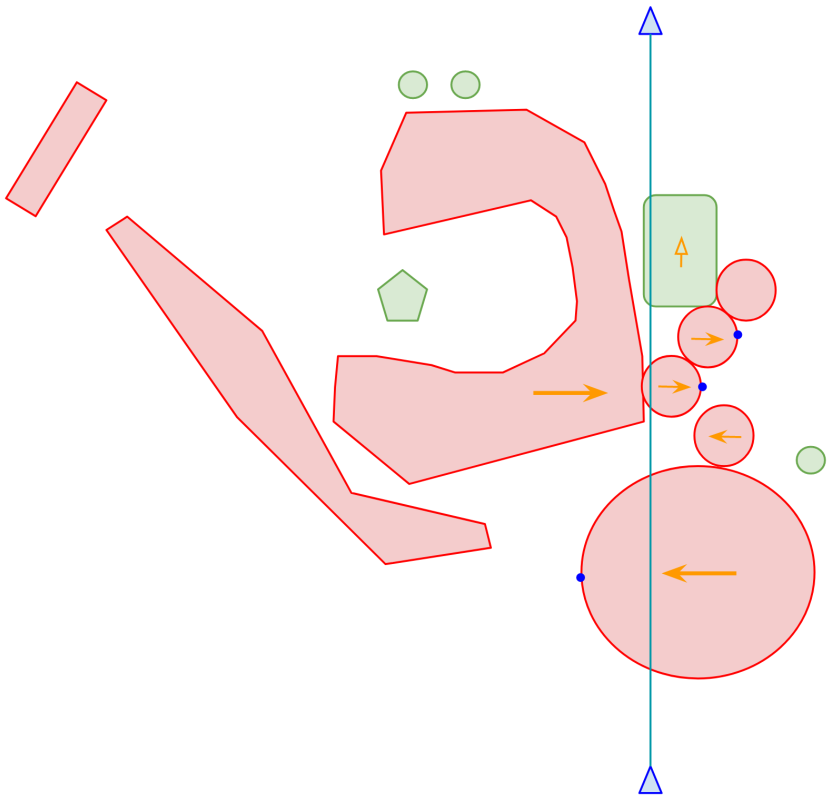

Figure 4-2: Example scenario, where red objects are obstacles and green objects are targets. (Ignore targets for the purposes of this thesis.) Homology constraints are indicated for some of the obstacles, and any obstacles without homology constraints should be ignored. Certain extrema are selected as candidate apexes.

by an apex sequence of alternating left and right apexes. And to be fair, this case accounts for many common racing scenarios; so-called "double apexes" are relatively uncommon. However, we must be robust to cases that require double or even triple apexes, such as in Figure 4-5. Our current approach would result in an infinite loop — in our current example, we would cycle between the two one-apex solutions in Figure 4-6 and Figure 4-7, never converging to the correct solution in Figure 4-5.

At this point, we throw out our ambition to update α independently of the previous α. We want each successive path π to be strictly better than the last, and thus, if on a previous step α contained some apex $(\tilde{p}, h)$, then the only valid reason for α to later not contain that apex would be if π is already in the correct homology class and would thus be overconstrained by that apex, i.e. $h(\pi, \tilde{p}) = h$.

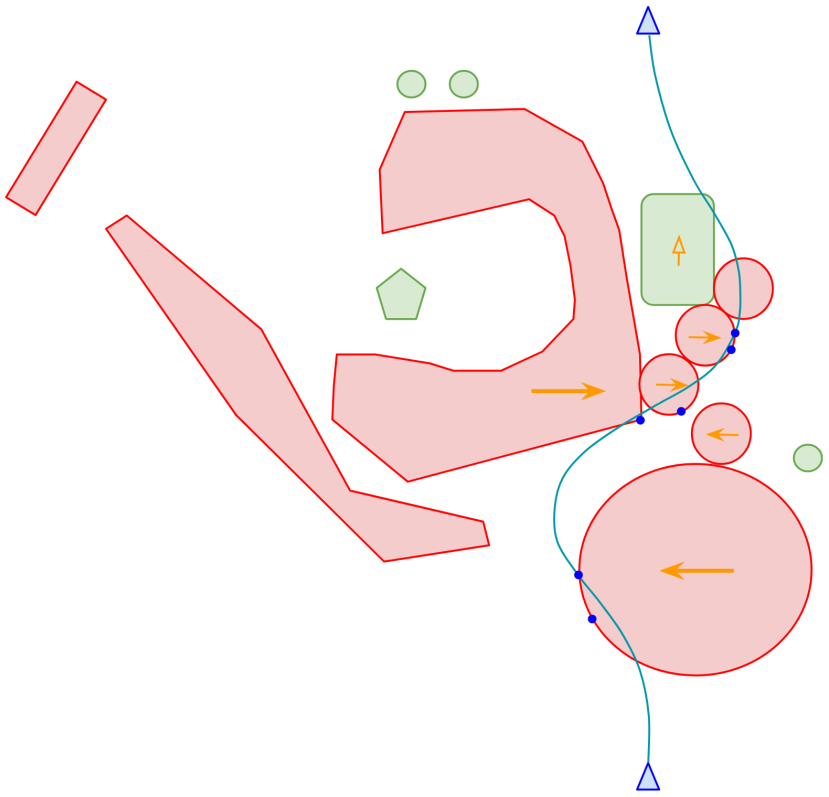

Figure 4-3: The next iteration after Figure 4-2. Candidate apexes are again indicated.

This approach is finalized in Algorithm 3. To summarize the changes from Algorithm 2, we introduce a new variable β that stores the apex sequence optimized in the previous iteration. Then, every time we compute our Spiro spline path π from our apex sequence α on line 6, we check if any apex in β is violated by our path π on line 7; if any are, then we insert them into α and recompute π. This process prevents any new path from undoing progress that was made in the previous iteration.

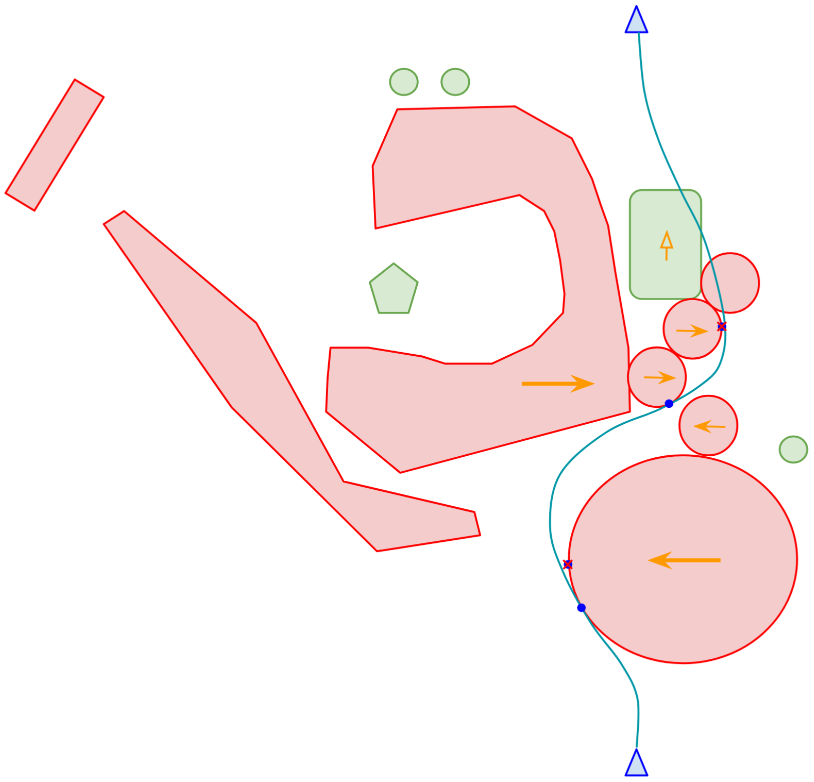

Figure 4-4: The next iteration after Figure 4-3, which replaces the previous apexes with new apexes. At this point, the path sufficiently respects the homology constraints.

