

Improving Parking Lot Efficiency through Autonomous Control and Assignment Strategies: A Microscopic Traffic Simulation Analysis

Alex Wong
Murat Arcaç, Ed.
Francesco Borrelli, Ed.



Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2023-166

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-166.html>

May 12, 2023

Copyright © 2023, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

I would first like to thank my co-chairs, Professor Francesco Borrelli and Professor Murat Arcaç. They provided valuable support and insight in the development of this work, and it has been a pleasure working in Professor Borrelli's Model Predictive Control Lab for the last two years.

I also extend a large thank you to Xu Shen, who initially brought me into the lab and has mentored me every step of the way. His guidance and support have been invaluable and incomparable.

Finally, thank you to my family and friends for helping me through it all.

Improving Parking Lot Efficiency through Autonomous Control and Assignment Strategies:
A Microscopic Traffic Simulation Analysis

by

Alexander Wong

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley



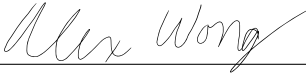
Committee in charge:

Professor Francesco Borrelli, Co-chair

Professor Murat Arcaç, Co-chair

Spring 2023

The dissertation of Alexander Wong, titled Improving Parking Lot Efficiency through Autonomous Control and Assignment Strategies: A Microscopic Traffic Simulation Analysis, is approved:

Co-chair		Date	<u>5/10/2023</u>
Co-chair		Date	<u>5/10/2023</u>
		Date	<u>5/11/2023</u>

University of California, Berkeley

Improving Parking Lot Efficiency through Autonomous Control and Assignment Strategies:
A Microscopic Traffic Simulation Analysis

Copyright 2023
by
Alexander Wong

Abstract

Improving Parking Lot Efficiency through Autonomous Control and Assignment Strategies:
A Microscopic Traffic Simulation Analysis

by

Alexander Wong

Master of Science in Computer Science

University of California, Berkeley

Professor Francesco Borrelli, Co-chair

Professor Murat Arcaç, Co-chair

This paper takes a holistic approach toward managing autonomous vehicle traffic in parking lots. Parking in lots is a crucial aspect of navigation, and a potentially time-consuming one, with drivers often having to circle around multiple times or manage complicated intersections. Parking large fleets of vehicles can be especially slow, thanks to the amount of congestion created and lack of vehicle coordination. To improve this, we present an autonomous framework for improving parking lot efficiency for a wide range of scenarios, including fleet parking. First, we develop a path planning and collision avoidance framework for individual vehicles as well as a simulation framework for managing a large group of vehicles in a decentralized manner. Then, we turn toward fleet management and improving parking efficiency for incoming fleets of autonomous vehicles. In particular, we focus on the approach of intelligently assigning spots for incoming vehicles to minimize the fleet's time to park. We pair this work with an existing parking lot dataset and run extensive simulations testing spot assignment strategies, including random assignment, assignment using a neural network, and assignment using an pre-trained driver intent prediction model. Overall, we find significant parking time savings compared to human driving, and we identify scenarios where different spot assignment strategies could be utilized.

To my parents

Contents

Contents	ii
List of Figures	iv
List of Tables	v
1 Introduction	1
2 Problem Formulation	3
2.1 Problem Setup	3
2.2 Approach	5
2.3 Dataset	7
3 Single-Vehicle Planning and Control Design	10
3.1 Path Planning	10
3.2 Path Following	13
3.3 Rule-Based Collision Avoidance	15
4 Multi-Vehicle Simulator Design	19
4.1 Simulation Types	19
4.2 Vehicle Spawning	20
4.3 Task Profile	20
4.4 Occupancy Management	21
5 Vehicle Assignment Strategy	22
5.1 Decentralized Assignment	22
5.2 Centralized Assignment at Entrance: Basic Strategies	24
5.3 Centralized Assignment at Entrance: Neural Network	27
6 Experiments	30
6.1 General Behavior	31
6.2 Occupied-Lot Recorded-Agent Experiments	32
6.3 Occupied-Lot Generated-Agent Experiments	34

6.4	Empty-lot Experiments	36
6.5	Analysis	37
7	Conclusion and Future Work	39
	Bibliography	41
A	Rule-Based Collision Avoidance Pseudocode	43
B	Electric Vehicle Charging Optimization	49

List of Figures

2.1	Simulator architecture. The parentheses indicate the sections that elaborate the design.	6
2.2	DLP dataset sample	7
2.3	DLP dataset map data	8
2.4	DLP dataset vehicle data sample	9
3.1	Path planning for cruising	11
3.2	All parking maneuver trajectories	13
3.3	“Left-north-up” trajectory and corresponding inputs	13
3.4	Collision avoidance flowchart when a vehicle is not braking	16
3.5	Collision avoidance flowchart when a vehicle is braking	17
5.1	Examples of intent prediction from [10]. The obstacles are plotted in blue, and the empty spots are plotted in green. The vehicle of interest is plotted in red with a “fading tail” as its motion history. The orange, purple, and dark green stars are the top-3 predicted intent at the current time step. The values beside each star show the probability of the corresponding intent.	23
5.2	A sample of the parking map during a simulation with hand-picked spot assignment. The circled numbers denote the assigned spot of each vehicle.	27
6.1	Sample DLP dataset scene (Scene 22)	33
6.2	Recorded-agent results with all algorithms	33
6.3	Scene 12 obstacles	34
6.4	Generated-agent hyperparameter survey	35
6.5	Generated-agent spawn interval survey	36
6.6	Empty-lot hyperparameter survey	37
6.7	Empty-lot spawn interval survey	38
A.1	Visualization of Algorithm 6, where vehicle i has passed vehicle j	46
A.2	A vehicle with its parking box (in red)	47
A.3	Visualization of Algorithm 9, where vehicle i has priority over vehicle j	48

List of Tables

4.1	Task parameters	21
4.2	Task types	21
6.1	DLP dataset scenes for recorded-agent experiments	32

Acknowledgments

I would first like to thank my co-chairs, Professor Francesco Borrelli and Professor Murat Arcak. They provided valuable support and insight in the development of this work, and it has been a pleasure working in Professor Borrelli's Model Predictive Control Lab for the last two years.

I also extend a large thank you to Xu Shen, who initially brought me into the lab and has mentored me every step of the way. His guidance and support have been invaluable and incomparable.

Finally, thank you to my family and friends for helping me through it all.

Chapter 1

Introduction

Autonomous vehicle (AV) control is one of the most fascinating fields in technology today. With the prevalence of driving in everyday life, AV development has the potential to change the way humans fundamentally operate. However, while much research and industry work has been done with AVs on road networks, there has been minimal research on these vehicles in tight environments, and particularly parking lot environments.

There has been previous research on certain elements of navigating parking lots. Some work focuses on localization, where vehicles try to determine their location using sensors and vision systems; this can be especially relevant in parking lots due to a potential lack of GPS in concrete structures. To that end, Qin et al. developed a SLAM (simultaneous localization and mapping) method for parking lots with semantic mapping [8], and Hou et al. introduced an underground parking lot dataset for this type of research [5]. Our work does not focus on localization, but it is nevertheless an interesting topic and may prove helpful to implement when certain data about vehicles is not available.

Other research targets autonomous control in parking lots. These works especially analyze path planning and collision avoidance, which can prove challenging in tight environments with a wide variety of vehicle activity. Li et al. proposed a behavior prediction-based algorithm for navigating lots with a central infrastructure for assigning spots, as well as a model predictive control (MPC) approach for local collision avoidance [7]. Chi et al. tackled a similar problem while using a modified version of A* search for path planning [1]. Similar to this work, these papers present novel control frameworks for parking lots, although they do not include much discussion about broader fleet management.

However, there has been work done toward developing improved policies for parking entire fleets of AVs. This is necessary because managing vehicle interactions becomes increasingly difficult (and potentially time- and energy-consuming) when large numbers of vehicles are involved. Therefore, additional strategies must be created and empirically tested in this specific environment. As one example, Shen et al. tested different fleet parking strategies in a small parking lot environment and introduced a grid-based form of collision avoidance [9].

Overall, this research identifies ways to solve individual aspects of managing AVs, but none tie together planning, collision avoidance, and fleet management with a large dataset for

experiments. This integration work is critical, as the combination of these elements results in extra considerations, such as developing efficient collision avoidance with multiple vehicles in a dense environment or planning paths that result in fast parking for fleets.

Compared to traditional road networks, there are additional challenges posed by controlling AVs in parking lots. First, the precise destination of a parking vehicle is not predetermined; it is deduced after entering the lot as the vehicle drives around and finds which spots are available. This adds an additional element of uncertainty to route planning. Second, parking lots have mostly informal rules, with no traffic lights and often minimal stop signs or lane markings. This makes it difficult to define a standardized driving and collision avoidance control paradigm for all parking lots. Third, while driving through the aisles (non-spots) of a parking lot could be considered a typical AV task, entering and exiting a parking spot is a complex process that often requires multiple turns and changes in direction.

Another topic in the realm of AV control in parking lots is the management of fleets. The possibilities of utilizing large autonomous fleets are endless, ranging from mass movement of inventory to more efficient public transportation. Like single vehicles, these fleets must be able to park as a group in a timely manner, but the design of parking lots poses a challenge; traffic is much choppier due to vehicles constantly stopping and starting, and traffic jams can quickly build up if a vehicle has to stop or park in an inopportune place. This demonstrates the necessity of an efficient fleet parking solution as AV technology develops.

It is easy to imagine solutions for fleet parking; for example, vehicles could park near the front, or spread out to minimize traffic jams. However, performing analysis on these policies would require a novel closed-loop parking simulator, as the interaction of vehicles in a parking environment, particularly large numbers of vehicles, is extremely complex. In this report, we work toward developing this solution, attacking the problem of fleet parking through the following contributions:

1. We present a closed-loop method for controlling AVs in parking lots, including realistic but systematic decision-making.
2. We present a survey of algorithms for optimizing fleet parking in a wide variety of parking and traffic scenarios.
3. We demonstrate our solution on an existing dataset, resulting in significant parking time saved across a large fleet.

Chapter 2

Problem Formulation

2.1 Problem Setup

We aim to minimize the time autonomous vehicles spend parking in parking lots. We address this problem by simulating the movement of a large fleet in a real-world parking lot and studying the effect of motion control design and different spot assignment strategies.

Inputs

A parking lot contains parking spots \mathcal{S} , static vehicles \mathcal{O} , moving vehicles \mathcal{A} , and an entrance e with coordinates $(x_e, y_e) \in \mathbb{R}^2$.

\mathcal{S} is the set of all parking spots. A parking spot $s \in \mathcal{S}$ is defined by its center $(x^{[s]}, y^{[s]}) \in \mathbb{R}^2$ and its width and length $(w^{[s]}, l^{[s]}) \in \mathbb{R}^2$. Its bounding box is defined by $\mathbb{B}_{\text{spot}}^{[s]}$. In this paper, we only work with parking spots that are perpendicular to driving lanes, so the heading angle of each parking spot is ignored.

