2023

Cooperative Driving Automation for Self-Interested Agents

Adam Morrissett
Virginia Commonwealth University

VCU College of Engineering
Electrical and Computer Engineering

Cooperative Driving Automation for Self-Interested Agents

Virginia Commonwealth University
College of Engineering
Department of Electrical and Computer Engineering
601 West Main Street
Richmond, Virginia 23284–3068

Cooperative Driving Automation for Self-Interested Agents

A dissertation submitted in partial fulfillment of the requirement for the degree of Doctor of Philosophy at Virginia Commonwealth University.

by

Adam Morrissett
Bachelor of Science in Computer Engineering, Virginia Commonwealth University, 2018

Director: Patrick J. Martin, Ph.D.
Assistant Professor, Department of Electrical and Computer Engineering

Virginia Commonwealth University
Richmond, Virginia
May, 2023

Acknowledgement

I thank my advisor, Dr. Patrick J. Martin, for his unending support and encouragement while I completed my work. His ability to juggle proposal writing, project management, and personal research while simultaneously interacting directly with all of his graduate and undergraduate students is a feat in and of itself. I'm still unsure how he accomplishes it. Despite his many responsibilities, Dr. Martin was always available to discuss research directions and ideas, resolve mental blocks, and revise papers. Like most graduate students, I contemplated cutting my Ph.D. journey short numerous times. His involvement in my work was critical in helping me reach the finish line.

I thank my committee members (ordered alphabetically): Dr. Sherif Abdelwahed, Dr. Nathaniel "Nate" Kinsey, Dr. Miloš Manić, and Dr. Ruixin Niu for serving on my committee. Their high expectations pushed me to reach my potential, and they each impacted me beyond my dissertation work. Dr. Abdelwahed's course on cyber-physical systems exposed the mathematical underpinnings of state machines, system composition, and interface specifications that I employ daily. His material emphasized the importance of creating well-defined interfaces between other systems and human users. Dr. Kinsey co-advised my undergraduate capstone project along with Dr. Ümit Özgür. Their guidance and notoriously high standards helped establish my foundational research skills and transformed my mindset to that of a scientist. In his neural networks and deep learning course, Dr. Manić bridged the gap between machine learning theory and applications. His final open-ended research project offered excellent academic writing practice and directly contributed to one of my publications. Dr. Niu's estimation and optimal filtering course was the most challenging class I had throughout my graduate program but also the most rewarding. His patience and thorough explanations helped me grasp complex but fundamental concepts in the robotics field.

I also thank Dr. Pavle Bujanović for offering me an internship with the U.S. Department of Transportation, Federal Highway Administration, my first break into the autonomous vehicle industry. He encouraged me to apply for the Dwight David Eisenhower Transportation Fellowship and served as an advisor after I received it.

I thank my parents, for without their support and encouragement—and nagging—my academic career would not have been so successful. I thank my partner Tamara Peña for her loving companionship and for sticking with me through my ups and downs as a graduate student. Finally, I thank my friends Dr. Steven M. Hernandez, Dr. Andrew "Andy" Fabian, and Lauren Linkous for being a solid social core.

This work was supported in part by the U.S. Department of Transportation, Federal Highway Administration under Agreement No. 693JJ32245201. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the Author(s) and do not necessarily reflect the view of the Federal Highway Administration.

This work was supported in part by the Commonwealth Cyber Initiative (CCI), an investment in advancing cyber R&D, innovation, and workforce development in Virginia. For more information about CCI, visit `www.cyberinitiative.org`.

Contents

List of Figures

List of Tables

List of Algorithms

Abstract

COOPERATIVE DRIVING AUTOMATION FOR SELF-INTERESTED AGENTS

By Adam Morrissett

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 2023.

Major Director: Patrick J. Martin, Ph.D.,
Assistant Professor, Department of Electrical and Computer Engineering

Driving is inherently interactive, and drivers must coordinate with other vehicles, cyclists, and pedestrians to avoid collisions. Furthermore, drivers typically prefer some interaction outcomes over others. People in a rush typically cut others off and drive aggressively, while those on a leisurely outing tend to move slowly and behave more altruistically. Even as passengers in autonomous vehicles, people may exhibit these preferences. Unfortunately, autonomous vehicles do not share these preferences, instead sharing only minimal information or operating in isolation. Existing coordination and management methods do not consider these preferences when making decisions. They focus specifically on minimizing the trip duration and maximizing throughput.

Cooperative driving automation is an emerging research field in which vehicles work together to achieve individual goals. Standards organizations such as SAE International have developed a taxonomy for various collaboration categories among connected autonomous vehicles (CAVs) with varying automation capabilities. The Federal Highway Administration is developing research platforms to facilitate cooperative driving algorithm development.

This research seeks to understand how sharing passenger preferences and other high-level information affects vehicles' coordination abilities and management performance. Specifically, we attempt to answer three main research questions. 1. Can CAVs achieve better interaction outcomes by sharing high-level information, such as their passengers' preferences? 2. Can CAVs solve challenging cooperative decision-making problems by sharing high-level

information, such as planned destinations? 3. How does optimizing for passenger preferences in interactive scenarios affect other traditional performance metrics?

We explore two application areas under the cooperative driving automation theme. The first scenario considers streams of autonomous vehicles simultaneously approaching an unmanaged intersection. Vehicle passengers have different preferences on how quickly they cross and in which order. Vehicles convey this information to an auction-based intersection management system installed at the intersection. The management system then assigns crossing durations and a crossing schedule to satisfy everyone's preferences as best as possible.

The second scenario investigates vehicles moving in a spatially-constrained environment such as an alleyway. In this work, we formulate a new type of finite multi-stage game we call a deadlock game. Additionally, we propose a solution method that solves general problem instances. This work provides the foundation for continued research into equilibria refinement and satisfying passengers' preferences on the outcomes.

Chapter 1

Introduction

Driving poses significant risks to humans. Of the 36,096 driving-related fatalities in 2019, an estimated 3,142 involved distracted drivers. Drowsy driving contributed to 697 fatalities, and 10,710 involved alcohol-impaired drivers [1]. In 2021, traffic congestion in large urban areas delayed commuters by an estimated 31,065,000 hours and wasted 3,868,000 gallons of gas, costing them \$6 784 000 000 [2].

Autonomous vehicle (AV) proponents claim that AVs would operate safer and more efficiently than humans, helping to reduce fatalities and congestion [3]. AVs—also known as *driving automation systems* or *Automated Driving Systems (ADS)* [4]—have faster reaction times, do not get drowsy, and do not get impaired, meaning safety thresholds (*e.g.*, following distances) can be relaxed.

While AVs can revolutionize driving, real-world experiences highlight several limitations. The Insurance Institute for Highway Safety (IIHS) conducted a case study on human-driven vehicle collisions [5]. They separated incident reports according to several driver-related factors: sensing and perception, prediction, planning and decision, execution and performance, and incapacitation. The study concluded that while AVs can help reduce incidents stemming from sensing-and-perception and incapacitation (accounting for 23 % and 10 % of incidents, respectively), they may struggle to reduce those caused by other factors.

Several high-profile crashes involving various AV manufacturers help support the IIHS's claims. The National Transportation Safety Board (NTSB) investigated several incidents (some fatal) involving Tesla vehicles operating under the Autopilot driving automation system. Reports describe the vehicles colliding with roadside infrastructure [6], semitrailers [7, 8], and emergency vehicles responding to calls [9]. In these incidents, drivers were over-reliant on Autopilot and responded inadequately to system failures.

Other vehicle manufacturers experienced similar incidents. Uber's Advanced Technologies Group was testing one of their AVs at night when it collided with a pedestrian crossing the street, killing her [10]. The investigation determined that the vehicle detected her presence but failed to classify and predict her movements, leading the system to make the wrong decision. A Cruise vehicle collided with a motorcyclist while trying to re-center itself after an aborted lane change [11]. The motorcyclist attempted to move into the gap previously left by the car as the car simultaneously maneuvered back into it.

Not all incidents involved collisions. An article from The Verge reported that a Waymo driverless taxi planned to turn right at an intersection, but construction cones blocked the destination lane [12]. The vehicle attempted to complete the turn, but the motion planning system could not determine a feasible route and stopped the car, blocking the entire road. A Waymo response team eventually removed the vehicle.

These incidents emphasize the performance gap between human drivers and AVs. Attentive human drivers would have prevented the Tesla and Uber collisions. Human drivers routinely navigate roads with lane closures, but the Waymo vehicle froze. Additionally, numerous disengagement reports with the California Department of Motor Vehicles describe times when safety drivers overrode the driving automation system because of unsafe decisions or failures [13]. Some vehicle companies have already removed safety drivers from their test vehicles (under restrictions) [14], but engineers need to close the performance gap before this becomes widespread.

Perception and decision-making (planning) issues are common themes in the above anec-

dotes. Driving automation systems fail to recognize objects that humans can easily discern, and in some cases, they falsely detect non-existent objects [15]. Unlike humans, AVs need complicated motion models to predict road users' behaviors. However, driving is an interactive task, and these models fail to capture influencing factors among different agents. Furthermore, human drivers commonly communicate directly through hand gestures or turn signals and indirectly through slow, elaborate movements. Current commercial AVs operate in isolation. They do not communicate with neighboring vehicles, which restricts them from cooperating or collaborating.

AV companies heavily invest in perception and prediction research but neglect an easily-accessible information source: inter-vehicle communication. Vehicular *ad hoc* networks (VANETs) and more general mobile *ad hoc* networks provide mesh networks for vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication [16]. Connected AVs (CAVs) can use these networks to exchange information with each other, supplementing the data they obtain from their onboard systems. This communication enables CAVs to create vehicle platoon [17] and coordinate their movements. These types of interactions are collectively referred to as cooperative driving automation (CDA).

Government institutions and standards organizations recognize CDA's potential and have been formalizing system architectures, data formats, and protocols to facilitate adoption and interoperability. Existing CDA research efforts generally share mundane information among vehicles, such as system states and planned trajectories. However, continued improvements to network technologies and increased capacity allow CAVs to share higher-level information, such as preferences over different possible CDA maneuvers and interaction outcomes.

AVs continually interact with other road users, but their differing preferences and self-interested nature creates conflicts in shared environments. They cannot all cross an intersection simultaneously, nor can two opposing vehicles traverse the same single-lane, two-way road together. Intersection management systems regulate crossing orders using first-come, first-served (FCFS) or fixed-policy methods, but AVs may prefer different schedules. Road-

ways use rights-of-way to establish priority among vehicles, but alleys and parking lots lack this structure, leaving drivers to figure out how to maneuver. Without explicit or implicit communication, drivers cannot agree on any course of action, much less one they both favor.

In this dissertation, we seek answers to the following research **questions**:

- Can CAVs achieve better interaction outcomes by sharing high-level information, such as their passengers' preferences?
- Can CAVs solve challenging cooperative decision-making problems by sharing high-level information, such as planned destinations?
- How does optimizing for passenger preferences in interactive scenarios affect other traditional performance metrics?

The work we present in this dissertation provides the following **contributions**:

- Path-constrained, time-assigned, control-minimal trajectory optimizer
- Socially-optimal auction-based intersection management system (IMS) where agents bid using cost functions instead of money
- Formalization of a highly-constrained interactive environment containing several agents, which we term a *deadlock game*
- Novel cooperative algorithm that solves arbitrary deadlock game instances under mild assumptions

This work explores two scenarios where vehicles' preferences can influence the chosen outcomes. In the first scenario, vehicles approach an intersection and express how quickly they want to cross and how long they are willing to wait before crossing. We propose an auction-based intersection management system that assigns a crossing duration to each vehicle and schedules the crossing based on vehicles' preferences.

The second scenario involves several vehicles moving around an extremely spatially-constrained environment with regions that can accommodate only one vehicle at a time. The vehicles need to coordinate their movements or they will reach a stalemate. Current solutions commonly employ game-theoretic receding horizon planners or a data-driven policy. However, these methods break down when vehicles mutually obstruct each others' paths.

We propose an algorithm that determines tactical movement plans to break the deadlock.

Contention is a common theme throughout this dissertation. All agents are self-interested with their own preferences on the scenario's outcome, which they express among themselves. Existing decision systems eschew preferences, choosing instead to optimize exclusively for metrics such as throughput. By exploiting these preferences, we seek solutions to CDA problems that are more favorable to the involved agents. The proposed methods leverage concepts from auction-theoretic intersection management and game-theoretic motion planning.

1.1 Autonomous Vehicle Technology

The AV industry is rife with misleading marketing terms, and the academic community uses overloaded and ambiguous ones. To help unify the two communities, SAE international and other standards organizations developed taxonomy and definitions documents to help convey the differences. This section provides an overview of those documents to provide context and scope for the rest of the dissertation.

1.1.1 Vehicle Autonomy Levels

Autonomous driving is complicated, and AV manufacturers automate different aspects of it to varying degrees. This variance hinders comparing and understanding each vehicle's capabilities (especially when referencing marking literature). To help remedy this, SAE International drafted a recommended practice document (J3016) that presents an automated driving taxonomy and describes and classifies the different levels of driving automation [4].

The document breaks down the notion of driving into six distinct tasks, collectively referred to as the *dynamic driving task (DDT)*:

- lateral motion control (*e.g.*, maintaining a following distance)
- longitudinal motion control (*e.g.*, staying within a lane)
- driving environment monitoring (*e.g.,* observing neighboring vehicles)[1]

- object and event response (*e.g.*, stopping for a pedestrian)[1]
- maneuver planning (*e.g.*, avoiding an obstacle)
- conspicuity enhancement (*e.g.*, turning on headlights at night)

AVs can automate each DDT to some extent. For example, Tesla's Autopilot system simultaneously performs lateral and longitudinal motion control while leaving the driver responsible for the rest of the subtasks.

Driving automation systems function within certain operating constraints, such as specific geographic locations or certain weather conditions. These constraints define the system's *operational design domain (ODD)*, which partially determines when a driving automation system can enable specific features. Waymo's driving automation system, Waymo Driver, performs the entire DDT, but only within a few geofenced areas.

The end goal for driving automation is to design a system that functions equivalent to or better than humans. However, they sometimes fail, requiring someone or something to take over control. The J3016 document defines two *DDT fallback* options for when the automation system stops functioning for any reason: human fallback and system fallback. In the former, a *fallback-ready user* assumes control and becomes the driver. With the latter, a safety system takes over and places the vehicle in a safe state (which ironically could mean stopping in the middle of the road [12]). The fallback method affects a driving automation system's automation level within the J3016 framework.

A driving automation system's automated DDT subtasks, ODD, and fallback method define its autonomy level. The J3016 document defines six levels, ranging from no automation to full automation. Table 1.1 summarizes the differences between them. Notice the jump in capabilities between levels 2 and 3; the driver performs OEDR in level 2, but the system performs it in level 3. Automated Driving Systems (ADSs) operate at level 3 and above and perform the entire DDT within an ODD.

Note that a *driver* is distinct from a *fallback-ready user*. By definition, a driver must

[1]Commonly combined and referred to collectively as *object and event detection and response (OEDR)*.

Table 1.1: SAE J3016 Driving Automation Levels[1]

Level	Lateral and Longitudinal Control	OEDR[2]	Fallback	ODD
0	Driver	Driver	N/A	N/A
1[3]	Driver & System	Driver	Driver	Limited
2	System	Driver	Driver	Limited
3	System	System	Fallback-Ready User	Limited
4	System	System	System	Limited
5	System	System	System	Unlimited

[1] Adapted from [4, Table 1].
[2] OEDR combines driving environment monitoring and object and event response.
[3] Level 1 vehicles control either lateral or longitudinal movement, but not both simultaneously.

perform OEDR; they must actively oversee the driving automation system. A fallback-ready user, however, need not actively monitor the driving environment. If the ADS must relinquish control of the vehicle (either through a failure or by leaving the ODD), the fallback-ready user will assume the role of driver and perform the DDT.

1.1.2 Cooperative Driving Classes

AVs are typically self-interested and act using only the information they obtain themselves. In some scenarios, however, cooperating with neighbors improves everyone's performance. Sharing perception data reduces blind spots, leading to more informed decisions [18]. Constructing vehicle platoons (road trains) reduces wind drag and thus reduces energy consumption [19].

Like automation levels, vehicles equip varying cooperative driving technologies. A vehicle's sensing, communication, and automation abilities affect its cooperative driving capabilities. SAE International developed a taxonomy to classify *cooperative driving automation (CDA)* based on the information exchanged among agents [20]. Note that agents include vehicles and roadside infrastructure. Within each class, vehicles' automation levels determine the extent to which they can cooperate. Table 1.2 summarizes the relationship between

Table 1.2: SAE J3216 CDA Classes and Cooperation Capabilities[1]

			Automation Level[2]	
Class	Nature	Description	1–2	3–5
A	Status Sharing	Current Status	Limited	Full
B	Intent Sharing	Future Plan	Limited	Full
C	Agreement Seeking	Work Together	None	Full
D	Prescriptive	Do as Directed	None	Full

[1] Adapted from [20, Table 1].
[2] Automation level 0 provides no CDA capabilities and is omitted.

CDA classes and automation levels. Each class defines a cooperation category, and a vehicle's autonomy level determines the extent to which it can cooperate within the category.

Driving automation systems operating at levels 1 and 2 delegate OEDR to the driver, meaning the system observes only a limited amount of the environment; it cannot perceive high-level indicators such as brake lights or turn signals. Limited perception hinders their capabilities to contribute to class A and B CDA. It completely prevents them from performing class C and D CDA. Meanwhile, ADSs perform the entire DDT, allowing them to fully contribute towards all CDA classes while operating within their ODDs.

Vehicles and infrastructure must communicate through a common interface to realize cooperative driving automation. However, standardization is troublesome because various automakers and infrastructure manufacturers might develop proprietary interfaces specific to their product lines. To promote widespread V2X adoption, SAE International published a standardized message set [21]. The messages enable agents to easily share maps, vehicle states, signal phase timing, and other information. The standard also describes how system designers can extend messages to provide supplemental information specific to certain vehicles or regions.

1.1.3 Reference Autonomy Architecture

AVs comprise several components responsible for specific functions: global positioning, object detection, throttle actuation, and more. Subsystems aggregate components based on their purpose, and these collections partition the system into logical layers. No standard exists yet that defines a canonical ADS architecture. SAE International is actively developing one [22]; however, many AV manufacturers [23, 24, 25] and open-source autonomy stacks [26, 27] architect their automation systems similar to the one described by Paden *et al.* in [28], recreated in Fig. 1.1. That work decomposes AVs into four main layers: route planning, behavior planning, motion planning, and feedback control. Other supporting subsystems not mentioned in their architecture provide perception, localization, prediction, and V2X capabilities.

Route planning involves generating a least-cost path in a map—represented as a weighted directed graph—starting from a given source node and terminating at a given destination node. Travel directions, speed limits, and other regulations usually determine the graph edges and corresponding weights.

Downstream from the route planner, the behavior planner determines an appropriate high-level task (*e.g.*, `stop`, `change_lanes`, and `turn_left`) for the vehicle to accomplish. This layer functions similarly to a finite state machine with transitions determined based on the perceived environment. Surrounding vehicles, road conditions, traffic signals, and other entities influence the selected behavior.

The motion planner translates the current behavior into the low-level path or trajectory needed to accomplish the task. Several methods exist to generate paths or trajectories, but the results must be feasible, comfortable, and safe.

As the lowest layer in the AV stack, the feedback controller generates actuator inputs to drive the vehicle. The controller tracks the reference trajectory as closely as possible and attempts to minimize modeling errors introduced by the higher layers. Designs for this layer focus on robustness and stability. Note that the name for this layer is somewhat of a

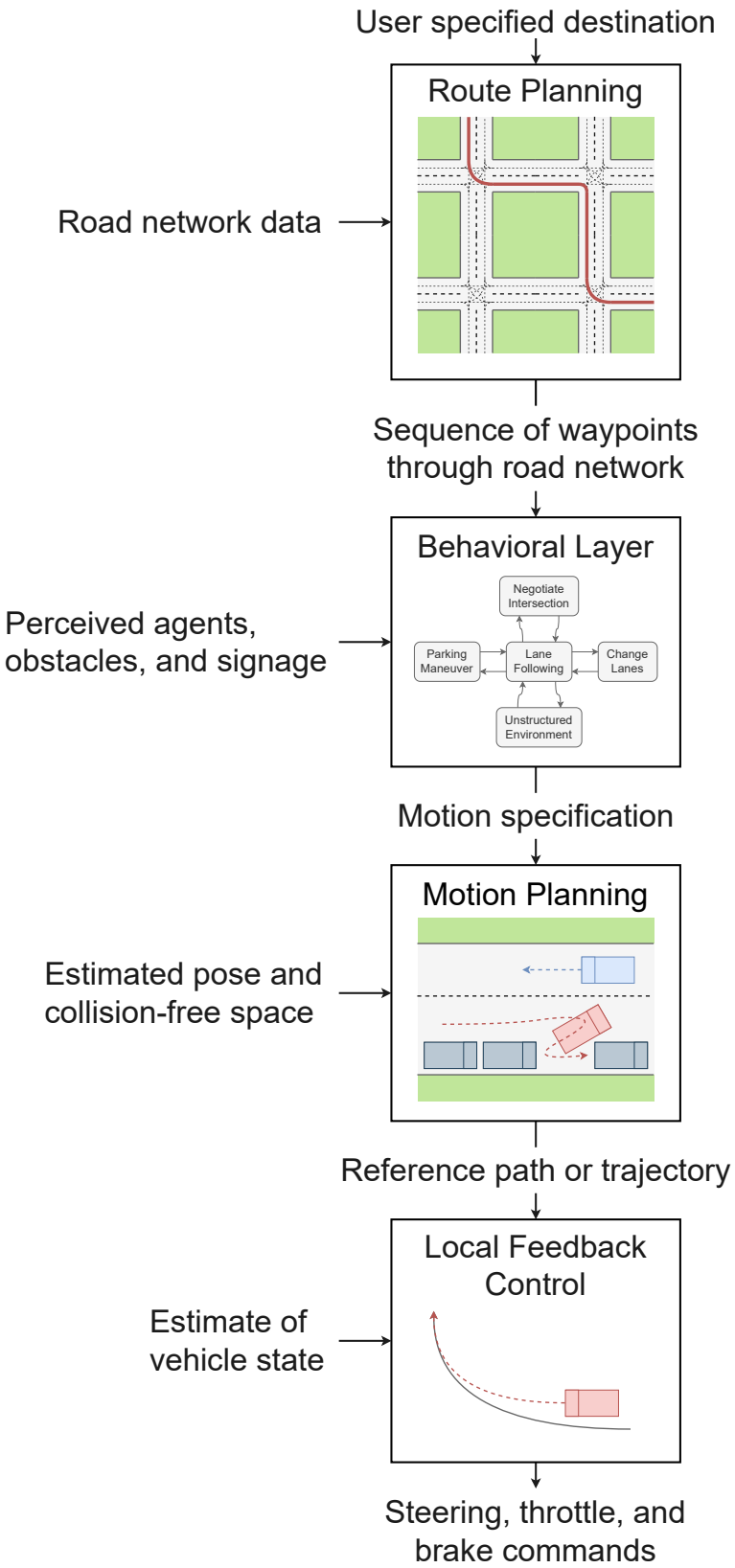


Figure 1.1: Reference autonomy architecture (adapted from [28, Fig. 1])

misnomer. Open-loop controllers operating in a receding horizon scheme (as is the case for model predictive control) offer an adequate alternative to feedback controllers [27].

Each layer is a distinct subfield within the robotics and automation communities, so contributing to the entire AV stack is infeasible. This dissertation focuses on the behavior planning and motion planning layers because they offer the most opportunities to incorporate inter-agent influences. Route planning layers generate high-level routes based on macroscopic traffic patterns; they ignore influencing factors from individual vehicles. Feedback (or receding horizon) control layers track low-level trajectories generated by motion planners. They ensure actuator inputs are safe and typically operate over short control horizons, much smaller than path planning horizons [27].

1.2 Intersection Management

Intersections are common throughout road networks, but they cause congestion and led to over 27 % of fatalities in 2018 [29]. Of those fatalities, approximately 32.7 % occurred at signalized intersections, and about 67.3 % occurred at unsignalized ones. Traffic signals regulate flow by iterating through *phases*: periods during which groups of nonconflicting movements (*e.g.*, left turn, right turn, straight) are allowed [30, Ch. 5]. Unsignalized intersections rely on drivers to follow road signs and rights-of-way.

Existing signalized intersections suffer several inefficiencies. Timer-only signals follow rigid schedules that ignore present traffic demands. Adaptive traffic signals need to be manually tuned and retuned as traffic patterns shift. Signal safety thresholds can be overly conservative, leading to delays and congestion. Additionally, phase schedules ignore passenger preferences. Unsignalized intersections face similar challenges. Delays from constant starting and stopping can accumulate. Crossing orders can be ambiguous, leading to collisions or stalemates. They may also ignore passenger preferences.

With the recent influx of vehicle-to-infrastructure (V2I) communication, emerging intelli-

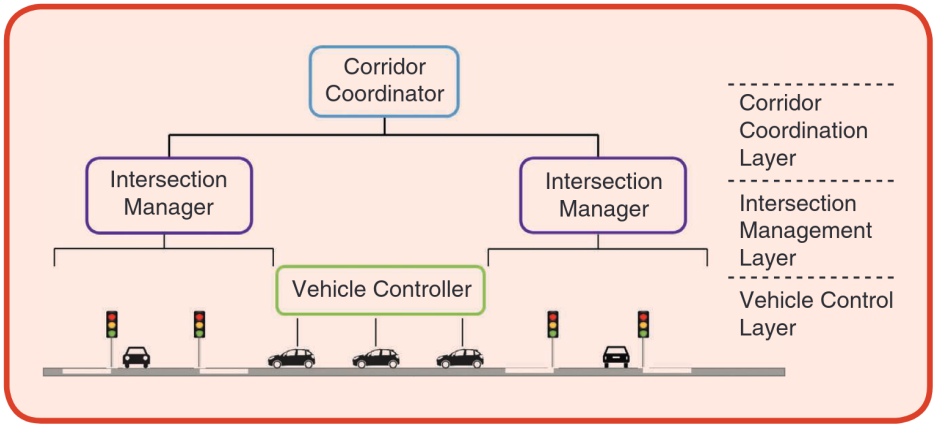


Figure 1.2: Intersection management hierarchy (from [31, Fig. 2]).

gent transportation systems (ITS) began to improve or supplement conventional intersection management methods. *signalized intersection management (SIM) systems* replace conventional ones at signalized intersections. *autonomous intersion management (AIM) systems* systems regulate unsignalized intersections [31]. This dissertation focuses on the latter as they provide more opportunities for collaboration among AVs.

SIMs and AIMs establish a hierarchy on management methods [31], visualized in Fig. 1.2. At the top, the *corridor coordination* layer deals with multiple intersections and optimizes traffic flow across a transportation network. The *intersection management* (or *trajectory planning*) layer operates on a single intersection; they sequence crossings based on vehicles' planned trajectories. The *vehicle control* layer deals with individual agents, and systems in this layer (*e.g.*, trajectory planners) generate crossing trajectories. Note that layers overlap at the borders. For example, an intersection management system may request that vehicles adjust their trajectories to satisfy some intersection-level goal. This dissertation focuses only on the lower two layers because those deal directly with AVs.

A key distinguishing factor among AIM systems is their mechanism for reserving intersection access [31], visualized in Fig. 1.3. *Intersection-based* methods allocate exclusive access to individual vehicles for a period. *Tile-based methods* partition the intersection into smaller regions and assign them to individual vehicles. Reservations are feasible when vehicles own disjoint tile sets. *Conflict-point-based* methods schedule vehicles so that their trajectories do

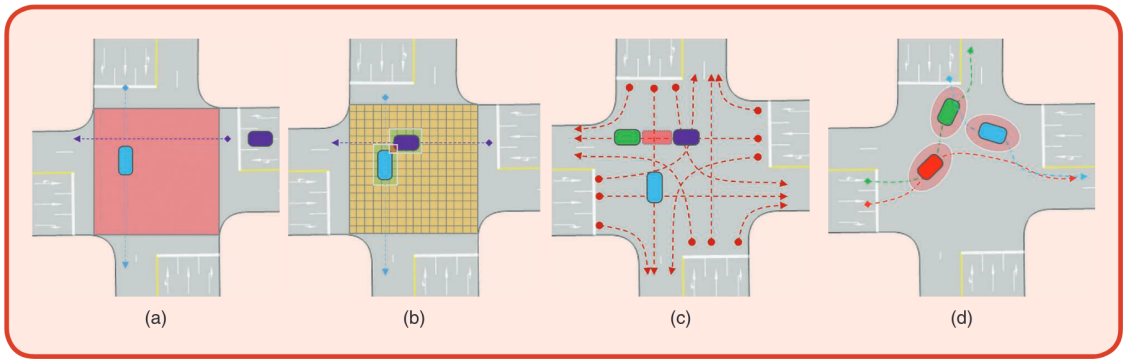


Figure 1.3: Intersection reservation types (from [31, Fig. 4]).

Table 1.3: Intersection Management Types

Method	Coupling	Crossing Schedule	Crossing Duration
Direct	Centralized	IMS	IMS
Scheduling	Less Centralized	IMS	Vehicle
Auction	Decentralized	Vehicle	Vehicle

IMS: determined by intersection management system
Vehicle: determined by vehicles

not overlap. *Vehicle-based* methods assign nonconflicting trajectories to each vehicle.