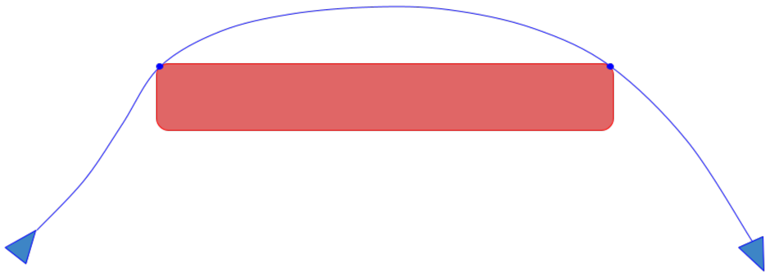

Figure 4-5: A double left apex maneuver.

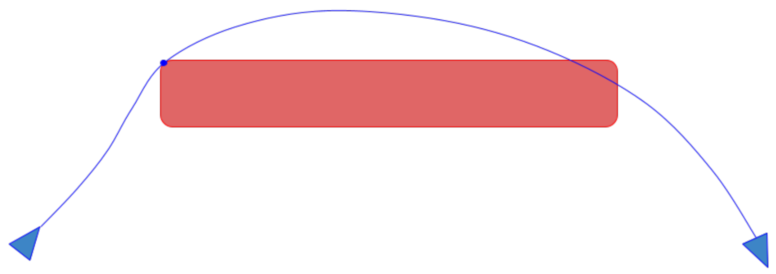

Figure 4-6: A failed attempt to solve Figure 4-5 with only one left apex.

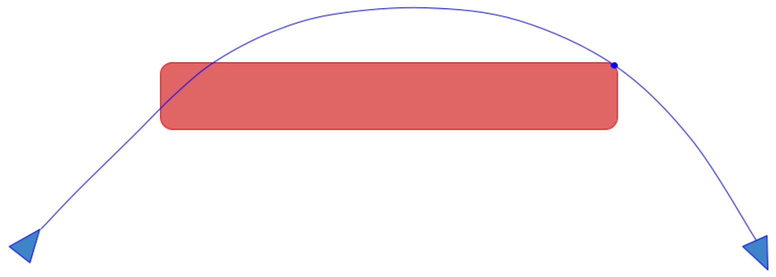

Figure 4-7: Another failed attempt to solve Figure 4-5 with only one left apex. A naive algorithm might oscillate between this solution and the solution in Figure 4-6, never converging to the correct solution in Figure 4-5.

Algorithm 3 The S* Algorithm (local, static environments, path only)

```
1: $\mathcal{O}_{\text{left}} \leftarrow \{\, o \in \mathcal{O} \mid \mathcal{H}(o) = \{\, \text{left} \,\} \,\}$
2: $\mathcal{O}_{\text{right}} \leftarrow \{\, o \in \mathcal{O} \mid \mathcal{H}(o) = \{\, \text{right} \,\} \,\}$
3: $\alpha,\ \beta \leftarrow$ empty apex sequences
4: **repeat**
5:     **repeat**
6:         $\pi \leftarrow$ Spiro spline path from $\tilde{q}_i$ to $\tilde{q}_f$ via $\alpha$
7:         $\alpha \leftarrow \alpha \cup \{\, (\tilde{p},\ h) \in \beta \mid h(\pi,\ \tilde{p}) \neq h \,\}$
8:     **until** $\Delta\alpha = 0$
9:     $\mathcal{C} \leftarrow$ free corridor between $\mathcal{O}_{\text{left}}$ and $\mathcal{O}_{\text{right}}$ wrt $\mathcal{F}(\pi) \mid \delta_{\text{safe}}$
10:    $\beta \leftarrow \alpha$
11:    $\alpha \leftarrow$ inflate $\pi \mid \mathcal{C}$
12: **until** $\Delta\alpha \approx 0$ **or** search is aborted
13: **return** $\pi$
```

Chapter 5

Extending to Trajectory Optimization via Decoupled Speed Profiles

So far, we have succeeded in computing a path that is likely to resemble the time-optimal trajectory through a static environment, but we have yet to actually compute this trajectory. We posit that because our path optimization step already approximates the time-optimal trajectory without directly computing it, it suffices here to develop a decoupled speed profiling step. In the racing world, computing this time-optimal speed profile is non-trivial because it is often necessary to hit the brakes ahead of a future high-curvature configuration in order to maintain recursive feasibility. In short, just as the path optimization module is responsible for finding apexes, this module must be capable of finding braking points.

We must also incorporate the speed profiler into each iteration of the optimization, instead of relegating it to an entirely downstream process. Computing the full trajectory τ during optimization serves a few purposes:

- When $\mu(\tau) > \mu_{\text{safe}}$, it signals that the underlying path π may have been constructed using unreasonable safety distance expectations. Thus, we can modulate our desired safety distance $\delta^* \leq \delta_{\text{safe}}$ in order to strike a reasonable tradeoff between $\mu(\tau)$ and $\delta(\tau, \mathcal{O})$.

- In Chapter 6, when we extend to dynamic environments, we will require a time-parameterized trajectory in order to make a comparison with dynamic obstacles.

- In Chapter 7, when we extend to multimodal optimization, we will use traversal time as a heuristic acquisition function for our branch-and-bound, requiring the computation of τ.

5.1 Assumptions

In this chapter, we add the following assumptions to the original problem formulation presented in Chapter 3:

- $\mathcal{H}(o) \in \{ \{ \text{left} \}, \{ \text{right} \} \}, \ \forall\, o \in \mathcal{O}$
- o is static, i.e. $o(t)$ is invariant with respect to t, $\forall\, o \in \mathcal{O}$

Also, we compute a single trajectory τ instead of a trajectory library $\mathcal{T}$.

5.2 Speed Optimization by Finding Braking Points

First, we focus on the singular task of computing the time-optimal trajectory τ along a path π subject to initial speed v_i, acceleration model $\mathcal{A}$, and maximum effort $\eta_{\max}$. Recall that our acceleration model $\mathcal{A}$ maps (q, v, η) to an acceleration a. Thus, in the simplest case where ego accelerates with constant effort η, speed profiling boils down to Algorithm 4, which computes the acceleration a associated with effort η at each state on line 4, and then forecasts the speed at the next state on line 5.

Algorithm 4 Speed Profiling (constant effort, assumes feasibility)

```
1: // fails if $\eta$ is infeasible
2: $v(0) \leftarrow v_i$
3: for $i = 0 \ldots N-1$ do
4:     $a(i) \leftarrow \mathcal{A}(q(i), v(i), \eta)$
5:     $v(i+1) \leftarrow$ forecast $v(i)$, $a(i)$
```

However, the problem gets more interesting when we observe that it may not be feasible to maintain that constant effort without later violating friction constraints.

Given our maximum effort $\eta_{\max}$, we set our minimum desired effort $\eta^*_{\min} = -|\eta_{\max}|$ because intuitively, a reasonably comfortable trajectory should accelerate and brake at the same friction coefficient except when acceleration is throttle-limited. For the next part, we assume it is feasible to maintain $\eta \in [\eta^*_{\min}, \eta_{\max}]$ along τ.

Algorithm 5 summarizes our approach, which boils down to a backward pass followed by a forward pass. In the backward pass (lines 3-8), we recursively compute the maximum speed at each configuration such that the remaining path is traversable without violating constraints on η. In the forward pass (lines 10-14), we forecast our speed from v_i while respecting the maximum speeds computed in the backward pass. So far, the approach described is a literature-standard forward-backward-solver for computing time-optimal speed profiles subject to a GGV performance envelope — a clear example can be found in section 3.5 of [3].