\mathcal{O} is the set of all static vehicles, hereby referred to as obstacles. These are vehicles that do not move for the entirety of the experiment time, so they are treated as static. An obstacle $o \in \mathcal{O}$ is defined by its center $(x^{[o]}, y^{[o]}) \in \mathbb{R}^2$, its width and length $(w^{[o]}, l^{[o]}) \in \mathbb{R}^2$, and its heading angle $\psi^{[o]} \in \mathbb{R}$. Its bounding box is defined by $\mathbb{B}_{\text{obs}}^{[o]}$.

\mathcal{A} is the set of all moving vehicles, hereby referred to as agents. These are vehicles that move during the experiment time. The state $z^{[i]}(t)$ of an agent $i \in \mathcal{A}$ at time t is defined by its center position $(x^{[i]}(t), y^{[i]}(t)) \in \mathbb{R}^2$, its width and length $(w^{[i]}, l^{[i]}) \in \mathbb{R}^2$, its heading angle $\psi^{[i]}(t) \in \mathbb{R}$, and its speed $v^{[i]}(t)$. Its bounding box is defined by $\mathbb{B}^{[i]}$. Each agent also has a time $T_{\text{start}}^{[i]}$ when it enters the parking lot.

For brevity, we group the agents into disjoint sets: $\mathcal{A}_{\text{enter}}$ are agents which start at the entrance e and end in a parking space, $\mathcal{A}_{\text{exit}}$ are agents which start in a parking space and exit at the entrance e , and $\mathcal{A}_{\text{other}}$ are agents with more complex behavior.

Outputs

For each agent $i \in \mathcal{A}$ at time t , we output control commands consisting of acceleration $a^{[i]}(t)$ and steering angle $\delta^{[i]}(t)$ which drive the vehicle along the path we have determined.

Objective

For an agent $i \in \mathcal{A}$, define $T_{\text{end}}^{[i]}$ as the time at which agent i finishes its maneuver (e.g. enter the lot and park). Its *elapsed driving time* in the lot is defined as $T_{\text{end}}^{[i]} - T_{\text{start}}^{[i]}$. We aim to minimize the elapsed driving time across all vehicles that enter the lot and park, as described in (2.1).

$$\sum_{i \in \mathcal{A}_{\text{enter}}} T_{\text{end}}^{[i]} - T_{\text{start}}^{[i]} \quad (2.1)$$

Theoretically, we can formulate this problem as a large optimization problem. However, it is infeasible in practice, as vehicle interactions in tight environments are chaotic and the problem would quickly become impossible to solve with efficiency. Therefore, instead of explicit optimization, we conduct simulation studies to decrease this objective value through the design of vehicle motion controllers and parking spot assignment algorithms.

Communication Assumptions

As we develop our parking controller, we make some assumptions about information availability in the parking lot.

1. Vehicle-to-Infrastructure (V2I): Each agent has information about a map of the parking lot, which includes driving lanes and parking spot locations.
2. Vehicle-to-Vehicle (V2V): Each agent has information about other vehicles in the lot, static and moving, as well as information related to their autonomous navigation (acceleration, braking, reference trajectory). This could be achieved in a variety of ways, including passing the information directly, using a central infrastructure in the parking lot, or collecting information from sensors and prediction models.

However, note that each agent still computes its trajectory independently and the central infrastructure only acts as an information hub. This is a conscious design choice. In theory, it is possible to have a central computer calculate the optimal control commands for all vehicles in the lot, and relay this to the individual vehicles. However, this has a couple of issues. First, the computational burden would be very large, as the space of optimization across all vehicles is massive. Second, this paradigm is not robust to changes in a dynamic parking lot, as an entering vehicle, perhaps at a previously unknown time, could drastically change the optimal path for all vehicles. We conclude that it is best to have all vehicles perform computations in a decentralized manner.

2.2 Approach

To analyze the problem of efficient fleet parking, we take two broad steps:

1. **Simulation Development:** Develop a simulated closed-loop environment where autonomous vehicles can navigate through a parking lot and interact with other agents, with access to appropriate V2I and V2V information.
2. **Vehicle Assignment Strategy:** Devise algorithms to assign parking spots to vehicles in the fleet such that the elapsed driving time can be minimized.

Simulation Development

Our multi-agent simulator can be viewed as a system for managing a set of independent vehicles as they navigate a parking lot. Each simulation proceeds as follows: the simulator gathers information about the parking lot it is simulating (spot locations, parked vehicles, etc.), then spawns a series of vehicles in the lot and acts as a central infrastructure for those vehicles. Each spawned vehicle, which is given a rough outline of its behavior by the simulator, then computes its navigation in a decentralized manner. A diagram of the simulator structure is presented in Figure 2.1.

Single-Vehicle Planning and Control Design

In Chapter 3, we present the control method used by each individual vehicle. The vehicle first plans its path (Section 3.1), determining the route it will take through the aisles of the lot (“cruising”) and the way it will enter into the spot from the aisle (“maneuvering”). Then, it follows that path (Section 3.2), using different controllers for cruising and maneuvering. The vehicle also avoids collisions while following the path using V2V communication (Section 3.3).

Multi-Vehicle Simulator Design

In Chapter 4, we present the system for managing the environment where the vehicles drive and interact. After creating the lot and loading information about the vehicles in the lot (Section 4.1), the simulator’s primary job is to spawn vehicles (Section 4.2) and assign them their overarching behavior, or “task profile” (Section 4.3). An example of a task profile is to spawn at the entrance, drive to a certain spot, and park there. Note that this is different from path planning and following, which take this task profile (that does not contain any actuation commands) and determine how to execute it. Overseeing all of the vehicles is an occupancy manager that ensures multiple vehicles do not attempt to enter the same spot or collide with an exiting vehicle in a spot (Section 4.4).

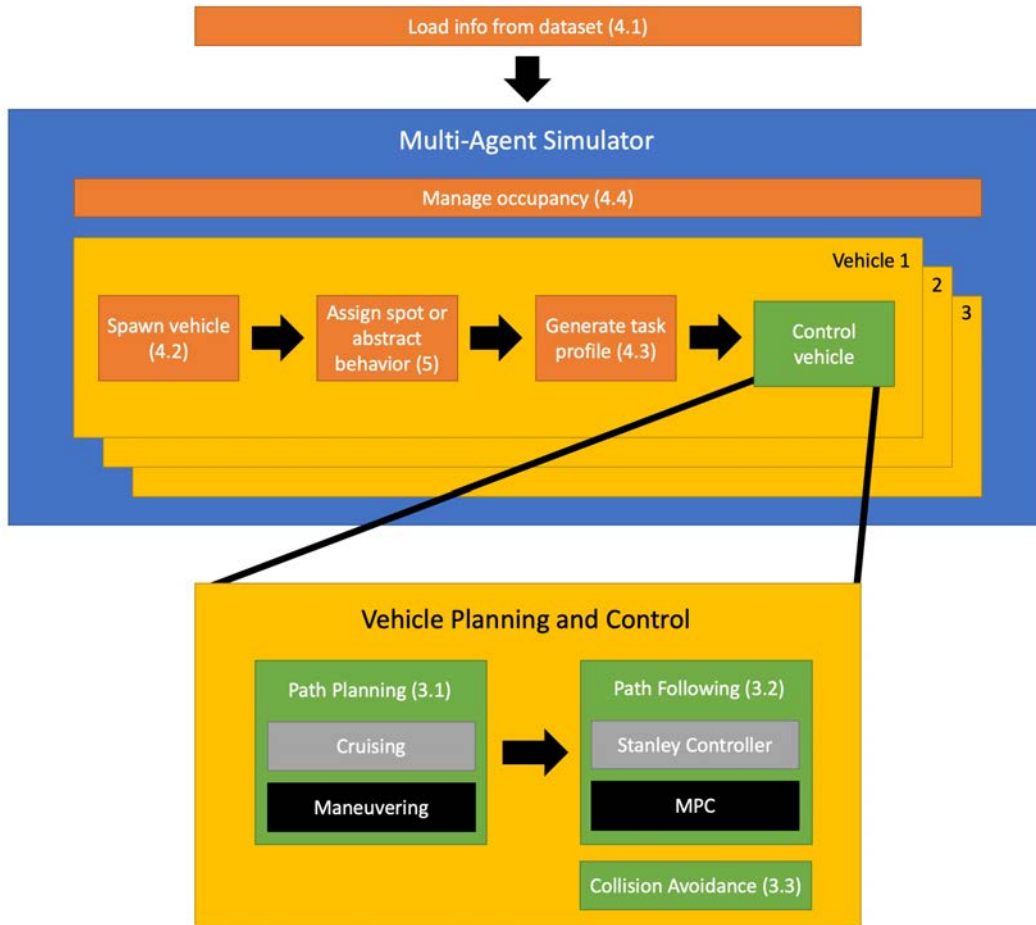


Figure 2.1: Simulator architecture. The parentheses indicate the sections that elaborate the design.

Vehicle Assignment Strategy

After developing a simulator, we use it as an environment to test fleet coordination strategies for efficient parking. There are many ways to tackle this problem; for instance, we could have vehicles take alternate, but more efficient routes to park or learn algorithms that take advantage of certain features of the lot. In this paper, the strategy we have chosen is to intelligently select which spot each incoming vehicle parks in. For instance, generally speaking, vehicles that park closer to the entrance will park faster than vehicles that park farther away.

We present the algorithms for spot assignment in Chapter 5. There are two types of strategies. One is to have each vehicle determine its parking spot in a decentralized manner

without coordination through the lot’s central infrastructure (Section 5.1). The other is to use the simulator’s central infrastructure, which contains knowledge about the lot and the vehicles in it, to intelligently pick a spot for each incoming vehicle when it enters the lot (Sections 5.2 and 5.3).

2.3 Dataset

In developing our multi-agent simulation, we worked in conjunction with the existing Dragon Lake Parking (DLP) ¹ dataset [10], using its parking lot map, trajectories of human-driven vehicles, and visualization features. Using a real-world dataset throughout our work was crucial for a few reasons. First, we needed to ensure our parking environment was realistic, from the design of the lot to the density and location of the vehicles in it. Second, it was important to have a human behavior-driven baseline, as it allowed us to easily create comparisons with autonomous control.

Dataset Info

The DLP dataset is one of the largest public datasets for parking in the world. Collected over the course of 3.5 hours, it contains annotated data of vehicles, pedestrians, and bicycles inside of a busy parking lot. A sample of the annotated data is presented in Figure 2.2. For our simulation, we only utilize the lot map and the trajectories of human-driven vehicles.