AIM systems can also differ in their scheduling policies. Common ones include *first-come, first-served (FCFS)*; *system optimal*; and *heuristic* policies. FCFS policies schedule vehicles in order of increasing arrival time. System-optimal policies generate schedules that optimize some performance metric, for example, maximizing throughput. Heuristic policies include other ways to inform the system better.

Management methods generally fall under three categories, summarized in Table 1.3. Direct control methods generate velocity profiles for each vehicle to follow, ensuring that they arrive staggered at the intersection. The system in [32, 33] used the unicycle kinematic motion model and vehicles' shared velocity constraints to generate appropriate velocity profiles. Vehicles crossed the intersection in an FCFS manner, and the system used the tile-based reservation method to avoid collisions. Researchers in [34] proposed a decentralized, game-theoretic method for generating velocity profiles. One vehicle determined a cooperative–competitive equilibrium among all vehicles, yielding collision-free trajectories

that balanced energy usage and time efficiency.

Unfortunately, direct methods tightly couple vehicles by assuming control objectives, vehicle models, or private information. The AIM system in [32, 33] assumed a motion model and that vehicles crossed the intersection at full speed. Such speed could be uncomfortable for some passengers. The game-theoretic approach in [34] was limited to only two vehicles, and it assumed that the determining vehicle knew the other's objective function.

Scheduling-based methods provide increased privacy to individual vehicles and looser coupling. They assign arrival times instead of velocity profiles and expect vehicles to generate their velocity profiles. In [35], vehicles published their state and desired arrival time, and the system scheduled crossings as close as possible to that time. The system formulated the scheduling problem as a mixed-integer linear program (MILP) with constraints to avoid collisions. However, the system still assumed some of the vehicles' dynamics, and it had parameters to balance between optimizing for time delay and vehicle preferences. Vehicles expressed their preferences through a scalar arrival time value.

Auction-based management methods provide the loosest coupling and offer the most expressivity [36, 37, 38]. Vehicles bid with money or tokens to win priority over the other participants. This management method comprises several variations. Some allow all vehicles within a lane to contribute toward a single, collective bid, while others restrict participants to only the lead vehicles. Auction outcomes can choose a single vehicle to cross or specify a crossing order for all participants in that round. The winner might pay the highest bid (a first-price auction) or the second-highest bid (a second-price or Vickrey auction). The latter option creates an incentive mechanism to keep bidders honest. In some cases, all bidders pay regardless of which won. Each vehicle may also implement a different bidding strategy: fixed, variable, or random amounts.

While auction mechanisms allow vehicles to convey their preferences explicitly, they have drawbacks. They attempt to maximize profits over social welfare, so the winning vehicle has the highest bid, not necessarily the greatest need. Consider two vehicles: one poor and one

rich. The poor vehicle wants to cross the intersection as fast as possible and thus has a high valuation V_{poor}; however, they can place only a low bid B_{poor}. In this case, their bid is smaller than their valuation ($B_{\text{poor}} < V_{\text{poor}}$). The rich vehicle is indifferent to when they cross, but they can afford a bid that matches their valuation ($B_{\text{rich}} = V_{\text{rich}}$). Assuming $V_{\text{poor}} > V_{\text{rich}}$ and $B_{\text{rich}} > B_{\text{poor}}$, the rich vehicle crosses first (maximizing profit) instead of the poor vehicle (maximizing social welfare). As with real economies, transportation economies can suffer from wealth inequality, contributing to unfair auctions. Therefore, many auction-based AIM systems subsidize bids for those that cannot afford them.

To summarize, direct control methods suffer from scalability issues and tight coupling among agents. Scheduling-based methods do not consider preferences on vehicle motion. Both methods impose size limitations on the problems (*e.g.*, excluding turn maneuvers). Money-based auctions maximize profits and can lead to wealth inequality among participants. Most importantly, no existing method optimizes specifically for social welfare to the best of our knowledge.

1.3 Motion Planning

Driving is complex, and static and dynamic obstacles litter roads. Drivers need to plan collision-free maneuvers while also adhering as close as possible to road laws. Recent advances in motion planning enable AVs to navigate safely around most static and simple dynamic obstacles. However, AVs share the road with many intelligent agents: other vehicles, cyclists, and pedestrians. They all influence each other; when one acts, others respond. These influences challenge motion prediction and planning because AVs need to consider coupling interactions when making decisions.

Conventional Motion Planning

Motion planning involves mapping the perceived environment using sensors in the *perception* layer to some behavior executed by controllers in the *act* layer. Planners share a common objective: generate a feasible, collision-free trajectory between an agent's current configuration and some goal configuration. Supplemental objectives include satisfying timing, comfort, or other constraints. In unobstructed environments, this task is straightforward, and the shortest path between configurations for AVs is typically a Dubins curve [39, Sec. 15.3.1].

The planning space in unstructured environments is expansive, but road networks impose structures that restrict trajectories. Vehicles must obey speed limits and lane boundaries. Additionally, they can make only certain maneuvers within their current lane. In some (emergency) situations, vehicles need to violate these road laws to stay safe. This structure often removes unstructured shortest paths, but it helps reduce the search space for determining other feasible paths.

Adding obstacles to the environment complicates planning. Problems become non-convex in general, which requires nonlinear solvers and guarantees only local solutions. AV motion planning methods fall into four main categories [28, 40]: variational, graph search, incremental search, and curve interpolation. Variational (trajectory optimization) methods formulate the path as a parameterized objective function and use nonlinear solvers to converge to locally-optimal solutions. They decompose further into two subclasses: direct (shooting, collocation, or pseudo-spectral) methods and indirect methods. Collocation methods solve differential equations by choosing points in the domain then selecting a candidate solution that satisfies those points. Shooting methods solve boundary value problems by solving the equations at sampled initial conditions and choosing the solution that satisfies desired boundary condition.

Graph-search algorithms discretize the vehicle's configuration space, forming a graph that encodes vehicle states as nodes and inputs as edges. The planner then uses Dijkstra, A^*, D^* [41], or another graph search algorithm to find a minimum cost path. Pregenerated

lane graphs encode road geometry, lanes boundaries, and traffic regulations obtained from high-resolution maps. While they suffice for normal operating conditions, planners must use a more general solution if the vehicle needs to deviate from an established path—for example, to avoid an obstacle. In these situations, geometric or sampling methods replace the lane graph.

Incremental search methods sample the vehicle's configuration space to form a reachability graph or tree that expands until the planner finds a path. The Rapidly-Exploring Random Trees (RRT) algorithm [42] and its variants are the most well-known implementations in this category.

Finally, interpolating curve methods use lines and circles or clothoid, polynomial, or Bézier curves to interpolate the path between a sequence of given reference points (called *knots*). A route planner or some other component provides the motion planner with several reference points, and the motion planner uses one of these smooth curves to connect those points.

Planning around dynamic obstacles presents an increased challenge because planners must now account for other agents' movements. Generated trajectories are typically open-loop, so planners must know or predict how objects move during the planning horizon [39, Ch. 7]. Motion prediction algorithms compose an entire research field; two general approaches are to use either a mathematical model (*e.g.*, the intelligent driver model [43]) or a data-driven one (*e.g.*, Waymo's VectorNet [44]). The type of model determines its prediction accuracy over short-, medium-, and long-term planning horizons [45].

If information about objects' movements is unavailable, planners need to use a feedback scheme to correct modeling errors. One technique is to run an open-loop planner in a receding horizon fashion. This approach is commonly integrated with a controller to yield model predictive control [28]. Obstacles are assumed to be stationary over the planning horizon, and they constitute state-space constraints. The controller then generates a tracking trajectory that best aligns with a reference one. When the planner executes in the future, the

obstacles' new positions are updated, and the process repeats. This method is reactionary and works best if the obstacles move slowly relative to the planning frequency.

The motion prediction methods described above have a significant limitation: they assume obstacles follow their trajectories independent of the ego vehicle's movements. This assumption is valid for a ball rolling in front of a car. However, it fails when dealing with intelligent agents, *e.g.*, a kid chasing after that ball. Planners need to incorporate others' possible decisions to improve their predictions. These scenarios necessitate a theory capable of modeling interactions among agents—game theory.

Game-Theoretic Motion Planning

Game theory models the interactions between two or more intelligent decision-makers, which naturally aligns with motion planning problems. Roads are highly interactive shared environments, and all road users possess some level of decision-making capabilities. Games are a general concept that can be specified using several categories: cooperative *vs.* noncooperative, simultaneous *vs.* sequential, and zero-sum *vs.* general-sum, among others.

When applied to motion planning problems, games take a dynamic form—they evolve based players' actions [46, Ch. 6]. Games with states that change over time are *dynamic games*. Parallel to control theory, they form dynamical systems with players' actions as the inputs. Difference equations describe state changes when time is discrete. For each time step (stage), agents play a simultaneous game generated by the state and actions from the previous stage. Discrete-time dynamic games are therefore called *multi-stage games*. As time tends toward a continuum, differential equations replace the difference equations, and the game transforms into a *differential game* with continuous state updates.

Depending on the agents' strategies, games are either cooperative or noncooperative. Agents in a *cooperative game* work together to satisfy a common goal, and a single optimization problem can determine a solution. *Noncooperative games*, however, feature self-interested agents that optimize only their objectives. Usually, agents' objectives conflict,

meaning there is no unique optimum. The *Pareto front* is a set that contains all incomparable but optimal outcomes that could suffice as a solution. Equilibrium is a common solution concept class used to solve noncooperative games. The Nash equilibrium (NE) is the most popular. It occurs when no agent has an incentive to unilaterally change their strategy, assuming all other agents are playing their optimal strategies.

Furthermore, games can be purely competitive *zero-sum games* or more relaxed *general-sum games*. Racing is commonly formulated as a zero-sum game because it establishes a clear winner and loser: the sum of rewards among all players equals zero [47, 48]. However, for everyday driving, agents do not necessarily compete. Equilibria can benefit multiple agents without necessarily hurting others. Therefore, general-sum game formulations are more appropriate for on-road driving [49]. This research focuses on dynamic noncooperative general-sum games for everyday driving.

Computing Nash equilibria for normal form general-sum games is PPAD-hard [50, Ch. 4], meaning an efficient polynomial-time algorithm for solving general instances does not exist (assuming $\mathsf{P} \neq \mathsf{NP}$). Instead, numerical methods approximate a solution or refine the problem to reduce complexity [51]. Researchers in [52, 53] approximate NE using an augmented Lagrangian formulation that converts hard constraints into soft ones. Another method, inspired by the iterative linear-quadratic regulator, linearizes the game's dynamics and assigns quadratic costs to each player [54, 55].

Games often admit several Nash equilibria, which presents another challenge [49]. Equilibria refinement techniques help prune NE by introducing additional optimality criteria (*e.g.*, stability, feasibility). Alternatively, mechanism design approaches modify the game's formulation to be better posed, thus reducing the number of admissible NE. For example, *urban driving games* [56] are potential games formulated for everyday driving. They exhibit an ordinal potential structure, which leads to a unique NE.

Game-theoretic motion planners commonly solve a copy of the problem locally: each vehicle solves an instance of the same problem and plays on behalf of their opponents [47,

52, 53]. However, independent game instances lead to vehicles occasionally converging to different, possibly conflicting NE. Consider a highway merging scenario involving an on-ramp car and an on-highway car. If the on-ramp car converges to an NE in which the on-highway one moves, but the on-highway car believes the on-ramp car will yield, the two cars will collide. While empirical results [53] suggest that solving the problem in a receding horizon fashion reduces this probability, no formal guarantees exist to prevent this. The question then becomes how to make all agents agree on the same or compatible NE.

Because game-theoretic motion planners compute the NE locally without communication, they need notions of how other agents behave. Almost all solution methods assume complete knowledge of their neighbors' objective functions. However, this approach is not practical as objectives vary among vehicles, and they may not be well-defined for human-driven vehicles. Some works attempt to relax this assumption by learning parameterized cost functions from observations. In particular, researchers in [57] use Stackelberg equilibria to estimate driver aggressiveness during lane changes. Other works formulate inverse dynamic games to learn more generic cost functions [58, 59].

Most game-theoretic motion planning literature focuses on three scenarios (visualized in Fig. 1.4): ramp merging, lane changing, and intersection navigation. These types of interactions lend themselves nicely to receding horizon control because vehicles can always choose actions that progress them toward their goals; no other vehicle can completely block them. Therefore, optimal paths within the planning horizon contribute toward the overall path, meaning they will converge to a solution.

To help promote feasibility, collision avoidance and staying on the road are typically the only hard constraints. Staying within lanes or obeying traffic laws is a soft constraint. Some approaches, such as the urban driving games framework, soften all constraints, which completely avoids infeasibility—a principle called *minimum violation planning*. Reformulating constraints might work for open roads where vehicles could temporarily drive in other lanes or on the road's shoulder, but in some scenarios, such as tight alleyways, violating position

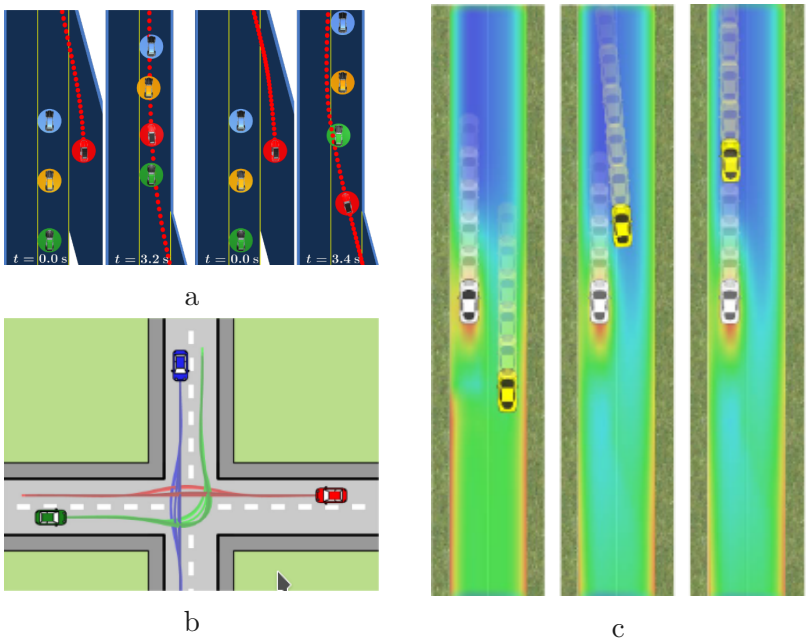


Figure 1.4: Common interaction scenarios. a Ramp merging (from [53, Fig. 1]. b Intersection navigation (from [56, Fig. 1]). c Lane changing (from [60, Fig. 1]).

constraints would damage vehicles.

Receding horizon planning methods perform adequately for interactions in common driving scenarios, but they provide neither convergence nor optimality guarantees. In problems that require longer planning horizons, vehicles may freeze because they cannot compute a feasible solution.

1.4 Dissertation Structure

This section introduces the dissertation and provides general background information. Chapter 2 presents a path-constrained, time-assigned, control-minimal trajectory optimizer that is a base component to our intersection management system. Chapter 3 then presents an auction-based intersection management system. Chapter 4 discusses cooperative motion planning for spatially-constrained environments. Finally, Chapter 5 concludes the dissertation and proposes future research directions.

Fig. 1.5 visualizes this dissertation's contributions. We investigate two application areas

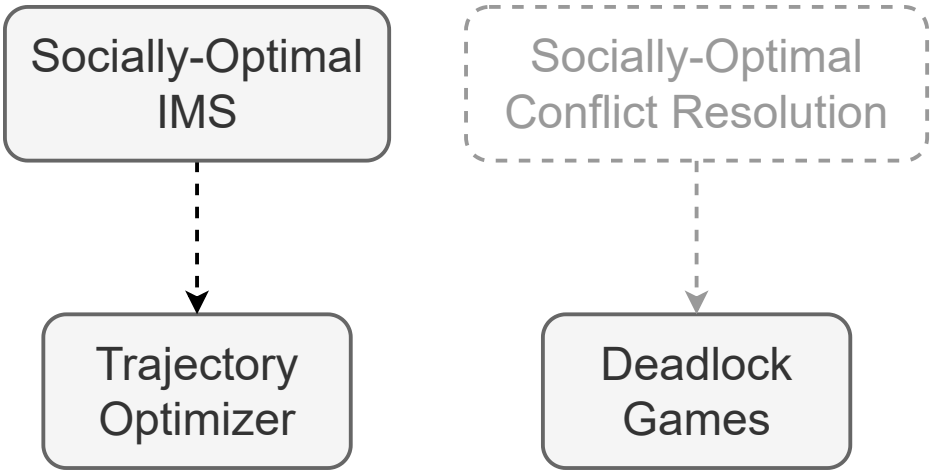


Figure 1.5: The dissertation's contributions.

under the cooperative driving automation context. The trajectory optimizer work provides a key component in the higher-level socially-optimal intersection management system. The deadlock game formulation and solution algorithm establishes the foundation for continued work on a higher-level socially-optimal conflict resolution system to resolve competing solutions.

Chapter 2

Trajectory Optimization

This chapter presents a path-constrained, time-assigned, control-minimal trajectory optimization problem. The work in this chapter was motivated by the initial versions of the work presented in Chapter 3 for the IMS. While chronologically, we are presenting the material out of order, this structure make more sense compositionally.

Within the IMS framework, vehicles are assigned crossing durations that dictate how long they need to take the crossing the intersection. This trajectory optimization problem is a component within the vehicle's planning stack to generate a reference trajectory that can be tracked.

Road networks are generally well-defined structures where autonomous vehicles need to follow two high-level rules: 1. avoid collisions; and 2. traverse the path (trip route) in minimal time, with minimal control, or a combination of these goals. Some transportation applications benefit from predictable, but not necessarily minimal, timing. In particular, scheduled transportation tasks with specific timing requirements may be the primary objective. Consider a bus route comprising several stops, each with a pre-arranged departure time. If the bus arrives too early, it wastes time and energy waiting at its stop. If it arrives too late, it will cause disruptions in the schedule. In such a use case, the bus's reference trajectory and control signals should ensure the bus arrives as close to the specified arrival time as possible.

One can generalize this scheduling concept by considering intersection management systems (IMS) that rely on accurate timing to predict vehicles' traversal times and safely schedule crossings [31]. In previous work, we developed an auction-based IMS in which vehicles bid using their cost functions [61]. The vehicles' motion planners generated reference trajectories and control inputs to drive through the intersection in so-called *crossing durations*. They also calculated a cost for each crossing duration and used those costs to define a *crossing cost* function. The management system used these functions to assign final crossing durations to each vehicle.

For each candidate crossing duration, vehicles solved a trajectory optimization problem. Their objective functions comprised a crossing duration cost, a reference tracking cost, and a control effort cost. This formulation sufficed for longer candidate crossing durations, but it caused issues when evaluating shorter ones. Vehicles' motion plans significantly deviated from their reference paths in order to satisfy the crossing duration constraint, sacrificing path tracking for a lower control cost. These reference tracking deviations often made the vehicles move into dangerous situations, such as cutting over sidewalks. To ensure that the IMS schedule assignments would truly be safe, we need vehicles to generate control inputs that kept them on their reference paths while also satisfying the IMS time assignment.

The above challenge motivated the work presented in this chapter. The motion planners still seek an effort-minimal reference trajectory to traverse the path. However, we reformulate the path tracking goal to a hard constraint so that the planners consider only control inputs that keep the vehicle on the path. We kept the crossing duration constraint, meaning vehicles have a finite time to traverse their path. Using this new approach, *control-minimal time-assigned path-constrained trajectory optimization*, the IMS may more accurately compare crossing costs.

2.1 Related Work

Solution categories for path-constrained trajectory planning include dynamic programming [39, Sec. 14.6.3], numerical integration [39, Sec. 14.6.3], reachability analysis (RA) [62], and convex optimization [63]. Numerical integration and convex optimization methods are particularly useful when performing time- or energy-optimal planning for robotic arms [63], biped robots [64], and mobile ground robots [65, 66].

Time-optimal trajectory planning [67, 68, 69] tries to minimize the path traversal time, while energy-optimal planning typically disregards it. In contrast to those goals, we seek the lowest control effort needed to move a robot along a path in a specific duration. Compared to other problem formulations, the time-assigned variant has received minimal attention.

Researchers in [70] were one of the first to study time-assigned path-constrained trajectory planning, and they solved the problem using nonlinear semi-infinite programming. More recently, the authors of [62] extended their RA framework for robotic arms to find trajectories of specified duration.[1] Their algorithm searched for a constraint-satisfying deformation between the time-maximal and time-minimal trajectories. However, the final result may not be control-minimal.

Another work developed a speed planner for a heavy vehicle crossing an intersection [69]. To avoid collisions, the optimization problem imposed time window constraints. The vehicle's crossing duration had to be within one of these windows. Our proposed formulation, in contrast, requires the vehicle to cross in a specific amount of time.

Compared to the other solution methods, convex optimization provides a framework that allows for flexible customization of objectives and constraints. Additionally, several solvers and front-ends exist that that ease its implementation.

[1] `https://web.archive.org/web/20220421233339/https://hungpham2511.github.io/toppra/python_api.html`

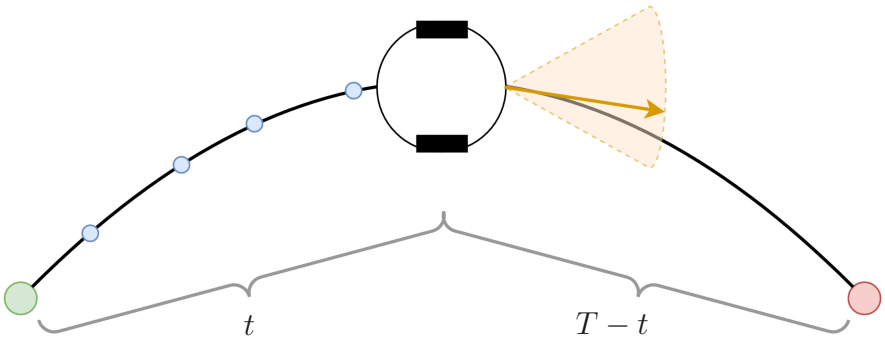


Figure 2.1: The robot traverses the path (black) starting from the beginning (green) and stopping at the end (red). Blue points are the robot's planning history. The planner chooses control inputs (orange arrow) to keep the robot on the path. The orange cone represents possible control inputs. The robot traverses the path in T seconds.

2.2 Problem Formulation

Consider the problem visualized in Fig. 2.1. We want a mobile ground robot to traverse a path close to a specific duration while minimizing its control effort. The robot must start from the path's beginning, stop at its end, and stay on this path while moving.

2.2.1 Robot Motion Model

We model the robot in Fig. 2.1 as a rigid body moving on a plane, making its configuration space $\mathcal{C}$ equal to the special Euclidean group $SE(2)$. The robot's configuration $\boldsymbol{q}(t) \in \mathcal{C}$ is $\boldsymbol{q}(t) \coloneqq (x(t), y(t), \theta(t))$, where x and y represent the robot's position on the plane, and θ is its heading with respect to the x-axis. The configuration and its components are functions of time $t \in \mathbb{R}_+$, where $\mathbb{R}_+$ is the set of nonnegative real numbers. For clarity, we will drop the explicit time dependence notation.

The robot moves according to a unicycle motion model under no-slip conditions. The following system of equations defines the configuration transformation for a single-order

kinematic unicycle [39, eq. (13.18)]:

$$\begin{aligned} \dot{x} &= v \cos\theta \\ \dot{y} &= v \sin\theta \\ \dot{\theta} &= \omega, \end{aligned} \tag{2.1}$$

where v is the robot's scalar linear velocity (forward is positive), and ω is its scalar angular velocity (counterclockwise is positive). Mark $\dot{\square}$ denotes the first time-derivative.

We could use the linear and angular scalar velocities of (2.1) as control inputs, but the resulting system would be unrealistic as robots do not start and stop instantaneously. Differentiating (2.1) with respect to time results in the second-order kinematic unicycle. The following system of equations defines the new configuration transformation:

$$\begin{aligned} \ddot{x} &= u_{t,\text{lin}} \cos\theta - \dot{\theta}\, v_{\boldsymbol{q}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \sin\theta \\ \ddot{y} &= u_{t,\text{lin}} \sin\theta + \dot{\theta}\, v_{\boldsymbol{q}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \cos\theta \\ \ddot{\theta} &= u_{t,\text{ang}}, \end{aligned} \tag{2.2}$$

where $u_{t,\text{lin}}$ is the scalar linear acceleration, $u_{t,\text{ang}}$ is the scalar angular acceleration, and $\ddot{\square}$ denotes the second time-derivative. We choose the scalar linear and angular accelerations as the system's inputs $\boldsymbol{u}_t \coloneqq (u_{t,\text{lin}}, u_{t,\text{ang}})$. Subscript $\square_t$ indicates that the system controls vary with time. Function $v_{\boldsymbol{q}} \colon \mathcal{C} \times \mathbb{R}^3 \to \mathbb{R}$ is the robot's scalar linear velocity in terms of its configuration and first time-derivative $\dot{\boldsymbol{q}}$. It is given by

$$v_{\boldsymbol{q}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \sqrt{\dot{x}^2 + \dot{y}^2}.$$

We use the $\square_{\boldsymbol{q}}$ subscript to denote functions of the robot's configuration and its derivatives.

Equation (2.2) is control-affine and may be restructured as

$$\ddot{\boldsymbol{q}} = \boldsymbol{f}_{\boldsymbol{q}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{G}_{\boldsymbol{q}}(\boldsymbol{q}) \boldsymbol{u}_t, \tag{2.3}$$

where

$$\boldsymbol{f_q}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \begin{bmatrix} -\dot{\theta} v_{\boldsymbol{q}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \sin\theta \\ \dot{\theta} v_{\boldsymbol{q}}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \cos\theta \\ 0 \end{bmatrix},$$

and

$$\boldsymbol{G_q}(\boldsymbol{q}) = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix}.$$

2.2.2 Path-Constrained Motion

We define a path as a continuous, collision-free function $\boldsymbol{\tau}\colon [0,1] \to \mathcal{C}_{\text{free}}$ that maps a point $s \in [0,1]$ to a robot configuration. Set $\mathcal{C}_{\text{free}}$ is the subset of the robot's configuration space that excludes obstacles. Point s, which we term the *path-position*, is the robot's position along the path. Value $\boldsymbol{\tau}(s=0)$ represents the path's beginning, and $\boldsymbol{\tau}(s=1)$ is its end. The path-position is an implicit function of time.

Suppose we constrain the robot's motion to a dynamically-feasible path. We express its configuration in terms of its path-position [39, Sec. 14.6.3]:

$$\boldsymbol{q} = \boldsymbol{\tau}(s). \tag{2.4}$$

We also relate the robot's velocity $\dot{\boldsymbol{q}}$ to the path by differentiating (2.4) with respect to time:

$$\dot{\boldsymbol{q}} = \frac{\mathrm{d}\boldsymbol{\tau}}{\mathrm{d}s}\frac{\mathrm{d}s}{\mathrm{d}t} = \boldsymbol{\tau}'(s)\dot{s}, \tag{2.5}$$

where $\square'$ denotes the first derivative with respect to the path-position, and $\dot{s}$ is the robot's *path-velocity*.

Furthermore, we express the robot's path-constrained acceleration by differentiating (2.5) with respect to time:

$$\ddot{\boldsymbol{q}} = \frac{\mathrm{d}^2\boldsymbol{\tau}}{\mathrm{d}s^2}\dot{s}^2 + \frac{\mathrm{d}\boldsymbol{\tau}}{\mathrm{d}s}\ddot{s} = \boldsymbol{\tau}''(s)\dot{s}^2 + \boldsymbol{\tau}'(s)\ddot{s}, \tag{2.6}$$

where $\square''$ denotes the second derivative with respect to the path-position, and $\ddot{s}$ is the robot's *path-acceleration*.

Now that we have the robot's configuration expressed in terms of the path, we substitute (2.4) and (2.5) into (2.3) to derive the path-constrained system dynamics:

$$\begin{aligned}\ddot{\boldsymbol{q}} &= \boldsymbol{f_q}\big(\boldsymbol{\tau}(s), \boldsymbol{\tau}'(s)\dot{s}\big) + \boldsymbol{G_q}\big(\boldsymbol{\tau}(s)\big)\boldsymbol{u}_t \\ &= \boldsymbol{f}_s(s, \dot{s}) + \boldsymbol{G}_s(s)\boldsymbol{u}_t, \end{aligned} \tag{2.7}$$

where

$$\boldsymbol{f}_s(s, \dot{s}) = \begin{bmatrix} -\tau_3'(s)\, \dot{s}\, v_s(s, \dot{s})\, \sin(\tau_3(s)) \\ \tau_3'(s)\dot{s} v_s(s, \dot{s}) \cos(\tau_3(s)) \\ 0 \end{bmatrix},$$

$$\boldsymbol{G}_s(s) = \begin{bmatrix} \cos(\tau_3(s)) & 0 \\ \sin(\tau_3(s)) & 0 \\ 0 & 1 \end{bmatrix},$$

and

$$v_s(s, \dot{s}) = \sqrt{[\tau_1'(s)\dot{s}]^2 + [\tau_2'(s)\dot{s}]^2}. \tag{2.8}$$

Subscript $\square_s$ denotes functions of the robot's path-position and its derivatives. Notation $\tau_i^{\square}$ denotes the i^{th} component of the path or its derivative. Furthermore, we equate (2.6) and (2.7) to express the path-constrained dynamics entirely in terms of the path-position, its derivatives, and the system input:

$$\boldsymbol{\tau}''(s)\dot{s}^2 + \boldsymbol{\tau}'(s)\, \ddot{s} = \boldsymbol{f}_s(s, \dot{s}) + \boldsymbol{G}_s(s)\boldsymbol{u}_t. \tag{2.9}$$

We now define the basic control-minimal, time-assigned, path-constrained trajectory op-

timization problem:

$$\underset{s,\,\boldsymbol{u}_t}{\text{minimize}} \quad \int_0^T \|\boldsymbol{u}_t(t)\|_2^2 \,\mathrm{d}t \tag{2.10a}$$

$$\text{subject to} \quad \boldsymbol{\tau}''(s)\dot{s}^2 + \boldsymbol{\tau}'(s)\ddot{s} = \boldsymbol{f}_s(s,\dot{s}) + \boldsymbol{G}_s(s)\boldsymbol{u}_t, \tag{2.10b}$$

$$\underline{\boldsymbol{u}} \leq \boldsymbol{u}_t \leq \overline{\boldsymbol{u}}, \tag{2.10c}$$

$$\text{for } t \in [0, T],$$

$$s(0) = 0, \tag{2.10d}$$

$$s(T) = 1 \tag{2.10e}$$

where $\underline{\boldsymbol{u}}$ and $\overline{\boldsymbol{u}}$ are the lower and upper actuator limits for $\boldsymbol{u}_t$, respectively. Constraint (2.10b) restricts the robot's motion to the path, (2.10c) bounds the control inputs, and (2.10d)–(2.10e) define the boundary constraints. The traversal time T is a design parameter that specifies how long the robot has to traverse the path.