Algorithm 5 Speed Profiling (assumes feasibility)

```
1:  $\eta^*_{\min} \leftarrow -|\eta_{\max}|$
2:  // backward pass
3:  for $i = 0 \dots N$ do
4:      $v(i) \leftarrow$ maximum feasible speed at $q(i) \mid \mathcal{A}, \eta^*_{\min}$
5:  for $i = N \dots 1$ do
6:      $a(i) \leftarrow \mathcal{A}(q(i), v(i), \eta^*_{\min})$
7:      $v' \leftarrow$ backprop $v(i), a(i)$
8:      $v(i-1) \leftarrow \min\{ v(i-1), v' \}$
9:  // forward pass, fails if $v_i > v(0)$
10: $v(0) \leftarrow v_i$
11: for $i = 0 \dots N-1$ do
12:     $a(i) \leftarrow \mathcal{A}(q(i), v(i), \eta)$
13:     $v' \leftarrow$ forecast $v(i), a(i)$
14:     $v(i+1) \leftarrow \min\{ v(i+1), v' \}$
```

There remains one issue, namely that the constraints on η may be infeasible. We observe this whenever we find that $v_i > v(0)$ on line 10 of Algorithm 5. In that case, it suffices to optimize for the maximum $\eta^*_{\min} \leq -|\eta_{\max}|$ that results in $v_i \leq v(0)$. Intuitively, this means that when ego cannot feasibly maintain effort $\eta \in [\eta^*_{\min}, \eta_{\max}]$, we simply brake harder until that constraint becomes feasible. The final iteration of the speed profiler is highlighted in Algorithm 6 — in essence, we repeat the backward pass until we find the maximum such $\eta^*_{\min}$.

Algorithm 6 Speed Profiling

```
1: $\eta^*_{\min} \leftarrow -|\eta_{\max}|$
2: repeat
3:     // backward pass
4:     for $i = 0 \ldots N$ do
5:         $v(i) \leftarrow$ maximum feasible speed at $q(i) \mid \mathcal{A},\, \eta^*_{\min}$
6:     for $i = N \ldots 1$ do
7:         $a(i) \leftarrow \mathcal{A}(q(i),\, v(i),\, \eta^*_{\min})$
8:         $v' \leftarrow$ backprop $v(i),\, a(i)$
9:         $v(i-1) \leftarrow \min\{\, v(i-1),\, v' \,\}$
10: until maximized $\eta^*_{\min} \le -|\eta_{\max}|$ s.t. $v_i \le v(0)$
11: // forward pass
12: $v(0) \leftarrow v_i$
13: for $i = 0 \ldots N-1$ do
14:     $a(i) \leftarrow \mathcal{A}(q(i),\, v(i),\, \eta)$
15:     $v' \leftarrow$ forecast $v(i),\, a(i)$
16:     $v(i+1) \leftarrow \min\{\, v(i+1),\, v' \,\}$
```

We initially integrate speed optimization into S* in Algorithm 7 — the only change is on line 9, where we compute a trajectory τ associated with path π. However, in the next section, we will discuss how we can take advantage of our computation of τ to make a tradeoff between $\delta < \delta_{\text{safe}}$ and $\mu > \mu_{\text{safe}}$.

Algorithm 7 The S* Algorithm (local, static environments, no tradeoff)

```
1: $\mathcal{O}_{\text{left}} \leftarrow \{\, o \in \mathcal{O} \mid \mathcal{H}(o) = \{\text{left}\}\,\}$
2: $\mathcal{O}_{\text{right}} \leftarrow \{\, o \in \mathcal{O} \mid \mathcal{H}(o) = \{\text{right}\}\,\}$
3: $\alpha,\, \beta \leftarrow$ empty apex sequences
4: repeat
5:     repeat
6:         $\pi \leftarrow$ Spiro spline path from $\tilde{q}_i$ to $\tilde{q}_f$ via $\alpha$
7:         $\alpha \leftarrow \alpha \cup \{\, (\tilde{p},\, h) \in \beta \mid h(\pi,\, \tilde{p}) \neq h \,\}$
8:     until $\Delta\alpha = 0$
9:     $\tau \leftarrow$ time-optimal trajectory along $\pi$ from $v_i \mid \mathcal{A},\, \eta_{\max}$
10:    $\mathcal{C} \leftarrow$ free corridor between $\mathcal{O}_{\text{left}}$ and $\mathcal{O}_{\text{right}}$ wrt $\mathcal{F}(\pi) \mid \delta_{\text{safe}}$
11:    $\beta \leftarrow \alpha$
12:    $\alpha \leftarrow$ inflate $\pi \mid \mathcal{C}$
13: until $\Delta\alpha \approx 0$ or search is aborted
14: return $\tau$
```

5.3 Trading Off Safety Distance and Friction

One of our goals is to make a decent tradeoff between penalizing $\delta < \delta_{\text{safe}}$ and penalizing $\mu > \mu_{\text{safe}}$. For example, it would be undesirable to maintain a large safety distance at the expense of losing control of the vehicle, as it would be undesirable to limit traction at the expense of grazing an obstacle.

We previously proposed to consider $\delta = \delta_{\text{safe}}$ and $\mu = \mu_{\text{safe}}$ as equally safe, and $\delta = 0$ and $\mu = \mu_{\max}$ as equally dangerous. Thus, given a value for μ, our desired safety distance δ^* should be

$$\delta^* = \delta_{\text{safe}} \times \min\left\{ 1, \frac{\mu_{\max} - \mu}{\mu_{\max} - \mu_{\text{safe}}} \right\}$$

Incorporating this into Algorithm 8 on lines 10-11, we end up converging to a point where $\delta(\tau, \mathcal{O})$ and $\mu(\tau)$ are equally penalized.

There is one additional detail in Algorithm 8 that becomes necessary because of this, namely the "deflation" in line 12. Because we are modulating δ^* from iteration to iteration, an apex planned in the previous iteration may be too conservative if δ^* has decreased in the current iteration. Thus, we correct this by nudging any overly conservative apexes back subject to the new desired safety distance δ^* — we call this process "deflation" because it makes intuitive sense as the reverse of inflation. It is actually possible for this to result in an infinite loop of inflation and deflation; we fix this easily by decaying the maximum displacement every iteration, though this is not depicted in the pseudocode. In Chapter 6, deflation will also double as a way to account for conservatism owing to the unique challenges of dealing with dynamic obstacles, as we will discuss.

Algorithm 8 The S* Algorithm (local, static environments)

```
 1: $\mathcal{O}_{\text{left}} \leftarrow \{\, o \in \mathcal{O} \;\mid\; \mathcal{H}(o) = \{\, \text{left} \,\}\, \}$
 2: $\mathcal{O}_{\text{right}} \leftarrow \{\, o \in \mathcal{O} \;\mid\; \mathcal{H}(o) = \{\, \text{right} \,\}\, \}$
 3: $\alpha,\, \beta \leftarrow$ empty apex sequences
 4: repeat
 5:     repeat
 6:         $\pi \leftarrow$ Spiro spline path from $\tilde{q}_i$ to $\tilde{q}_f$ via $\alpha$
 7:         $\alpha \leftarrow \alpha \,\cup\, \{\, (\tilde{p},\, h) \in \beta \;\mid\; h(\pi,\, \tilde{p}) \neq h \,\}$
 8:     until $\Delta\alpha = 0$
 9:     $\tau \leftarrow$ time-optimal trajectory along $\pi$ from $v_i \;\mid\; \mathcal{A},\, \eta_{\max}$
10:     $\delta^* \leftarrow \delta_{\text{safe}} \times \min\left\{ 1,\, \frac{\mu_{\max} - \mu(\tau)}{\mu_{\max} - \mu_{\text{safe}}} \right\}$
11:     $\mathcal{C} \leftarrow$ free corridor between $\mathcal{O}_{\text{left}}$ and $\mathcal{O}_{\text{right}}$ wrt $\mathcal{F}(\pi) \;\mid\; \delta^*$
12:     $\beta \leftarrow$ deflate $\alpha \;\mid\; \mathcal{C}$
13:     $\alpha \leftarrow$ inflate $\pi \;\mid\; \mathcal{C}$
14: until $\max\{\, \Delta\alpha,\, \Delta\beta \,\} \approx 0$ or search is aborted
15: return $\tau$
```

Chapter 6

Extending to Dynamic Environments

We now set our eyes on a new horizon: dynamic environments. Considering that the path optimization of S* has clearly been predicated on geometric concepts that make sense only in static environments, you would think this would be a huge stretch. And you'd be right — while the solution to this ambition, Algorithm 9, looks fairly similar in pseudocode to Algorithm 8 from the previous chapter, this observation neglects the fact that Algorithm 8 was meticulously formulated and iterated on in order to easily extend to dynamic environments.

First, we will extend the geometric concepts from Chapter 4 to spacetime. The only corresponding change to the pseudocode will be the shift from $\mathcal{F}(\pi)$ to $\mathcal{F}(\tau)$ on line 15 of Algorithm 9; and yet, this singular change belies a fairly considerable shift in thinking.