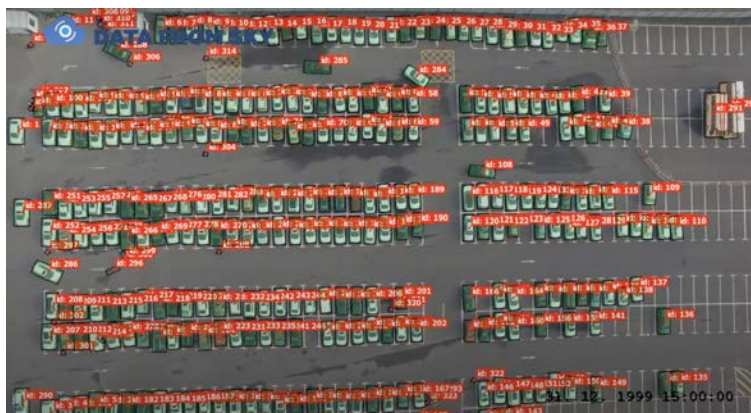


Figure 2.2: DLP dataset sample

¹<https://sites.google.com/berkeley.edu/dlp-dataset>

Map Data

The DLP dataset contains information about the Dragon Lake parking lot map. Namely, it includes the dimensions of the lot, the location and orientation of each parking spot, and a set of waypoints that define the driving lanes (“aisles”) of the lot. Figure 2.3 is a matplotlib [6] visualization of the lot, with parking spots outlined in gray and waypoints in green.

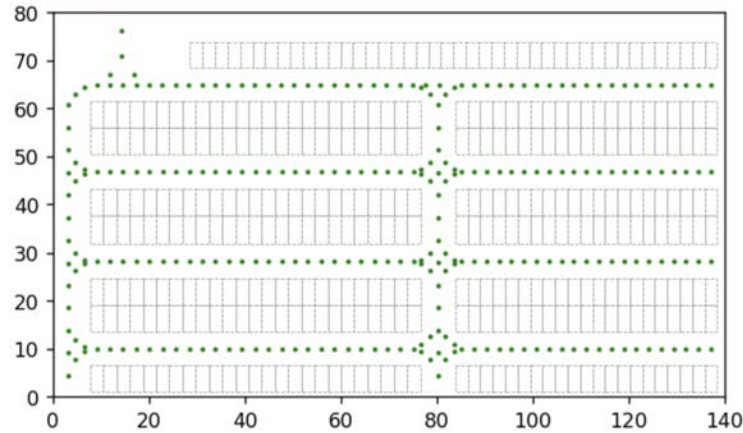


Figure 2.3: DLP dataset map data

Trajectories of Human-Driven Vehicles

The behaviors of human-driven vehicles are recorded as a sequence of frames over time. Each frame contains the size, location, heading angle, velocity, and acceleration of all vehicles in the lot at that time. Vehicles are divided into static obstacles and non-static agents, similar to our formulation. Figure 2.4 visualizes a sample frame, with obstacles drawn as blue rectangles and agents drawn as yellow rectangles.

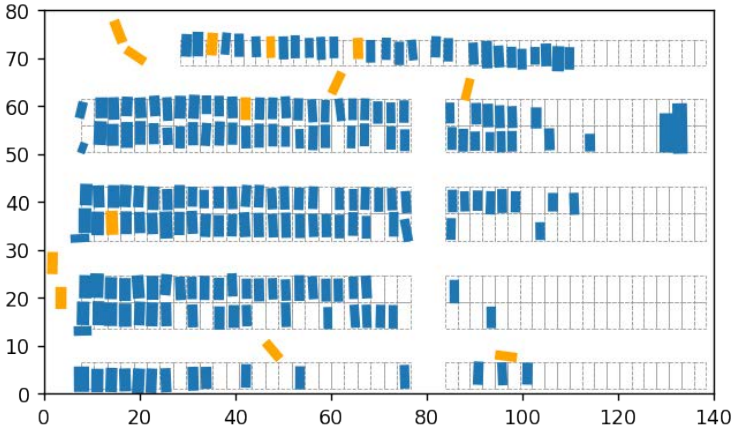


Figure 2.4: DLP dataset vehicle data sample

Chapter 3

Single-Vehicle Planning and Control Design

Before we can develop techniques for fleet parking, we must create an efficient control method for single vehicles. Each agent (moving vehicle) in the parking lot must accomplish the same core mission: navigate from one part of the lot to another, possibly entering or exiting a parking spot, while avoiding collisions with other vehicles. We split up the navigation task into two parts: planning an optimal path through the lot (Section 3.1), then following that path with feedback controllers (Section 3.2). Collision avoidance, which we perform by outlining a set of rules based on a vehicle and its surroundings, is presented in Section 3.3.

3.1 Path Planning

Planning the path of travel in parking lots can be broken up into two distinct steps:

1. Cruising: Driving through the aisles of the parking lot while navigating from one part of the lot to another.
2. Maneuvering: Entering or exiting a parking spot.

We use A* route planning and optimization-based collision avoidance (OBCA) [11] for these tasks, respectively.

Cruising

Most of a vehicle's time in a parking lot is spent traversing the lot's aisles while navigating to or looking for a spot. This task can be re-framed as traveling from one location to another location along waypoints that follow the aisles of the lot, such as those presented in the DLP dataset.

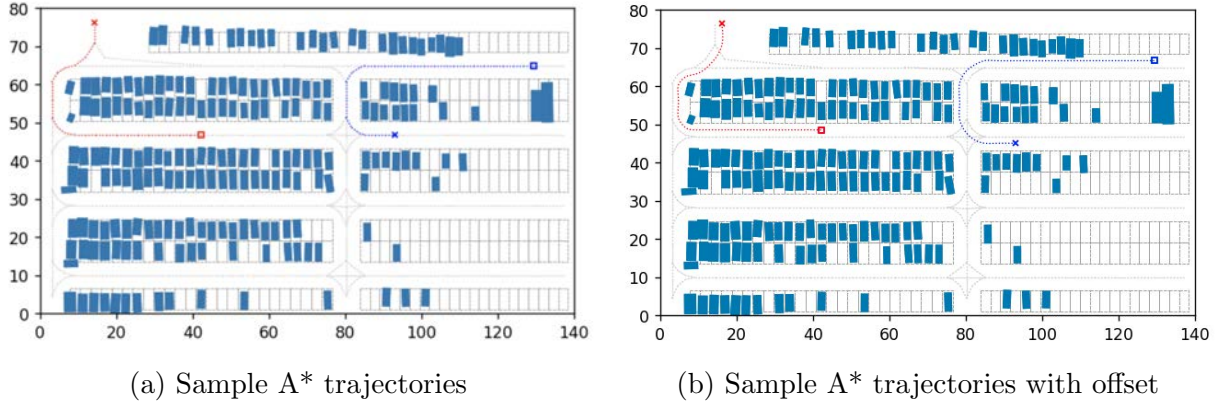


Figure 3.1: Path planning for cruising

For planning the path of travel, we use A* search [3]. Figure 3.1a shows two sample A* paths for the DLP dataset, with the path in red going from a spot to the entrance and the path in blue going between spots.

One complication is that vehicles cannot drive directly in the middle of the aisles, since two vehicles traveling in the same aisle, but in opposite directions, would collide. So, after searching a reference path p of length N_{ref} (i.e. the paths in Figure 3.1a), we build in an offset $\Delta_{\text{off}} \in \mathbb{R}$ from the center of the lane such that passing vehicles do not intertwine, resulting in a final reference path p^* that the vehicle follows. For each point $(x_{\text{ref}}, y_{\text{ref}}, \psi_{\text{ref}}) \in p$, the corresponding point in p^* is given by (3.1). A visualization of the paths in Figure 3.1a with $\Delta_{\text{off}} = 1.75$ is shown in Figure 3.1b.

$$(x_{\text{ref}} - \sin(\psi_{\text{ref}})\Delta_{\text{off}}, y_{\text{ref}} + \cos(\psi_{\text{ref}})\Delta_{\text{off}}, \psi_{\text{ref}}) \quad (3.1)$$

Maneuvering

Once a vehicle has traveled to its intended spot and wants to park, it must intentionally leave its aisle to enter the spot, a task that cannot be accomplished with the existing waypoints along the road. Instead, we must create new paths that model how a vehicle travels from its A* terminus into a spot. Typically, the spaces around the parking spots would be tight, so the paths should be dynamically feasible and collision-free for vehicles to follow.

Since all parking spots in a given lot are similarly oriented, the paths to enter or leave the spots can be pre-determined independent of the spot, meaning they can be calculated once offline for each parking lot. For the DLP dataset, we compute 8 such sets of paths, hereby called “parking maneuvers.” Each maneuver is built on three criteria:

1. Starting location (left or right): on which side of the spot does the vehicle start its maneuver?

2. Chosen spot (north or south): does the vehicle need to travel north (up on the visualization) or south (down on the visualization) to park?
3. Final heading (up or down): which direction does the vehicle end the maneuver facing?¹

To calculate the path for a given maneuver, we define the vehicle’s state at the start and end of the maneuver to be z_0 and z_F , and use the kinematic bicycle model $\dot{z} = f(z, u)$ as the vehicle dynamics:

$$\dot{z} = f(z, u) := \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} v \cos(\psi) \\ v \sin(\psi) \\ \frac{v}{l_{\text{wb}}} \tan(\delta) \\ a \end{bmatrix}, \quad u = \begin{bmatrix} a \\ \delta \end{bmatrix}. \quad (3.2)$$

Then, we find the optimal path and corresponding control commands by solving the following trajectory planning problem:

$$\min_{\mathbf{z}, \mathbf{u}, T} J = \int_{t=0}^T c(z(t), u(t)) dt \quad (3.3a)$$

$$\text{s.t. } \dot{z}(t) = f(z(t), u(t)), \quad (3.3a)$$

$$z(t) \in \mathcal{Z}, u(t) \in \mathcal{U}, \quad (3.3b)$$

$$z(0) = z_0, z(T) = z_F, \quad (3.3c)$$

$$\text{dist} \left(\mathbb{B}(z(t)), \mathbb{B}_{\text{obs}}^{[o]} \right) \geq d_{\min}, \forall o, \quad (3.3d)$$

where the state z and input u are constrained under operation limits \mathcal{Z} and \mathcal{U} . We denote by $\mathbb{B}(z(t))$ the vehicle body at time t and ask it to maintain a safety distance d_{\min} away from all obstacles $o \in \mathcal{O}$. The stage cost $c(\cdot, \cdot)$ can encode the amount of actuation, energy consumption, and time consumption. The optimal solution $\{\mathbf{z}^*, \mathbf{u}^*\}$ is the optimal path for the given maneuver. All eight maneuvers and their given inputs are shown in Figure 3.2. Figure 3.3 shows a sample trajectory and the corresponding optimal inputs.