We want to find an optimal path-position function $s^\star$, which defines a time scaling along the path [39, Sec. 7.1.3], and an optimal control input function $\boldsymbol{u}_t^\star$ that achieves the time scaling. In contrast to time-minimal or energy-minimal problem variants, we seek a time scaling that causes the robot to finish traversing the path in exactly T seconds.

The resulting control signal could be fed into the system. Alternatively, the time scaling could be composed with the path to form a reference trajectory that is passed into a tracking controller.

2.2.3 Convex Reformulation

Optimization problem (2.10) is nonlinear in the path-position and its derivatives, but we reformulate it into a convex problem using a nonlinear change of variables. We follow the reformulation in [63], which presented a convex optimization solution for time-minimal, path-constrained trajectory optimization with a robotic arm. Our work studies the time-assigned

problem variant for a mobile robot.

Before introducing the nonlinear change of variables, we simplify the path-constrained dynamics from (2.9). Function v_s is linear in the path-velocity, so we extract $\dot{s}$ from the radical in (2.8) to form (with a slight abuse of notation) an alternate equation:

$$v_s(s, \dot{s}) = \dot{s} v_s(s),$$

where

$$v_s(s) = \sqrt{[\tau_1'(s)]^2 + [\tau_2'(s)]^2}. \tag{2.11}$$

By substituting (2.11) into (2.9), we express the path-constrained system dynamics with a more concise model:

$$\boldsymbol{h}_s(s)\ddot{s} = \boldsymbol{f}_s(s)\dot{s}^2 + \boldsymbol{G}_s(s)\boldsymbol{u}_t,$$

where

$$\boldsymbol{h}_s(s) = \boldsymbol{\tau}'(s),$$

and

$$\boldsymbol{f}_s(s) = \begin{bmatrix} -\tau_1''(s) - \tau_3'(s) v_s(s) \sin(\tau_3(s)) \\ -\tau_2''(s) + \tau_3'(s) v_s(s) \cos(\tau_3(s)) \\ -\tau_3''(s) \end{bmatrix}.$$

Next, we change the objective function's integration variable from time (t) to the path-position (s):

$$\int_0^T \|\boldsymbol{u}_t(t)\|_2^2 \, \mathrm{d}t = \int_{s(0)}^{s(T)} \frac{\|\boldsymbol{u}_t(t)\|_2^2}{\mathrm{d}s/\mathrm{d}t} \, \mathrm{d}s = \int_0^1 \frac{\|\boldsymbol{u}_s(s)\|_2^2}{\dot{s}} \, \mathrm{d}s \,.$$

As a consequence of changing the integration variable, we must also introduce a new control input $\boldsymbol{u}_s \colon [0, 1] \to \mathbb{R}^2$ that is a function of the path-position instead of time. Propagating the change of variables into the rest of the optimization problem, we treat s, $\dot{s}$, and $\ddot{s}$ as decision variables.

Finally, we introduce the nonlinear change of variables [63, eqs. (17) and (18)]:

$$z(s) \coloneqq \dot{s}^2$$

$$\nu(s) \coloneqq \ddot{s}.$$

In the resulting differential-algebraic system of equations (DAE), z is the differential state, $\boldsymbol{u}_s$ is the algebraic state, and ν is the control input. As derived in [63], the system has linear dynamics defined by [63, eq. (19)]

$$z'(s) = 2\nu(s).$$

With the change of variables, we redefine problem (2.10) as a convex one:

$$\begin{aligned}
\underset{z,\,\boldsymbol{u}_s,\,\nu}{\text{minimize}} \quad & \int_0^1 \frac{\|\boldsymbol{u}_s(s)\|_2^2}{\sqrt{z(s)}}\,\mathrm{d}s && \text{(2.12a)}\\
\text{subject to} \quad & \boldsymbol{h}_s(s)\nu(s) = \boldsymbol{f}_s(s)z(s) + \boldsymbol{G}_s(s)\boldsymbol{u}_s(s), && \text{(2.12b)}\\
& z'(s) = 2\nu(s), && \text{(2.12c)}\\
& 0 < z(s) \leq \overline{z}, && \text{(2.12d)}\\
& \underline{\boldsymbol{u}} \leq \boldsymbol{u}_s(s) \leq \overline{\boldsymbol{u}}, && \text{(2.12e)}\\
& \text{for } s \in [0,1], \\
& \int_0^1 \frac{1}{\sqrt{z(s)}}\,\mathrm{d}s \leq T && \text{(2.12f)}
\end{aligned}$$

When reformulating problem (2.10), we lost the traversal time constraint T; therefore, it is reintroduced as (2.12f).

Intuitively, the solver minimizes the objective cost by reducing z and $\boldsymbol{u}_s$; however, decreasing them too much will violate the traversal time constraint (2.12f).

2.3 Solution Description

We solve problem (2.12) using trapezoidal collocation [63, 71]. First, we discretize s into a grid of $K+1$ collocation points. Since the system's dynamics are linear in s, we approximate ν as a piecewise-constant function, z as a linear spline, and $\boldsymbol{u}_s$ as a nonlinear spline.

The objective functional (2.12a) is approximated as

$$\int_0^1 \frac{\|\boldsymbol{u}_s(s)\|_2^2}{\sqrt{z(s)}}\,\mathrm{d}s \approx \sum_{k=0}^{K}\left[\left\|\boldsymbol{u}_s^k\right\|_2^2 \int_{s^k}^{s^{k+1}} \frac{1}{\sqrt{z(s)}}\,\mathrm{d}s\right], \tag{2.13}$$

where s^k is the value of s at grid point k, and $\boldsymbol{u}_s^k$ is the value of $\boldsymbol{u}_s$ at collocation point s^k. The analytical integral for the second component of (2.13) is

$$\int_{s^k}^{s^{k+1}} \frac{1}{\sqrt{z(s)}}\,\mathrm{d}s = \frac{2\left(s^{k+1}-s^k\right)}{\sqrt{z^{k+1}}+\sqrt{z^k}},$$

where z^k is the value of z at collocation point s^k.

Given the above formulation, the trajectory optimization problem (2.12) is approximated

as a convex program:

$$\underset{\substack{z^0,\ldots,z^K \\ \boldsymbol{u}_s^0,\ldots,\boldsymbol{u}_s^{K-1} \\ \nu^0,\ldots,\nu^{K-1}}}{\text{minimize}} \quad \sum_{k=0}^{K-1} \frac{2\left(s^{k+1}-s^k\right)\left\|\boldsymbol{u}_s^k\right\|_2^2}{\sqrt{z^{k+1}}+\sqrt{z^k}} \tag{2.14a}$$

$$\text{subject to} \quad \boldsymbol{h}_s(s^k)\nu^k = \boldsymbol{f}_s(s^k)z^k + \boldsymbol{G}_s(s^k)\boldsymbol{u}_s^k, \tag{2.14b}$$

$$z^{k+1}-z^k = 2\nu^k\left(s^{k+1}-s^k\right), \tag{2.14c}$$

$$\underline{\boldsymbol{u}}_s \le \boldsymbol{u}_s^k \le \overline{\boldsymbol{u}}_s, \tag{2.14d}$$

$$0 \le z^k \le \overline{z}, \tag{2.14e}$$

$$\text{for } k = 0, 1, \ldots, K-1,$$

$$z^0 = \dot{s}_0^2, \tag{2.14f}$$

$$z^K = \dot{s}_1^2, \tag{2.14g}$$

$$\sum_{k=0}^{K-1} \frac{2\left(s^{k+1}-s^k\right)}{\sqrt{z^{k+1}}+\sqrt{z^k}} \le T \tag{2.14h}$$

2.3.1 Second-Order Cone Reformulation

We reduce program (2.14) into a more efficient, second-order cone program (SOCP) using the reformulation method described in [63]. The reformulation also provides implementation improvement. Convex program solvers that rely on disciplined convex programming, such as CVXPY, may fail to verify the convexity of complicated expressions.[2] The general SOCP structure requires a linear objective function, affine equality constraints, and second-order cone inequality constraints [72, Sec. 4.4.2].

Reformulating the convex program into an SOCP requires the introduction of additional decision variables $a^0, a^1, \ldots, a^K$; $b^0, b^1, \ldots, b^{K-1}$; and $c^0, c^1, \ldots, c^{K-1}$.

[2] `https://web.archive.org/web/20211018022509/https://www.cvxpy.org/tutorial/dcp/index.html`

We introduce constraints

$$a^k \leq \sqrt{z^k} \quad \text{for } k = 0, 1, \ldots, K. \tag{2.15}$$

Then, we re-express objective function (2.14a) as

$$\sum_{k=0}^{K-1} 2\left(s^{k+1} - s^k\right) b^k$$

and introduce constraints

$$\frac{\left\|\boldsymbol{u}_s^k\right\|_2^2}{a^{k+1} + a^k} \leq b^k \quad \text{for } k = 0, 1, \ldots, K-1. \tag{2.16}$$

Similarly, introducing constraints

$$\frac{1}{a^{k+1} + a^k} \leq c^k \quad \text{for } k = 0, 1, \ldots, K-1, \tag{2.17}$$

allows us to express (2.14h) as

$$2\sum_{k=0}^{K-1} \left(s^{k+1} - s^k\right) c^k \leq T. \tag{2.18}$$

Constraints (2.15) and (2.16)–(2.18) will become active as the program converges to a solution. When this happens, the inequalities will become equalities, and the SOCP will resemble the convex program (2.14).

Constraint (2.15) is expressed in its conic form with

$$\left\| \begin{matrix} 2a^k \\ z^k - 1 \end{matrix} \right\|_2 \leq z^k + 1; \tag{2.19}$$

constraint (2.16) becomes

$$\left\| \begin{matrix} 2u^k_{s,\text{lin}} \\ 2u^k_{s,\text{ang}} \\ a^{k+1} + a^k - b^k \end{matrix} \right\|_2 \leq a^{k+1} + a^k + b^k; \tag{2.20}$$

and (2.17) is

$$\left\| \begin{matrix} 2 \\ a^{k+1} + a^k - c^k \end{matrix} \right\|_2 \leq a^{k+1} + a^k + c^k. \tag{2.21}$$

The components in the left side of inequalities (2.19)–(2.21) form column vectors, and $\|\cdot\|_2$ represents the Euclidean norm of those vectors.

Finally, the SOCP reformulation of (2.14) is given as:

$$\underset{\substack{z^0,\ldots,z^K \\ \boldsymbol{u}_s^0,\ldots,\boldsymbol{u}_s^{K-1} \\ \nu^0,\ldots,\nu^{K-1} \\ a^0,\ldots,a^K \\ b^0,\ldots,b^{K-1} \\ c^0,\ldots,c^{K-1}}}{\text{minimize}} \quad \sum_{k=0}^{K-1} 2\left(s^{k+1}-s^k\right) b^k \tag{2.22a}$$

$$\text{subject to} \quad \boldsymbol{h}_s(s^k)\nu^k = \boldsymbol{f}_s(s^k)z^k + \boldsymbol{G}_s(s^k)\boldsymbol{u}_s^k, \tag{2.22b}$$

$$\underline{\boldsymbol{u}}_s \leq \boldsymbol{u}_s^k \leq \overline{\boldsymbol{u}}_s, \tag{2.22c}$$

$$0 \leq z^k \leq \overline{z}, \tag{2.22d}$$

$$\left\| \begin{matrix} 2a^k \\ z^k - 1 \end{matrix} \right\|_2 \leq z^k + 1, \tag{2.22e}$$

$$\text{for } k = 0, 1, \ldots, K,$$

$$z^{k+1} - z^k = 2\nu^k\left(s^{k+1}-s^k\right), \tag{2.22f}$$

$$\left\| \begin{matrix} 2u_{s,\text{lin}}^k \\ 2u_{s,\text{ang}}^k \\ a^{k+1} + a^k - b^k \end{matrix} \right\|_2 \leq a^{k+1} + a^k + b^k, \tag{2.22g}$$

$$\left\| \begin{matrix} 2 \\ a^{k+1} + a^k - c^k \end{matrix} \right\|_2 \leq a^{k+1} + a^k + c^k, \tag{2.22h}$$

$$\text{for } k = 0, 1, \ldots, K-1,$$

$$2\sum_{k=0}^{K-1}\left(s^{k+1}-s^k\right)c^k \leq T, \tag{2.22i}$$

$$z^0 = \dot{s}_0^2, \tag{2.22j}$$

$$z^K = \dot{s}_1^2 \tag{2.22k}$$

Some combinations of reference path, traversal time, and control bounds may render this problem infeasible. We assume the reference path is dynamically feasible and that the planner implementation will report if the problem is infeasible. To reduce computation

time, a practitioner may place a time-optimal module in front of our system to determine the minimum feasible traversal time.

2.4 Experiments

We evaluated our solution against three different path types that vehicles commonly encounter at intersections: a left turn, a right turn, and a straight path. These paths are represented as follows:

$$\tau_{\text{left}}(s) = \begin{bmatrix} \alpha_{\text{left}} \cos(\pi s/2) - \alpha_{\text{left}} \\ \alpha_{\text{left}} \sin(\pi s/2) \\ \arctan\big(\tau'_{\text{left},2}(s)/\tau'_{\text{left},1}(s)\big) \end{bmatrix}$$

$$\tau_{\text{right}}(s) = \begin{bmatrix} -\alpha_{\text{right}} \cos(\pi s/2) \\ \alpha_{\text{right}} \sin(\pi s/2) \\ \arctan\big(\tau'_{\text{right},2}(s)/\tau'_{\text{right},1}(s)\big) \end{bmatrix}$$

$$\tau_{\text{straight}}(s) = \begin{bmatrix} 1 \\ \alpha_{\text{straight}} s \\ \arctan\big(\tau'_{\text{straight},2}(s)/\tau'_{\text{straight},1}(s)\big) \end{bmatrix}$$

Parameters α_{left}, α_{right}, and α_{straight} represent the arc radius (for turns) and length (for straight paths). Table 2.1 shows the path parameters, traversal time ranges, and actuator limits used in our evaluations.

Our solution is implemented in Python using CasADi [73] to generate the path and its derivatives and CVXPY [74] to model and solve the SOCP problem. The code for our experiments is available in our lab's GitHub repository.[3]

For the collocation implementation, we performed a hyper-parameter sweep over $K \in [5, 200]$, the grid resolution, and found that $K = 20$ provided both reasonable approximation

[3] `https://github.com/the-hive-lab/trajectory_optimization`

Table 2.1: Simulation Parameters

Parameter	Min. Value	Max. Value	Step	Unit
α_{left} and α_{right}	5	15	1	m
α_{straight}	5	15	1	m
$\overline{\boldsymbol{u}}_{t,\text{lin}}$ and $\overline{\boldsymbol{u}}_{s,\text{lin}}$	-	2.5	-	m/s^2
$\underline{\boldsymbol{u}}_{t,\text{lin}}$ and $\underline{\boldsymbol{u}}_{s,\text{lin}}$	-2.5	-	-	m/s^2
$\overline{\boldsymbol{u}}_{t,\text{ang}}$ and $\overline{\boldsymbol{u}}_{s,\text{ang}}$	-	2.5	-	rad/s^2
$\underline{\boldsymbol{u}}_{t,\text{ang}}$ and $\underline{\boldsymbol{u}}_{s,\text{ang}}$	-2.5	-	-	rad/s^2
T	5	25	1	s

and solver stability.

We simulated the vehicle using a phase space variant of the second-order unicycle [39, eq. (13.46)]. The model's state vector $\boldsymbol{\xi} \in \mathbb{R}^5$ is $\boldsymbol{\xi} \coloneqq (x, y, \theta, v, \omega)$, and its state transition is defined by

$$\dot{x} = v \cos\theta$$

$$\dot{y} = v \sin\theta$$

$$\dot{\theta} = \omega$$

$$\dot{v} = u_{t,\text{lin}}$$

$$\dot{\omega} = u_{t,\text{ang}}.$$

After solving SOCP problem (2.22), the system control is approximated with a third-order spline using cubic interpolation. This interpolated control function is denoted $\hat{\boldsymbol{u}}_s \colon [0,1] \to \mathbb{R}^2$. We also approximated the transformed differential state using the same interpolation method and refer to the interpolant as $\hat{z} \colon [0,1] \to \mathbb{R}$.

The simulated robot was actuated using the control signal generated by the motion planner. We created a grid over s using $J+1$ points and integrated the system over the grid point intervals using Runge–Kutta fourth-order (RK4) integration. The control signal $\hat{\boldsymbol{u}}_s$ is held constant within each interval $[s^j, s^{j+1}]$, for $j = 0, 1, \ldots, J-1$.

Table 2.2: Traversal Time Error Metrics

Path	Mean Error	Standard Deviation	Runs
τ_{left}	6.7501×10^{-8}	8.6470×10^{-8}	231
τ_{right}	6.8975×10^{-8}	9.0152×10^{-8}	231
τ_{straight}	1.1068×10^{-7}	1.0731×10^{-7}	231
All	8.2594×10^{-8}	9.7135×10^{-8}	693

Each run is a combination of specific $\alpha_{\square}$ and T values.

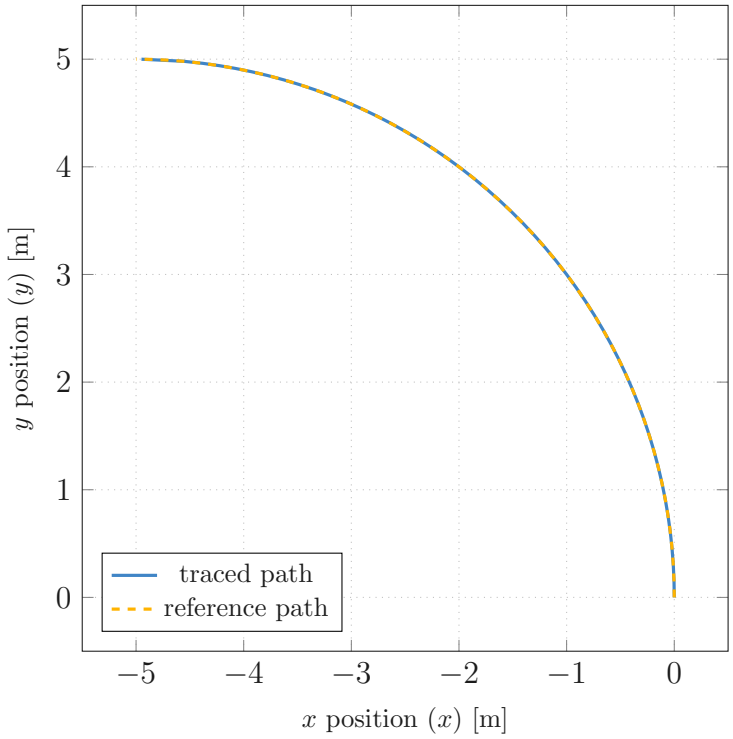


Figure 2.2: Simulated robot's path trace. The dashed gold line represents a left-turn reference path, and the solid blue line is the robot's path.

2.4.1 Experimental Results

Table 2.2 shows the mean traversal time error and standard deviation for all three path types over the different path parameters and traversal times. As shown in the table, the robot's traversal durations matched its assigned times and with minimal error.

Fig. 2.2 visualizes the simulated robot's trace for one of the test paths. Fig. 2.3 visualizes the system trajectory and control inputs for the same test path. Due to space

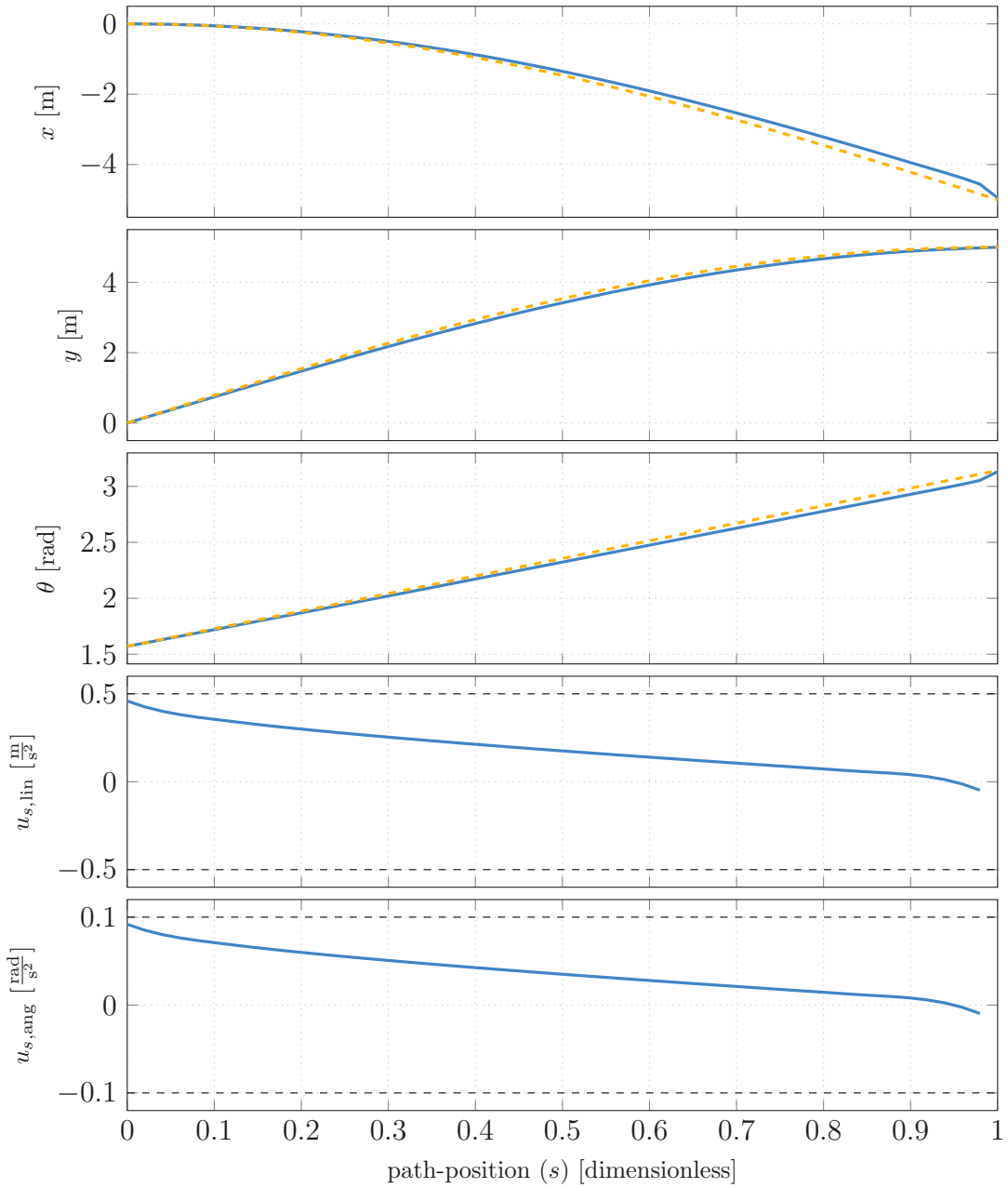


Figure 2.3: State trajectory from an example simulation. The dashed gold lines are the reference components of the path. The solid blue lines are the system's trace from the simulation.

constraints, we omit the scalar linear and angular velocity plots. The first three plots show a slight tracking error, which we believe is due to accumulated approximation errors in the simulation.

The experiment computer was equipped with an Intel Core i5-8250U processor and 16 GB

of memory. The solver runtime across all runs averaged 27.44 ms with a standard deviation of 51.86 ms, suggesting it could be performed in online.

2.5 Conclusion

In this chapter, we proposed a convex optimization solution to the control-minimal time-assigned path-constrained trajectory optimization problem. Using a nonlinear change of variables, the original problem formulation is converted into a convex problem then solved using trapezoidal collocation. From the experimental results, we conclude that our proposed method works as expected. The robot traversed the given paths in the specified time and tracked it with minimal error.

One limitation to our approach is that it may not extend to other motion models. Expressing the robot's dynamics in a form that was linear in the squared path-velocity and the path-acceleration was critical to this approach, but other motion models may not have this structure. Another limitation is that we assume a collision-free reference path. If an obstacle blocks the path, a higher-level path planner will have to find a new one. Fortunately, our runtime is low enough that this re-planning process could be performed online.

The trajectory optimizer is a key component in the intersection management system (IMS) that we will introduce in Ch. 3. As we explain in the next chapter, the IMS assigns crossing durations to each vehicle wanting to cross an intersection. Vehicles move along the centerline of an imaginary turn lane, which is their reference path, and they must plan a trajectory that traverses the path in exactly (or sufficiently close to) their assigned duration.

Chapter 3

Intersection Management

When multiple vehicles arrive simultaneously at an unsignalized intersection, the crossing order is ambiguous. This ambiguity causes a stalemate that persists until one driver decides to cross. Several intersection management systems (IMSes) exist to determine crossing orders based on a variety of policies. In auction-based IMSes, vehicles use monetary bids to gain crossing priority over others. However, monetary auctions may disenfranchise those that cannot afford to bid. We propose an alternative auction formulation where vehicles bid with cost functions instead of money. Our resulting IMS allocates crossing durations and determines crossing schedules based on passengers' preferences and their vehicles' capabilities. The IMS also allows system designers to impose constraints that alter duration and schedule assignment. We evaluated our system in a simulated environment and against several configuration variants. The results reveal an interesting trade-off between passenger preferences and intersection throughput.

3.1 Introduction

Intersections often congest road networks and have been shown to contribute to over twenty-eight percent of collisions in 2019 [75]. Unsignalized crossings can be particularly frustrating as simultaneous arrivals lead to ambiguous crossing orders. This ambiguity creates an awk-

ward standoff and delay until one driver signals their intent, e.g. slowly creeping their vehicle into the intersection.

Traffic signals have long been the solution to managing intersections, but improvements in vehicle-to-everything (V2X) communication technology have enabled researchers to develop more sophisticated management methods. By communicating directly with individual vehicles, new intersection management systems (IMSes) may use vehicle state information to safely and efficiently assign crossing orders before cars arrive at the intersection.

Existing IMSes regulate traffic using multiple approaches, but they primarily optimize for throughput or similar metrics [76], sometimes requiring vehicles to drive through the intersection at full speed [77]. These methods do not typically include the inherent preferences of drivers and passengers on different driving aspects. For example, drivers may prefer to travel under the speed limit while others typically exceed it.

Drivers and passengers also exhibit different preferences on how quickly to cross and intersection. They also sometimes demonstrate an awareness of improving social welfare by choosing to yield to others at an intersection. Alternatively, there are other drivers who behave aggressively by cutting other vehicles off to get through an intersection first.

In addition to these preferences, vehicle dynamics should be considered in IMSes. For example, sports cars may cross an intersection quickly, but semi-trucks have limited acceleration and require much more time. These limitations restrict passengers' preferences to what their vehicles are physically capable of achieving. Existing works typically assume vehicles have identical dynamics.

Current research efforts into intersection management focus on intersections with sparse traffic and vehicles traveling at speed. First come, first served scheduling algorithms work well under these conditions because the arrivals are staggered. However, when traffic reaches a critical density, it results in stop-and-go driving, and determining which vehicle arrived "first" becomes more difficult or impossible. Consider a large number of cars converging on a parking lot with one entrance or a road detour that ushers vehicles through a lower-capacity

auxiliary road.

This chapter proposes an auction-based IMS that optimizes for passenger preferences while also considering vehicle dynamics and other constraints. Vehicles submit, as their bid, cost functions for how quickly they want to cross the intersection and how soon. The IMS allocates a crossing duration interval to each vehicle. Each vehicle must cross the intersection within the amount of time they are assigned. For example, if the IMS assigns a five-second crossing duration, the vehicle must cross the intersection in five seconds. The system also schedules vehicles according to their preferences on how soon they want to cross. Vehicles' cost functions abstract their dynamics, allowing for a loose coupling with the IMS and promoting scalability.

Since our proposed system optimizes for passenger preferences, we expect there to be a trade-off between throughput and satisfaction. We hypothesize that our system will produce more favorable passenger outcomes (*i.e.*, preferred crossing durations and orders) at the expense of traffic throughput.