Second, we will introduce the yield maneuver, i.e. slowing down for an obstacle instead of nudging to either side, rounding out the remaining changes in Algorithm 9. Thanks to our convenient formulation of speed optimization in Chapter 5, this will be easily integrated.

Algorithm 9 The S* Algorithm (local, dynamic environments)

```
 1: $\mathcal{O}_{\text{left}} \leftarrow \{\, o \in \mathcal{O} \mid \mathcal{H}(o) = \{\text{left}\}\,\}$
 2: $\mathcal{O}_{\text{right}} \leftarrow \{\, o \in \mathcal{O} \mid \mathcal{H}(o) = \{\text{right}\}\,\}$
 3: $\mathcal{O}_{\text{yield}} \leftarrow \{\, o \in \mathcal{O} \mid \mathcal{H}(o) = \{\text{yield}\}\,\}$
 4: $\alpha,\, \beta \leftarrow$ empty apex sequences
 5: $\eta^*_{\max} \leftarrow \eta_{\max}$
 6: repeat
 7:     repeat
 8:         $\pi \leftarrow$ Spiro spline path from $\tilde{q}_i$ to $\tilde{q}_f$ via $\alpha$
 9:         $\alpha \leftarrow \alpha \cup \{\, (\tilde{p},\, h) \in \beta \mid h(\pi,\, \tilde{p}) \neq h \,\}$
10:     until $\Delta\alpha = 0$
11:     repeat
12:         $\tau \leftarrow$ time-optimal trajectory along $\pi$ from $v_i \mid \mathcal{A},\, \eta^*_{\max}$
13:         $\delta^* \leftarrow \delta_{\text{safe}} \times \min\left\{ 1,\, \frac{\mu_{\max}-\mu(\tau)}{\mu_{\max}-\mu_{\text{safe}}} \right\}$
14:     until maximized $\eta^*_{\max} \leq \eta_{\max}$ s.t. $\delta(\tau,\, \mathcal{O}_{\text{yield}}) \geq \delta^*$
15:     $\mathcal{C} \leftarrow$ free corridor between $\mathcal{O}_{\text{left}}$ and $\mathcal{O}_{\text{right}}$ wrt $\mathcal{F}(\tau) \mid \delta^*$
16:     $\beta \leftarrow$ deflate $\alpha \mid \mathcal{C}$
17:     $\alpha \leftarrow$ inflate $\tau \mid \mathcal{C}$
18: until $\max\{\, \Delta\alpha,\, \Delta\beta \,\} \approx 0$ or search is aborted
19: return $\tau$
```

6.1 Assumptions

In this chapter, the only assumption we add to the original problem formulation presented in Chapter 3 is that

$$\mathcal{H}(o) \in \{\, \{\text{left}\}\,,\, \{\text{right}\}\,,\, \{\text{yield}\}\,\}\,, \quad \forall\, o \in \mathcal{O}$$

Also, we compute a single trajectory τ instead of a trajectory library $\mathcal{T}$.

6.2 Extending Apex Optimization to Spacetime

The central incompatibility that dynamic obstacles present to our current formulation in Algorithm 8 is that our intuition of selecting extrema in the Frenet frame of our path as apexes makes little sense in spacetime. However, one intuition remains valid — even if ego and an obstacle are in motion, the most extreme left or right displacement ego would need in order to avoid the obstacle over the course of their respective trajectories would be a sensible candidate for a new apex.

Thus, we are motivated to extend the concept of the Frenet frame to spacetime, which has conveniently already been discussed in the definitions of Chapter 3 — please compare the space and spacetime definitions of Frenet frame $\mathcal{F}$ and corridor $\mathcal{C}$ to better understand this leap of logic. Every other step of our apex optimization framework translates well to spacetime in line 15 of Algorithm 9, aside from two issues that have conveniently been solved by previous innovations:

- When we modify the path every iteration, we also modify our speed profile along that path, and thus an apex that made sense in the previous iteration may be overly conservative in the current iteration. For example, an attempted overtake may turn too sharply in one iteration, and need correcting in the next iteration because the act of turning naturally slows ego down. However, this is already solved by the deflation mechanic described in the previous chapter.
- With dynamic obstacles in particular, inflation and deflation are more prone to infinite loops; however, we already described a method of decaying maximum displacement in the previous chapter.

6.3 The Yield Maneuver

Recall that we defined the yield maneuver in Chapter 3 as a maneuver where ego maintains its nominal path but slows down in order to avoid collision. Our goal is to modify the speed profiler to yield for any obstacles o such that $\mathcal{H}(o) = \{ \text{yield} \}$.

Thankfully, this is fairly easy because of our formulation in Chapter 4. On line 12 of Algorithm 9, we see that any value of $\eta^*_{\max} < \eta_{\max}$ corresponds to a trajectory that, compared to the trajectory associated with $\eta_{\max}$, is both slower and less extreme at braking points, since we set $\eta^*_{\min} = -|\eta_{\max}|$ in the nominal case. In other words, the variable $\eta^*_{\max} \leq \eta_{\max}$ corresponds to a family of speed profiles that get progressively slower and less aggressive. The yield maneuver thus boils down to a binary search over the maximum $\eta^*_{\max} \leq \eta_{\max}$ satisfying $\delta(\tau, \mathcal{O}_{\text{yield}}) \geq \delta^*$, as we see on line 14.

Chapter 7

Extending to Multimodal Optimization

Finally, we tackle the central motivation for developing this non-numerical optimization technique — tractable multimodal trajectory optimization. In this chapter, we will borrow techniques from the field of multimodal optimization in order to efficiently search through many homology classes in a best-first fashion. By this point, we have eliminated all simplifying assumptions; we aim to solve the full extent of the problem formulated in Chapter 3. This means that for any obstacle o, $\mathcal{H}(o)$ may contain multiple candidate homology classes, and it is up to the algorithm to determine which homology classes are worth optimizing under limited computation.

Naturally, our solution will resemble a parallelizable branch-and-bound, resulting in a search tree where each node represents a set of homology constraints $\mathcal{H}$, and each node's children partition $\mathcal{H}$ into disjoint possibilities. In order to design this branch-and-bound, we need to answer the following questions:

- When do we choose to expand a node by spawning children? And what homology constraints should those children be subject to?

- How do we select the next node to optimize? In other words, what *acquisition function* do we use to select the next node?

Over this course of this chapter, we will work up to the full pseudocode detailed in Algorithm 10, which is the final iteration in this thesis.

7.1 Node Expansion

In order to understand what it means to expand a node in the context of S*, we should first discuss how our current working version of S* deals with cases where for some obstacle o, $|\mathcal{H}(o)| > 1$. If we plan a trajectory τ and it maintains a sufficiently high safety distance from o, then our algorithm works as normal by ignoring o. However, if τ is unsafe because of o, we now cannot proceed with our usual apex optimization and speed optimization because we have not yet decided how ego should respond to o. Should ego nudge left? Nudge right? Yield? The best we can do for now is pretend o does not exist and proceed with optimization. This is our motivation for node expansion — if there are multiple candidate homology classes to consider, we should create child nodes for each one and offload the task of optimizing those children to some priority queue capable of discerning the most promising node to optimize first.

However, it does not suffice to spawn children for every obstacle o that the current trajectory τ violates, for every candidate homology class contained in $\mathcal{H}(o)$. For example, if the very first trajectory planned — the trajectory interpolating no apexes — collides with a series of 20 obstacles, each of which ego can nudge left, nudge right, or yield for, then a naive algorithm would spawn 3^{20} children to search over. No acquisition function would be able to accurately pinpoint the most promising node among that many candidates.

Instead, we set a few ground rules:

- Each child $\mathcal{H}'$ may make at most one decision, i.e. for at most one obstacle o may it be true that $|\mathcal{H}(o)| > 1$ and $|\mathcal{H}'(o)| = 1$.
- Given two potential decisions, e.g. "nudge left for obstacle 1" and "nudge left for obstacle 2", if making the first decision could remove the need to make the second decision, then we should prioritize making the first decision, and only consider the second decision if we prohibit the first decision, i.e. enforce "cannot nudge left for obstacle 1". In particular, if the displacement required to nudge left for obstacle 1 is greater than the displacement required to nudge left for obstacle 2, we claim that nudging left for obstacle 1 may remove the need to

explicitly nudge left for obstacle 2, while the opposite is not true.