Even though each parking maneuver assumes the vehicle is approaching the spot from the left side, these 8 maneuvers are enough to accommodate any possible parking scenario². If the vehicle is approaching from the right, we simply mirror one of the existing maneuvers. Also, to calculate the trajectory of a vehicle exiting a spot into an aisle, we can use the same parking maneuver to enter the spot, except in reverse.

From this point, the act of entering a spot from an aisle will be referred to as “in-parking” and the act of exiting a spot into an aisle will be referred to as “un-parking.” Both fall under the umbrella of maneuvering.

¹To mirror the natural variety of parking by backing into or driving forward into a spot, we have half of the simulated vehicles park in the “up” heading and half park in the “down” heading.

²For the DLP dataset, some parking spots would require different parking maneuvers to enter and exit because of the nature of the nearby waypoints. For simplicity, we do not have vehicles park in these spots.

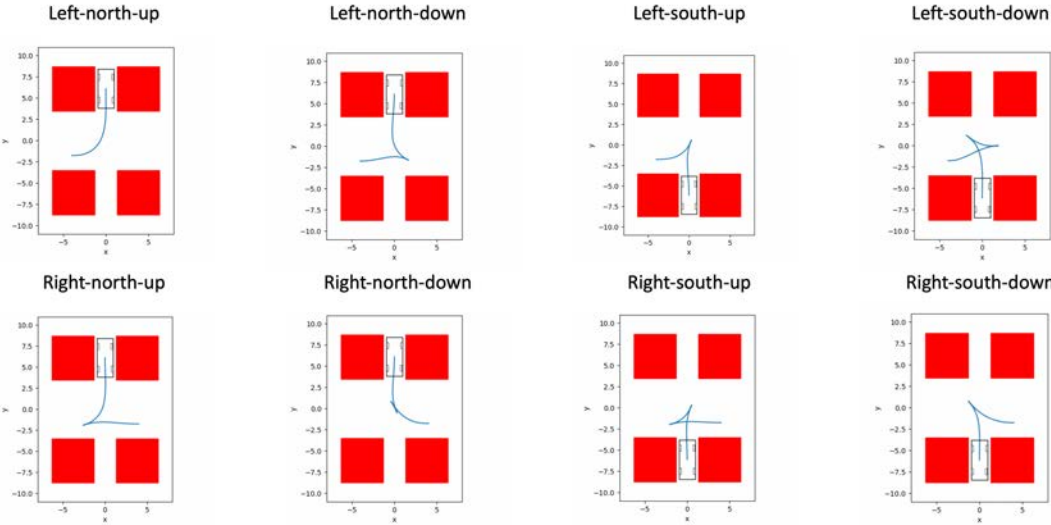


Figure 3.2: All parking maneuver trajectories

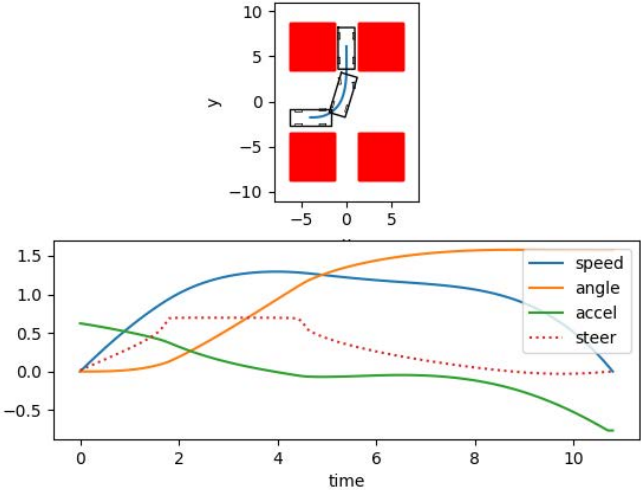


Figure 3.3: “Left-north-up” trajectory and corresponding inputs

3.2 Path Following

After planning a vehicle’s path by defining a set of waypoints, we use two algorithms to follow that path: a Stanley controller for cruising and an MPC controller for parking maneuvers.

Stanley Controller

There are many ways to follow a long trajectory defined by a set of waypoints, such as the one returned by our A* search. One of the best controllers is the Stanley controller [4], which was built for following rapidly-varying trajectories in real-time. Given a set of waypoints, it defines a control law to determine the vehicle's acceleration a and steering angle δ :

$$a(t) = k_p(v(t) - v_{ref}(t)) + k_i e_{int}(t), \quad (3.4a)$$

$$e_{int}(t) = e_{int}(t-1) + v(t) - v_{ref}(t), \quad (3.4b)$$

$$\delta(t) = \psi(t) + \arctan \frac{ke(t)}{v(t)}, \quad (3.4c)$$

where for each vehicle, v is velocity, v_{ref} is the reference velocity, e_{int} is the integrated velocity error, e is the crosstrack error, and ψ is the heading error of the vehicle with respect to the closest segment of the reference trajectory. For parameters, we set $k_i = 0$ (ignoring the integrated error), $k = 0.5$, and $k_p = 1$ if the vehicle is not braking and $k_p = 5$ if the vehicle is braking. We also enforce maximum inputs of $|a| \leq 10 \text{ m/s}^2$ and $|\delta| \leq 40^\circ$.

Each vehicle uses this controller to navigate through the aisles of the parking lot, following the offset reference trajectory defined in Equation (3.1).

Model Predictive Control

The Stanley controller is great for following a path efficiently with a simple control law. However, it does not handle collision avoidance. While driving through aisles, this issue can be resolved through a rule-based system, presented in Section 3.3. For maneuvering, we desire more precise control because of the added presence of static obstacles (parked cars).

To optimize our path following of a parking maneuver while keeping our cost of computation reasonable, we use model predictive control (MPC). MPC aims to follow a reference trajectory as close as possible while satisfying constraints by constantly solving an optimal control problem over a limited time horizon.

For our problem, a maneuvering vehicle's MPC trajectory is taken from the corresponding offline parking maneuver, and its constraints are to avoid other vehicles. The MPC formulation is as follows:

$$\begin{aligned}
\min_{\mathbf{z}_{\cdot|t}, \mathbf{u}_{\cdot|t}} \quad & J = \sum_{k=0}^{N-1} (x_{k|t} - x_{\text{ref}}(k|t))^2 + (y_{k|t} - y_{\text{ref}}(k|t))^2 \\
\text{s.t.} \quad & z_{k+1|t} = f(z_{k|t}, u_{k|t})\Delta_t, & (3.5a) \\
& z_{k|t} \in \mathcal{Z}, u_{k|t} \in \mathcal{U}, & (3.5b) \\
& z_{0|t} = z(t), & (3.5c) \\
& \text{dist} \left(\mathbb{B}(z_{k|t}), \mathbb{B}_{\text{obs}}^{[o]} \right) \geq d_{\min}, \forall o, & (3.5d) \\
& \forall k = 0, \dots, N-1,
\end{aligned}$$

where t is the current time relative to the beginning of the parking maneuver, f is the kinematic bicycle model (3.2), Δ_t is the time step size, and the state $z = [x, y, \psi, v]^T$ and input u are constrained under operation limits \mathcal{Z} and \mathcal{U} . As before, we denote by $\mathbb{B}(z_{k|t})$ the vehicle body at time k and ask it to maintain a safety distance d_{\min} away from all obstacles $o \in \mathcal{O}$. The stage cost is solely dependent on the squared distance to the corresponding part of the parking maneuver, given by $z_{\text{ref}} = [x_{\text{ref}}, y_{\text{ref}}, \psi_{\text{ref}}, v_{\text{ref}}]^T$. To create the collision avoidance constraints for Equation (3.5d), we use the formulation devised by Firoozi et al. [2]. After solving (3.5), we apply the first input $u_{0|t}^*$ to the vehicle.

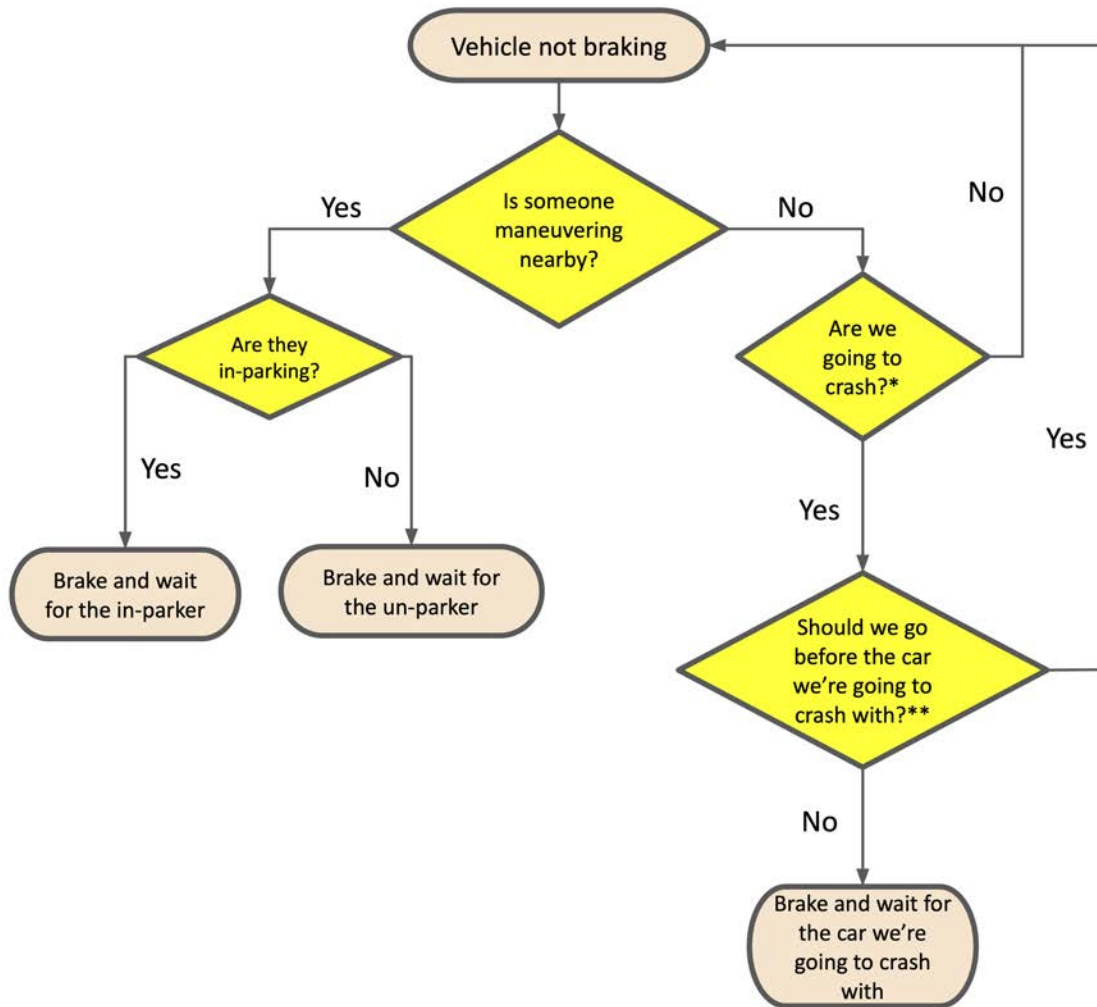
Note: Despite choosing a limited time horizon of one second, the computation required to solve the parking MPC problem is substantially more than the Stanley controller. While this would not be a problem in a real-life decentralized hardware experiment, it can cause a slowdown in a simulation, where all control is eventually computed by the CPU of one machine. To account for this, it is also acceptable to directly teleport the vehicle along the offline-planned parking maneuver in large-scale simulations. Since the offline maneuvers are computed to be kinematically feasible and collision-free, teleportation can still lead to realistic simulation results.