3.2 Related Work

Auction-based intersection management systems typically rely on real currency or virtual tokens to determine a crossing order. Vehicles place bids to gain crossing priority over others, and the auctioneer determines a winner based on the auction's rules. In [36], vehicles use real money in a sealed-bid, second-price auction to determine which vehicle crosses first; participants in [37, 38] used virtual tokens instead. Vehicles in those works had only a finite amount of currency, so the intersection management system subsidized bids for vehicles with insufficient funds. Variations to auction-based management systems include different payment systems: *all pay*, in which all participants pay their bid, and *winner pays*, in which only the winner pays their bid. Crossing variations can include letting only the winner cross (*i.e.* one crossing per auction) or letting all participants crossing in decreasing order of

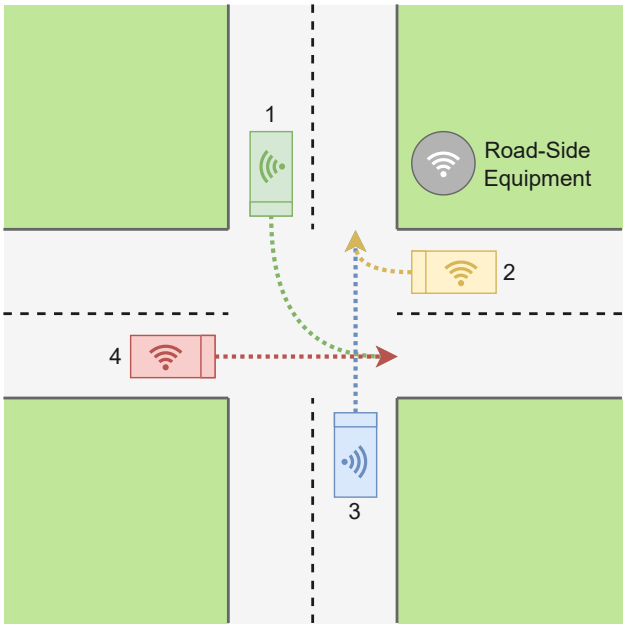


Figure 3.1: Four vehicles approach an intersection. The dotted lines indicate each vehicle's desired path, but they do not indicate when the vehicles will cross. Only one vehicle crosses the intersection at a time.

bids. Compared to direct control and scheduling based approaches, vehicles in auction-based systems cross the intersection in their preferred durations. However, a concern with monetary auctions is that intersection crossings are determined by economic status rather than vehicles' abilities. While there is no existing study on transportation economies, to the best of our knowledge, we suspect this might lead to wealth imbalances that disadvantage poorer bidders. Subsidies built into current management systems indicate that these imbalances are possible.

Existing IMSes typically use a variety of methods to optimize for efficiency metrics including throughput, travel delay, congestion [76]. However, few systems consider passenger preferences on when to cross the intersection. None, to the best of our knowledge, consider preferences on how quickly to cross. While traffic may be optimal with respect to throughput, it may be *suboptimal* with respect to passenger satisfaction. The bids in auction-based methods serve as a proxy for passengers' crossing order preferences. However, auction-based IMSes do not consider crossing duration preferences. Our proposed system resolves this shortcoming by including passenger preferences in the optimization process.

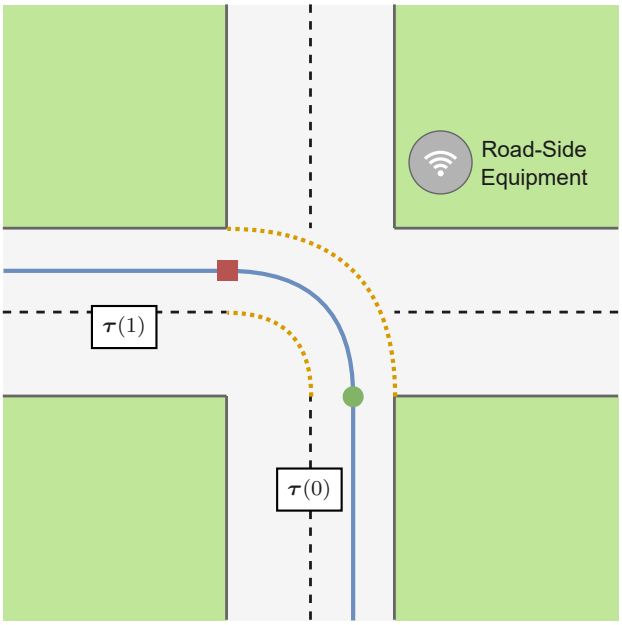


Figure 3.2: Imaginary intersection lane (bounded by orange dotted lines) and corresponding centerline (denoted by a solid blue line) for a left turn. The path starts at the green dot and ends at the red square. The blue lines beyond the red and green marks visualize other lanes' centerlines.

3.3 Problem Formulation

Our road network comprises M roads that lead to the same M-way, unsignalized intersection with a stop sign at each entrance. Additionally, N autonomous vehicles drive on the network, with disjoint subsets driving on each road. A management system installed at the intersection regulates the traffic and determines the crossing order. Fig. 3.1 visualizes an example scenario with a four-way intersection and four vehicles.

Vehicles approach the intersection simultaneously, or within some small finite time window $\epsilon \geq 0$, and the crossing order initially is undetermined. Like real-world intersections, vehicles may be perceived to have arrived simultaneously even if their exact arrival times differ. Let t_i and t_j be the arrival times of two different vehicles i and j, respectively. We consider the two vehicles to have arrived simultaneously if $|t_j - t_i| \leq \epsilon$, where ϵ is determined by the system designer.

All incoming vehicles must first stop at the intersection's entrance before crossing. This

allows pedestrians and other vulnerable road users time to cross the street. We define the *waiting duration* as the elapsed duration between when the vehicle stops at the intersection and when it begins crossing.

Definition 3.3.1 (Waiting Duration)**.** The waiting duration $\Delta^i_{\text{wait}} \coloneqq t^i_{\text{start}} - t^i_{\text{stop}}$ is the length of time (in seconds) that vehicle $i = 1, \ldots, N$ waits at the intersection before crossing. The vehicle stops at the intersection at time t^i_{stop} and begins crossing at time t^i_{start}. We assume both times are positive and that the start time is greater than or equal to the stop time. ■

3.3.1 Vehicle Motion

Each vehicle $i = 1, 2, \ldots, N$ moves according to a dynamics model $\boldsymbol{f}^i \colon \mathbb{R}^{p^i} \times \mathcal{C}^i \times \mathcal{U}^i \to \mathbb{R}^{p^i}$ defined by

$$\ddot{\boldsymbol{q}}^i(t) = \boldsymbol{f}^i\big(\dot{\boldsymbol{q}}^i(t), \boldsymbol{q}^i(t), \boldsymbol{u}^i(t)\big), \tag{3.1}$$

where $\boldsymbol{q}^i(t) \in \mathcal{C}^i$ and $\boldsymbol{u}^i(t) \in \mathcal{U}^i$ are the model's configuration and control vectors, respectively, at time $t \in \mathbb{R}_{\geq 0}$. Notations $\dot{\square}$ and $\ddot{\square}$ indicate the first and second time derivatives, respectively. Sets $\mathcal{C}^i$ and $\mathcal{U}^i$ are the vehicle's configuration and control spaces, respectively. We represent the set of nonnegative real numbers with $\mathbb{R}_{\geq 0}$. The particular motion model may differ among vehicles, but we assume all models are time invariant. The configurations and controls are functions of time, but we drop the explicit notation in the rest of the paper to preserve clarity.

A vehicle can cross the intersection within different time durations based on the its dynamics. For example, sports cars may cross quickly, but semi-trucks with fully loaded trailers would require significantly more time. We define a *crossing duration* as the length of time a vehicle takes to cross the intersection. A crossing duration interval contains a vehicle's feasible crossing durations.

Definition 3.3.2 (Crossing Duration)**.** The crossing duration $\Delta^i_{\text{cross}} \in \mathcal{D}^i_{\text{cross}}$ is the length of time (in seconds) that vehicle $i = 1, \ldots, N$ takes to cross the intersection. Interval

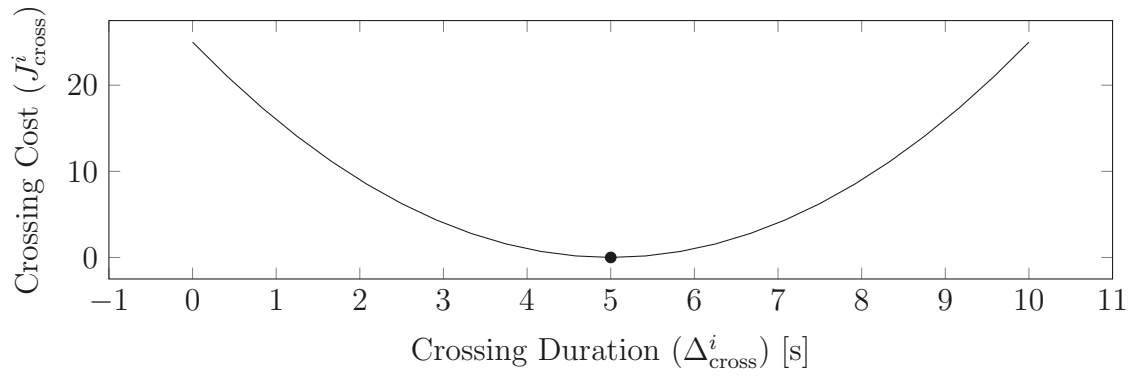
$\mathcal{D}^i_{\text{cross}} \coloneqq [\underline{\Delta^i_{\text{cross}}}, \overline{\Delta^i_{\text{cross}}}] \subseteq \mathbb{R}_{>0}$ contains the dynamically feasible crossing durations for vehicle i, where $\underline{\Delta^i_{\text{cross}}}$ and $\overline{\Delta^i_{\text{cross}}}$ are the lower and upper bounds, respectively. Set $\mathbb{R}_{>0}$ contains the positive reals. We assume $\underline{\Delta^i_{\text{cross}}}$ is positive and less than or equal to $\overline{\Delta^i_{\text{cross}}}$. ■

When crossing the intersection, each vehicle follows a continuous reference path $\boldsymbol{\tau}^i \colon [0, 1] \to \mathcal{C}^i$ beginning at $\boldsymbol{\tau}(0)$ and ending at $\boldsymbol{\tau}(1)$. On road networks, imaginary lanes within the each intersection connect entrances to exits and those lanes' centerlines serve as the reference paths. Fig. 3.2 illustrates the reference path for a left turn. The orange dotted lines visualize the imaginary lane's boundaries, and the blue solid line is the lane's centerline. The path begins at the green dot and ends at the red square. Road lane graphs generated from road maps contain these desired driving paths [28]. In real-world applications, driving automation systems use mapping libraries, such as `lanelet2` [78], to access a lane's centerline.

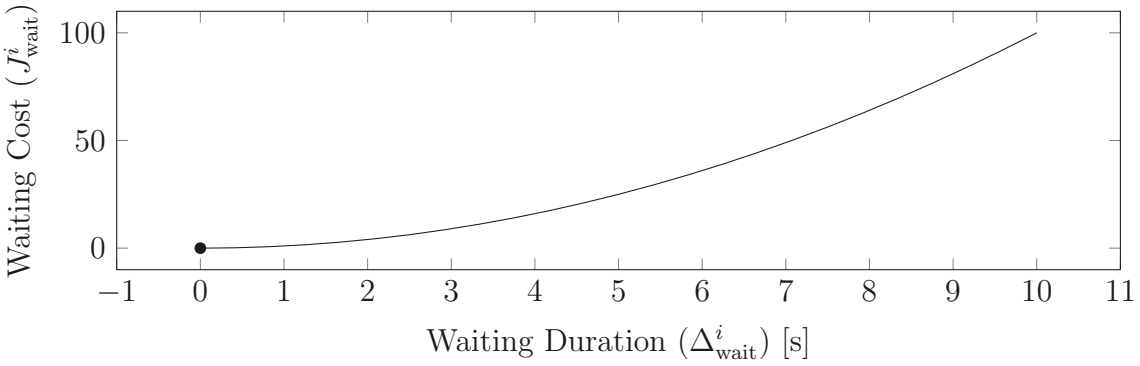
Using a reference path, vehicles' motion planners determine the lower crossing duration bound by solving a time-optimal path parameterization problem [62]. Alternatively, the motion planners could conduct a binary search for a lower crossing duration using trajectory optimization method described in Ch. 2. The upper crossing duration is based on the vehicle's speed when it is in gear without input into the accelerator. For practical purposes, we assume the upper duration bound is lower than this "coasting duration" because no reasonable driver would coast in-gear through an intersection.

3.3.2 Passenger Preferences

Each vehicle's passengers want to cross the intersection in a specific crossing duration, Δ^i_{cross}. We represent passengers' preferences over crossing durations with a *crossing cost* function. Fig. 3.3a plots a quadratic crossing cost where five seconds is the desired crossing duration. As illustrated in the figure, the vehicle may physically be able to cross the intersection slower or faster, but those times are less desirable. For example, some passengers get motion sick if the car moves too quickly. However, they do not want to spend an unreasonable amount of time crossing because that would unnecessarily delay their overall trip. Individuals may



a



b

Figure 3.3: a An example crossing cost function with a quadratic structure. b An example waiting cost function with a nonlinear increasing structure. For both functions, a dot indicates the desired duration.

have different preferences, but we assume each vehicle's crossing cost function aggregates its passengers' preferences (*i.e.*, one crossing cost function per vehicle).

Definition 3.3.3 (Crossing Cost)**.** The crossing cost $J^i_{\text{cross}} \colon \mathcal{D}^i_{\text{cross}} \to \mathbb{R}$ for vehicle $i = 1, 2, \ldots, N$ is the cost associated with crossing the intersection in $\Delta^i_{\text{cross}} \in \mathcal{D}^i_{\text{cross}}$ seconds. ■

Passengers also have a collective preference over different waiting durations. We represent this preference with a *waiting cost* function. Fig. 3.3b plots a nonlinear increasing waiting cost function where zero seconds is the preferred waiting duration.

Definition 3.3.4 (Waiting Cost)**.** The waiting cost $J^i_{\text{wait}} \colon \mathbb{R}_{\geq 0} \to \mathbb{R}$ for vehicle $i = 1, 2, \ldots, N$ is the cost associated with waiting at the intersection for $\Delta^i_{\text{wait}} \in \mathbb{R}_{\geq 0}$ seconds before starting to cross. ■

3.3.3 Crossing Order

Vehicles cross the intersection in a sequence specified by a crossing order. Let $\mathcal{N} \coloneqq \{1, 2, \ldots, N\}$ be the set of vehicles on the road network. Let $\mathcal{L} \coloneqq \{1, 2, \ldots, L\} \subseteq \mathcal{N}$ denote the set of *lead vehicles* stopped at the intersection. Permutation $\pi \in \Pi_{\mathcal{L}}$ over the set of lead vehicles represents a possible order. We define $\pi \colon \mathcal{L} \to \mathcal{L}$ using Cauchy's two-line notation

$$\begin{pmatrix} 1 & 2 & \cdots & L \\ \pi(1) & \pi(2) & \cdots & \pi(L) \end{pmatrix}, \tag{3.2}$$

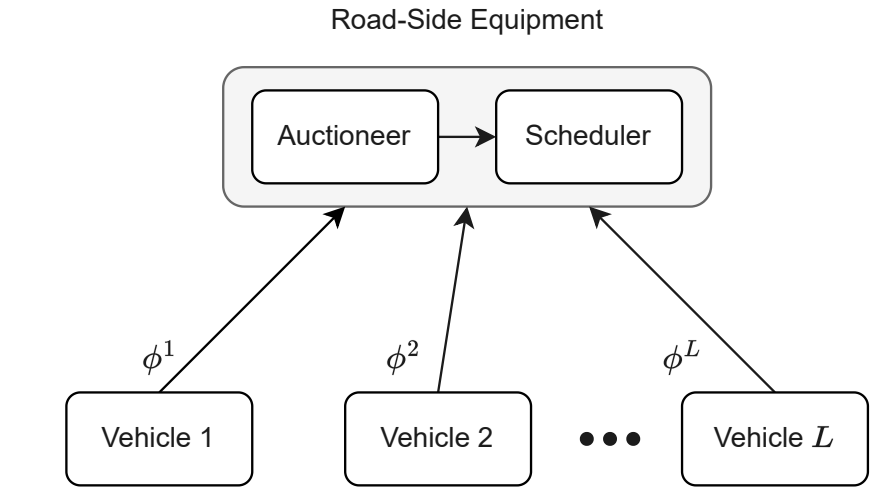
where $\pi(i)$ is the i^{th} vehicle to cross the intersection under the permutation. Set $\Pi_{\mathcal{L}}$ contains all permutations over the set of lead vehicles.

Given the problem formulation, we seek crossing durations and a crossing order that minimize vehicles' crossing and waiting costs. We refer to such a cost-minimal vector of crossing durations and associated crossing order as a *socially-optimal* outcome.

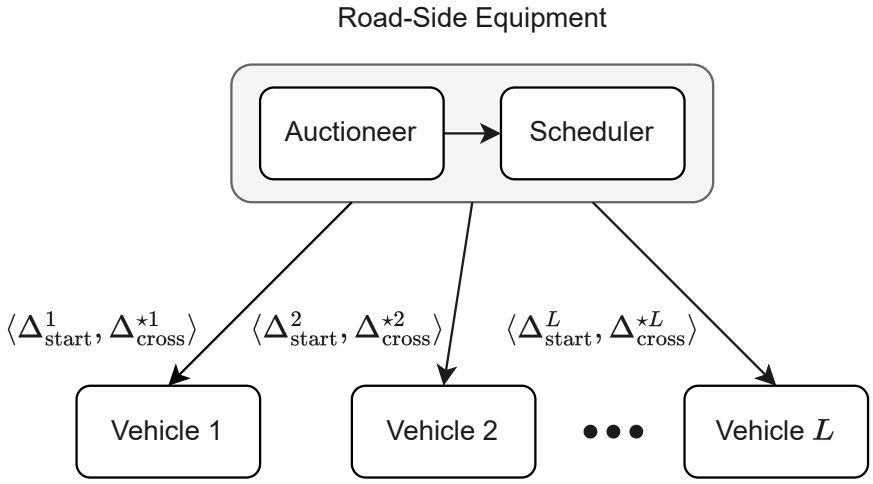
3.4 Socially-Optimal IMS Framework

We propose an auction-based intersection management system (IMS) to solve the problem defined in Section 3.3. The roadside equipment (RSE), shown in Fig. 3.1, serves as the auctioneer. It receives bids and assigns crossing durations and schedules to all participating vehicles. Our current work assumes that the intersection has fixed roadside infrastructure; however, the participating vehicles alternatively could elect one of themselves to serve as the auctioneer. We defer this election process to future research.

As opposed to other auction-based IMSes, vehicles in our system bid using cost functions instead of money or tokens. Each vehicle submits a crossing cost function representing their willingness to cross the intersection in specific durations. They also submit a waiting cost function representing their willingness to wait at the intersection for specific durations before crossing.



a



b

Figure 3.4: a Vehicles transmit their bids as they near the intersection. b The IMS transmits each vehicle's optimal starting time and crossing duration.

We assume only one vehicle crosses the intersection at a time, and only the lead vehicles for each road participate in the auction. Fig. 3.4 visualizes the dataflow between vehicles and the IMS.

Our goal is to determine a crossing duration for each vehicle and a crossing order that maximizes the social welfare for all vehicles crossing the intersection. We use a utilitarian social cost function, which is the summed cost (negative utility) for each vehicle. A utilitarian social cost function sums the (possibly weighted) costs of all agents, meaning changes the

one agent's cost will affect the overall cost. Minimizing this social cost function given any constraints on crossing durations or the order produces a socially-optimal solution [79, Sec. 22.C].

3.4.1 Auction Mechanism

We based our solution on a concept from economics and game theory called mechanism design [79, Ch. 23]. A mechanism is a collection of strategy sets and an outcome function. Auctions are one well known type of mechanism. The auctioneer allocates goods to vehicles based on its outcome function, and vehicles place bids from their strategy sets to win those goods.

As opposed to other auction-based IMSs, where agents bid using money, the vehicles in our system bid using crossing and waiting cost functions. Vehicles calculate their cost curves for different crossing and waiting durations then submit them as their bids. We define vehicle i's bid ϕ^i by the tuple

$$\phi^i \coloneqq \langle J^i_{\text{cross}}, J^i_{\text{wait}} \rangle. \tag{3.3}$$

The IMS uses this information to determine an appropriate crossing duration for each vehicle and a crossing schedule. Algorithm 3.1 provides an overview of the algorithms used by the IMS and vehicles.

Algorithm 3.1 IMS overview

for all $i \in \mathcal{L}$ **do** ▷ Vehicle i
  $\underline{\Delta^i_{\text{cross}}} \leftarrow$ solve minimum duration problem
  $\mathcal{D}^i_{\text{cross}} \leftarrow [\underline{\Delta^i_{\text{cross}}}, \overline{\Delta^i_{\text{cross}}}]$
  $\phi^i \leftarrow \langle J_{\text{cross}}, J_{\text{wait}} \rangle$
  Submit ϕ^i as bid
  Wait for $\langle t^{\pi^\star(i)}_{\text{start}}, \Delta^{\star\pi^\star(i)}_{\text{cross}} \rangle$
  $\sigma^\star \leftarrow$ solve (3.14)
  Track $\boldsymbol{\tau}^i \circ \sigma^\star$
end for

Wait for all ϕ^i, $i = 1, 2, \ldots, L$ ▷ Auctioneer
$\boldsymbol{\Delta}^\star_{\text{cross}} \leftarrow$ solve (3.5)

$\pi^\star \leftarrow$ solve (3.7) ▷ Scheduler
$t^{\pi^\star(0)}_{\text{start}} \leftarrow 0$
$\Delta^{\star\pi^\star(0)}_{\text{cross}} \leftarrow 0$
for all $i = 1, \ldots, L$ **do**
  $t^{\pi^\star(i)}_{\text{start}} \leftarrow t^{\pi^\star(i-1)}_{\text{start}} + \Delta^{\star\pi^\star(i-1)}_{\text{cross}}$
  Transmit $\langle t^{\pi^\star(i)}_{\text{start}}, \Delta^{\star\pi^\star(i)}_{\text{cross}} \rangle$ to vehicle i
end for

We want the auction to be *ex post* (or Pareto) efficient, meaning its outcome is Pareto optimal (efficient) given the agents' utility functions. An outcome is Pareto optimal if no agent's utility can be improved without reducing another agent's utility.

Vehicles' bids represent and abstract private information that only they can observe, meaning auctions can be susceptible to strategic agents. To avoid this, auctions can be designed to incentivize agents to bid truthfully, a property call *incentive compatibility*. A sealed-bid, second-price auction (or Vickrey auction) is an example of an incentive compatible mechanism. For this work, we assume agents bid truthfully. Future work will analyze our auction formulation for incentive compatibility and modify it if necessary.

To demonstrate that our auction formulation is *ex post* efficient, we need to show that the crossing duration allocations and crossing schedule are Pareto optimal. As briefly mentioned earlier, optimizing a utilitarian social welfare (cost) function results in a Pareto optimal outcome.

Authors in [80] proposed a framework for comparing different market-based coordination methods for distributed energy systems. We use the same comparison framework to summarize our system:

- **Agent preference:** Vehicle preferences (objectives) are defined by Definitions 3.3.3 and 3.3.4, and auctioneer preferences (objectives) are defined by (3.4).
- **Control decision:** For vehicle i, a trajectory $(\boldsymbol{x}^{\star i}, \boldsymbol{u}^{\star i})$ that minimizes J^i_{cross} for a given Δ^i_{cross}. For the IMS, a vector of optimal crossing durations $\boldsymbol{\Delta}^\star_{\text{cross}}$ and an optimal crossing schedule $\pi^\star$.
- **Information structure:** Type independence among agents; Decision dependence from auctioneer to vehicles.
- **Solution concept:** Auction-based optimization problem.

3.4.2 Auctioneer

The auctioneer seeks to maximize social utility by assigning crossing durations that best satisfy everyone's preferences. After receiving vehicles' bids, the auctioneer assigns crossing durations according to vehicles' crossing cost functions. The auctioneer may consider other constraints or costs that affect crossing duration assignment. For example, we could impose an upper limit on the intersection clearing time (the time it takes for all lead vehicles to cross) to ensure all vehicles cross in a timely manner. We formalize the auctioneer's objective with the auction cost function:

$$J_{\text{auction}}(\boldsymbol{\Delta}_{\text{cross}}) \coloneqq \sum_{i=1}^{L} J^i_{\text{cross}}(\Delta^i_{\text{cross}}) + J_{\text{other}}(\boldsymbol{\Delta}_{\text{cross}}), \tag{3.4}$$

where vector $\boldsymbol{\Delta}_{\text{cross}} \coloneqq (\Delta^1_{\text{cross}}, \Delta^2_{\text{cross}}, \ldots, \Delta^L_{\text{cross}})$ contains each vehicle's crossing duration.

The auctioneer assigns crossing durations by solving the following optimization problem:

$$\underset{\boldsymbol{\Delta}_{\text{cross}}}{\text{minimize}} \quad J_{\text{auction}}(\boldsymbol{\Delta}_{\text{cross}}) \tag{3.5a}$$

$$\text{subject to} \quad \Delta^i_{\text{cross}} \in \mathcal{D}^i_{\text{cross}} \quad i = 1, \ldots, L, \tag{3.5b}$$

$$\boldsymbol{g}(\boldsymbol{\Delta}_{\text{cross}}) = \mathbf{0}, \tag{3.5c}$$

$$\boldsymbol{h}(\boldsymbol{\Delta}_{\text{cross}}) \leq \mathbf{0} \tag{3.5d}$$

Function $J_{\text{other}} \colon \mathcal{D}^1_{\text{cross}} \times \mathcal{D}^2_{\text{cross}} \times \cdots \times \mathcal{D}^L_{\text{cross}} \to \mathbb{R}$ represents other objectives or soft constraints that a system designer may want to impose. If there are none, J_{cross} equals zero. Constraints (3.5c) and (3.5d) are hard constraints that a system designer may want to impose, such as an upper bound on the intersection clearing time. It is the system designer's responsibility to ensure the additional constraints do not make the problem infeasible.

The auctioneer determines a crossing duration allocation $\boldsymbol{\Delta}^\star_{\text{cross}} \in \mathcal{D}^1_{\text{cross}} \times \mathcal{D}^2_{\text{cross}} \times \cdots \times \mathcal{D}^L_{\text{cross}}$ that minimizes the utilitarian social cost function subject to any constraints, thus producing a Pareto optimal outcome. If the outcome was not Pareto optimal, there would exist another allocation benefiting all agents and further reducing the auction cost, thus creating a contradiction. Note that because (3.5) is a nonlinear optimization problem, our approach guarantees a *local* Pareto efficiency.

3.4.3 Scheduler

After the auctioneer assigns optimal crossing durations to each vehicle, the scheduler determines the crossing order. We schedule vehicles in an order that maximizes social welfare with respect to each vehicle's waiting cost function, *i.e.*, it minimizes them. We call this a *socially-optimal* scheduling algorithm.

A socially-optimal crossing order (permutation) $\pi^\star \in \Pi_{\mathcal{L}}$ minimizes the vehicles' summed waiting costs, which we term the *schedule cost*. The schedule cost is a functional $J_{\text{schedule}} \colon \Pi_{\mathcal{L}} \to$

$\mathbb{R}$ that we define by

$$J_{\text{schedule}}(\pi) \coloneqq \sum_{i=1}^{L} J_{\text{wait}}^{\pi(i)}\left(t_{\text{start}}^{\pi(i)}\right) \tag{3.6}$$

The scheduler solves the following optimization problem to determine the crossing schedule:

$$\begin{aligned}
\min_{\pi \in \Pi_{\mathcal{L}}} \quad & J_{\text{schedule}}(\pi) && \text{(3.7a)}\\
\text{s.t.} \quad & t_{\text{start}}^{\pi(i)} = t_{\text{start}}^{\pi(i-1)} + \Delta_{\text{cross}}^{\star\pi(i-1)} \quad i = 1, \ldots, L, && \text{(3.7b)}\\
& t_{\text{start}}^{\pi(0)} = 0, && \text{(3.7c)}\\
& \Delta_{\text{cross}}^{\star\pi(0)} = 0 && \text{(3.7d)}
\end{aligned}$$

Algorithm 3.2 describes an implementation of the socially-optimal scheduling algorithm. The implementation is globally Pareto optimal because it exhaustively searches the set of permutations. If the set contains several schedules with equal cost, the algorithm will return the first on it encountered.

While Algorithm 3.2 has a runtime complexity of $\mathcal{O}(LL!)$, the number of vehicles per auction and scheduling iteration realistically is low enough to maintain tractability. Additionally, a parallelized implementation could evaluate permutations concurrently to reduce computation time.