- If τ violates an obstacle o for which $|\mathcal{H}(o)| > 1$, but the degree of violation is dwarfed by the violation of other obstacles with known homology classes, then it is not yet the right time to make a decision on the homology class with respect to o.

We realize these constraints in lines 26-43 of Algorithm 10. Note that if $\mathcal{D}$ is empty, or equivalently if we do not spawn any children, then we have the option on lines 45-46 to submit the current node to the priority queue for re-optimization, if it requires further improvement.

7.2 Node Acquisition

In order to select the next node to optimize, the simplest acquisition function is simply the traversal time of the parent trajectory $T(\tau)$ — the lower the traversal time, the more likely a child node is to lead to the globally optimal trajectory. Ideally, we would design our acquisition function to more accurately estimate the potential of a node based on other factors like its displacement from its parent or lower bounds on penalties for safety distance and friction. However, our simple solution performs well enough.

There is one exception to this rule, which is that we always select child nodes that yield for a new obstacle over child nodes that nudge for a new obstacle. This choice is for two reasons: because it requires less computation, and because a good multimodal trajectory optimizer should reliably output emergency stop maneuvers even in the most extreme of circumstances.

7.3 The Full S* Algorithm

Finally, we attempt a line-by-line breakdown of Algorithm 10, summarizing the innovations across the last four chapters.

Algorithm 10 The S* Algorithm (multimodal, dynamic environments)

```
1: $\mathcal{T} \leftarrow$ empty trajectory library
2: $\mathcal{Q} \leftarrow$ empty priority queue
3: $\alpha_0,\ \beta_0 \leftarrow$ empty apex sequences
4: push $(\mathcal{H},\ \alpha_0,\ \beta_0,\ \eta_{\max})$ to $\mathcal{Q}$ with priority $\infty$
5: repeat (in parallel)
6:     $(\mathcal{H},\ \alpha,\ \beta,\ \eta^*_{\max}) \leftarrow$ pop top priority from $\mathcal{Q}$
7:     $\mathcal{O}_{\text{left}} \leftarrow \{\ o \in \mathcal{O}\ \mid\ \mathcal{H}(o) = \{\ \text{left}\ \}\ \}$
8:     $\mathcal{O}_{\text{right}} \leftarrow \{\ o \in \mathcal{O}\ \mid\ \mathcal{H}(o) = \{\ \text{right}\ \}\ \}$
9:     $\mathcal{O}_{\text{yield}} \leftarrow \{\ o \in \mathcal{O}\ \mid\ \mathcal{H}(o) = \{\ \text{yield}\ \}\ \}$
10:    $\mathcal{O}_{\text{undecided}} \leftarrow \{\ o \in \mathcal{O}\ \mid\ \{\ \text{yield}\ \} \subsetneq \mathcal{H}(o) \subseteq \{\ \text{left, yield, right}\ \}\ \}$
11:    repeat
12:        $\pi \leftarrow$ Spiro spline path from $\tilde{q}_i$ to $\tilde{q}_f$ via $\alpha$
13:        $\alpha \leftarrow \alpha\ \cup\ \{\ (\tilde{p},\ h) \in \beta\ \mid\ h(\pi,\ \tilde{p}) \neq h\ \}$
14:    until $\Delta\alpha = 0$
15:    repeat
16:        repeat
17:            $\tau \leftarrow$ time-optimal trajectory along $\pi$ from $v_i\ \mid\ \mathcal{A},\ \eta^*_{\max}$
18:            $\delta^* \leftarrow \delta_{\text{safe}} \times \min\left\{\ 1,\ \frac{\mu_{\max}-\mu(\tau)}{\mu_{\max}-\mu_{\text{safe}}}\ \right\}$
19:        until maximized $\eta^*_{\max} \leq \eta_{\max}$ s.t. $\delta(\tau,\ \mathcal{O}_{\text{yield}})\ \geq\ \delta^*$
20:        $\mathcal{T}(\mathcal{H}) \leftarrow \tau$
21:        // continued in part 2
```

```
22:         // continued from part 1
23:         $\mathcal{C} \leftarrow$ free corridor between $\mathcal{O}_{\text{left}}$ and $\mathcal{O}_{\text{right}}$ wrt $\mathcal{F}(\tau)$ $\mid$ $\delta^*$
24:         $\beta \leftarrow$ deflate $\alpha$ $\mid$ $\mathcal{C}$
25:         $\alpha \leftarrow$ inflate $\tau$ $\mid$ $\mathcal{C}$
26:         $\mathcal{D} \leftarrow$ empty set of new homology decisions
27:         **for** $o \in \mathcal{O}_{\text{undecided}}$ $\mid$ $\delta(\tau, o) < \delta^* - \max\{\Delta\alpha, \Delta\beta\}$ **do**
28:             **if** left $\in \mathcal{H}(o)$ **then**
29:                 $\mathcal{C}'_{\text{left}} \leftarrow$ free corridor between $\mathcal{O}_{\text{left}} \cup \{o\}$ and $\mathcal{O}_{\text{right}}$ wrt $\mathcal{F}(\tau)$ $\mid$ $\delta^*$
30:                 $\alpha'_{\text{left}} \leftarrow$ inflate $\tau$ $\mid$ $\mathcal{C}'_{\text{left}}$
31:                 push $(o, \text{left}, \alpha'_{\text{left}})$ to $\mathcal{D}$
32:             **if** right $\in \mathcal{H}(o)$ **then**
33:                 $\mathcal{C}'_{\text{right}} \leftarrow$ free corridor between $\mathcal{O}_{\text{left}}$ and $\mathcal{O}_{\text{right}} \cup \{o\}$ wrt $\mathcal{F}(\tau)$ $\mid$ $\delta^*$
34:                 $\alpha'_{\text{right}} \leftarrow$ inflate $\tau$ $\mid$ $\mathcal{C}'_{\text{right}}$
35:                 push $(o, \text{right}, \alpha'_{\text{right}})$ to $\mathcal{D}$
36:         **for** $(o, h, \alpha') \in \mathcal{D}$ in descending order of $\Delta\alpha'$ **do**
37:             $\mathcal{H}' \leftarrow \mathcal{H}$ but $\mathcal{H}'(o) = \{h\}$
38:             push $(\mathcal{H}', \alpha', \beta, \eta^*_{\max})$ to $\mathcal{Q}$ with priority $-T(\tau)$
39:             $\mathcal{H}(o) \leftarrow \mathcal{H}(o) \setminus \{h\}$
40:             **if** $\mathcal{H}(o) = \{\text{yield}\}$ **then**
41:                 $\mathcal{O}_{\text{undecided}} \leftarrow \mathcal{O}_{\text{undecided}} \setminus \{o\}$
42:                 $\mathcal{O}_{\text{yield}} \leftarrow \mathcal{O}_{\text{yield}} \cup \{o\}$
43:                 **break**
44:     **until** $\mathcal{D}$ is empty
45:     **if** $\max\{\Delta\alpha, \Delta\beta\} \gg 0$ **then**
46:         push $(\mathcal{H}, \alpha, \beta, \eta^*_{\max})$ to $\mathcal{Q}$ with priority $-T(\tau)$
47: **until** all workers are idle **or** search is aborted
48: **return** $\mathcal{T}$
```

- (line 1) We start with an empty trajectory library $\mathcal{T}$, where $\mathcal{T}(\mathcal{H})$ stores at most one trajectory τ corresponding to homology constraints $\mathcal{H}$.

- (line 2) We start with an empty priority queue $\mathcal{Q}$, which we will periodically push tasks to of the form ($\mathcal{H}$, α, β, $\eta^*_{\max}$), for homology constraints $\mathcal{H}$, inflated apex sequence α, deflated apex sequence β, and maximum effort $\eta^*_{\max}$.

- (line 3) We initialize our apex sequence to the empty sequence; this means our first path will interpolate no apexes, and will simply be the Spiro spline connecting $\tilde{q}_i$ to $\tilde{q}_f$.

- (line 4) We push our first task to $\mathcal{Q}$.