3.3 Rule-Based Collision Avoidance

Collision avoidance in parking lots is different from standard streets; with no streetlights and often no stop signs, traffic is much more fluid and avoiding crashes is more complicated. We have devised a decentralized rule-based collision avoidance algorithm to encourage as quick of traffic flow as possible. The algorithm operates on the following principles:

1. Vehicles should not brake unless they are going to collide with another vehicle in the immediate future.
2. At an intersection, the vehicle with the right of way is the vehicle that is further into the intersection.
3. Cruising vehicles in aisles have priority over vehicles that are maneuvering.

A rough flowchart of the rule-based collision avoidance algorithm is provided in the figures below. A vehicle is in one of two states: either it is driving normally along its prescribed path toward its destination (Figure 3.4), or it is braking to avoid a potential collision and waiting for another vehicle to no longer be a collision threat (Figure 3.5).

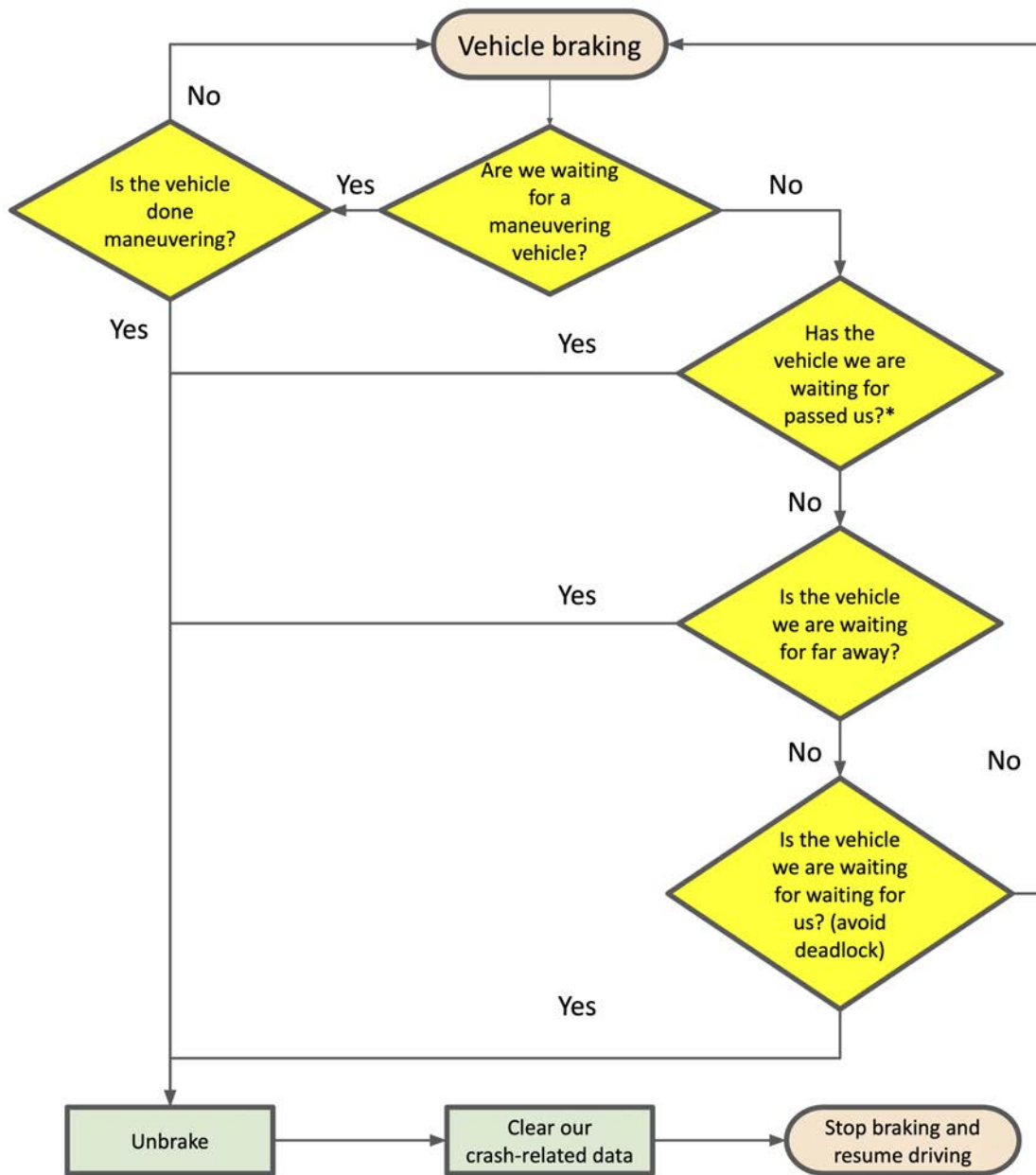


*Will crash = our polytope will intersect with another vehicle’s polytope

**Should go before = the angle between the line connecting the center of the vehicles and our heading is greater than the angle between the line connecting the center of the vehicles and the other vehicle’s heading

Figure 3.4: Collision avoidance flowchart when a vehicle is not braking

Also, Algorithm 1 provides the pseudocode for the algorithm for a vehicle $i \in \mathcal{A}$. Lines 4-8 manage reference speed, lines 9-10 detect if i should brake for nearby parking vehicles,



*Passed us = the angle between each of the rear corners of the vehicle we are waiting for and each of the front corners of our vehicle is greater than 90 degrees

Figure 3.5: Collision avoidance flowchart when a vehicle is braking

lines 11-14 detect if i is going to crash into any other vehicles (and should brake as a result), and lines 17-21 describe logic for resuming driving when in a braked state. For

constants, v_{cruise} is a i 's default speed, v_{end} is the speed i slows to as it approaches its assigned parking spot, and d_{buffer} , which is used for maneuvering collision avoidance, is the distance a vehicle must drive past a maneuvering vehicle before we no longer consider it in the collision avoidance logic. Sub-functions `should_resume_driving`, `close_to_assigned_spot`, `within_parking_box`, `has_passed`, and `should_go_before` are defined in Appendix A.

Algorithm 1 `standard_driving_control(i : vehicle)`

```

1:           ▷ Drive along designated waypoints to parking spot, avoiding collisions
2:           ▷ Constants:  $v_{\text{end}}$ ,  $v_{\text{cruise}}$ ,  $d_{\text{buffer}}$ 
3: if  $v_{\text{ref}}^{[i]} > 0$ , meaning  $i$  is not braking then
4:   if  $i$  is near the spot it will park in, given by close_to_assigned_spot( $i$ ) then
5:     Set  $v_{\text{ref}}^{[i]} = v_{\text{end}}$  in (3.4)
6:   else
7:     Set  $v_{\text{ref}}^{[i]} = v_{\text{cruise}}$  in (3.4)
8:   end if
9:   if  $\exists$  vehicle  $j \in \mathcal{A}$  that is maneuvering, and it is nearby, given by
   within_parking_box( $i, j$ ), and  $i$  has not already passed by  $j$ , given by not
   has_passed( $i, j, d_{\text{buffer}}$ ) then
10:    Brake and wait for  $j$ , set  $v_{\text{ref}}^{[i]} = 0, k_p = 5$  in (3.4)
11:   else           ▷ No nearby maneuvering vehicles
12:      $s_{\text{crash}} \leftarrow \text{will\_crash\_with}( $i$ )$ , which computes vehicles  $i$  is close to colliding with
13:     if  $i$  has lowest priority amongst all vehicles in  $s_{\text{crash}}$ , given by not
     should_go_before( $i, j$ )  $\forall j \in s_{\text{crash}}$  then
14:       Brake and wait for a random  $j \in s_{\text{crash}}$ , set  $v_{\text{ref}}^{[i]} = 0, k_p = 5$  in (3.4)
15:     end if
16:   end if
17: else           ▷  $i$  is braking and waiting for another vehicle  $j \in \mathcal{A}$ 
18:   if  $j$  is no longer a threat to crash with  $i$ , given by should_resume_driving( $i, j$ ) then
19:     Stop braking, set  $v_{\text{ref}}^{[i]} = v_{\text{cruise}}, k_p = 1$  in (3.4)
20:   end if
21: end if

```

Chapter 4

Multi-Vehicle Simulator Design

After developing a single vehicle control framework, we create the environment to test entire fleets of vehicles. This involves loading a parking lot map and the pre-recorded trajectories of human drivers (Section 4.1), spawning vehicles in that lot (Section 4.2) and assigning them high-level behavior abstractions in the form of a “task profile” (Section 4.3), and maintaining a central occupancy manager to ensure vehicles do not have conflicting selections of parking spots (Section 4.4).

4.1 Simulation Types

We have access to the entire DLP dataset, which includes parking lot specifications as well as full trajectory data for obstacles and agents. We use varying amounts of this data to generate different real-world scenarios from which we measure our objective, the average parking time of entering vehicles. We have two primary types of simulations:

1. Occupied-lot: the lot begins partially occupied with some static obstacles that never leave their parking spot. The obstacle locations are taken from the dataset. This simulation type is further broken down into two subtypes, which determine how agents are spawned.
 - a) Recorded-agent: agents spawn at the same time and location as in the dataset. This is useful for producing a direct comparison of our algorithms to human driving.
 - b) Generated-agent: we determine the spawn time and location of agents¹. This is useful for creating our own traffic patterns, but simulating the occupancy of a busy lot.

¹Vehicles from the dataset have a given size that mirrors their real-world counterparts. For generated agents, we assume all vehicles have the same size; for a generated vehicle i , we set $w^{[i]} = 1.85$ and $l^{[i]} = 4.6$.

2. Empty-lot: neither obstacles nor agents from the dataset are used; the lot starts empty (except for exiting vehicles we generate). This is useful for simulating an empty lot or for testing mass entrances by forcing the vehicles to make tougher decisions about where they park.