Algorithm 3.2 Socially optimal scheduling

1: $J^{\star}_{\text{schedule}} \leftarrow \infty$
2: $\pi^{\star} \leftarrow$ undefined
3: **for all** $\pi \in \Pi_{\mathcal{L}}$ **do**
4: $\quad t^{\pi(0)}_{\text{start}} \leftarrow 0$
5: $\quad t^{\pi(0)}_{\text{cross}} \leftarrow 0$
6: $\quad$ **for all** $i = 1, \ldots, L$ **do**
7: $\quad\quad t^{\pi(i)}_{\text{start}} \leftarrow t^{\pi(i-1)}_{\text{start}} + \Delta^{\pi(i-1)}_{\text{cross}}$
8: $\quad$ **end for**
9: $\quad J_{\text{schedule}} \leftarrow \sum_{i=1}^{L} c^{\pi(i)}_{\text{wait}}\left(t^{\pi(i)}_{\text{start}}\right)$
10: $\quad$ **if** $J_{\text{schedule}} < J^{\star}_{\text{schedule}}$ **then**
11: $\quad\quad J^{\star}_{\text{schedule}} \leftarrow J_{\text{schedule}}$
12: $\quad\quad \pi^{\star} \leftarrow \pi$
13: $\quad$ **end if**
14: **end for**
15: **return** $\pi^{\star}$

After generating an optimal crossing schedule $\pi^{\star}$, the scheduler assigns a start time $t^{\pi^{\star}(i)}_{\text{start}}$ to each vehicle. The start time for vehicle $\pi^{\star(i)}$ is the sum of the previous vehicle's start time and crossing duration:

$$t^{\pi^{\star}(i)}_{\text{start}} = \begin{cases} t^{\pi^{\star}(i-1)}_{\text{start}} + \Delta^{\pi^{\star}(i-1)}_{\text{cross}} & i = 2, \ldots, L \\ 0 & i = 1 \end{cases}. \tag{3.8}$$

The IMS then transmits to each vehicle their assigned start time and crossing duration as illustrated in Fig. 3.4b.

3.4.4 Combined Auctioneer and Scheduler

The proposed IMS design imposes an implicit hierarchy on the crossing and waiting cost functions. In other words, the system prioritizes minimizing crossing costs over waiting costs. As we show in the experimental results, this can cause high average waiting costs and thus high average total costs. Vehicles that prefer to cross the intersection slowly will contribute toward longer waiting durations. Longer waiting durations can be costly

depending on vehicle's waiting cost functions.

We therefore propose an alternative system design that combines the auctioneer and scheduler into a single entity. The resulting optimization problem determines crossing durations and start times simultaneously, minimizing the summed crossing and waiting costs. We combine problems (3.5) and (3.7) to form the combined problem

$$\min_{\boldsymbol{\Delta}_{\text{cross}},\, \pi} \quad J_{\text{auction}}(\boldsymbol{\Delta}_{\text{cross}}) + J_{\text{schedule}}(\pi) \tag{3.9a}$$

$$\text{s.t.} \quad \Delta^i_{\text{cross}} \in \mathcal{D}^i_{\text{cross}} \quad i = 1, \ldots, L, \tag{3.9b}$$

$$\boldsymbol{g}(\boldsymbol{\Delta}_{\text{cross}}) = \mathbf{0}, \tag{3.9c}$$

$$\boldsymbol{h}(\boldsymbol{\Delta}_{\text{cross}}) \leq \mathbf{0}, \tag{3.9d}$$

$$t^{\pi(i)}_{\text{start}} = t^{\pi(i-1)}_{\text{start}} + \Delta^{\pi(i-1)}_{\text{cross}} \quad i = 1, \ldots, L, \tag{3.9e}$$

$$t^{\pi(0)}_{\text{start}} = 0, \tag{3.9f}$$

$$\Delta^{\pi(0)}_{\text{cross}} = 0 \tag{3.9g}$$

Constraints (3.9b)–(3.9d) are equivalent to those in problem (3.5), and constraints (3.9e)–(3.9g) are equivalent to those in problem (3.7).

Algorithm 3.3 implements problem (3.9). For each permutation (crossing order), the algorithm assigns cost-minimal crossing durations. The algorithm returns the lowest-cost permutation and associated crossing durations. Note that the combined auctioneer and scheduler can provide only local Pareto optimality.

Algorithm 3.3 Combined System

$J^\star_{\text{schedule}} \leftarrow \infty$
$\pi^\star \leftarrow$ undefined
$\boldsymbol{\Delta}_{\text{cross}} \leftarrow$ undefined
for all $\pi \in \Pi_{\mathcal{L}}$ **do**
    $J_{\text{schedule}}, \pi, \boldsymbol{\Delta}_{\text{cross}} \leftarrow$ solve (3.9)
    if $J_{\text{schedule}} < J^\star_{\text{schedule}}$ **then**
        $J^\star_{\text{schedule}} \leftarrow J_{\text{schedule}}$
        $\pi^\star \leftarrow \pi$
        $\boldsymbol{\Delta}^\star_{\text{cross}} \leftarrow \boldsymbol{\Delta}_{\text{cross}}$
    end if
end for
return $\pi^\star, \boldsymbol{\Delta}^\star_{\text{cross}}$

3.5 Trajectory Optimization

Once vehicles receive their assigned crossing duration and start time, they need to plan their motion. Initially, we used a multi-objective optimization approach where vehicles tried to minimize both reference tracking deviations and control effort. This structure resembled model predictive control (MPC) with a planning horizon equal to the crossing duration.

As we discuss in the next section, this approach is susceptible to large tracking errors when the IMS assigns short crossing durations. In the subsequent section, we recapitulate the path-constrained trajectory optimizer from Ch. 2, which performed significantly better than our original method. While the MPC-style approach generated undesirable motion plans, we discuss it here because the work was a key motivator for trajectory optimizer.

3.5.1 Multi-Objective Optimization

Given the assigned crossing duration Δ^i_{cross}, we discretize the planning horizon into a sequence of stages $k = 0, 1, \ldots, N(\Delta^i_{\text{cross}})$, where function $N \colon \mathcal{D}^i_{\text{cross}} \to \mathbb{Z}_{>0}$ is defined as

$$N(\Delta^i_{\text{cross}}) \triangleq \left\lceil \frac{\Delta^i_{\text{cross}}}{\delta^i} \right\rceil.$$

Parameter $\delta^i \in \mathbb{R}_{>0}$ controls the discretization sampling period. The intuition is that the planning horizon becomes longer as the vehicle has more time to cross the intersection (*i.e.*, a larger Δ^i_{cross}).

We define the discretized vehicle dynamics as

$$\boldsymbol{x}^i_{k+1} = \boldsymbol{f}^i_d(\boldsymbol{x}^i_k, \boldsymbol{u}^i_k) \qquad \text{for } k = 0, 1, \ldots, N(\Delta^i_{\text{cross}}),$$

where $\boldsymbol{x}^i_k$ and $\boldsymbol{u}^i_k$ are the system's state and input vectors, respectively, at stage k.

Each vehicle tracks a reference path $\boldsymbol{\tau}$, such as a turning arc or straight line, as they drive through the intersection. The reference path is the lane's center line, which is provided to the planner (*e.g.*, through a perception system). Let $\boldsymbol{r}^i \colon \mathbb{R}_{\geq 0} \to \mathcal{X}^i$ be a time-varying reference function (trajectory) defined as

$$\boldsymbol{r}^i(t) \triangleq \boldsymbol{\tau}^i\left(\frac{t}{\Delta^i_{\text{cross}}}\right).$$

We denote the discretized version as

$$\boldsymbol{r}^i_k \triangleq \boldsymbol{r}^i(k\delta^i) \qquad \text{for } k = 0, 1, \ldots, N(\Delta^i_{\text{cross}}).$$

The motion planning problem is then defined as

$$\begin{aligned}
\underset{\substack{\boldsymbol{x}^1,\ldots,\boldsymbol{x}^{N(\Delta^i_{\text{cross}})} \\ \boldsymbol{u}^1,\ldots,\boldsymbol{u}^{N(\Delta^i_{\text{cross}})}}}{\text{minimize}} \quad & \sum_{k=1}^{N(\Delta^i_{\text{cross}})} \left(\|\boldsymbol{x}^i_k - \boldsymbol{r}^i_k\|_{\boldsymbol{Q}} + \|\boldsymbol{u}^i_k\|_{\boldsymbol{R}} \right) \\
\text{subject to} \quad & \boldsymbol{x}^i_{k+1} = \boldsymbol{f}^i_d\left(\boldsymbol{x}^i_k, \boldsymbol{u}^i_k\right), \\
& \underline{\boldsymbol{x}}^i_k \leq \boldsymbol{x}^i_k \leq \overline{\boldsymbol{x}}^i_k, \\
& \underline{\boldsymbol{u}}^i_k \leq \boldsymbol{u}^i_k \leq \overline{\boldsymbol{u}}^i_k
\end{aligned}$$

where matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ weight the reference tracking and control effort objectives, respec-

tively.

One issue with this approach, and multi-objective optimization in general, is the trade-off between objectives. Prioritizing the control effort cost will cause the vehicle to significantly deviate from the reference path, opting to travel in a straight line. Focusing on path tracking can lead to unnecessarily large control inputs. Finding the proper balance between objectives is difficult, and different crossing durations require different weightings.

The other issue is the implicit time scaling for the reference trajectory. It assumes the vehicle makes uniform progress along the reference path. However, this is unrealistic as the vehicle will make little progress at the path's beginning and end. Vehicles need time to accelerate and decelerate.

Ideally, we want the minimum control effort required to track the reference path. Therefore, we need the path-constrained trajectory optimizer presented in Ch. 2, which we briefly review in the next section.

3.5.2 Path-Constrained Motion Planning

This task follows the same idea as the time optimal path parameterization problem mentioned earlier, but now we want the vehicle to traverse the path in a specific amount of time. We summarize the work presented in Ch. 2, modifying the notation slightly to fit within the intersection management context.

Given an assigned duration $\Delta^{\star i}_{\text{cross}}$, we want the vehicle's configuration to be $\boldsymbol{\tau}(0)$ when it begins crossing the intersection. We want its configuration at the end of its traversal $(t^i_{\text{start}} + \Delta^{\star i}_{\text{cross}})$ to be $\boldsymbol{\tau}(1)$.

The reference path is a purely geometric concept, as it does not convey any time-related information. Therefore, we need a time scaling $\sigma^i \colon [0, \Delta^{\star i}_{\text{cross}}] \to [0, 1]$ that maps times $t_{\text{cross}} \in [0, \Delta^{\star i}_{\text{cross}}]$ to path-positions $s \in [0, 1]$. The time scaling describes how the vehicle moves along the path as a function of time. We require $\sigma^i(t^i_{\text{start}}) = 0$ and $\sigma^i(t^i_{\text{start}} + \Delta^{\star i}_{\text{cross}}) = 1$ because we want the vehicle to start and stop at the path's boundaries. As with (3.1) (the

vehicle's dynamics model), we use the short-hand notation $\sigma^i \coloneqq \sigma^i(t)$ to preserve clarity.

To plan the vehicle's motion, we consider only configurations that lie along this path. We express each vehicle's configuration in terms of its path-position [39, Sec. 14.6.3]:

$$\boldsymbol{q}^i \coloneqq \boldsymbol{\tau}^i(\sigma^i). \tag{3.10}$$

We also relate the vehicle's velocity $\dot{\boldsymbol{q}}^i$ to the path by differentiating (3.10) with respect to time:

$$\dot{\boldsymbol{q}}^i \coloneqq \frac{\mathrm{d}\boldsymbol{\tau}^i}{\mathrm{d}\sigma^i}\frac{\mathrm{d}\sigma^i}{\mathrm{d}t} = \boldsymbol{\tau}'^i(\sigma^i)\dot{\sigma}^i, \tag{3.11}$$

where $\square'$ denotes the first derivative with respect to the path-position, and $\dot{\sigma}^i$ is the vehicle's *path-velocity*.

Furthermore, we express the vehicle's path-constrained acceleration by differentiating (3.11) with respect to time:

$$\ddot{\boldsymbol{q}}^i \coloneqq \frac{\mathrm{d}^2\boldsymbol{\tau}^i}{\mathrm{d}{\sigma^i}^2}\left[\dot{\sigma}^i\right]^2 + \frac{\mathrm{d}\boldsymbol{\tau}^i}{\mathrm{d}\sigma^i}\ddot{\sigma}^i = \boldsymbol{\tau}''^i(\sigma^i)\left[\dot{\sigma}^i\right]^2 + \boldsymbol{\tau}'^i(\sigma^i)\ddot{\sigma}^i, \tag{3.12}$$

where $\square''$ denotes the second derivative with respect to the path-position, and $\ddot{\sigma}^i$ is the vehicle's *path-acceleration*.

Substituting (3.10)–(3.12) into (3.1) , yields the path-constrained dynamics for vehicle i:

$$\boldsymbol{\tau}''^i(\sigma^i)\left[\dot{\sigma}^i\right]^2 + \boldsymbol{\tau}'^i(\sigma^i)\ddot{\sigma}^i = \boldsymbol{f}^i\big(\boldsymbol{\tau}^i(\sigma^i), \boldsymbol{\tau}'^i(\sigma^i)\dot{\sigma}^i, \boldsymbol{u}^i\big) \tag{3.13}$$

We assume that motion planners minimize control effort while satisfying the assigned

crossing duration, which informs the following optimization problem:

$$\min_{\sigma,\,\boldsymbol{u}} \quad \int_{t^i_{\text{start}}}^{t^i_{\text{start}}+\Delta^{\star i}_{\text{cross}}} \|\boldsymbol{u}\|_2^2 \,\mathrm{d}t \tag{3.14a}$$

$$\text{s.t.} \quad \boldsymbol{\tau}''^i(\sigma)\dot{\sigma}^2 + \boldsymbol{\tau}'^i(\sigma)\ddot{\sigma} = \boldsymbol{f}^i\big(\boldsymbol{\tau}^i(\sigma), \boldsymbol{\tau}'^i(\sigma)\dot{\sigma}, \boldsymbol{u}\big), \tag{3.14b}$$

$$\underline{\boldsymbol{u}}^i \leq \boldsymbol{u} \leq \overline{\boldsymbol{u}}^i, \tag{3.14c}$$

$$\sigma(t^i_{\text{start}}) = 0, \tag{3.14d}$$

$$\sigma(t^i_{\text{start}} + \Delta^{\star i}_{\text{cross}}) = 1 \tag{3.14e}$$

where $\|\cdot\|_2$ is the Euclidean norm, and $\underline{\boldsymbol{u}}^i$ and $\overline{\boldsymbol{u}}^i$ are the lower and upper vehicle control bounds, respectively, that depend on the vehicle's dynamics. In problem (3.14), the motion planner seeks a dynamically feasible time scaling $\sigma^\star$ and control signal $\boldsymbol{u}^\star$ that minimize the control effort. A system integrator could pass the resulting control signals directly to the vehicle. They could alternatively compose the path and time scaling to create a reference input for a trajectory tracking controller.

As in Ch. 2, we employed direct collocation and a nonlinear change of variables to convert problem (3.14) into a second-order cone program (SOCP). From there, we solved the program using a conventional SOCP solver.

3.6 Experiments

We conducted two types of experiments. In the initial version of this work, we did a numerical test to ensure our proposed IMS assigned desirable crossing durations and a crossing schedule. The expanded test evaluated our system against a continuous stream of vehicles with different dynamics and passenger preferences.

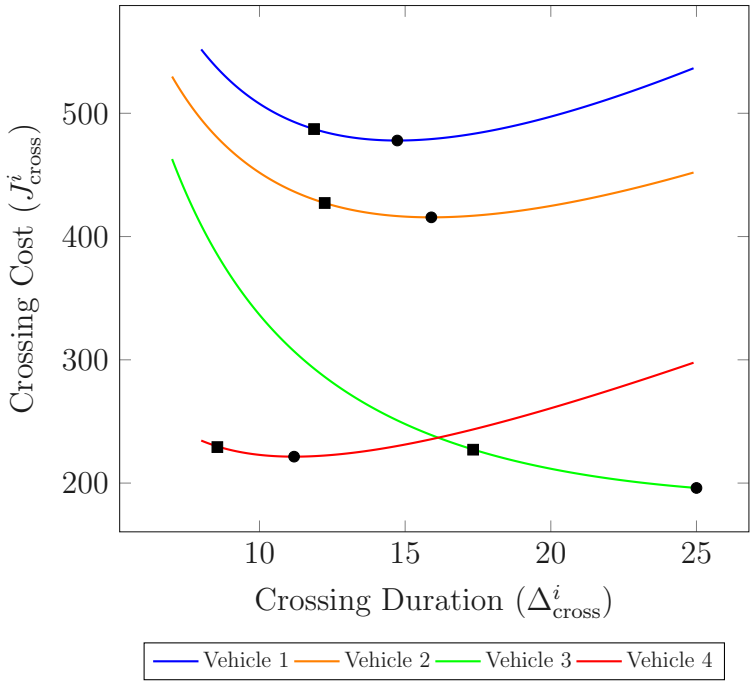


Figure 3.5: Crossing costs for each vehicle. Black circles are assigned durations without intersection constraints. Black squares are assigned durations with intersection constraint $T = 50$.

3.6.1 Numerical Experiments

As an initial test, we evaluated our IMS on the scenario shown in Fig. 3.1. We chose crossing cost functions that exemplify three passenger types: slow, average, and fast.

Fig. 3.5 plots the crossing costs for each vehicles' passengers, and it shows assigned crossing durations. Passengers in vehicles 1 (blue) and 2 (orange) were average while those in vehicle 3 (green) were slow. Vehicle 4's (red) passengers wanted to cross quickly. The black circles represent each vehicle's assigned traversal durations without any intersection-level constraints. We can see that in this case, all vehicles received their preferred durations. When we impose a clearing time constraint of 50 seconds to ensure all vehicles cross in a timely manner, the IMS assigns slightly faster durations, which increases vehicles' costs.

Fig. 3.6 visualizes the schedule costs for the vehicles when given random waiting cost functions. We choose a random crossing schedule as a baseline to imitate how vehicles non-deterministically break stalemates at the intersection. From the figure, we can see that

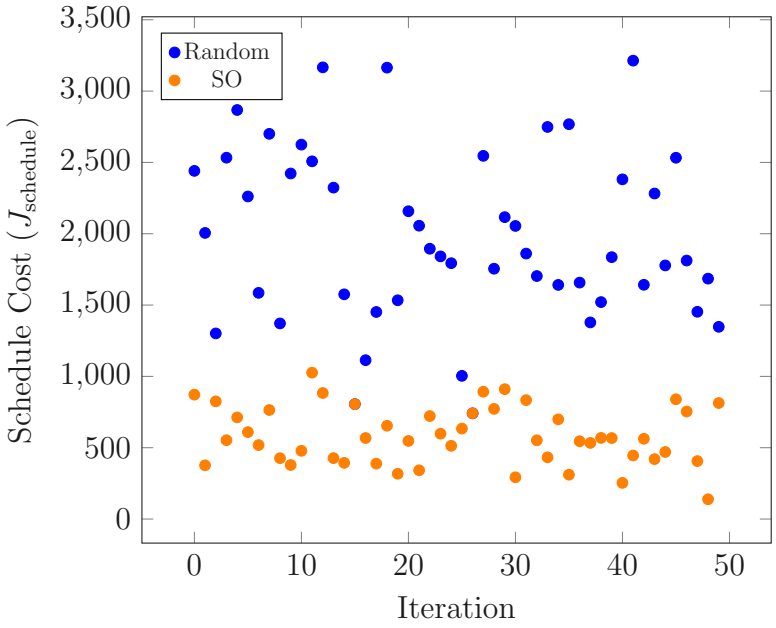


Figure 3.6: Schedule costs for random and socially-optimal schedules.

the socially-optimal (SO) scheduling algorithm assigns more preferable (or equivalently-preferable) crossing schedules in all instances.

3.6.2 Simulation Experiments

We evaluated our auction-based IMS on the four-way intersection shown in Fig. 3.7 that has lane width of 3.2 meters and an intersection width of 14.4 meters. Our system was developed in Python using CVXPY [74, 81] to solve the path-constrained trajectory optimization problems. We used CasADi [73] to represent the paths and their derivatives as well as to solve the auction problems. Our code is publicly available in our lab's GitHub repository[1].

We simulated the intersection environment using Eclipse Simulation of Urban MObility (SUMO) [82] and integrated it with our system using the SUMO Traffic Control Interface (TraCI). To better adhere to Python development best practices, we created a thin wrapper[2] around TraCI that provides a "Pythonic" interface.

During the simulation, SUMO added vehicles to the road network at a constant rate.

[1] `https://github.com/the-hive-lab/intersection_auctions`
[2] `https://github.com/the-hive-lab/pythonic_traci`

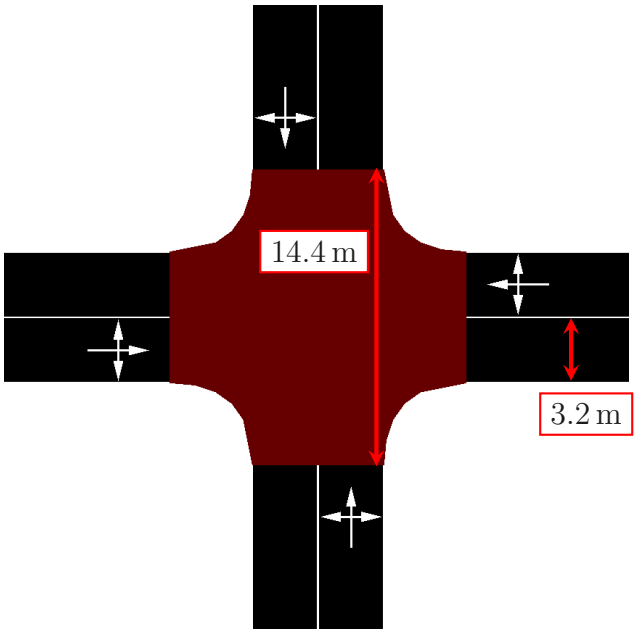


Figure 3.7: Screenshot of the intersection used in SUMO for the simulations.

Each vehicle had a randomly-determined type (slow, average, fast) with its own control bounds. Our software randomly selected crossing and waiting cost functions and turning directions from pre-specified lists.

Each vehicle stopped at the intersection, submitted its bid to the auctioneer, then waited for the auction results. Once the auctioneer received bids from all participants, it determined the best crossing durations and start times. Then it sent the results to each participating vehicle. Vehicles continued along their chosen route in their assigned order, crossing the intersection in their assigned duration. The stop-auction-move process repeated until all vehicles crossed the intersection. We simulated with ten vehicles on each lane (forty in total).

Table 3.1: Vehicle Control Bounds

Vehicle Type	Linear Acceleration	Angular Acceleration
Slow	3	10
Average	5	10
Fast	15	10

3.6.3 Setup

Each vehicle moved according to second-order unicycle dynamics. To derive this model, we started with the conventional single-order (or kinematic) unicycle [39, Eq. (3.18)]

$$\dot{x}^i(t) = v^i(t)\cos\big(\theta^i(t)\big) \tag{3.15a}$$

$$\dot{y}^i(t) = v^i(t)\sin\big(\theta^i(t)\big) \tag{3.15b}$$

$$\dot{\theta}^i(t) = \omega^i(t). \tag{3.15c}$$

For the rest of the derivation, we drop the explicit dependence on time in (3.15). The configuration vector $\boldsymbol{q}^i \coloneqq (x^i, y^i, \theta^i) \in \mathcal{C}_{\text{unicycle}}$ includes the vehicle's position (x^i and y^i) and heading θ^i. Set $\mathcal{C}_{\text{unicycle}}$ is the vehicle's configuration space. The system control vector $\boldsymbol{u} \coloneqq (v^i, \omega^i) \in \mathbb{R}^2$ comprises the scalar linear velocity v^i and scalar angular velocity ω^i.

Equation (3.1) is a second-order motion model, but system (3.15) describes a first-order one. We differentiate (3.15) to obtain the second-order kinematic unicycle

$$\ddot{x}^i = u^i_{\text{linear}}\cos\big(\theta^i\big) - \dot{\theta}^i\sin\big(\theta^i\big)\sqrt{(\dot{x}^i)^2 + (\dot{y}^i)^2} \tag{3.16a}$$

$$\ddot{y}^i = u^i_{\text{linear}}\sin\big(\theta^i\big) + \dot{\theta}^i\cos\big(\theta^i\big)\sqrt{(\dot{x}^i)^2 + (\dot{y}^i)^2} \tag{3.16b}$$

$$\ddot{\theta}^i = u^i_{\text{angular}}, \tag{3.16c}$$

where u^i_{linear} and u^i_{angular} are the scalar linear and angular accelerations, respectively.

We defined three different vehicle types: slow (*e.g.*, a semi-trailer truck), average, and fast (*e.g.*, a sports car). Each vehicle type had its own control bounds, shown in Table 3.1.

Vehicles entering the intersection could turn left, turn right, or go straight. All reference paths comprised a straight segment followed by either a turning segment or another straight segment. This ensured the rear axle was coincident with the intersection's start; otherwise, the vehicle would clip the sidewalk when turning. We represented the second segment for each reference path with a function $\boldsymbol{\tau}_{\{\text{left,right,straight}\}} \colon [0,1] \to \mathcal{C}_{\text{unicycle}}$ defined by

$$\boldsymbol{\tau}_{\text{left}}(s) = \begin{bmatrix} 8.8\cos\left(\frac{\pi}{2}s + \frac{3\pi}{2}\right) \\ 8.8\sin\left(\frac{\pi}{2}s + \frac{3\pi}{2}\right) + 8.8 \\ \arctan\left(\tau'_{\text{left},2}(s)/\tau'_{\text{left},1}(s)\right) \end{bmatrix} \tag{3.17}$$

$$\boldsymbol{\tau}_{\text{right}}(s) = \begin{bmatrix} 5.6\cos\left(\frac{\pi}{2}s + \frac{3\pi}{2}\right) \\ -5.6\sin\left(\frac{\pi}{2}s + \frac{3\pi}{2}\right) - 5.6 \\ \arctan\left(\tau'_{\text{right},2}(s)/\tau'_{\text{right},1}(s)\right) \end{bmatrix} \tag{3.18}$$

$$\boldsymbol{\tau}_{\text{straight}}(s) = \begin{bmatrix} 14.4s \\ 0 \\ \arctan\left(\tau'_{\text{straight},2}(s)/\tau'_{\text{straight},1}(s)\right) \end{bmatrix}. \tag{3.19}$$

Notations $\boldsymbol{\tau}'_{\square,1}$ and $\boldsymbol{\tau}'_{\square,2}$ denote the first and second components of the path's s-derivative.

These paths were implemented as offsets from the vehicle's starting orientation. This allowed us to use a single reference path implementation and then align it to individual vehicles' orientations when stopped. We approximated solutions to problem (3.14) with a direct collocation scheme and second-order cone program as described in Ch. 2 [83].

We chose crossing cost functions to model three types of (idealized) passengers: cautious, average, and aggressive. Cautious passengers preferred to cross the intersection slowly, so their cost decreased as the crossing duration increased. Average passengers avoided crossing too quickly or too slowly. Their function was a quadratic. Aggressive passengers preferred crossing as quickly as possible, preferring shorter durations. We chose these driver types for exposition, but other passenger types and functions could be used instead.

Like the crossing cost functions, we modeled three types of passengers: impatient, av-

erage, and patient. Impatient passengers preferred to start crossing as soon as possible. These are passengers that are in a hurry. Patient passengers wanted to start crossing at a later time. A real-world equivalent would be passengers that let others cross before them. Intuitively, this could be someone who is ahead of schedule, so they want to pad their trip with extra time. Average passengers wanted to cross somewhere in between. They do not mind waiting, but they do not want to wait too long.

3.6.4 IMS Configurations

We evaluated our IMS against different auctioneer and scheduler configurations to better understand how the two components individually affected performance. Our nominal system, denoted PO, assigned preferred crossing durations and socially optimal crossing schedules. Table 3.2 contains all configurations we used in the evaluations.

To simulate vehicles interacting at an unmanaged intersection, we set the auctioneer to assign preferred durations and set the scheduler to assign random orders. This provides a realistic baseline because, without external forces, vehicles will cross the intersection in however long they want. They will also break stalemates nondeterministically. Another configuration assigned preferred crossing durations and a fixed-order crossing schedule. This was equivalent to the right-most-first rule starting from a common reference point.

These two configurations served as baselines because they best represented how real vehicles would interact. Other configurations included assigning time-minimum crossing durations with random, fixed, and socially optimal schedules. Another imposed a clearing time constraint on the auctioneer. The final configuration replaced the individual auctioneer and scheduler subsystems with the combined one from Section 3.4.4

3.6.5 Constrained Auctioneer

To demonstrate our system's performance under intersection-level constraints, we imposed an upper bound on the intersection clearing time. The clearing time is the sum of crossing

durations. This ensured that all vehicles crossed the intersection in a timely manner, sacrificing passenger preferences for traffic throughput. We used this auction problem for the constrained, optimal (CO) IMS configuration.

In this configuration, we required all vehicles within an auction round to collectively cross the intersection within $T = 15$ seconds. Note that we could have use other values for T. The modified optimization problem for the auctioneer was

$$\underset{\boldsymbol{\Delta}_{\text{cross}}}{\text{minimize}} \quad \sum_{i=1}^{L} J^i_{\text{cross}}(\Delta^i_{\text{cross}}) + \gamma\alpha \tag{3.20a}$$

$$\text{subject to} \quad \Delta^i_{\text{cross}} \in \mathcal{D}^i_{\text{cross}} \quad i = 1, \ldots, L, \tag{3.20b}$$

$$\sum_{i=1}^{L} \Delta^i_{\text{cross}} \leq T + \alpha, \tag{3.20c}$$

$$\alpha \geq 0 \tag{3.20d}$$

To ensure the optimization problem was feasible, we used a slack variable α to make the clearing time (summed crossing durations) a soft constraint. We also added a positive penalty parameter γ to encourage the numerical solver to strictly adhere to the desired upper limit. In our experiments, we set γ equal to 100.