- (lines 5-6) We generalize our branch-and-bound as a parallelizable process where workers continuously pop tasks from $\mathcal{Q}$, in order of priority.

- (lines 7-10) For a given $\mathcal{H}$, we can partition $\mathcal{O}$ into $\mathcal{O}_{\text{left}} \cup \mathcal{O}_{\text{right}} \cup \mathcal{O}_{\text{yield}} \cup \mathcal{O}_{\text{undecided}}$.

- (lines 11-14) We first compute the Spiro spline path π that starts from $\tilde{q}_i$, ends at $\tilde{q}_f$, and interpolates α. Then, we recompute π as long as there exist any apexes in β that are violated by π and thus should be inserted into α.

- (lines 16-19) For a given maximum effort $\eta^*_{\max}$, we compute trajectory τ as the time-optimal trajectory along π from initial speed v_i satisfying acceleration model $\mathcal{A}$ and maximum effort $\eta^*_{\max}$. This loop repeats this process until $\eta^*_{\max}$ is maximized under the constraint that $\delta(\tau, \mathcal{O}_{\text{yield}}) \geq \delta^*$, i.e. ego is not dangerously close to any obstacle it should be yielding to. We compute δ^*, our desired safety distance, as a function of the maximum friction coefficient $\mu(\tau)$ exerted during τ, in order to strike a tradeoff between safety distance and friction.

- (line 20) We save our trajectory τ to the library $\mathcal{T}$.

- (line 23) In order to update our apex sequences, we first compute the free corridor $\mathcal{C}$ between $\mathcal{O}_{\text{left}}$ and $\mathcal{O}_{\text{right}}$ with respect to the $\mathcal{F}(\tau)$, the "Frenet frame" of τ, and subject to our desired safety distance δ^*. This corridor may be degenerate, in which case we will need to widen it in order to loosen our safety distance constraints.

- (line 24) We deflate our current apex sequence to obtain deflated apex sequence β, which we will use next iteration in order to prevent our next path from undoing progress made in this iteration.

- (line 25) We inflate our trajectory to obtain inflated apex sequence α. Keep in mind that α is only based on obstacles in $\mathcal{O}_{\text{left}}$ and $\mathcal{O}_{\text{right}}$, and thus completely ignores obstacles in $\mathcal{O}_{\text{undecided}}$.

- (line 26) We will process obstacles in $\mathcal{O}_{\text{undecided}}$ by constructing a set of new homology decisions $\mathcal{D}$, where each element $(o,\, h,\, \alpha')$ consists of an obstacle o, a homology $h \in \{\text{ left, right }\}$, and a new apex sequence $\alpha' \neq \alpha$ that incorporates the homology decision $(o,\, h)$.

- (line 27) We only process an obstacle $o \in \mathcal{O}_{\text{undecided}}$ if its violation by τ is extreme enough to exceed the violation by τ of $\mathcal{O}_{\text{left}}$ and $\mathcal{O}_{\text{right}}$. This is to delay node expansion until absolutely necessary.

- (lines 28-35) We populate $\mathcal{D}$ by considering each undecided obstacle o and recomputing our free corridor $\mathcal{C}'$ and inflated apex sequence α' after considering whether to nudge left or right for o.

- (line 36) We process our homology decisions in descending order of displacement, since if decision 1 requires a higher displacement than decision 2, then decision 2 can never remove the need to consider decision 1, but decision 1 may remove the need to consider decision 2.

- (lines 37-38) When we process a homology decision, we spawn a new task for it that incorporates the new homology constraints and the new inflated apex sequence. We assign a priority to this task based on how low our traversal time $T(\tau)$ is.

- (lines 39-43) If we choose not to pursue some homology decision, then we must prohibit making that choice in the future before moving on to the next candidate decision. If it turns out that by prohibiting that choice, the only homology class left for some obstacle is the yield maneuver, then we make the choice to yield. Instead of spawning a new task for this, we instead jump to line 15 to start anew, reusing our path π since it does not change due to a yield maneuver.

- (lines 44-46) If we have no new homology decisions to consider, we may consider pushing a new task to $\mathcal{Q}$ that corresponds to re-optimizing our current node. We should only do this if the current node needs significant changes.

- (lines 47-48) Finally, when all modes have been explored or the search is aborted, we return our trajectory library $\mathcal{T}$, enabling a downstream process to consume the globally optimal trajectory or decide between any of the stored trajectories at its own discretion.

Chapter 8

Results

In this chapter, we will present results that demonstrate the S* algorihm in action. Unfortunately, we were unable to collect results for the most recent version of S*, which is capable of robustly navigating complex dynamic environments, within the time frame of the thesis. Thus, the following results reflect two different preliminary versions of S*, which we will name V1 and V2.

V1 is characterized by the following simplifications:

- V1 only nudges for static obstacles, i.e. ego may only yield to dynamic obstacles.
- V1 achieves the yield maneuver through a jerky and fallible method that has since been discontinued.
- V1 used the cubic spline method described in Chapter 4 instead of Spiro splines, resulting in occasional steering jerks (which are thankfully corrected somewhat by downstream controls) and suboptimal racing.
- V1 fails whenever it is infeasible to maintain safety distance δ_{safe} because two obstacles are not sufficiently far apart.
- V1 will not attempt to make a tradeoff between safety distance and friction, instead attempting to enforce safety distance δ_{safe} at any cost.
- V1 will occasionally generate an overconstrained trajectory with an unnecessary apex, like in Figure 8-4.

- V1 is poorly optimized and does not handle parallel computation.

V1 also has a feature that is not within the scope of the thesis due to time constraints — namely, in addition to nudging for static obstacles and yielding for dynamic obstacles, V1 is capable of nudging to hit virtual targets spawned by the Roborace Metaverse. We will see results that showcase this, but the method behind it will remain a topic for the future.

V2 is characterized by the following simplifications:

- V2 fails whenever it is infeasible to maintain safety distance δ_{safe} because two obstacles are not sufficiently far apart.
- V2 will not attempt to make a tradeoff between safety distance and friction, instead attempting to enforce safety distance δ_{safe} at any cost.
- While V2 can typically navigate simple dynamic environments, it fails in more complex environments, due to various convergence issues solved by post-V2 insights sprinkled throughout this thesis.
- V2 only handles local optimization, as the multimodal optimization in V1 did not translate well to V2 and we opted to solve a smaller problem first.
- V2 is poorly optimized.

However, despite these limitations, we have observed very promising results, both as a standalone multimodal trajectory optimizer and integrated into an autonomous vehicle stack running in simulation and on a full-size Roborace Devbot 2.0 racecar navigating multiple mixed-reality dynamically-perceived obstacle courses at up to 100 mph.

8.1 V1 in Randomly Generated Scenes

V1 was thoroughly tested for static scenes of obstacles and targets, including shapes like circles, rectangles, rounded rectangles, and even track boundaries. In over 50

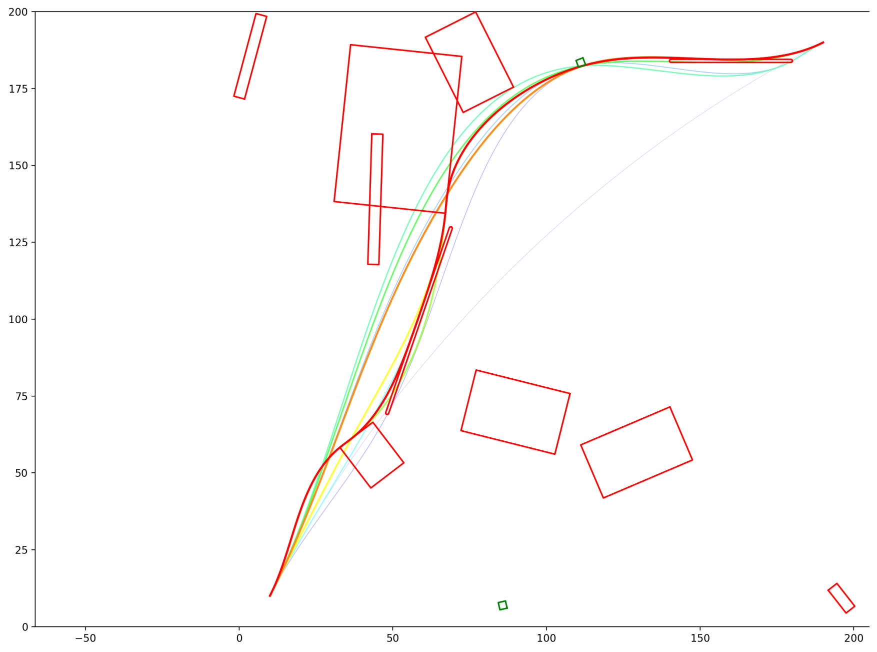

Figure 8-1: Sparse obstacle course with narrow corridors. Warmer colors are lower cost. Obstacles are red and targets are green. All trajectories generated during the optimization are displayed, not just the ones in the trajectory library.