4.2 Vehicle Spawning

For all simulation types except recorded-agents, we must decide how to spawn vehicles, both at the entrance and in spots. We use an exponential distribution with mean λ_{spawn} , which generates some randomness while allowing us to control the general amount of traffic in the lot. Once an agent $i \in \mathcal{A}_{\text{enter}}$ spawns at time $T_{\text{start}}^{[i]}$, the next entering agent $j \in \mathcal{A}_{\text{enter}}$ will spawn at $T_{\text{start}}^{[j]} = T_{\text{start}}^{[i]} + T_{\text{interval}}$, where $T_{\text{interval}} \sim \exp(\frac{1}{\lambda_{\text{spawn}}})$. The same rule applies for exiting agents. Entering agents spawn at the entrance e and exiting agents spawn in random spots.

As a small technicality, if i cannot spawn at time $T_{\text{start}}^{[i]}$ because there is another entering agent blocking the entrance, i will be placed on a queue Q . Once the entrance is free and there is no one in front of i in the queue, it will spawn, and $T_{\text{start}}^{[i]}$ will be set to the current time.

4.3 Task Profile

One of the core questions we are trying to answer is the effectiveness of autonomous driving in saving elapsed driving time. Answering this through a simulation requires modeling the high-level behavior abstraction of each vehicle, which would then allow for direct comparisons of accomplishing the same behaviors between human driving and autonomous driving.

To implement this, each agent $i \in \mathcal{A}$ maintains a “task profile” $\rho^{[i]}$ that contains its intended behavior. Each task profile contains a list of tasks, where each task $\tau \in \rho^{[i]}$ is named IDLE, CRUISE, PARK, or UNPARK, depending on the intended behavior of the agent. For example, an agent that enters the lot, parks, waits, then exits the lot would have task profile [CRUISE, PARK, IDLE, UNPARK, CRUISE] with appropriate parameters for each task. During a simulation, after an agent is spawned, it executes its task profile in order. More information about each task is located in Tables 4.1 and 4.2.

For simulation types where we generate agents, those agents always fall under $\mathcal{A}_{\text{enter}}$ and $\mathcal{A}_{\text{exit}}$, which have task profiles [CRUISE, PARK] and [UNPARK, CRUISE], respectively. For the recorded-agents simulation type, where our simulated agents mirror the task profiles of their real-world counterparts, the task profile could be arbitrarily complex.

Parameter	Description
name	Task name
v_{cruise}	Max driving speed
s_{target}	Spot to cruise to/park in
$(x_{\text{target}}, y_{\text{target}})$	Coordinates to cruise to
ΔT_{idle}	Idle task duration
T_{next}	Time to end idle task

Table 4.1: Task parameters

name	v_{cruise}	s_{target}	$(x_{\text{target}}, y_{\text{target}})$	ΔT_{idle}	T_{next}
IDLE	n/a	n/a	n/a	optional	optional
CRUISE	required	optional	optional	n/a	n/a
PARK	n/a	required	n/a	n/a	n/a
UNPARK	n/a	required	n/a	n/a	n/a

Table 4.2: Task types

4.4 Occupancy Management

To ensure vehicles do not collide or enter deadlock by driving to the same parking spot, the centralized infrastructure maintains an occupancy database $\mathbb{O} = \{0, 1\}^{|\mathcal{S}|}$, where $\mathbb{O}^{[s]}$ is 1 if parking spot $s \in \mathcal{S}$ is occupied. The database obeys the following rules:

1. At simulation start, for each $s \in \mathcal{S}$, $\mathbb{O}^{[s]} = 1$ if there is an $o \in \mathcal{O}$ such that $(x^{[o]}, y^{[o]}) \in \mathbb{B}_{\text{spot}}^{[s]}$ or an $i \in \mathcal{A}$ such that $(x^{[i]}, y^{[i]}) \in \mathbb{B}_{\text{spot}}^{[s]}$.
2. When an agent $i \in \mathcal{A}_{\text{enter}}$ spawns with intent to park in spot $s \in \mathcal{S}$, $\mathbb{O}^{[s]}$ is set to 1.
3. When an agent $i \in \mathcal{A}_{\text{exit}}$ spawns in spot $s \in \mathcal{S}$, $\mathbb{O}^{[s]}$ is set to 1. Once the agent finishes its unparking maneuver, $\mathbb{O}^{[s]}$ is set to 0.
4. When an agent $i \in \mathcal{A}_{\text{other}}$ spawns with intent to park in spot $s \in \mathcal{S}$, $\mathbb{O}^{[s]}$ is set to 1. Once it has parked, then unparked, $\mathbb{O}^{[s]}$ is set to 0.

Using these rules, any spawning agents identifying a place to park will not go to a spot already occupied (or soon to be occupied) by another agent. However, it still allows multiple agents to park in the same spot over the course of a simulation.

Chapter 5

Vehicle Assignment Strategy

As Chapter 6 will prove, applying the single-vehicle autonomous driving framework alone leads to significant time savings compared to recorded human driving due to increased efficiency in route planning and path following. However, it is possible to generate even more time savings when taking the entire fleet into account. In this chapter, we examine one of the ways to save fleet parking time: by making smart decisions about which spots to park in. For instance, two vehicles entering a lot at around the same time could save time by parking in parallel in different rows instead of parking next to each other and having one vehicle wait for the other.

As our objective is to minimize elapsed driving time for entering agents, we can generate savings by altering the spot assignments (where each vehicle chooses to park) for agents in $\mathcal{A}_{\text{enter}}$ ¹. Below, we present strategies to improve fleet parking through assigning spots in a decentralized (Section 5.1) or centralized manner (Sections 5.2 and 5.3).

5.1 Decentralized Assignment

When human-driven vehicles enter a parking lot, they are only influenced by their immediate surroundings—there is no central infrastructure that influences their decision-making. We present vehicle assignment strategies that mirror this lack of centrality. Pre-recorded assignment is a strategy when simulating an existing scenario from the dataset (including the same obstacles and agents), while intent-based assignment can be used in any kind of scenario, including new, artificially-generated scenarios.

¹For recorded-agent simulations where we do not control the spawn location of each vehicle, we determine an agent $i \in \mathcal{A}$ is in $\mathcal{A}_{\text{enter}}$ if its task profile is [CRUISE, PARK] and its initial y-position $y^{[i]}(T_{\text{start}}^{[i]}) > y_{\text{enter}}$, where y_{enter} marks the entrance of the lot in the DLP dataset.

Pre-Recorded

For recorded-agent simulations where we have full trajectory data, one option is to have vehicles park in the same spots that human drivers selected in the recorded data. When an agent enters the lot, we assign it the spot that it parked in in the dataset. With this spot assignment, the time savings are derived from improved path planning and following due to autonomous control and improved vehicle-to-vehicle interaction with our rule-based collision avoidance algorithm.

Driver Intent-Based Assignment

For decentralized assignment in a new scenario, we present a strategy that acts more similarly to human driving. For humans, spot assignment is not determined at lot entrance, but is dynamically decided as they drive through the aisles. This is because humans do not have access to information about the entire lot; they can only see and make decisions based off what is in their local area. We use this concept of local vision to motivate this algorithm.

Intent Prediction Model

Our intent-based algorithm is built upon ParkPredict+ by Shen et al. [10], an intent and trajectory prediction model. Given a sequence of a vehicle’s past states and the current state of the area near the vehicle (in the form of a rasterized birds-eye view image), ParkPredict+ can predict the intended location and future trajectory of the vehicle. We use the intent prediction element of this work to guide our agents’ decision-making.

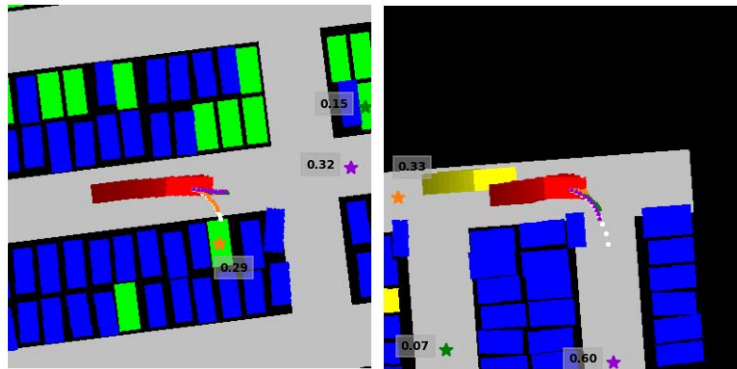


Figure 5.1: Examples of intent prediction from [10]. The obstacles are plotted in blue, and the empty spots are plotted in green. The vehicle of interest is plotted in red with a "fading tail" as its motion history. The orange, purple, and dark green stars are the top-3 predicted intent at the current time step. The values beside each star show the probability of the corresponding intent.

Algorithm

The intent-based assignment algorithm is similar to entrance assignment, except instead of being assigned a spot at the entrance and driving directly to it, an agent will constantly update its intended location $\iota = (x_\iota, y_\iota) \in \mathbb{R}^2$ and drive toward it. Fig. 5.1 shows some examples of intent prediction while driving. The model will predict a distribution over all intents that are currently viable for the vehicle, which indicates the likelihood of choosing each one of them. We pick ι by sampling from this distribution. Then, once ι is inside of a spot, the agent will park there. Pseudocode is provided in Algorithms 2 and 3.

For Algorithm 2, lines 5-9 generate an intent prediction and lines 10-20 take appropriate action based on the returned intent. Constant t_{predict} is the interval between intent predictions (e.g. a vehicle re-predicts its intent every t_{predict} seconds).

For Algorithm 3, line 2 calls the intent prediction model, which returns a list of possible intent locations and their corresponding probabilities. Lines 5-15 determine which of the intent locations are valid (e.g. the vehicle cannot park in an occupied spot). Lines 16-26 sample an intent from the list of valid intent locations. Constant r_{near} is the radius around the vehicle in which it cannot select a new intent, since allowing it to do so could cause the vehicle to not advance further and stop in the middle of an aisle.

5.2 Centralized Assignment at Entrance: Basic Strategies

Our primary strategy to improve elapsed driving time is to select smarter parking spots for entering agents as they come into the lot. This way, they will not only reap the benefits of choosing more optimal spots with respect to time, but by knowing their final parking locations immediately, they can take optimal paths to their spots. In practice, this would involve a central infrastructure passing the assigned spot and trajectory to each agent as it enters. We examine multiple types of spot assignment algorithms.

We define the set of unoccupied spots $\mathcal{S}_{\text{empty}} = \{s \in \mathcal{S}, \mathbb{O}^{[s]} = 0\}$.

Random

A spot is assigned at random from the set of all unoccupied spots $\mathcal{S}_{\text{empty}}$.

Closest Spot

The assigned spot \hat{s} minimizes $(x^{[\hat{s}]} - x_e)^2 + (y^{[\hat{s}]} - y_e)^2$ across all spots in $\mathcal{S}_{\text{empty}}$, i.e. it is the closest spot from the entrance as the crow flies.