3.6.6 Evaluation Metrics

We compared the different configurations using average trip duration, average crossing cost, average waiting cost, and average total cost. The averages were over all vehicles in the simulation. The average trip duration indicated each configuration's ability to manage traffic flow. The average total cost summed the average crossing and waiting costs, providing insight on much passengers preferred their assigned outcomes. Using the average crossing and waiting costs individually, we could better understand how each subsystem influenced the overall costs.

Table 3.2: IMS Configuration and Performance Metrics

Cross. Duration	Cross. Order	Abbrev.	Avg. Trip Duration	Avg. Cross. Cost	Avg. Wait. Cost	Avg. Total Cost
Preferred	Random	PR	76.10	1.48	63.76	65.24
Preferred	Fixed	PF	73.29	1.65	62.30	63.95
Preferred	**Socially optimal**	**PO**	71.27	1.67	13.31	14.98
Min. duration	Random	MR	23.66	5.54	7.96	13.50
Min. duration	Fixed	MF	15.36	5.78	9.35	15.13
Min. duration	Socially optimal	MO	15.53	5.78	2.03	7.80
Constrained	Socially optimal	CO	41.83	1.79	4.96	6.75
Combined	Combined	Combo.	53.33	2.20	1.83	4.04

3.6.7 Discussion

Our experiments revealed that the system's performance depends on passenger preferences. As shown in Fig. 3.8, assigning time-minimal crossing durations with the MR, MF, and MO configurations increased passengers' crossing costs. However, assigning preferred durations contributed to larger waiting costs (Fig. 3.9) and in turn higher total costs (Fig. 3.10). This makes sense as not all vehicles' passengers wanted to cross as quickly as possible.

When fixing the crossing duration assignment method and varying the schedule assignments, we saw that assigning socially optimal schedules produced the most favorable (least costly) outcomes. See Fig. 3.9 for a visual comparison and Table 3.2 for the numerical values. This suggests that the socially optimal scheduling method is globally beneficial.

These results highlight our previous discussion on how the system design creates an implicit cost function hierarchy. We see in Fig. 3.10 that the crossing costs were optimized for PR, PF, and PO, but the waiting costs were not. When we used the combined auctioneer and scheduler configuration, the resulting average waiting cost reduced drastically with only a minuscule crossing cost increase.

It is important to emphasize how vehicles' cost functions influenced the results. In the best case scenario, vehicles' crossing cost functions resemble their waiting cost functions. If all vehicles preferred to cross faster, then the resulting starting times would be sooner. This would be most beneficial to impatient drivers (those who prefer to cross sooner). In the worse case scenario, the functions are opposites.

The experimental results and above discussion help establish two configuration extremes. At one extreme, the system assigns minimal crossing durations, optimizing traffic throughput at the expense of passenger preferences. At the other end, the system assigns preferred durations, which maximizes preferences but contributes to longer average trip durations. Figs. 3.10 and 3.11 show this inverse relation between passenger satisfaction and trip duration.

Fortunately, our system allows a designer to impose intersection-level constraints to slide between the two extremes. In this paper, we set a fixed upper bound on clearing time.

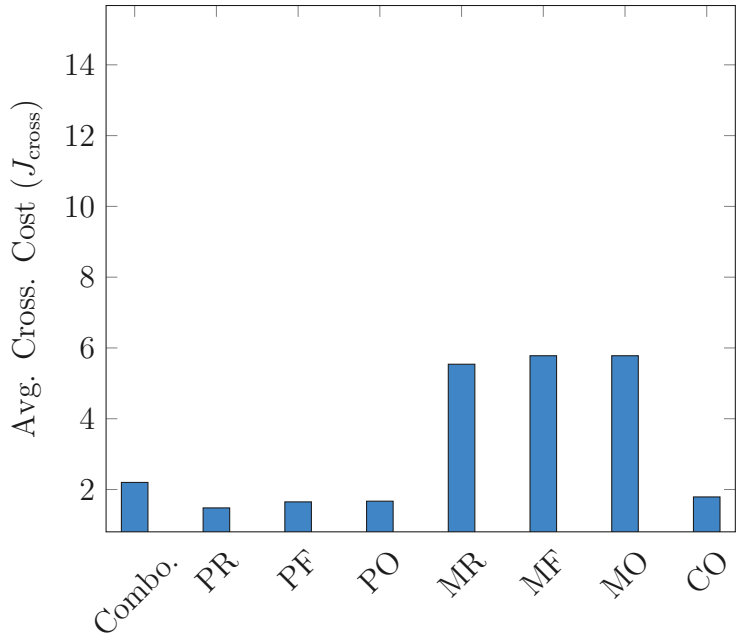


Figure 3.8: Average crossing costs for each IMS configuration.

A future iteration of our system could integrate traffic flow or congestion monitors and dynamically adjust the clearing time constraint. As traffic flow decreases, the clearing time bound could be lowered to alleviate congestion. The bound would then increase as traffic flow increases. Other intersection-level objective adjustments could include prioritizing more congested lanes.

3.7 Conclusion

In this chapter, we proposed an auction-theoretic intersection management system (IMS) that used cost functions instead of money as bids. Vehicle passengers have preferences on how quickly they cross an intersection, their *crossing cost*, and how long they wait before crossing, their *waiting cost*. As vehicles simultaneously approach an intersection, they share with the IMS their crossing and waiting cost functions. The auctioneer inside the IMS assigns socially-optimal (*i.e.*, cost-minimal) crossing durations. The scheduler subsystem then assigns a socially-optimal (*i.e.*, cost-minimal) crossing schedule.

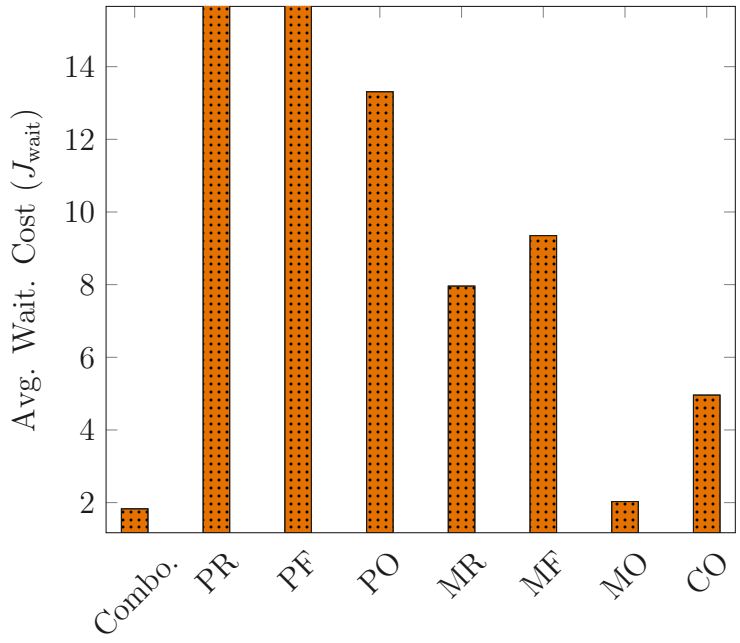


Figure 3.9: Average waiting costs for each IMS configuration. Note that the PR and PF values are clipped because they are much larger than the other values. See Table 3.2 for the values.

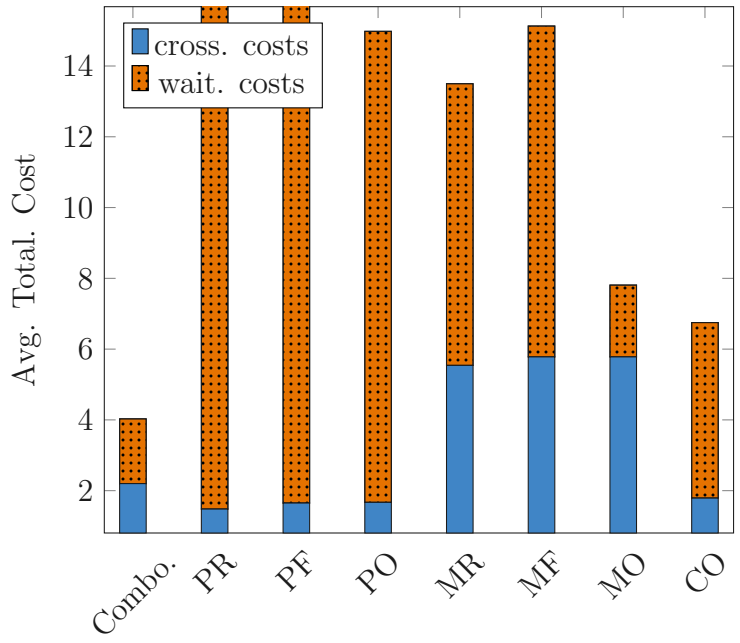


Figure 3.10: Average total costs for each IMS configuration. This figure combines data from Figs. 3.8 and 3.9. Note that the PR and PF values are clipped because they are much larger than the other values. See Table 3.2 for the values.

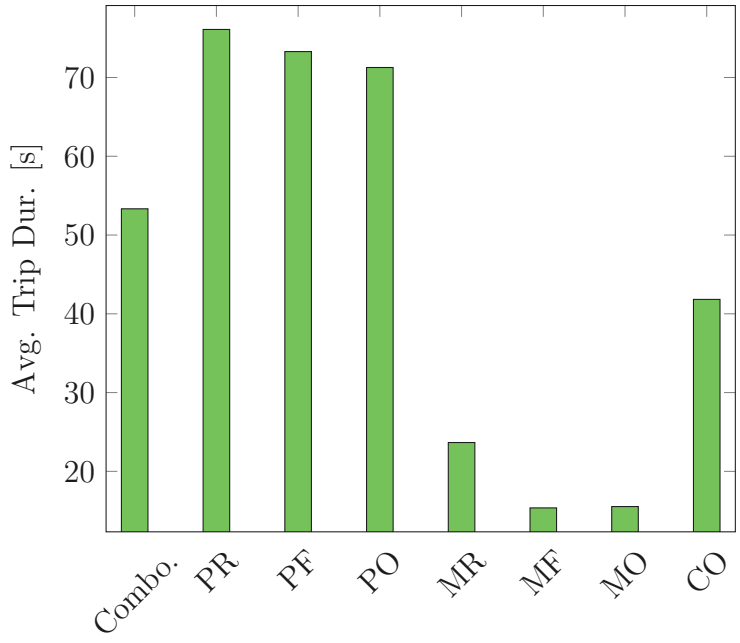


Figure 3.11: Average trip durations for each IMS configuration.

We evaluated our proposed system on a four-way intersection against several IMS configurations. Our results show that the system's performance depends on vehicles' cost functions. Results also show a trade-off between intersection throughput and passenger satisfaction. System designers can balance this by imposing constraints, such an upper bound on intersection clearing time. Experimental results demonstrated that a modest clearing time constraint can significantly improve intersection throughput with only a small decrease in satisfaction (cost increase).

This work provides several directions for future research. The current implementation assumes the intersection has fixed infrastructure to facilitate the auction, but vehicles could instead elect one of themselves to hold the auction. Additionally, we could integrate a traffic flow sensor and dynamically adjust the clearing time bound, assigning preferable crossing durations in light traffic and time-minimal durations in heavy traffic. Finally, we can analyze the auction mechanism for incentive compatibility and adapt it if necessary. Our current system design assumes vehicles report their preferences truthfully, which strategic agents may exploit this and bid untruthfully to gain unfair advantages over others.

Chapter 4

Game-Theoretic Motion Planning

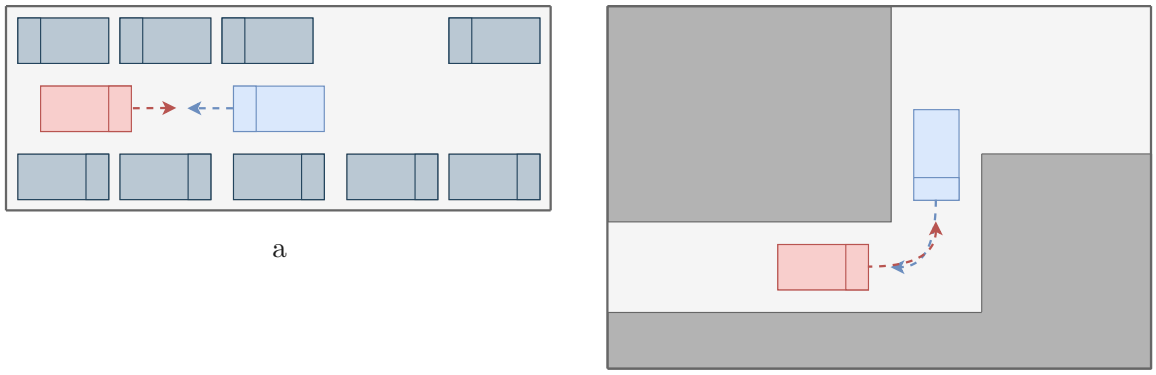
4.1 Introduction

Compared to rural or suburban roads, urban environments are significantly more cramped. When driving through narrow alleyways, other vehicles regularly and suddenly pull in off main roads, blocking the exits. This forces both drivers to somehow maneuver their cars so that they can pass each other. Human drivers easily and routinely resolve this conflict, but the problem is under explored for autonomous vehicles.

Common interaction scenarios that autonomous vehicles face include ramp merging, intersection crossing, and lane changing. Fig. 1.4 visualizes these. All of these scenarios provide sufficient room for each vehicle to maneuver, meaning their motion planners need to only determine a time scaling along their original reference path. However, when vehicles interact with each other in spatially-constrained environments such as parking lots, narrow side streets, or alleyways, vehicles need to make tactical decision and re-plan their paths and trajectories.

This chapter investigates complex vehicle interactions where self-interested vehicles mutually block each other from reaching their destinations, resulting in a deadlock—we term the resulting game a *deadlock game*.[1] Agents can resolve the deadlock only by coordinating

[1]The term *deadlock game* is slightly overloaded in the game theory literature, but other uses come from



a b

Figure 4.1: Deadlock interaction scenarios. a Depicts two vehicles trying to cross the same one-lane, two-way road. b Depicts two vehicles driving down an alley from different directions.

their movements. Fig. 4.1 illustrates two example scenarios. In Fig. 4.1a, the red vehicle wants to drive to the right end of the road while the blue vehicle's goal is to move to the left end. Neither one can to back up off the road because that would mean backing up into oncoming traffic. The only solution is for one of the vehicles to maneuver into the open space and allow the other to pass. Fig. 4.1b presents a similar situation for a narrow alleyway. Essentially, one vehicle needs to act locally suboptimal to achieve their globally optimal objective—they sacrifice their original optimal solution for the "greater good."

This chapter presents our preliminary research into solving deadlock games. We begin by formalizing the world representation, the agent movement models, and the game components. Then, we present a tactical decision making algorithm to solve general problem instances under a mild assumption. Finally, we compare our algorithm's performance against a data-driven policy synthesized using reinforcement learning. This work provides a foundation for continued expansions including equilibrium refinement and more complex scenarios.

areas outside of motion planning.

4.2 Related Work

Game-theoretic motion planning literature studies mainly ramp merging, lane changing, and intersection navigation. However, vehicles often interact in other ways: tight alleyways; one-lane, two-way roads; and small parking lots. During these interactions, drivers commonly need to work against their objective (*e.g.*, one vehicle needs to back up to let the other through). When optimizing over a short, fixed time horizon as in [47, 53, 55, 56, 84], not moving may be an optimal trajectory. Alternatively, the equilibria over the entire scenario could be for the vehicles to oscillate back and forth between game instances without actually making progress. Instead, an optimal long-term plan is to back into a parking spot then drive toward the original goal once the other car passes. Because existing motion planners do not consider long-term (mission-level) paths or planning horizons, they converge only to local optima.

Cooperative driving in spatially-constrained environment is an under-researched problem in the autonomous driving community; however, computational theorists and theoretical computer scientists have spent decades studying similar problems under the context of puzzles and games.

Rush Hour is a children's game that closely resembles the problem we present in this chapter. The game board is a 6×6 grid with different length cars arranged vertically or horizontally. Vehicles can move only left-to-right or up-and-down and cannot cross over each other. The goal is to arrange the cars so that a specially-designated car can escape the board through an opening on the perimeter. Figure 4.2a shows a picture of the game. Sliding-block puzzles are a similar game that generalizes Rush Hour planning problem. The puzzle consists of several rectangles that can move vertically or horizontally, and the player attempts to arrange the pieces in a specific sequence. Figure 4.2b shows an example sliding-block puzzle.

Rush Hour and sliding-block puzzles are both PSPACE-complete, meaning a polynomial time algorithm does not exist to find a solution (assuming $\mathsf{P} \neq \mathsf{NP}$) [86, 87]. One of the first

a b

Figure 4.2: a Image of the Rush Hour game. b Example sliding-block puzzle (from [85]).

solutions to the Rush Hour game problem involved creating a graph of game configurations where each node represented a different configuration [88]. A path from the game's initial configuration to its terminal configuration would reveal the solution strategy. Unfortunately, this approach scales combinatorially in time with game size and traffic density, meaning slightly increasing the game size would render the problem intractable.

Piece density significantly contributes to making Rush Hour and sliding-block puzzles challenging. Increasing game pieces increases the number of individual tactical moves the player must make. Fortunately, deadlock game scenarios, such as alleyways and narrow side streets, have few vehicles. Additionally, these environments offer slightly more space, for vehicles to maneuver. We exploit this structure in our algorithm.

We conjecture that general deadlock game instances are easier to solve than Rush Hour or sliding puzzle games because there is more free space available. Intuitively, sliding puzzles are challenging because the playing surface is densely populated with pieces. Moving one piece requires first moving a sequence of other pieces. In contrast, deadlock games have sparsely-populated driving areas, meaning most vehicles can freely move to unoccupied areas. Some movements require moving other vehicles first, but the movement sequences are shorter than in Rush Hour or sliding puzzles.

In the autonomous vehicle and robotics communities, machine learning and data-driven methods for creating motion planners abound. A recent work focusing specifically on

spatially-constrained environments developed a motion planner using reinforcement learning (RL) [89, 90]. The researchers trained a Deep Q-Network to generate conflict resolution policies for several vehicles in a parking lot. While the approach worked well for the trained scenarios, failed to transfer to similar ones, as we demonstrate in the experiment section. Additionally, it requires a lengthy training period. Our algorithm, however, easily transfers to various constrained environments and does not require training.

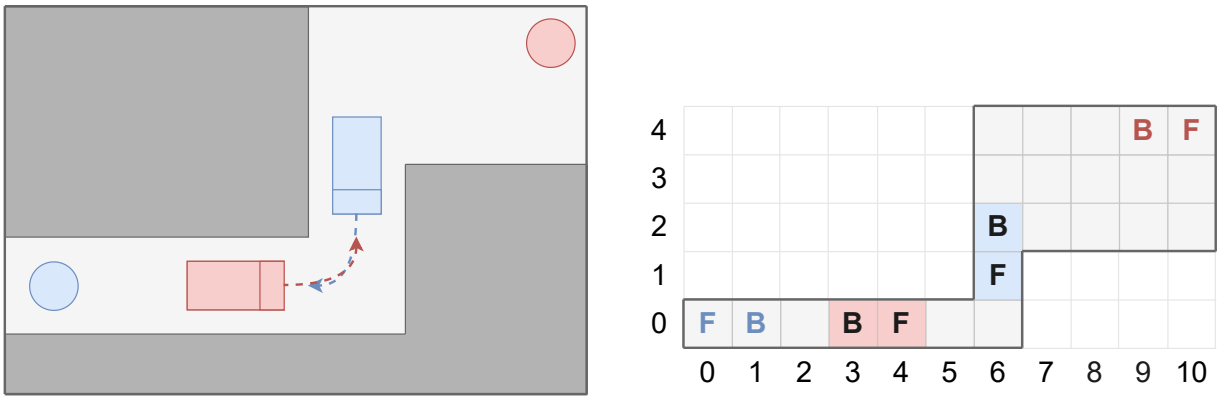
4.3 Problem Formulation

We make the following assumption: the ego agent will have a feasible path without moving once all vehicles move off the path. Breaking this assumption significantly increases the problem's complexity because it more closely resembles Rush Hour and sliding block puzzles mentioned in the previous section.

We consider driving interactions in extremely spatially-constrained roadways. Consider the alleyway depicted in Fig. 4.3a. The red car wants to drive to the red circle, while the blue car wants to drive to the blue circle. However, they cannot share the road; one vehicle must yield to the other.

4.3.1 World Representation

We represent the environment as a grid world $\mathcal{W} \subseteq \mathbb{N}_0 \times \mathbb{N}_0$ where each grid cell is a square with side lengths equal the car's width. Fig. 4.3 visualizes a continuous world and its grid world equivalent. Note that the grid world contains only cells representing the road surface; we exclude walls and other non-drivable areas because they serve no purpose in the discretized world.



a b

Figure 4.3: a Example extremely spatially-constrained environment. b Grid world representation of the environment. Note that only gray cells (and the red and blue cells) are part of the grid world. Letters F and B represent the vehicle's front and back, respectively. Colored squares represent vehicles, and colored letters represent each vehicle's destination.

4.3.2 Vehicle Representation

We model each vehicle as a rigid body moving on the grid world, and we assume all vehicles are homogeneous. Each vehicle's configuration space $\mathcal{C}$ is a discrete subset of the special Euclidean group $SE(2)$:

$$\mathcal{C} := \mathbb{Z} \times \mathbb{Z} \times \hat{\mathbb{S}}^1,$$

where

$$\hat{\mathbb{S}}^1 := \left\{0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi, \frac{5\pi}{4}, \frac{3\pi}{2}, \frac{7\pi}{4}, 2\pi\right\} / \sim, \qquad 0 \sim 2\pi.$$

The identification symbol $\sim$ indicates that 0 and 2π are equivalent (*i.e.*, the set "wraps around"). Vehicle i's configuration vector $\boldsymbol{q}^i \in \mathcal{C}$ is $\boldsymbol{q}^i := (x^i, y^i, \theta^i)$, where $x^i \in \mathbb{Z}$ and $y^i \in \mathbb{Z}$ represent its position on the plane, and $\theta^i \in \hat{\mathbb{S}}^1$ is its heading with respect to the x-axis. We set the vehicle's position reference to its rear axle, which is the back (B) cell in the grid world representation. The control vector $\boldsymbol{u}^i \in \mathcal{U}$ is $\boldsymbol{u}^i := (v^i, \omega^i)$, where v^i is the scalar linear velocity, and ω^i is the scalar angular velocity. The discrete control space $\mathcal{U}$ is defined as

$$\mathcal{U} := \left\{\left(-1, -\frac{\pi}{4}\right), (-1, 0), \left(-1, \frac{\pi}{4}\right), (0, 0), \left(1, -\frac{\pi}{4}\right), (1, 0), \left(1, \frac{\pi}{4}\right)\right\}.$$

Each vehicle moves according to the motion model

$$\begin{aligned}
\Delta x^i &= \begin{cases} \operatorname{sgn}\left(v^i \cos\theta^i\right) & v^i \geq 0 \vee (v^i < 0 \wedge \omega^i = 0) \\ \operatorname{sgn}\left(v^i \cos(\theta^i + \omega^i)\right) & \text{otherwise} \end{cases} \\
\Delta y^i &= \begin{cases} \operatorname{sgn}\left(v^i \sin\theta^i\right) & v^i \geq 0 \vee (v^i < 0 \wedge \omega^i = 0) \\ \operatorname{sgn}\left(v^i \sin(\theta^i + \omega^i)\right) & \text{otherwise} \end{cases} \\
\Delta\theta^i &= \omega^i,
\end{aligned} \tag{4.1}$$

where $\operatorname{sgn}(\cdot)$ is the sign function. Given the configuration and control spaces, each vehicle's movement under motion model (4.1) is equivalent to a discretized variant of the continuous kinematic bicycle model. We assume vehicles move slow enough that we can neglect friction and slipping.

Vehicles have a multi-cell footprint, meaning some parts may protrude beyond the grid world's edge even though the vehicle's configuration is valid. To restrict vehicles to in-world configurations, we first define a projection that maps a vehicle's configuration to its footprint in the grid world. Using notation similar to LaValle's in [39, Sec. 3.2], let $\mathcal{A}^i(\boldsymbol{q}^i)$ be vehicle i's footprint parameterized by its current configuration $\boldsymbol{q}^i \in \mathcal{C}$:

$$\mathcal{A}^i(\boldsymbol{q}^i) \coloneqq \left\{ \left(x^i, y^i\right), \left(x^i + \operatorname{sgn}(\cos\theta^i), y^i + \operatorname{sgn}(\sin\theta^i)\right) \right\}.$$

We slightly abuse notation to keep the mathematical text concise, so we take a moment to explain the subtleties. Set $\mathcal{A}^i$ represents agent i's footprint at their current configuration, implicitly assumed to be $\boldsymbol{q}^i \in \mathcal{C}$. We express this footprint using the short-hand notation

$$\mathcal{A}^i \coloneqq \mathcal{A}^i(\boldsymbol{q}^i).$$

We represent the collection of all N agents in the grid world with the overloaded set symbol

$\mathcal{A}$ (no superscript):

$$\mathcal{A} \coloneqq \{\mathcal{A}^1, \mathcal{A}^2, \ldots, \mathcal{A}^N\}.$$

Because all agents have identical footprints, we represent an arbitrary agent's footprint under a specific configuration $\boldsymbol{q} \in \mathcal{C}$ as $\mathcal{A}(\boldsymbol{q})$.

We define the in-world configuration space $\mathcal{C}_{\mathcal{W}}$ as the set of all configurations such that the vehicle is fully contained in the world (*i.e.*, no parts protrude beyond the world's boundaries):

$$\mathcal{C}_{\mathcal{W}} \coloneqq \{\boldsymbol{q} \in \mathcal{C} \mid \mathcal{A}(\boldsymbol{q}) \subseteq \mathcal{W}\} \subseteq \mathcal{C}.$$

We restrict all vehicles' configuration space to be $\mathcal{C}_{\mathcal{W}}$.

Now that we defined the configuration and control spaces for each agent, we define a deadlock game using the definition framework from from Başar and Olsder [91, Def. 5.1].

Definition 4.3.1 (Deadlock game)**.** A deadlock game is a type of N-person discrete-time deterministic finite dynamic game of prespecified fixed duration involving:

1. index set $\mathcal{N} = \{1, \ldots, N\}$ called the *players' set*, where N is the number of players
2. index set $\mathcal{K} = \{1, \ldots, K\}$ denoting *game stages*, where K is the maximum number of moves a player is allowed to make in the game
3. finite set $\mathcal{X} \coloneqq \mathcal{C}^1_{\mathcal{W}} \times \mathcal{C}^2_{\mathcal{W}} \times \cdots \times \mathcal{C}^N_{\mathcal{W}}$ called the game's *state space*
4. finite set $\mathcal{U}^i_k \subseteq \mathcal{U}$ called agent $\mathcal{A}^i$'s *control space* at stage k
5. function $\boldsymbol{f}_k \colon \mathcal{X} \times \mathcal{U}^1_k \times \cdots \times \mathcal{U}^N_k \to \mathcal{X}$ called the *state equation* and defined for each $k \in \mathcal{K}$ as

   $$\boldsymbol{x}_{k+1} = \boldsymbol{f}_k(\boldsymbol{x}_k, \boldsymbol{u}^1_k, \boldsymbol{u}^2_k, \ldots, \boldsymbol{u}^N_k),$$

   where $\boldsymbol{x}_1 \in \mathcal{X}$ is the game's *initial state*
6. finite set $\eta^i_k \in \mathcal{H}^i_k$ called the *information structure* and defined for each $k \in \mathcal{K}$ and $i \in \mathcal{N}$ as

   $$\eta^i_k \coloneqq \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k; \boldsymbol{u}^1_1, \boldsymbol{u}^1_2, \ldots, \boldsymbol{u}^1_{k-1}; \boldsymbol{u}^2_1, \boldsymbol{u}^2_2, \ldots, \boldsymbol{u}^2_{k-1}; \ldots; \boldsymbol{u}^N_1, \boldsymbol{u}^N_2, \ldots, \boldsymbol{u}^N_{k-1}\},$$

which establishes closed-loop perfect state information for agent $\mathcal{A}^i$

7. set $\mathcal{H}_k^i$ defined for each $k \in \mathcal{K}$ and $i \in \mathcal{N}$ as

$$\mathcal{H}_k^i \coloneqq \{(\mathcal{X}_1^1 \times \cdots \times \mathcal{X}_k^1) \times \cdots \times (\mathcal{X}_1^N \times \cdots \times \mathcal{X}_k^N) \times (\mathcal{U}_1^1 \times \cdots \times \mathcal{U}_{k-1}^1) \times \cdots \times (\mathcal{U}_1^N \times \cdots \times \mathcal{U}_{k-1}^N)\},$$

which is agent $\mathcal{A}^i$'s *information space* at stage k

8. class Γ_k^i of mappings $\boldsymbol{\gamma}_k^i \colon \mathcal{H}_k^i \to \mathcal{U}_k^i$ called agent $\mathcal{A}^i$'s *permissible strategies* at stage k
9. functional $L^i \colon \mathcal{H}_k^i \to \mathbb{R}$ called agent $\mathcal{A}^i$'s *cost functional*

■

We exclude the definitions for the observation space and state measurement equations from Def. 4.3.1 because agents can observe each others' full states. Therefore, the observation space is the game's state space, and the measurement model is the identity function.

4.4 Solution Description

We solve the deadlock game problem using a movement hierarchy and a recursive algorithm. Before presenting our solution, we introduce some key terms.