randomly generated scenes (including sparse scenes, dense scenes, scenes with different shapes, and scenes with varying numbers of objects), with a fixed initial configuration, flat road, no aerodynamics, and traversal time as the singular objective function, we produce visually verifiable results, with the only notable error being shown in Figure 8-4, where an unnecessary apex near (175, 150) overconstrains the reported globally optimal trajectory. Even for dense scenes like Figure 8-3, which require more computation because they have more reasonable modes to explore, V1 averages 500 milliseconds of serial computation for 10 objects, and 60 seconds of serial computation for 22 objects.

8.2 V1 in Mixed-Reality Obstacle Courses

We also demonstrate V1 integrated into MIT Driverless's autonomous racing stack racing a full-size Roborace Devbot 2.0 racecar on mixed-reality obstacle courses at up to 100 mph as part of Roborace Season Beta Round 7 and Round 10. Both rounds

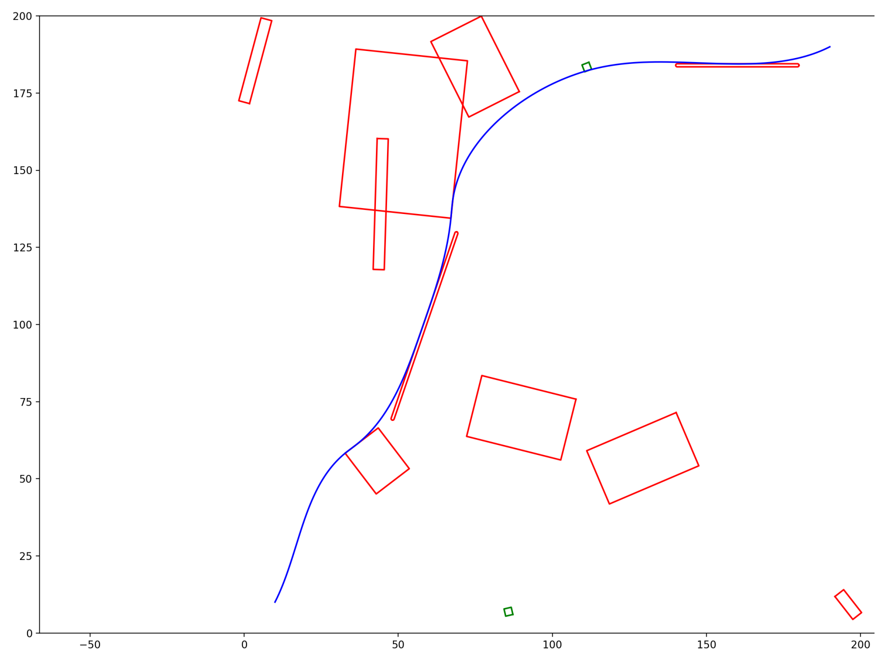

Figure 8-2: Globally optimal trajectory for Figure 8-1.

were held at the Las Vegas Motor Speedway's Outside Road Course. Virtual objects were revealed to ego by a V2X system 150 m away, and obstacles penalized 30 seconds, while targets rewarded 2 seconds. In Round 7, ego yielded to all dynamic obstacles, while in Round 10, ego ignored all dynamic obstacles. We finished in third place for both events, out of six teams. In Round 7, we hit the second-most number of targets and the only obstacle we hit was a dynamic obstacle owing to a failure in our yielding implementation. In Round 10, we again hit the second-most number of targets, and still finished with the fastest raw elapsed time and peak speed.

We highlight a couple of implementation details needed to integrate V1 into our autonomy stack:

- At every time step, our terminal configuration $\tilde{q}_f$ is chosen from a lookahead point on the global racing line, a precomputed cyclic path traversing the entire track. In addition, we ignore all obstacles that are closer to $\tilde{q}_f$ than to $\tilde{q}_i$, which is important because otherwise, our choice of $\tilde{q}_f$ may actually be infeasible.

- At every time step, our initial configuration $\tilde{q}_i$ is not set to the vehicle's measured configuration, but rather to a projected configuration based on the previously

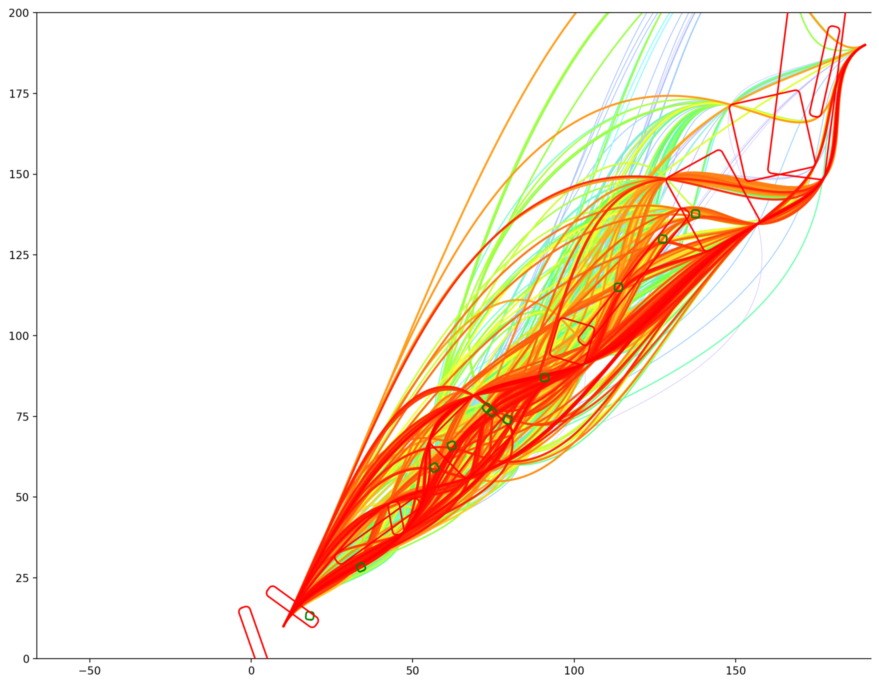

Figure 8-3: Dense obstacle course.

planned trajectory. This is to ensure stability when paired with a downstream controller.

- Since we do not make a tradeoff between safety distance and friction, V1 is capable of outputting dangerously extreme trajectories in order to correct minor safety distance hazards. We mitigate this by simulating a "blind spot" in the vicinity of ego, i.e. a circle around ego where ego does not see any obstacles.

- In order to forecast the motion of other virtual agents, we run V1 on each agent but without any obstacles besides the track boundaries, since agents are known to ignore virtual obstacles and targets. We then use these forecasted trajectories when planning ego's trajectories, in what amounts to a Stackelberg game.