Algorithm 2 `intent_based_control(i : vehicle)`

```

1:                                     ▷ Drive using an intent prediction model to determine behavior
2:                                     ▷ Constants:  $t_{\text{predict}}$ 
3:  $\iota \leftarrow \text{None}$                                      ▷ Intended location, initialized to None
4:  $\iota_{\text{spot}} \leftarrow \text{None}$    ▷ Intended spot, set to None until a spot has been selected to park in
5: if A new intent prediction has not been made in  $t_{\text{predict}}$  seconds and no spot has been
   selected yet, given by  $\iota_{\text{spot}} = \text{None}$  then
6:   if The intent predict model suggests a new intent location, given by
   solve_intent( $i$ )  $\neq \text{None}$  then
7:      $\iota, \iota_{\text{spot}} \leftarrow \text{solve\_intent}(i)$ , which sets the new intent
8:   end if
9: end if
10: if A spot has been selected, given by  $\iota_{\text{spot}} \neq \text{None}$  then
11:   Perform parking maneuver
12: else
13:   if Arrived at  $\iota$  then
14:     Determine parking maneuver
15:     Wait for area to be free of other vehicles before in-parking
16:   else
17:     Calculate trajectory to intent using A* search
18:     Drive along trajectory using Algorithm 1
19:   end if
20: end if

```

Hand-Picked

Defined as a list $S_{\text{hp}} \in \mathbb{R}^{|S|}$, in this method, the order of spot assignment is hand-picked to be as close to optimally efficient as possible according to heuristics. As revealed by Shen et al. [9], spacing vehicles out at a proper distance can let vehicles perform their parking maneuvers in parallel without blocking each other, which saves time for the overall fleet. Therefore, for the hand-picked assignment, we make sure that the spots in S_{hp} have enough distance relative to each other while staying close to the entrance. Figure 5.2 demonstrates this heuristic; notice how a large number of vehicles are able to park in parallel despite being close to the entrance.

This is a lot-specific order, so the order we select only applies for the DLP parking lot. Also, for simplicity, we only run this assignment method in empty-lot simulations to reduce the amount of manual experimentation for finding efficient spot orders. This assignment method works primarily as a baseline to measure other methods against and provides insights for parking large, dense fleets.

Algorithm 3 solve_intent(i : vehicle)

```

1:                                     ▷ Determine a vehicle's intended location
2:                                     ▷ Constants:  $r_{\text{near}}$ 
3:  $\iota_{\text{list}} \leftarrow$  intent prediction model results
4:  $\iota_{\text{valid}} \leftarrow \emptyset$ 
5: for  $\iota$  in  $\iota_{\text{list}}$  do                                     ▷ Filter through the intent predictions
6:   if  $\iota \in \mathbb{B}_{\text{spot}}^{[s]}$  for some spot  $s \in \mathcal{S}$  and  $\mathbb{O}^{[s]} = 0$  then   ▷ Don't park in a spot where
   there is already a vehicle
7:     continue
8:   else if  $\iota$  is behind  $i$  then                                     ▷ Don't turn around mid-aisle
9:     continue
10:  else if  $\iota$  is within radius  $r_{\text{near}}$  of  $i$  then                                     ▷ Don't stall
11:    continue
12:  else
13:     $\iota_{\text{valid}} \leftarrow \iota_{\text{valid}} \cup \iota$                                      ▷ Add this intent to the list of valid intent locations
14:  end if
15: end for
16: if  $|\iota_{\text{valid}}| = 0$  then                                     ▷ No valid intent locations
17:  return None
18: else
19:   $\hat{\iota} \leftarrow$  random weighted sample from  $\iota_{\text{valid}}$ 
20:  if  $\hat{\iota} \in \mathbb{B}_{\text{spot}}^{[s]}$  for some spot  $s \in \mathcal{S}$  then ▷ If the intent location is in a spot, return the
   spot
21:     $\hat{\iota}_{\text{spot}} \leftarrow s$ 
22:  else
23:     $\hat{\iota}_{\text{spot}} \leftarrow$  None
24:  end if
25:  return  $\hat{\iota}, \hat{\iota}_{\text{spot}}$ 
26: end if

```

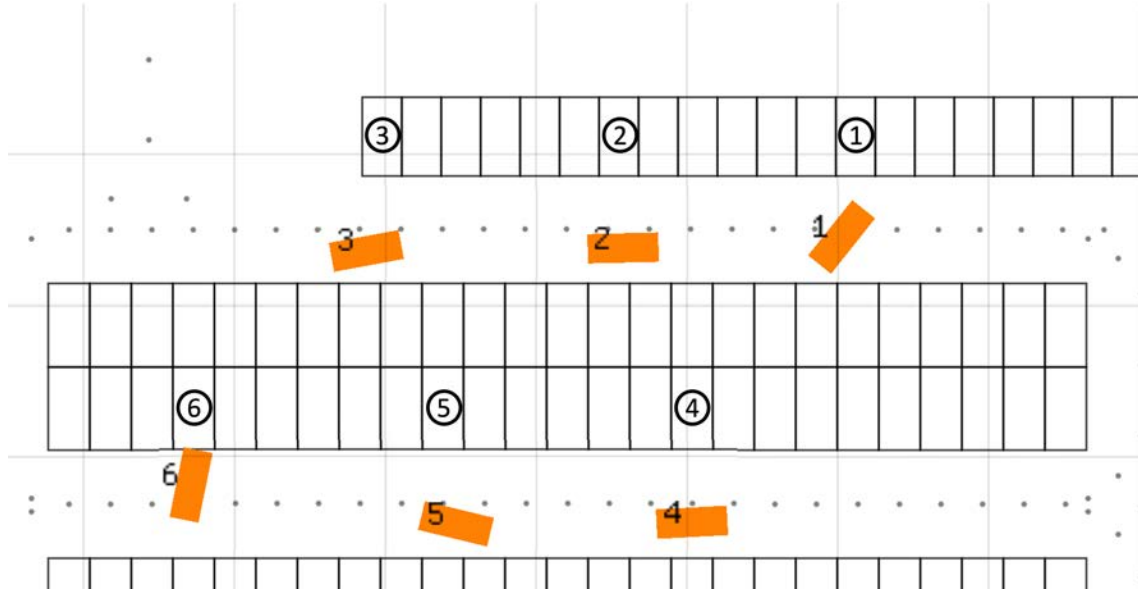


Figure 5.2: A sample of the parking map during a simulation with hand-picked spot assignment. The circled numbers denote the assigned spot of each vehicle.

5.3 Centralized Assignment at Entrance: Neural Network

When assigning spots at the entrance, for each individual agent, its objective is to choose the optimal spot to park in that helps minimize elapsed driving time for the entire fleet. This could be viewed as learning a complex function that takes as input the state of the parking lot and outputs a parking spot to go to, making the problem a prime candidate to be solved with neural networks.

Using a neural network to assign parking spots requires some problem-specific details, which we break down here.

Output

The initial instinct may be to make the neural network directly output a spot. However, this does not translate well to machine learning, as the spot number does not correlate with features of the spot. Instead, we output the time it takes for an entering vehicle to travel to a given spot, then assign based on the times for each spot. For this paper, we assign the spot $s \in \mathcal{S}_{\text{empty}}$ with the shortest predicted elapsed driving time $t_{\text{proj}}^{[s]}$, but there are potentially more possibilities, such as manipulating the times into a distribution and sampling.

Loss

By outputting the time, creating a differentiable loss is straightforward by comparing the projected and actual elapsed driving times for each entering vehicle. We use squared-error loss for agent $i \in \mathcal{A}_{\text{enter}}$ and assigned spot \hat{s} , which gives Equation (5.1), or the square of the difference between the projected elapsed driving time and actual elapsed driving time.

$$L(\hat{s}, i) = t_{\text{proj}}^{[\hat{s}]} - (T_{\text{end}}^{[i]} - T_{\text{start}}^{[i]})^2 \quad (5.1)$$

Features

Choosing input features for a spot-assigning neural network requires identifying characteristics about the lot that may affect the time it takes for an agent to park. These are the features we chose for a given spot $s \in \mathcal{S}$:

1. Length of the A* search-learned trajectory from the entrance e to the waypoint closest to $(x^{[s]}, y^{[s]})$. In other words, the distance the agent must cruise before performing a parking maneuver.
2. The x-coordinate of the spot $x^{[s]}$.
3. The y-coordinate of the spot $y^{[s]}$.
4. The number of other agents along the trajectory to s . This is calculated by discretizing the parking lot into blocks of roughly 10 meters by 10 meters, then determining the blocks that an agent must travel through to reach s and counting the number of other agents in those blocks. The intent of this feature is to measure the traffic to a given spot that could slow an agent down.
5. The number of agents maneuvering in nearby spots to s . We define two spots to be “nearby” if the closest waypoints for each of the spots are at most 10 meters apart. This feature is included because any agent arriving to s must first wait for any maneuvering agents near s to finish their maneuver, which could cause large delays.
6. The average spawn time of the lot, λ_{spawn}^2 . This is intended to potentially induce different behavior depending on the level of traffic in the lot.
7. The queue length $|Q|$. This is another feature included to potentially reduce traffic by accounting for the number of vehicles waiting to enter.

The neural network is a simple multi-layer perceptron (MLP) with two hidden layers. The hidden layers have sizes 84 and 10, and ReLU is used for the activation.

²For recorded-agent simulations, $\lambda_{\text{spawn}} = 2 \frac{\max_{i \in \mathcal{A}} T_{\text{start}}^{[i]}}{|\mathcal{A}|}$ (the factor of 2 is included because entering and exiting vehicles arrive independently with the same spawn interval mean).

Training

The network is trained using simulations, where we set $|\mathcal{A}_{\text{enter}}|$, $|\mathcal{A}_{\text{exit}}|$, and λ_{spawn} to different values for each simulation. The network's parameters are updated for each entering agent individually using its projected and actual elapsed driving time. Optimization is performed with Adam and a learning rate of 0.01.

We use a mix of on-policy training, where neural network spot assignment is used during training simulations, and off-policy training, where another method (such as random assignment) is used.

Chapter 6

Experiments

In our experiments, we demonstrate that with our single-vehicle motion controller and spot assignment algorithm, we can generate significant time savings for parking fleets. We break down our experiments according to the simulation types:

1. Occupied-lot recorded-agent: using simulated versions of real-world vehicles and their corresponding behaviors (in the form of task profiles), we demonstrate the efficacy of applying our algorithms in an actual parking lot.
2. Occupied-lot generated-agent: by simulating fleet entrances into a busy lot, we can demonstrate the ability for our algorithms to adapt to a wide variety of parking environments and still park fleets efficiently.
3. Empty-lot: by making the entering fleet the only vehicles in the lot, we can gain insights into ways to optimize mass parking and learn how to make autonomous driving more efficient.