Definition 4.4.1 (Ego Agent)**.** The *ego agent* is the vehicle currently planning their movement. ■

Definition 4.4.2 (Immovable Agent)**.** An *immovable agent* is an agent that cannot be forced to move. Immovable agents are above the ego agent in the movement hierarchy. ■

Definition 4.4.3 (Movable Agent)**.** A *movable agent* is an agent that can be forced to move if they (partially) cover the ego agent's planned path. Movable agents are below the ego agent in the movement hierarchy. ■

Definition 4.4.4 (Blocking Agent)**.** A *blocking agent* is a movable agent that (partially) covers the ego agent's planned path. ■

Our solution imposes a mild assumption on the game instances. We assume each ego agent can find a clear path to their target configuration without moving from their current position, assuming all movable agents are free from the path. Violating this assumption significantly increases the game's difficulty as the problem more closely resembles Rush Hour or a sliding-block puzzle.

4.4.1 Algorithm Entry

The algorithm begins by arbitrarily designating one vehicle as the *ego agent* $\mathcal{A}^{\text{ego}}$, which has movement priority over all agents that have not yet reached their destinations. The immovable agents set $\mathcal{A}_{\text{immovable}}$ initially contains all agents that started at their destinations:

$$\mathcal{A}_{\text{immovable}} \coloneqq \{\mathcal{A}^i \in \mathcal{A} \mid \boldsymbol{q}^i = \boldsymbol{q}^i_{\text{dest}}\} \qquad \text{for } i = 1, \ldots, N.$$

Configuration $\boldsymbol{q}^i \in \mathcal{C}_{\mathcal{W}}$ is agent i's current configuration. For any realistic deadlock game instance, the immovable agents set initially would be equal to the empty set, but we initialize it this way for completeness. We initialize the movable agents set $\mathcal{A}_{\text{movable}}$ to include all agents that are not in the immovable agents set:

$$\mathcal{A}_{\text{movable}} \coloneqq \mathcal{A} \setminus \{\mathcal{A}^{\text{ego}}\} \setminus \mathcal{A}_{\text{immovable}}$$

Later in the algorithm's progression, the immovable agents set will expand as we move down the movement hierarchy. Similarly, the movable agents set will shrink as we move down the movement hierarchy. Algorithm 4.1 shows the pseudocode for the algorithm's entry point. The CurrentConfig function returns agent $\mathcal{A}^i$'s current configuration, which is $\boldsymbol{q}^i$.

Algorithm 4.1 Deadlock Game Algorithm Entry Point

1: **function** SolveProblem($\mathcal{W}$, $\mathcal{A}^1$, …, $\mathcal{A}^N$)
2: $\quad\mathcal{N} \leftarrow \{1, \ldots, N\}$
3: $\quad$**for** $i \in \mathcal{N}$ **do**
4: $\quad\quad\mathcal{A}_{\text{immovable}} \leftarrow \{\mathcal{A}^j \mid \text{CurrentConfig}(\mathcal{A}^j) = \boldsymbol{q}^j_{\text{dest}}\}, \qquad \text{for } j \in \mathcal{N}$
5: $\quad\quad\mathcal{A}_{\text{movable}} \leftarrow \{\mathcal{A}^j \mid \mathcal{A}^j \neq \mathcal{A}^i \wedge \mathcal{A}^j \notin \mathcal{A}_{\text{immovable}}\}, \qquad \text{for } j \in \mathcal{N}$

6: $\quad\quad$**if** $\neg$AttemptMove($\mathcal{W}, \mathcal{A}^i, \mathcal{A}_{\text{movable}}, \mathcal{A}_{\text{immovable}}$) **then**
7: $\quad\quad\quad$**return** False
8: $\quad\quad$**end if**
9: $\quad$**end for**

10: $\quad$**return** True
11: **end function**

After initialization, the algorithm calls the AttemptMove function, shown in Algorithm 4.2. In the function, the ego agent plans a path $\mathcal{P}^{\text{ego}} \coloneqq \{\boldsymbol{q}^{\text{ego}}, \ldots, \boldsymbol{q}^{\text{ego}}_{\text{dest}}\}$ from its current configuration to its destination configuration $\boldsymbol{q}^{\text{ego}}_{\text{dest}} \in \mathcal{C}_{\mathcal{W}}$ while assuming no movable agents exist in the world. The ego agent must navigate around immovable agents. The algorithm generates a set of blocking agents $\mathcal{A}_{\text{blocking}}$ that (partially) cover the ego agent's planned path:

$$\mathcal{A}_{\text{blocking}} \coloneqq \{\mathcal{A}^i \in \mathcal{A}_{\text{movable}} \mid \exists \boldsymbol{q} \in \mathcal{P}^{\text{ego}} \text{ s.t. } \mathcal{A}^i \cap \mathcal{A}(\boldsymbol{q}) \neq \emptyset\}, \qquad \text{for } i = 1, \ldots, |\mathcal{A}_{\text{movable}}|.$$

All blocking agents must move off the ego agent's planned path. The algorithm iterates through the set of blocking agents, and recursively calls the AttemptMove function for each one. In the recursive step, the set of immovable agents expands to the include the ego agent:

$$\mathcal{A}_{\text{immovable}} \leftarrow \mathcal{A}_{\text{immovable}} \cup \{\mathcal{A}^{\text{ego}}\}. \tag{4.2}$$

Additionally, the movable agents set shrinks to exclude the currently-indexed blocking agent:

$$\mathcal{A}_{\text{movable}} \leftarrow \mathcal{A}_{\text{movable}} \setminus \{\mathcal{A}^i_{\text{blocking}}\} \qquad \text{for } i = 1, \ldots, |\mathcal{A}_{\text{blocking}}|. \tag{4.3}$$

Then the currently-index blocking agent becomes the ego agent:

$$\mathcal{A}^{\text{ego}} \leftarrow \mathcal{A}^{i}_{\text{blocking}} \qquad \text{for } i = 1, \ldots, |\mathcal{A}_{\text{blocking}}|. \tag{4.4}$$

Algorithm 4.2 Attempt to Move Ego Agent

```
1: function AttemptMove(W, A^ego, A_movable, A_immovable)
2:     if CurrentConfig(A^ego) = q^ego_dest then        ▷ Ego agent already at destination
3:         return True
4:     end if

5:     P, A_blocking ← FindPath(W, A^ego, A_movable, A_immovable)

6:     if P = ∅ then                                    ▷ Ego agent cannot move
7:         return False
8:     end if

9:     if A_blocking = ∅ then                           ▷ No blocking agents
10:        MoveTo(A^ego, End(P))
11:        PlannedPath(A^ego) ← {End(P)}
12:        return True
13:    end if

14:    for i = 1, ..., |A_blocking| do                  ▷ All blocking agents need to move
15:        PlannedPath(A^ego) ← P
16:        A_movable ← A_movable \ {A^i_blocking}
17:        A_immovable ← A_immovable ∪ {A^ego}
18:        if ¬AttempMove(W, A^i, A_movable, A_immovable) then
19:            return False
20:        end if
21:    end for

22:    MoveTo(A^ego, End(P))
23:    PlannedPath(A^ego) ← {End(P)}
24:    return True
25: end function
```

4.4.2 Recursive Steps

Each recursive step traverses down the movement hierarchy. Within the step, the ego agent must move off all immovable agents' planned paths. The algorithm generates a set of con-

figurations that (partially) overlap any immovable agent's planned path:

$$\mathcal{C}_{\text{avoid}} \coloneqq \{\boldsymbol{q} \in \mathcal{C}_{\mathcal{W}} \mid \exists \boldsymbol{q}_{\text{path}} \in \mathcal{P}^i \text{ s.t. } \mathcal{A}(\boldsymbol{q}) \cap \mathcal{A}(\boldsymbol{q}_{\text{path}}) \neq \emptyset\} \qquad \text{for } i = 1, \ldots, |\mathcal{A}_{\text{immovable}}|.$$

From this set of "avoid" configurations, the algorithm generates a set of configurations in which the ego agent will not overlap with any paths, which we call *open configurations*:

$$\mathcal{C}_{\text{open}} \coloneqq \mathcal{C}_{\mathcal{W}} \setminus \mathcal{C}_{\text{avoid}}.$$

Next, the algorithm generates a set of reachable configurations:

$$\mathcal{C}_{\text{reachable}} \coloneqq \{\boldsymbol{q} \in \mathcal{C}_{\text{open}} \mid \textsc{HasPath}(\mathcal{C}_{\mathcal{W}'}, \boldsymbol{q}^{\text{ego}}, \boldsymbol{q})\},$$

where $\mathcal{C}_{\mathcal{W}'}$ is the set of in-world configurations excluding the immovable agents' configurations. Intuitively, the ego agent can cross over any path on their way to an open configuration, but they cannot touch any of them after they finish moving.

Finally, the ego agent can plan their path. If the reachable set contains the ego agent's destination configuration $\boldsymbol{q}^{\text{ego}}_{\text{dest}}$, they plan a path to it. Otherwise, the agent plans a path to one of the open configurations. If the reachable set is empty, the algorithm reports a planning failure.

The algorithm searches for blocking agents on the ego agent's path and recursively calls AttemptMove on all discovered ones. Before taking the recursive step, the algorithm updates the agent sets and ego agent using (4.2)–(4.4). Once the recursive calls propagates back up to the current ego vehicle, the algorithm will check the reported results from lower-level agents. If all agents moved off the path, the ego agent moves to their destination; otherwise, the algorithm returns a planning failure.

Fig. 4.4 visualizes the algorithm steps for a constrained environment involving two opposing vehicles.

Algorithm 4.3 Find Path to Destination Configuration or Open Configuration

function FINDPATH($\mathcal{W}$, $\mathcal{A}^{\text{ego}}$, $\mathcal{A}_{\text{movable}}$, $\mathcal{A}_{\text{immovable}}$)
  $\mathcal{C}_{\text{avoid}} \leftarrow \emptyset$
  for $i = 1, \ldots, |\mathcal{A}_{\text{immovable}}|$ **do**
    for $\boldsymbol{q}^i \in$ PLANNEDPATH($\mathcal{A}^i$) **do**
      $\mathcal{C}_{\text{touch}} \leftarrow \{\boldsymbol{q} \in \mathcal{C}_{\mathcal{W}} \mid \mathcal{A}(\boldsymbol{q}) \cap \mathcal{A}(\boldsymbol{q}^i) \neq \emptyset\}$
      $\mathcal{C}_{\text{avoid}} \leftarrow \mathcal{C}_{\text{avoid}} \cup \mathcal{C}_{\text{touch}}$
    end for
  end for

  $\mathcal{C}_{\text{open}} \leftarrow \mathcal{C}_{\mathcal{W}} \setminus \mathcal{C}_{\text{avoid}}$
  $\mathcal{C}_{\mathcal{W}'} \leftarrow \mathcal{C}_{\mathcal{W}} \setminus \{\boldsymbol{q}^i_{\text{immovable}}\}$ for $i = 1, \ldots, |\mathcal{A}_{\text{immovable}}|$
  $\boldsymbol{q}^{\text{ego}} \leftarrow$ CURRENTCONFIG($\mathcal{A}^{\text{ego}}$)
  $\mathcal{C}_{\text{reachable}} \leftarrow \{\boldsymbol{q} \in \mathcal{C}_{\text{open}} \mid$ HASPATH($\mathcal{C}_{\mathcal{W}'}, \boldsymbol{q}^{\text{ego}}, \boldsymbol{q}$)$\}$

  $\mathcal{P} \leftarrow \emptyset$
  if $\boldsymbol{q}^{\text{ego}}_{\text{dest}} \in \mathcal{C}_{\text{reachable}}$ **then**
    $\mathcal{P} \leftarrow$ SHORTESTPATH($\mathcal{C}_{\mathcal{W}'}, \boldsymbol{q}^{\text{ego}}, \boldsymbol{q}^{\text{ego}}_{\text{dest}}$)
  else if $\mathcal{C}_{\text{reachable}} \neq \emptyset$ **then**
    $\boldsymbol{q}_{\text{reach}} \in \mathcal{C}_{\text{reachable}}$
    $\mathcal{P} \leftarrow$ SHORTESTPATH($\mathcal{C}_{\mathcal{W}'}, \boldsymbol{q}^{\text{ego}}, \boldsymbol{q}_{\text{reach}}$)
  end if

  $\mathcal{A}_{\text{blocking}} \leftarrow \emptyset$
  for $\boldsymbol{q} \in \mathcal{P}$ **do**
    $\mathcal{A}_{\text{blocking}} \leftarrow \mathcal{A}_{\text{blocking}} \cup \{\mathcal{A}^i_{\text{movable}} \mid \mathcal{A}^i_{\text{movable}} \cap \mathcal{A}(\boldsymbol{q}) \neq \emptyset\}$ for $i = 1, \ldots, |\mathcal{A}_{\text{movable}}|$
  end for

  return $\mathcal{P}, \mathcal{A}_{\text{blocking}}$
end function

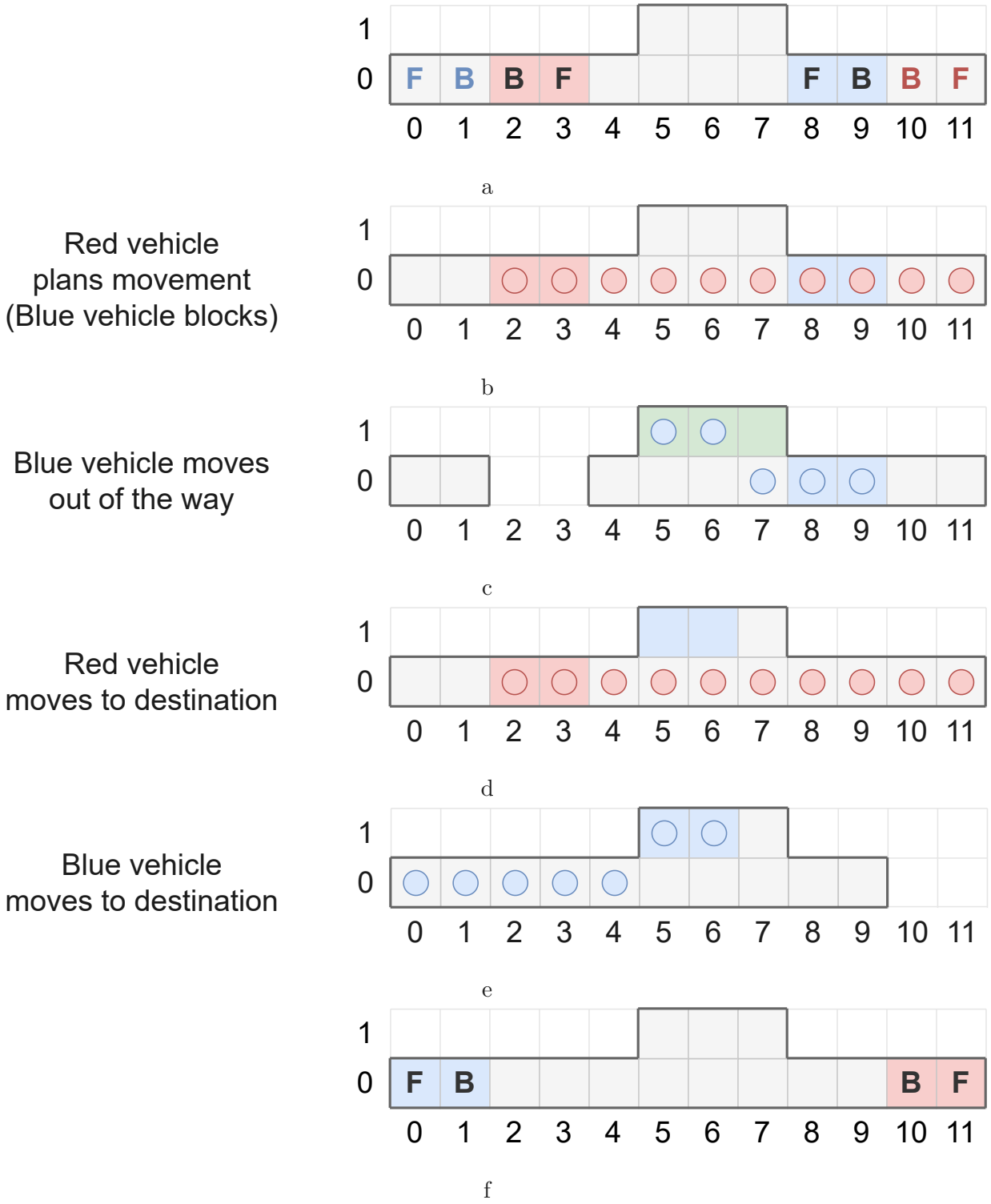


Figure 4.4: Deadlock game algorithm visualization

4.4.3 Infeasible Orderings

Some deadlock game instances do not admit a feasible solution for all movement orderings. For example, the scenario in Fig. 4.3 is infeasible if the blue car attempts to move first. In that ordering, the blue car would completely block the red car from moving off the planned path. The problem has only one solution: give priority to the red car.

To resolve the infeasible orderings issue, we can iterate through all order permutations and return an ordering that admits a solution. Note that this process increases the algorithm's time complexity, but each permutation is independent. Therefore, we can evaluate them in parallel.

4.5 Experiments

4.5.1 Setup

We implemented our algorithm in Python 3 using the NetworkX library [92]. We represented the grid world and configuration spaces as unweighted undirected graphs. For the grid world graph, each node represented a cell denoted by position coordinates (x, y). Configuration graphs represented configurations from the configuration space, and configuration vectors (x, y, θ) identified individual nodes. The configuration graph contained edges between two nodes if there was a feasible transition based on the vehicle's motion model. Each vehicle had their own configuration graph. Vehicles planned their paths using Dijkstra's shortest path algorithm.

Fig. 4.5 visualizes the grid world graph and its agents. Each node displays its coordinates in the 2D plane. Node colors indicate an agent's current position (projected from their current configuration). Node border colors denote a vehicle's planned destination. For both node and border colors, light shades are the vehicles' fronts, and dark ones are their backs.

We evaluated our algorithm on several grid world environments with varying vehicle

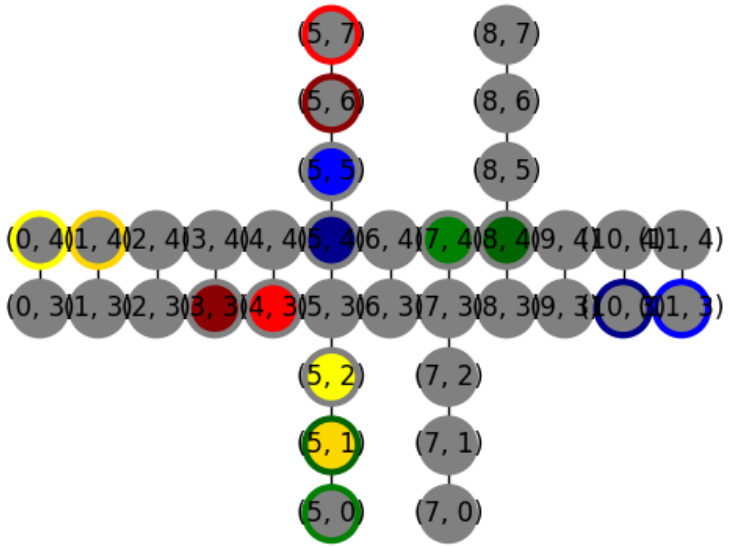


Figure 4.5: Undirected graph representation for a deadlock game instance.

configurations. Fig. 4.6 shows the different environments we used during our experiments. The reinforcement learning (RL) based policy from [89, 90] served as a baseline comparison. Fig. 4.6a visualizes the environment they used for testing and evaluation, which we recreated so that we could evaluate our algorithm on it. We trained the RL policy on the environment the authors gave and according to their instructions. We then applied the same policy, without modification, to our new test environments.

4.5.2 Results and Discussion

Table 4.1 summarizes our experimental results. In our experiments, the RL trained policy solved the problem it used for training, but it failed to generalize to other environments. It failed even for subjectively simpler environments, such as the one in Fig. 4.6f. Our algorithm, however, solved all of the scenarios without needing any type of training.

We want to emphasize a limitation of our comparison against the RL based policy. It is not definitive proof that machine learning based policies in general fail to transfer to arbitrary environments. We trained the policy only on one example world, so it likely would have transferred better if it was trained with more examples. However, we believe the

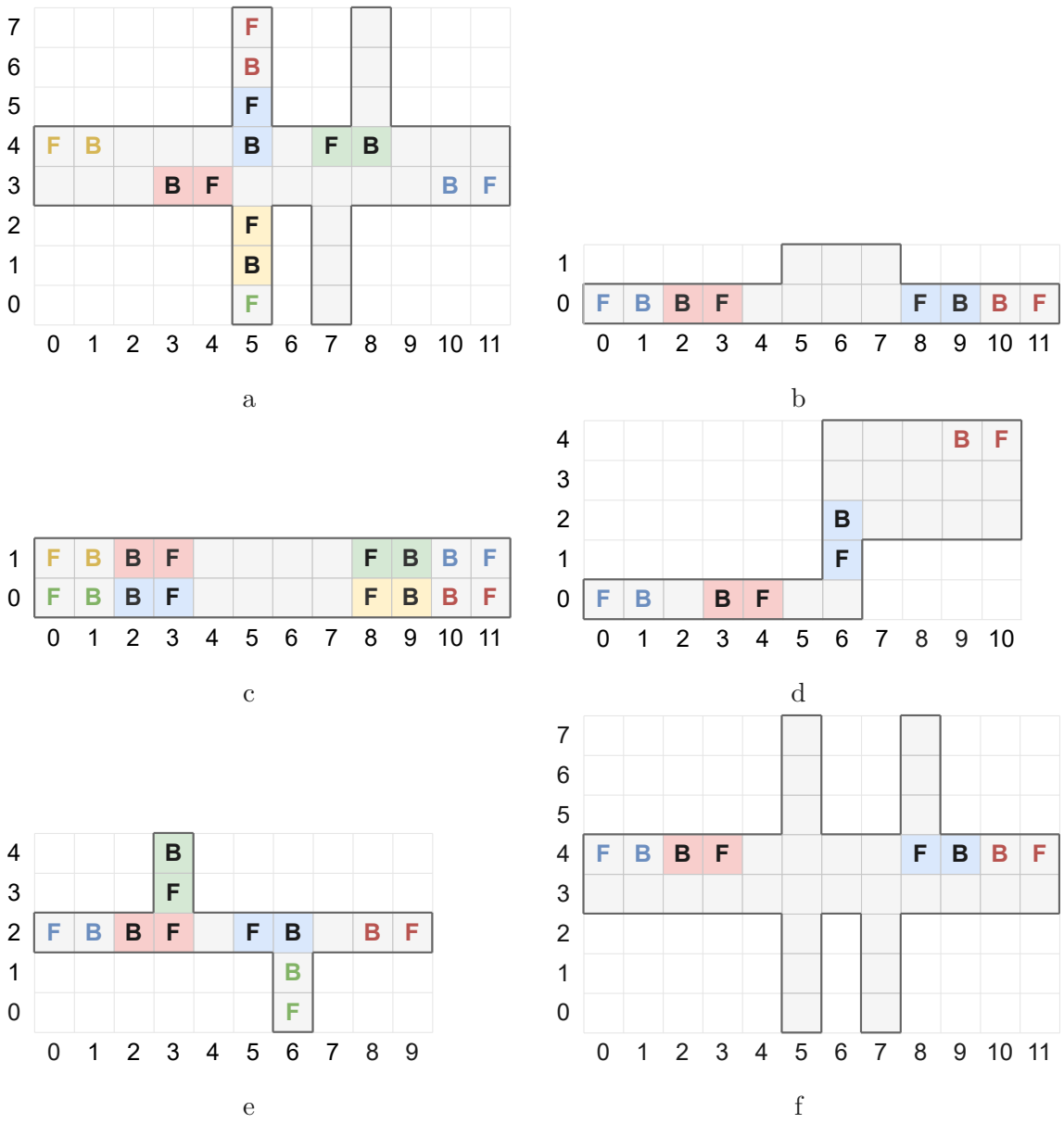


Figure 4.6: Grid worlds used during our experiments.

Table 4.1: Deadlock Games Experimental Results

Scenario	RL Policy	Our Algorithm
4.6a	✓	✓
4.6b		✓
4.6c		✓
4.6d		✓
4.6e		✓
4.6f		✓

results suggest that machine learning based policies can struggle to solve *a priori* unknown environments whereas a more engineered approach does not. The performance results in scenario 1 (Fig. 4.6a) versus scenario 6 (Fig. 4.6f) showcase this struggle. The latter scenario used the same environment structure but had fewer agents, hypothetically making it an easier problem to solve.

4.6 Conclusion

Driving interactions sometimes occur in spatially-constrained environments, such as narrow roads or alleyways. In these cases, drivers (or autonomous vehicles) need to coordinate their movements so they can maneuver around each other. Otherwise, they will be left in a deadlock, unable to move.

This chapter presented the notion of a deadlock game as a type of discrete dynamic game. We then proposed an algorithm that exploited the problem structure to solve arbitrary deadlock game instances under a mild assumption. Our experiments evaluated the algorithm on several challenging driving environments and compared its performance to a recently-proposed RL based policy. The results indicate that machine learning based policies struggle to solve previously unseen environments whereas our solution solved all challenges. Our solution also does not need a training period.

The work we presented in this chapter is a foundational component to a higher-level conflict resolution system. Because deadlock games are a type of game, they admit several (possible incomparable) solutions. Following this dissertation's theme, we need a higher-level system to refine the Pareto optimal set of equilibria by choosing a single solution that best satisfies all agent's preferences. We leave this continued investigation for future work.

Another extension to this work includes relaxing the clear-path assumption imposed in our problem formulation. For example, consider the scenario in Fig. 4.6e. By reversing each agent's target configuration, we drastically increase the problem's difficulty. This subtle

change now requires the ego agent to move to an intermediate configuration so that movable agents can maneuver around them.

Chapter 5

Conclusion

5.1 Conclusion

In this dissertation, we developed an auction-based intersection management that assigned crossing durations and crossing schedules that optimize vehicles' passengers' preferences. This work begins a push toward decision making that focuses more on social welfare than traditional performance metrics. We also formulated the notion of a deadlock game and presented an algorithm to solve general instances. The deadlock game work forms the foundation for a higher-level socially-optimal conflict resolution system that we leave for future work. This dissertation's main contributions are:

- **Path-constrained, time-assigned, control-minimal trajectory optimizer.** We developed a trajectory optimizer that determines a feasible time scaling mapping time values to path-positions. Composing the time scaling with the associated reference path yields a reference trajectory that, when tracked, traverses the path in a specific duration. This contribution is a core component to the later-proposed intersection management system.

- **Auction-based intersection management system.** We developed an intersection management system where connected vehicles big using cost functions. As vehicles ap-

proach the intersection, the send their cost functions to the auctioneer. The auctioneer assigned crossing durations and a schedule based on the functions. To the best of our knowledge, our proposed system is the first of its kind. In our extensive testing, we show that they system successfully optimizes for social welfare at the expense of traffic flow. Furthermore, we proposed an intersection-level constraint that allows system designers to balance between passenger satisfaction and intersection throughput.

- **Deadlock games and solution algorithm.** We explored contention in spatially-constrained environments where self-interested agents mutually block each other from reaching their destinations. Under the definition framework of dynamic games, we formatted the concept of a *deadlock game* as a type of discrete-time finite game. Additionally, we developed an algorithm to solve arbitrary instances and evaluated it against several challenging example scenarios. Our experimental results demonstrate our algorithm's superior performance against state-of-the-art solutions that use machine learning to train a control policy.

The following publications support the dissertation's technical contributions: [61, 83, 93].

5.2 Future Directions

The work we presented in this dissertation serves as a starting point for a transition into social welfare focused optimization and decision making. While we presented several significant and novel contributions throughout the dissertation, each chapter leaves avenues for further improvements and continued research.

We designed the trajectory optimizer in Chapter 2 specifically for the second order unicycle motion model. One extension to this framework include applying it to other motion models, such as the second-order bicycle model. An application-oriented extension involves implementing the optimizer into ROS 2, simulating it in Gazebo, and deploying it on a physical robot.

The IMS we developed in Chapter 3 assumes the intersection has fixed infrastructure to facilitate the auction, but vehicles could instead elect one of their themselves to hold the auction. Additionally, we could integrate a traffic flow sensor to dynamically adjust the clearing time bound, assigning preferable crossing durations in light traffic and time-minimal durations in heavy traffic. Finally, we can analyze the auction mechanism for incentive compatibility and adapt it if necessary. Our current system design assumes vehicles report their preferences truthfully, which strategic agents may exploit and bid untruthfully to gain unfair advantages over others.

The deadlock game solution algorithm we presented in 4 imposes an assumption on game instances' starting configurations. Vehicles must start in a configuration where all movable agents can move off the ego agent's planned path without the ego agent moving. Violating this assumption dramatically increases the game's complexity because it more closely resembled Rush Hour and sliding-block puzzle instances. Instead of moving directly to their planned destinations, ego vehicles would have to move to an intermediate configuration so that movable agents could continue moving off the planned paths. Another future direction includes using the deadlock games formulation as a foundation for a socially-optimal conflict resolution system. The current algorithm searches for a feasible solution, but the game admits several possible solutions, some better than others. Under this dissertation's theme, a conflict resolution system would incorporate vehicles' costs over the outcome and settle on one that satisfies everyone's preference as best as possible.