8.3 V2 in Simple Dynamic Scenes

Finally, we demonstrate a simple dynamic environment for which V2 does not fail in Figure 8-6. In this scenario, we are testing the "right undertake" maneuver, which

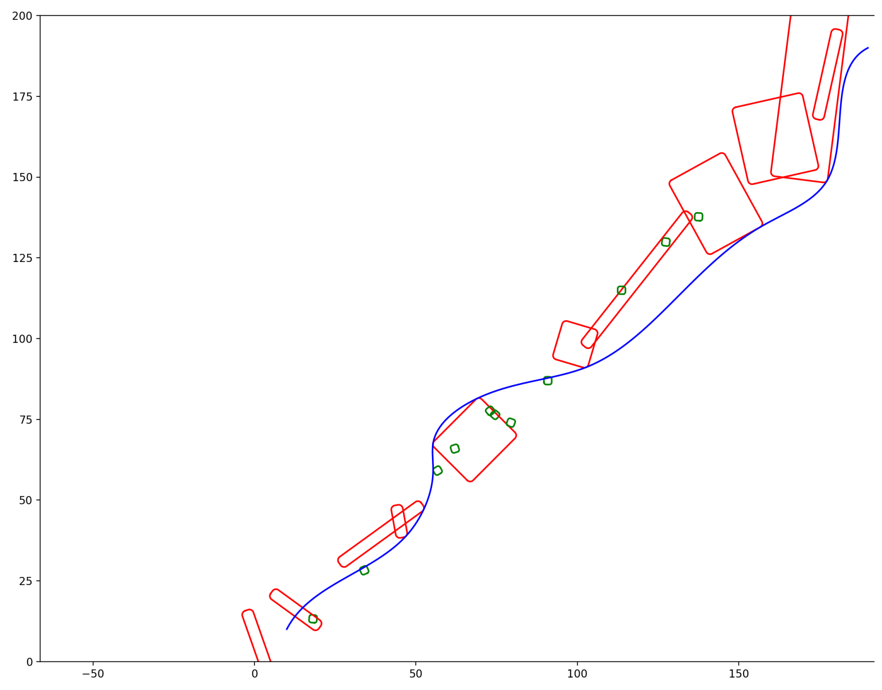

Figure 8-4: Globally optimal trajectory for Figure 8-3 as computed by V1. Unfortunately, V1 overconstrains this trajectory by including an unnecessary apex near (175, 150).

we describe as "nudging to the right to prepare for an overtake, but staying behind our opponent for the time being"; this maneuver was actually motivated by the Indy Autonomous Challenge race at the Las Vegas Motor Speedway, where one of the rules for 1v1 racing was that for the first few laps, competitors were only allowed to overtake during a specified passing zone in view of the spectators. Unfortunately, this amount of complexity approaches the limit of what V2 could solve without failing to converge, so this is our only result for V2.

Figure 8-5: MIT Driverless's run, streamed on Twitch during Roborace Season Beta Round 7.

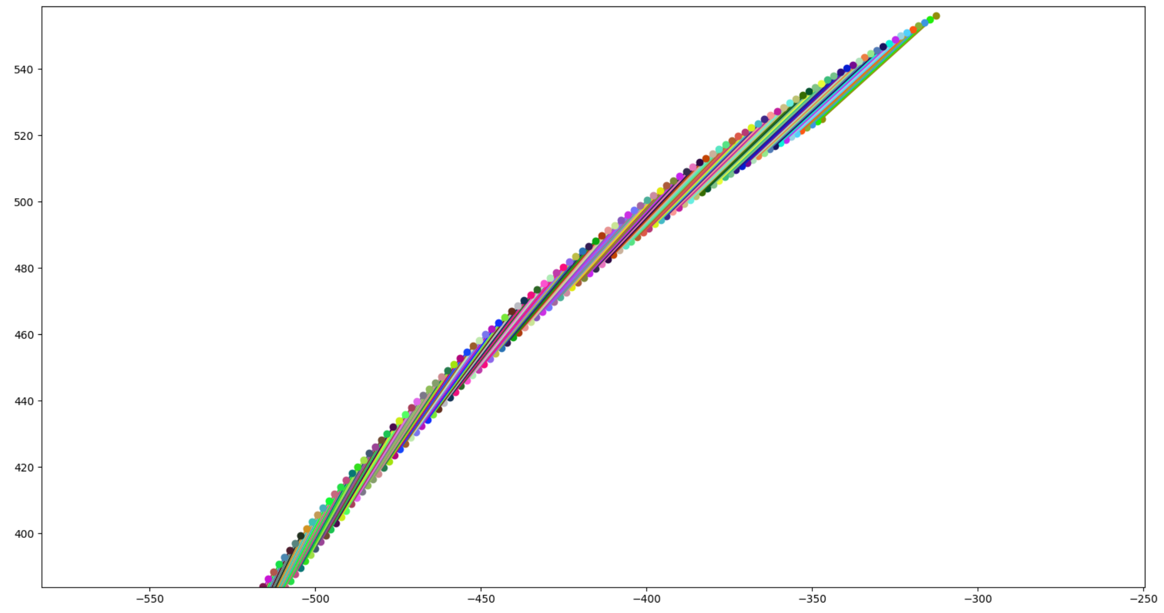

Figure 8-6: A "right undertake" maneuver. Both ego and its opponent start in the upper right corner, and ego nudges to the right and closes the gap in order to prepare to overtake its opponent while staying behind the opponent. The color coding indicates the synchronized positions of ego and its opponent with respect to time.

Chapter 9

Future Work

In this chapter, we touch on future work for S*.

9.1 Recent Results

By far the most pressing future work after this thesis is to collect results of the final version of S* navigating complex dynamic environments, including randomly generated scenes, simulated environments, and real racing scenarios from the Indy Autonomous Challenge and Roborace. Without these results, it is hard to say with certainty that the innovations in this thesis are correct and robust. When these results are collected, we also hope to release the source code.

9.2 Rigorous Evaluation

Next, we hope to be more rigorous in our evaluation of the algorithm. In particular, aside from visually verifiable results, we would like to collect quantitative results showcasing its anytime qualities and benchmark it extensively against other methods. We also hope to analyze data from any races that deploy S* in the future.

9.3 Extending to Targets

While arguably not very important due to a lack of motivating use cases in industry, we look forward to one day re-exploring the challenge of navigating toward targets, instead of simply navigating around obstacles. V1 was already capable of this, and ideally the final implementation will be capable of this too.

9.4 Multimodal Motion Forecasting

We believe S^* will be able to solve multimodal motion forecasting. We have already used S^* V1 to solve unimodal motion forecasting for dynamic agents with known behaviors (e.g. follow the racing line) in convex corridors such as empty racetracks; it suffices to learn the acceleration model of the agent and then solve normally. However, we hope to eventually infer likely behaviors and scene awareness directly within the multimodal trajectory optimization. While trajectories are scored by cost for the motion planning problem, they would be scored here by probability.

9.5 Joint Motion Forecasting and Planning

Furthermore, we eventually hope to model interactions between ego and non-ego agents as Nash games instead of Stackelberg games, which is what we have currently implemented. By conditioning predictions on ego plans, ego can confidently overtake an agent before a turn while expecting that the agent will not drive ego off the track. The hope is that the computational efficiency of the S^* algorithm will enable this type of joint motion prediction and planning to be more tractable as ever, even on a latency-critical system like an autonomous racecar.

Bibliography

[1] A. Brouillard and Paradigm Shift Driver Development. *The Perfect Corner: A Driver's Step-by-Step Guide to Finding Their Own Optimal Line Through the Physics of Racing*. The Science of Speed Series. Paradigm Shift Motorsport Books, 2016.

[2] John Connors and Gabriel Elkaim. Analysis of a spline based, obstacle avoiding path planning algorithm. In *2007 IEEE 65th Vehicular Technology Conference - VTC2007-Spring*, pages 2565–2569, 2007.

[3] Alexander Heilmeier, Alexander Wischnewski, Leonhard Hermansdorfer, Johannes Betz, Markus Lienkamp, and Boris Lohmann. Minimum curvature trajectory planning and control for an autonomous race car. *Vehicle System Dynamics*, 58(10):1497–1527, 2020.

[4] Markus Kuderer, Christoph Sprunk, Henrik Kretzschmar, and Wolfram Burgard. Online generation of homotopically distinct navigation paths. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6462–6467, 2014.

[5] Raphael Linus Levien. *From Spiral to Spline: Optimal Techniques in Interactive Curve Design*. PhD thesis, EECS Department, University of California, Berkeley, Dec 2009.

[6] Christoph Roesmann, Wendelin Feiten, Thomas Woesch, Frank Hoffmann, and Torsten Bertram. Trajectory modification considering dynamic constraints of autonomous robots. In *ROBOTIK 2012; 7th German Conference on Robotics*, pages 1–6, 2012.

[7] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. Planning of multiple robot trajectories in distinctive topologies. In *2015 European Conference on Mobile Robots (ECMR)*, pages 1–6, 2015.