Vita

Adam Morrissett was born in Chesterfield, VA, USA, on December 26, 1995. He graduated from the Governor's Academy for Engineering Studies at Lloyd C. Bird High School, Chesterfield, Virginia, USA in 2014. In the summer of 2016 between May and August, he worked as an undergraduate researcher developing a collaborative formation flying algorithm for a group of fixed-wing model aircraft. He received the bachelor of science in computer engineering with a minor in computer science from Virginia Commonwealth University, Richmond, Virginia, USA in May 2018. In the summer of 2021 between June and August, he interned with the U.S. Department of Transportation, Federal Highway Administration developing software for their open-source autonomous vehicle platform, Cooperative Automation Research Mobility Applications (CARMA) Platform. He received the doctor of philosophy in engineering with a concentration in electrical and computer engineering from Virginia Commonwealth University, Richmond, Virginia, USA in May 2023. His research work focused on cooperative driving automation, trajectory optimization, and multi-vehicle motion planning.

Publication History

Socially-Optimal Auction-Theoretic Intersection Management

Authors. Adam Morrissett, Patrick J. Martin

Journal. Submitted for review (2023)

Abstract. When multiple vehicles arrive simultaneously at an unsignalized intersection, the crossing order is ambiguous. This ambiguity causes a stalemate that persists until one driver decides to cross. Several intersection management systems (IMSes) exist to determine crossing orders based on a variety of policies. In auction-based IMSes, vehicles use monetary bids to gain crossing priority over others. However, monetary auctions may disenfranchise those that cannot afford to bid. We propose an alternative auction formulation where vehicles bid with cost functions instead of money. Our resulting IMS allocates crossing durations and determines crossing schedules based on passengers' preferences and their vehicles' capabilities. The IMS also allows system designers to impose constraints that alter duration and schedule assignment. We evaluate our system in a simulated environment and against several configuration variants. The results reveal an interesting trade-off between passenger preferences and intersection throughput.

Control-Minimal Time-Assigned Path-Constrained Trajectory Optimization

Authors. Adam Morrissett, Patrick J. Martin

Conference. 2023 American Control Conference (ACC)

Abstract. Path-constrained trajectory optimization research normally focuses on time or energy optimality. However, some applications seek reference trajectories that satisfy other constraints. In this paper, we formulate a control-minimal time-assigned path-constrained trajectory optimization problem: a mobile ground robot must traverse a given path in a specific amount of time using minimal control effort. Through a nonlinear change of variables,

we solve this problem using convex optimization. We evaluate our solution with an intelligent transportation scenario where an autonomous vehicle must cross an intersection in a specific amount of time while following the turn lane's geometric center.

Socially-Optimal Auction-Theoretic Intersection Management System

Authors. Adam Morrissett, Patrick J. Martin, Sherif Abdelwahed

Conference. 2022 IEEE Intelligent Vehicles Symposium (IV)

Abstract. Unsignalized intersections are often sources of congestion and collisions. When human-driven vehicles arrive simultaneously, determining which one goes first can be difficult; drivers typically creep out into the intersection or wave each other through to break stalemates. While intuitive for human drivers, this can be challenging for autonomous vehicles (AVs). Current AVs typically operate in isolation without explicitly communicating their intentions to others. In this paper, we propose an auction-based intersection management system (IMS) that determines a crossing schedule based on vehicle input. Vehicles bid for crossing time using a cost function over different possible crossing times, and the IMS assigns crossing times that maximize social utility. We evaluate our system with an ambiguous crossing scenario and demonstrate its usefulness in determining socially-optimal crossing schedules [61].

OpenCity: An Open Architecture Testbed for Smart Cities

Authors. Nasibeh Zohrabi, Patrick J. Martin, Murat Kuzlu, Lauren Linkous, Roja Eini, **Adam Morrissett**, Mostafa Zaman, Ashraf Tantawy, Oezguer Gueler, Maher Al Islam, Nathan Puryear, Halil Kalkavan, Jonathan Lundquist, Erwin Karincic, Sherif Abdelwahed

Conference. 2021 IEEE International Smart Cities Conference (ISC2),

Abstract. This paper presents an open architecture testbed for smart cities, called OpenCity,

which is hosted at Virginia Commonwealth University (VCU). The OpenCity platform consists of data collection and processing units, database management, distributed performance management algorithms, and real-time data visualization. This smart city testbed aims to support various educational and research activities related to smart city development. The testbed provides a near-real-life platform to allow students to learn about the unique features of smart cities and explore supporting technologies. In addition, it allows researchers to develop, deploy, and validate new techniques, tools, and technologies to support future smart city developments. The OpenCity platform will support various ongoing important research directions in smart cities, including smart homes and buildings, urban mobility, smart grid, and water management. In addition, it will be extendable to include other potential applications and components as needed. The testbed will be validated by developing and deploying a management system that focuses on users' experience and resource efficiency. The management system incorporates learning techniques and model-based predictive control approaches to take into account the current and future information of uncertain parameters as well as the subjective data (e.g., user-related data) in the design. The OpenCity management structure enables real-time control and monitoring of complex components in the testbed [94].

A Review of Non-Lane Road Marking Detection and Recognition

Authors. Adam Morrissett, Sherif Abdelwahed

Conference. 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)

Abstract. Environment perception is a critical function used by driving automation systems, or self-driving cars, for detecting objects such as obstacles, lane markings, and road signs. In order to replace human drivers, self-driving cars will need to safely operate in parking lots, private roads, underground, or any other environment with poor GPS signals or uncharted infrastructure. While much attention has been spent on recognizing lane mark-

ings, non-lane road markings have received considerably less attention. Current perception systems can recognize only a small subset of markings and often only under favorable weather conditions. This limitation is exacerbated by the current quality of scene segmentation data sets. Only a select few of existing data sets have annotations for non-lane road markings, and the ones that do only have them for a small number of marking types. Additionally most of the data sets were generated under one type of driving condition. Finally, it is difficult to determine if current recognition systems can satisfy real-time requirements. This paper investigates the current limitations and challenges for non-lane road marking detection and recognition including recognition capabilities, data set quality, and inference times [95].

A Physical Testbed for Intelligent Transportation Systems

Authors. Adam Morrissett, Roja Eini, Mostafa Zaman, Nasibeh Zohrabi, Sherif Abdelwahed

Conference. 2019 12th International Conference on Human System Interaction (HSI)

Abstract. Intelligent transportation systems (ITSs) and other smart-city technologies are increasingly advancing in capability and complexity. While simulation environments continue to improve, their fidelity and ease of use can quickly degrade as newer systems become increasingly complex. To remedy this, we propose a hardware- and software-based traffic management system testbed as part of a larger smart-city testbed. It comprises a network of connected vehicles, a network of intersection controllers, a variety of control services, and data analytics services. The main goal of our testbed is to provide researchers and students with the means to develop novel traffic and vehicle control algorithms with higher fidelity than what can be achieved with simulation alone. Specifically, we are using the testbed to develop an integrated management system that combines model-based control and data analytics to improve the system performance over time. In this paper, we give a detailed description of each component within the testbed and discuss its current developmental state. Additionally, we present initial results and propose future work [96].

A Physical Testbed for Smart City Research

Authors. Adam Morrissett, Sherif Abdelwahed

Conference. 2018 IEEE/ACS 15th International Conference on Computer Systems and Applications (AICCSA)

Abstract. City infrastructure is deteriorating, traffic management systems are becoming increasingly inefficient due to volume, and resources are becoming scarce. In the era of information and analytics, the idea of smart cities has been increasingly proposed as a solution to inefficient public services and resource management. While some cities have had success with beginning to transform into smart cities, the process has revealed significant barriers. One of which is the communication infrastructure necessary to create an interconnected network of sensors, actuators, and analytics systems. This barrier is discussed, and a physical testbed for smart city research is proposed. The current progress of the testbed development is reported, and a plan for continued work is outlined [97].

References

[1] U.S. Department of Transportation, National Highway Traffic Safety Administration, "Overview of motor vehicle crashes in 2019," U.S. Department of Transportation, National Highway Traffic Safety Administration, Research Note DOT HS 813 060, Dec. 2020. [Online]. Available: https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/813060

[2] D. Schrank, L. Albert, B. Eisele, and T. Lomax, "2021 urban mobility report," The Texas A&M Transportation Institute, Tech. Rep., Jun. 2021. [Online]. Available: https://static.tti.tamu.edu/tti.tamu.edu/documents/mobility-report-2021.pdf

[3] U.S. Department of Transportation, "Preparing for the future of transportation: Automated vehicles 3.0," U.S. Department of Transportation, Washington, D.C., USA, Tech. Rep., Sep. 2018. [Online]. Available: https://www.transportation.gov/av/3/preparing-future-transportation-automated-vehicles-3

[4] SAE International, "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," SAE International, Tech. Rep. J3016_202104, Apr. 2021. [Online]. Available: https://www.sae.org/standards/content/j3016_202104/

[5] A. S. Mueller, J. B. Cicchino, and D. S. Zuby, "What humanlike errors do autonomous vehicles need to avoid to maximize safety?" *J. of Saf. Res.*, vol. 75, pp. 310–318, Dec. 2020.

[6] National Transportation Safety Board, "Collision between a sport utility vehicle operating with partial driving automation and a crash attenuator," National Transportation Safety Board, Highway Accident Report NTSB/HAR-20/01, Feb. 2020. [Online]. Available: https://www.ntsb.gov/investigations/Pages/HWY18FH011.aspx

[7] ——, "Collision between a car operating with automated vehicle control systems and a tractor-semitrailer truck," National Transportation Safety Board, Highway Accident Report NTSB/HAR-17/02, Sep. 2017. [Online]. Available: https://www.ntsb.gov/investigations/Pages/HWY16FH018.aspx

[8] ——, "Collision between car operating with partial driving automation and truck-tractor semitrailer," National Transportation Safety Board, Highway Accident Brief NTSB/HAB-20/01, Jan. 2020. [Online]. Available: https://www.ntsb.gov/investigations/Pages/HWY19FH008.aspx

[9] ——, "Rear-end collision between a car operating with advanced driver assistance systems and a stationary fire truck," National Transportation Safety Board, Highway Accident Brief NTSB/HAB-19/07, Aug. 2019. [Online]. Available: https://www.ntsb.gov/investigations/Pages/HWY18FH004.aspx

[10] ——, "Collision between vehicle controlled by developmental automated driving system and pedestrian," National Transportation Safety Board, Highway Accident Report NTSB/HAR-19/03, Nov. 2019.

[11] E. Baron, "Blame game: Self-driving car crash highlights tricky legal question," *The Mercury News*, Jan. 2018. [Online]. Available: https://www.mercurynews.com/2018/01/23/motorcyclist-hit-by-self-driving-car-in-s-f-sues-general-motors/

[12] A. J. Hawkins, "A driverless Waymo got stuck in traffic and then tried to run away from its support crew," *The Verge*, May 2021. [Online]. Available: https://www.theverge.com/2021/5/14/22436584/waymo-driverless-stuck-traffic-roadside-assistance-video

[13] State of California Department of Motor Vehicles, "2020 disengagement reports," State of California Department of Motor Vehicles, Tech. Rep., 2020. [Online]. Available: https://www.dmv.ca.gov/portal/vehicle-industry-services/autonomous-vehicles/disengagement-reports/

[14] ——, "Autonomous vehicle testing permit holders," 2021. [Online]. Available: https://www.dmv.ca.gov/portal/vehicle-industry-services/autonomous-vehicles/autonomous-vehicle-testing-permit-holders/

[15] National Highway Traffic Safety Administration, "Part 573 safety recall report," National Highway Traffic Safety Administration, Safety Recall 21V-846, Oct. 2021. [Online]. Available: https://static.nhtsa.gov/odi/rcl/2021/RCLRPT-21V846-7836.PDF

[16] G. Karagiannis, O. Altintas, E. Ekici, G. Heijenk, B. Jarupan, K. Lin, and T. Weil, "Vehicular networking: A survey and tutorial on requirements, architectures, challenges, standards and solutions," *IEEE Commun. Surv. & Tut.*, vol. 13, no. 4, pp. 584–616, Jul. 2011.

[17] J. Loftus and R. Tershak, "Truck platooning: The state of the industry and future research topics," Jan. 2018.

[18] S.-W. Kim, B. Qin, Z. J. Chong, X. Shen, W. Liu, M. H. Ang, E. Frazzoli, and D. Rus, "Multivehicle cooperative driving using cooperative perception: Design and experimental validation," *IEEE Trans. Intell. Transport. Syst.*, vol. 16, no. 2, pp. 663–680, Apr. 2015.

[19] A. Davila, E. del Pozo, E. Aramburu, and A. Freixas, "Environmental benefits of vehicle platooning," in *Symp. on Int. Automot. Technol. 2013*, Jan. 2013.

[20] SAE International, "Taxonomy and definitions for terms related to cooperative driving automation for On-Road motor vehicles," SAE International, Tech. Rep. J3216_202107, Jul. 2021. [Online]. Available: https://www.sae.org/standards/content/j3216_202107/

[21] ——, "V2X communications message set dictionary," SAE International, Tech. Rep. J2735_202007, Jul. 2020. [Online]. Available: https://www.sae.org/standards/content/j2735_202007/

[22] ——, "Automated driving reference architecture," SAE International, Tech. Rep. J3131, Feb. 2016, work in progress. [Online]. Available: https://www.sae.org/standards/content/j3131/

[23] "Autopilot." [Online]. Available: https://www.tesla.com/autopilot

[24] "Technology." [Online]. Available: https://www.getcruise.com/technology

[25] "Waymo Driver." [Online]. Available: https://waymo.com/waymo-driver/

[26] "Apollo." [Online]. Available: https://apollo.auto/

[27] "Autoware.Auto." [Online]. Available: https://www.autoware.org/autoware-auto

[28] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Trans. Intell. Veh.*, vol. 1, no. 1, pp. 33–55, Mar. 2016.

[29] U.S. Department of Transportation, Federal Highway Administration, "About intersection safety," Mar. 2021. [Online]. Available: https://safety.fhwa.dot.gov/intersection/about/

[30] T. Urbanik, A. Tanaka, B. Lozner, E. Lindstrom, K. Lee, S. Quayle, S. Beaird, S. Tsoi, P. Ryus, D. Gettman, S. Sunkari, K. Balke, D. Bullock, National Cooperative Highway

Research Program, Transportation Research Board, and National Academies of Sciences, Engineering, and Medicine, *Signal Timing Manual*, 2nd ed. Washington, D.C., USA: Transportation Research Board, Sep. 2015.

[31] Z. Zhong, M. Nejad, and E. E. Lee, II, "Autonomous and semiautonomous intersection management: A survey," *IEEE Intell. Transp. Syst. Mag.*, vol. 13, no. 2, pp. 53–70, Oct. 2021.

[32] M. A. Guney and I. A. Raptis, "Scheduling-driven motion coordination of autonomous vehicles at a multi-lane traffic intersection," in *2018 Annu. Amer. Control Conf. (ACC)*. Milwaukee, WI, USA: IEEE, Jun. 2018, pp. 4038–4043.

[33] ——, "Scheduling-based optimization for motion coordination of autonomous vehicles at multilane intersections," *J. of Robot.*, vol. 2020, Mar. 2020.

[34] M. Stryszowski, S. Longo, E. Velenis, and G. Forostovsky, "A framework for self-enforced interaction between connected vehicles: Intersection negotiation," *IEEE Trans. on Intell. Transp. Syst.*, vol. 22, no. 11, pp. 6716–6725, Nov. 2021.

[35] S. A. Fayazi and A. Vahidi, "Mixed-integer linear programming for optimal scheduling of autonomous vehicle intersection crossing," *IEEE Trans. on Intell. Veh.*, vol. 3, no. 3, pp. 287–299, Sep. 2018.

[36] D. Carlino, S. D. Boyles, and P. Stone, "Auction-based autonomous intersection management," in *16th Int. IEEE Conf. on Intell. Transp. Syst. (ITSC 2013)*. The Hague, NL: IEEE, Oct. 2013, pp. 529–534.

[37] G. Cabri, L. Gherardini, and M. Montangero, "Auction-based crossings management," in *Proc. 5th EAI Int. Conf. on Smart Objects and Technol. for Social Good*. Valencia, ES: ACM, 2019, pp. 183–188.

[38] G. Cabri, L. Gherardini, M. Montangero, and F. Muzzini, "About auction strategies for intersection management when human-driven and autonomous vehicles coexist," *Multimedia Tools and Appl.*, vol. 80, no. 10, pp. 15 921–15 936, Apr. 2021.

[39] S. M. LaValle, *Planning Algorithms*. New York, NY, USA: Cambridge University Press, 2006.

[40] D. Gonzalez, J. Perez, V. Milanes, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Trans. Intell. Transport. Syst.*, vol. 17, no. 4, pp. 1135–1145, Apr. 2016.

[41] A. Stentz, "Optimal and efficient path planning for partially-known environments," in *IEEE Int. Conf. on Robot. and Automat.*, May 1994.

[42] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Iowa State University, Ames, IA, USA, Tech. Rep. 98-11, Oct. 1998.

[43] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Phys. Rev. E*, vol. 62, no. 2, pp. 1805–1824, Aug. 2000.

[44] J. Gao, C. Sun, H. Zhao, Y. Shen, D. Anguelov, C. Li, and C. Schmid, "VectorNet: Encoding hd maps and agent dynamics from vectorized representation," in *2020 IEEE/CVF Conf. on Comput. Vision and Pattern Recognit. (CVPR)*. Seattle, WA, USA: IEEE, Jun. 2020, pp. 11 522–11 530.

[45] S. Lefèvre, D. Vasquez, and C. Laugier, "A survey on motion prediction and risk assessment for intelligent vehicles," *Robomech J*, vol. 1, no. 1, Dec. 2014.

[46] A. Haurie, J. B. Krawczyk, and G. Zaccour, *Games and Dynamic Games*, ser. World Scientific-Now Publishers Series in Business. World Scientific, Mar. 2012, vol. 1.

[47] M. Wang, Z. Wang, J. Talbot, C. J. Gerdes, and M. Schwager, "Game theoretic planning for self-driving cars in competitive scenarios," in *Robot.: Sci. and Syst. XV*. Robotics: Science and Systems Foundation, Jun. 2019.

[48] Z. Wang, T. Taubner, and M. Schwager, "Multi-agent sensitivity enhanced iterative best response: A real-time game theoretic planner for drone racing in 3D environments," *Robot. and Auton. Syst.*, vol. 125, Mar. 2020.

[49] A. Zanardi, S. Bolognani, A. Censi, and E. Frazzoli, *Game Theoretical Motion Planning: Tutorial ICRA 2021*. Zürich, CH: ETH Zurich, Aug. 2021.

[50] Y. Shoham and K. Leyton-Brown, *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge, NY, USA: Cambridge University Press, 2009.

[51] N. L. De Forest, *The Square Root of 4 to a Million Places*. Minneapolis, MN, USA: Filiquarian Publishing, LLC, Jul. 2010.

[52] S. Le Cleac'h, M. Schwager, and Z. Manchester, "ALGAMES: A fast solver for constrained dynamic games," *Robot.: Sci. and Syst. XVI*, Jul. 2020.

[53] ——, "ALGAMES: A fast augmented lagrangian solver for constrained dynamic games," *arXiv:2104.08452 [cs]*, May 2021.

[54] D. Fridovich-Keil, V. Rubies-Royo, and C. J. Tomlin, "An iterative quadratic method for general-sum differential games with feedback linearizable dynamics," in *2020 IEEE Int. Conf. on Robot. and Automat. (ICRA)*. Paris, FR: IEEE, May 2020, pp. 2216–2222.

[55] D. Fridovich-Keil, E. Ratner, L. Peters, A. D. Dragan, and C. J. Tomlin, "Efficient iterative linear-quadratic approximations for nonlinear multi-player general-sum differential games," in *2020 IEEE Int. Conf. on Robot. and Automat. (ICRA)*. Paris, FR: IEEE, May 2020, pp. 1475–1481.

[56] A. Zanardi, E. Mion, M. Bruschetta, S. Bolognani, A. Censi, and E. Frazzoli, "Urban driving games with lexicographic preferences and socially efficient nash equilibria," *IEEE Robot. Autom. Lett.*, vol. 6, no. 3, pp. 4978–4985, Jul. 2021.

[57] Q. Zhang, R. Langari, H. E. Tseng, D. Filev, S. Szwabowski, and S. Coskun, "A game theoretic model predictive controller with aggressiveness estimation for mandatory lane change," *IEEE Trans. Intell. Veh.*, vol. 5, no. 1, pp. 75–89, Mar. 2020.

[58] L. Peters, D. Fridovich-Keil, V. Rubies-Royo, C. J. Tomlin, and C. Stachniss, "Inferring objectives in continuous dynamic games from noise-corrupted partial state observations," *arXiv:2106.03611 [cs]*, Aug. 2021.

[59] S. Le Cleac'h, M. Schwager, and Z. Manchester, "LUCIDGames: Online unscented inverse dynamic games for adaptive trajectory prediction and planning," *IEEE Robot. Autom. Lett.*, vol. 6, no. 3, pp. 5485–5492, Jul. 2021.

[60] J. F. Fisac, E. Bronstein, E. Stefansson, D. Sadigh, S. S. Sastry, and A. D. Dragan, "Hierarchical game-theoretic planning for autonomous vehicles," in *2019 Int. Conf. on Robot. and Automat. (ICRA)*. Montreal, QC, CA: IEEE, May 2019, pp. 9590–9596.

[61] A. Morrissett, P. J. Martin, and S. Abdelwahed, "Socially-optimal auction-theoretic intersection management system," in *2022 IEEE Intell. Veh. Symp. (IV)*. Aachen, DE: IEEE, Jun. 2022, pp. 1340–1346.

[62] H. Pham and Q.-C. Pham, "A new approach to time-optimal path parameterization based on reachability analysis," *IEEE Trans. Robot.*, vol. 34, no. 3, pp. 645–659, Jun. 2018.

[63] D. Verscheure, B. Demeulenaere, J. Swevers, J. D. Schutter, and M. Diehl, "Time-optimal path tracking for robots: A convex optimization approach," *IEEE Trans. Autom. Control*, vol. 54, no. 10, pp. 2318–2327, Oct. 2009.

[64] Q.-C. Pham and O. Stasse, "Time-optimal path parameterization for redundantly actuated robots: A numerical integration approach," *IEEE/ASME Trans. Mechatron.*, vol. 20, no. 6, pp. 3257–3263, Dec. 2015.

[65] P. Shen, H. Zou, X. Zhang, Y. Li, and Y. Fang, "Platoon of autonomous vehicles with rear-end collision avoidance through time-optimal path-constrained trajectory planning," in *2017 11th Int. Workshop on Robot Motion and Control (RoMoCo)*, Jul. 2017, pp. 232–237.

[66] P. Shen, X. Zhang, Y. Fang, and M. Yuan, "Real-time acceleration-continuous path-constrained trajectory planning with built-in tradeoff between cruise and time-optimal motions," *IEEE Trans. Autom. Sci. Eng.*, vol. 17, no. 4, pp. 1911–1924, Oct. 2020.

[67] B. Gopalakrishnan, A. K. Singh, and K. M. Krishna, "Time scaled collision cone based trajectory optimization approach for reactive planning in dynamic environments," in *2014 IEEE/RSJ Int. Conf. Intell. Robot. and Syst.*, Sep. 2014, pp. 4169–4176.

[68] A. K. Singh and Q.-C. Pham, "Reactive path coordination based on time-scaled collision cone," *J. Guid. Control, and Dyn.*, vol. 41, no. 9, pp. 2031–2038, Sep. 2018.

[69] S. Frölander and R. Hasan, "Convex optimization-based design of a speed planner for autonomous heavy duty vehicles," Master's thesis, KTH Royal Institute of Technology, Stockholm, SE, 2019.

[70] C. Guarino Lo Bianco and M. Romano, "Optimal velocity planning for autonomous vehicles considering curvature constraints," in *Proc. 2007 IEEE Int. Conf. on Robot. and Automat.*, Apr. 2007, pp. 2706–2711.

[71] M. Kelly, "An introduction to trajectory optimization: How to do your own direct collocation," *SIAM Rev.*, vol. 59, no. 4, pp. 849–904, 2017.

[72] S. Boyd and L. Vandenberghe, *Convex Optimization.* New York, NY, USA: Cambridge University Press, 2004.

[73] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Math. Program. Comput.*, vol. 11, no. 1, pp. 1–36, 2019.

[74] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *J. Mach. Learn. Res.*, vol. 17, no. 83, pp. 1–5, 2016.

[75] U.S. Department of Transportation, "About intersection safety," 3 2021. [Online]. Available: https://safety.fhwa.dot.gov/intersection/about/

[76] A. Gholamhosseinian and J. Seitz, "A comprehensive survey on cooperative intersection management for heterogeneous connected vehicles," *IEEE Access*, vol. 10, pp. 7937–7972, Jan. 2022.

[77] Q. Jin, G. Wu, K. Boriboonsomsin, and M. Barth, "Multi-agent intersection management for connected vehicles using an optimal scheduling approach," *2012 Int. Conf. on Connected Veh. and Expo (ICCVE)*, pp. 185–190, May 2012.

[78] F. Poggenhans, J.-H. Pauls, J. Janosovits, S. Orf, M. Naumann, F. Kuhnt, and M. Mayr, "Lanelet2: A high-definition map framework for the future of automated driving," in *2018 21st Int. Conf. Intell. Trans. Syst. (ITSC).* Maui, HI, USA: IEEE, Nov. 2018, pp. 1672–1679.

[79] A. Mas-Colell, M. D. Whinston, and J. R. Green, *Microeconomic Theory.* New York, NY, USA: Oxford University Press, 1995.

[80] S. Li, J. Lian, A. J. Conejo, and W. Zhang, "The market-based coordination of distributed energy resources," *IEEE Control Syst. Mag.*, vol. 40, no. 4, pp. 26–52, 2020.

[81] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd, "A rewriting system for convex optimization problems," *J. Control Decis.*, vol. 5, no. 1, pp. 42–60, 2018.

[82] P. Alvarez Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using SUMO," in *2018 21st Int. Conf. Intell. Trans. Syst. (ITSC)*. Maui, HI, USA: IEEE, Nov. 2018, pp. 2575–2582.

[83] A. Morrissett and P. J. Martin, "Control-minimal time-assigned path-constrained trajectory optimization," in *2023 Amer. Control Conf.* San Diego, CA, USA: IEEE, May 2023, to appear in.

[84] D. Sadigh, S. Sastry, S. A. Seshia, and A. D. Dragan, "Planning for autonomous cars that leverage effects on human actions," in *Robot.: Sci. and Syst. XII*. Robotics: Science and Systems Foundation, 2016.

[85] R. A. Hearn, "The complexity of sliding-block puzzles and plank puzzles," in *Tribute to a Mathemagician*, B. Cipra, E. D. Demaine, M. L. Demaine, and T. Rodgers, Eds. Wellesley, MA, USA: A K Peters, Ltd., 2005, pp. 173–184.

[86] R. A. Hearn and E. D. Demaine, "PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation," *Theor. Comput. Sci.*, vol. 343, no. 1–2, pp. 72–96, Oct. 2005.

[87] G. W. Flake and E. B. Baum, "Rush Hour is PSPACE-complete, or "Why you should generously tip parking lot attendants"," *Theor. Comput. Sci.*, vol. 270, no. 1–2, pp. 895–911, Jan. 2002.

[88] M. Stamp, B. Engel, M. Ewell, and V. Morrow, "Rush Hour® and Dijkstra's algorithm," *Graph Theory Notes of New York XL*, pp. 23–30, 2001.

[89] X. Shen and F. Borrelli, "Multi-vehicle conflict resolution in highly constrained spaces by merging optimal control and reinforcement learning," *arXiv:2211.01487v2 [cs.RO]*, Nov. 2022.

[90] ——, "Reinforcement learning and distributed model predictive control for conflict resolution in highly constrained spaces," *arXiv:2302.01586v1 [cs.RO]*, Nov. 2023.

[91] T. Başar and G. J. Olsder, *Dynamic Noncooperative Game Theory*, 2nd ed. Philadelphia, PA, USA: SIAM, 1998.

[92] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proc. 7th Python in Sci. Conf. (SciPy2008)*, Pasadena, CA, USA, Aug. 2008, pp. 11–15.

[93] A. Morrissett and P. J. Martin, "Socially-optimal auction-theoretic intersection management," 2023, submitted for publication.

[94] N. Zohrabi, P. J. Martin, M. Kuzlu, L. Linkous, R. Eini, A. Morrissett, M. Zaman, A. Tantawy, O. Gueler, M. Al Islam, N. Puryear, H. Kalkavan, J. Lundquist, E. Karincic, and S. Abdelwahed, "Opencity: An open architecture testbed for smart cities," in *2021 IEEE Int. Smart Cities Conf. (ISC2)*. Manchester, UK: IEEE, Sep. 2021.

[95] A. Morrissett and S. Abdelwahed, "A review of non-lane road marking detection and recognition," in *2020 IEEE 23rd Int. Conf. on Intell. Transp. Syst. (ITSC)*. Rhodes, GR: IEEE, Sep. 2020.

[96] A. Morrissett, R. Eini, M. Zaman, N. Zohrabi, and S. Abdelwahed, "A physical testbed for intelligent transportation systems," in *2019 12th Int. Conf. on Human Syst. Interact. (HSI)*. Richmond, VA, USA: IEEE, Jun. 2019, pp. 161–167.

[97] A. Morrissett and S. Abdelwahed, "A physical testbed for smart city research," in *2018 IEEE/ACS 15th Int. Conf. on Comput. Syst. and Appl. (AICCSA)*. Aqaba, JO: IEEE, Oct. 2018.