We then test using different spot assignment algorithms (in parentheses is their shorthand, used in analysis and the legend of charts):

1. Human driving (Human Driving): real-life elapsed driving times.
2. Autonomous driving, pre-recorded human spot selection (Pre-Recorded): autonomous vehicle control, but same parking spot selections for entering vehicles as pre-recorded in the dataset.
3. Autonomous driving, intent-based spot assignment (Intent): autonomous vehicle control, parking driven by intent prediction model presented in Section 5.1.
4. Autonomous driving, random spot assignment (Random): autonomous vehicle control, parking spot selection chosen randomly at entrance.

5. Autonomous driving, closest spot assignment (Closest): autonomous vehicle control, parking spot selected is closest to entrance.
6. Autonomous driving, neural network-based spot assignment (NN): autonomous vehicle control, neural network presented in Section 5.3 chooses parking spot.
7. Autonomous driving, hand-picked spot assignment (Hand-Picked): autonomous vehicle control, entering vehicle parking spots selected by dataset-specific hand-picked order.

Not all algorithms are used for all simulation types. Human Driving and Pre-Recorded are only used in recorded-agent experiments, as the other simulation types are not based on pre-recorded behaviors of human drivers. Hand-Picked is just used for empty-lot experiments, as the heuristics will become unclear in a partially occupied parking lot.

All results are averaged over ten simulations with equal parameters.

6.1 General Behavior

Before we discuss the numerical results of our experiments, we will enumerate common behaviors we observed from each algorithm across simulation types and parameters. This is to improve understanding of findings and avoid repetition in analysis.

- Human driving performs poorly in general due to frequent stop-and-go, suboptimal trajectory to parking spots, and unskilled parking maneuvers.
- Pre-Recorded makes vehicles travel to their pre-recorded spots with the shortest path and smooth speed profiles, which increases performance in recorded-agent experiments.
- Random distributes the vehicles evenly around the lot, resulting in relatively fluid traffic patterns regardless of scenario, although at the cost of having some vehicles park very far from the entrance.
- Closest has vehicles park as close to the entrance as possible, resulting in a heavy density of vehicles there (“traffic jam”). This often results in a long line of vehicles at the entrance, each waiting for the one in front of it to park, which can slow down fleet parking time.
- Intent has similar behavior to Closest, as the intent prediction model has a tendency towards parking in open spots in its field of view. This results in traffic jams near the entrance, which slows down parking time.
- NN generally directs vehicles to park close to the front of the lot, although it does a better job of redirecting traffic away from traffic jams at the entrance. It tends to have vehicles park in empty aisles if they are available, which can relieve some traffic at the entrance.

- Hand-Picked was designed to have vehicles park close to the entrance, but spread out enough in each aisle such that they can park in parallel. This means that despite not cruising very far, vehicles almost never have to wait to park, resulting in good parking times overall.

6.2 Occupied-Lot Recorded-Agent Experiments

For recorded-agent experiments, we run simulations using both obstacles (static vehicles) and agents (recorded human-driven vehicles) from the dataset. Our simulated agents are spawned at the same place and assigned the same tasks as their real-world counterparts (e.g. start from the entrance, park), although we may alter some parts of their navigation (e.g. parking spot).

Dataset Specifics

The DLP dataset, which takes place over 3.5 hours, is split into 30 “scenes” of around 7–8 minutes. For our experiments, we use three of the most vehicle-filled scenes to test our framework. In these scenes, there are not many spots available close to the entrance, but vehicle flow is not too dense and traffic jams are rare. More specific information about the three scenes, such as number of vehicles and average spawn mean, is presented in Table 6.1, and Figure 6.1 is a sample scene, with obstacles in blue and agents in yellow.

Scene	Length	$ \mathcal{A}_{\text{enter}} $	$ \mathcal{A}_{\text{exit}} $	$ \mathcal{A}_{\text{other}} $	λ_{spawn}
Scene 22	7:19	14	6	39	14.8
Scene 24	7:56	11	6	38	16.6
Scene 25	8:29	10	9	34	18.4

Table 6.1: DLP dataset scenes for recorded-agent experiments

In Figure 6.2, we test Human Driving, Pre-Recorded, Intent, Random, Closest Spot, and NN for each scene.

Results

First, by comparing the Human Driving and Pre-Recorded series, observe that introducing our single-vehicle motion control and rule-based collision avoidance reduces parking time by around half. This is due to efficient path planning, consistent speed profiles, improved collision avoidance protocols, and ignoring pedestrians in our simulations. Then, Intent improves times even further, as entering vehicles have more of a tendency to select spots closer to the entrance compared to real-life spot selection due to the nature of the intent prediction model.

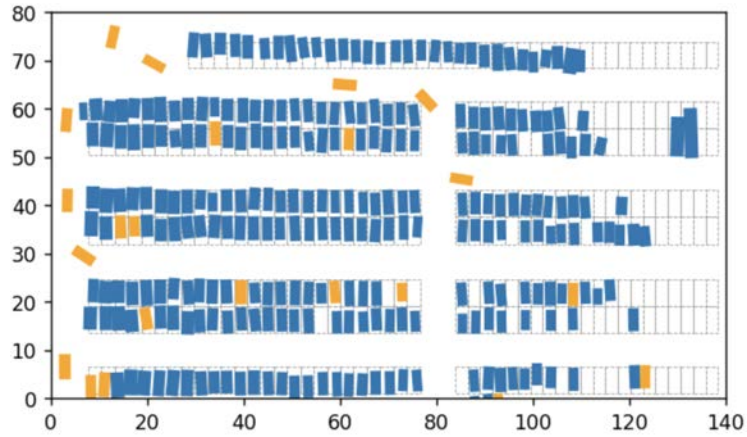


Figure 6.1: Sample DLP dataset scene (Scene 22)

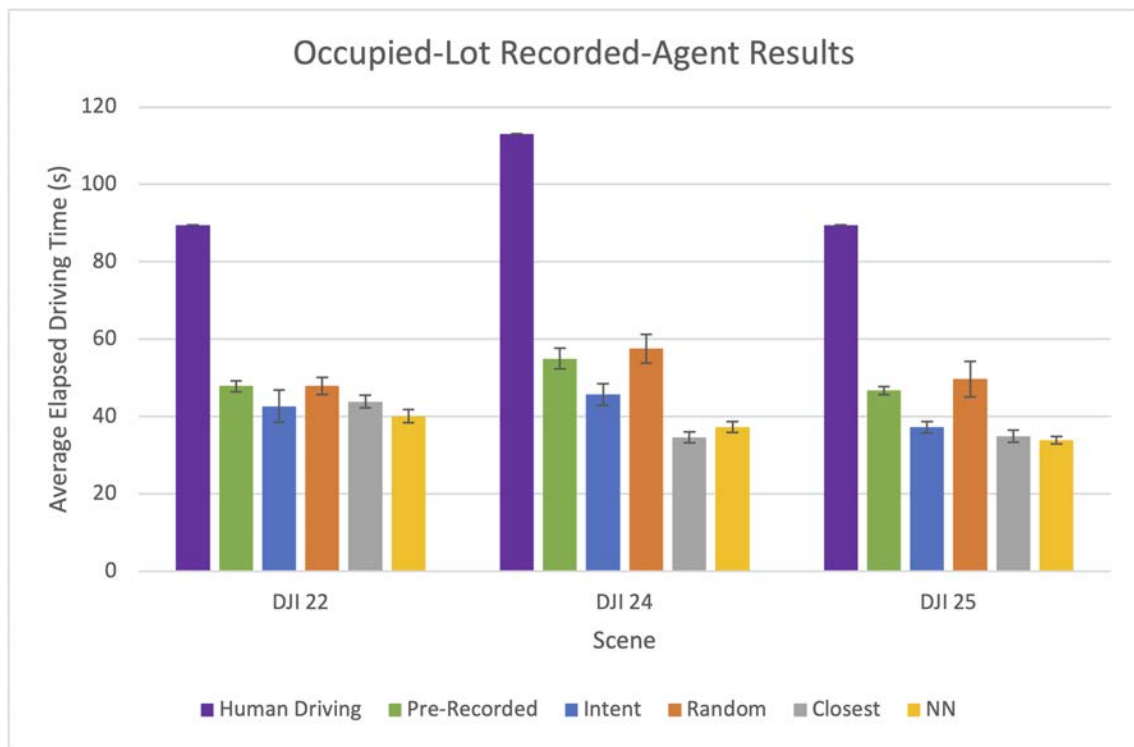


Figure 6.2: Recorded-agent results with all algorithms

For the algorithms that select spots for agents in a centralized manner, Random performs the worst out of any selection strategy. Closest and NN perform better, and actually perform quite similarly. This is a trend with heavily-occupied but less-busy lots, like the DLP

dataset—since there are not many spots available close to the entrance, there is not much variance in strategy that the neural network could exploit because the closest spot to the entrance is usually the best spot to park in. Furthermore, there are not many traffic jams, which could possibly divert entering agents away from closer spots.

6.3 Occupied-Lot Generated-Agent Experiments

For generated-agent experiments, we only use the obstacles from a given scene and generate the agents ourselves. The experiments involve varying two factors:

1. Lot traffic density: we can tweak this with the number entering and exiting vehicles $|\mathcal{A}_{\text{enter}}|$ and $|\mathcal{A}_{\text{exit}}|$ as well as the average vehicle spawn interval mean λ_{spawn} .
2. Spot assignment algorithm

Our choice of DLP dataset scene for these experiments is Scene 12, which is a busy scene but has slightly more spaces available than the scenes presented in Section 6.2.

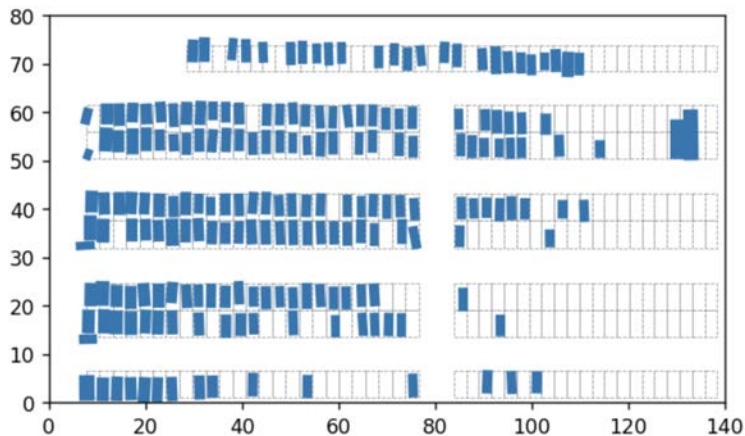


Figure 6.3: Scene 12 obstacles

In Figure 6.4, we test four algorithms: Intent, Random, Closest, and NN. We also test over five different combinations of $|\mathcal{A}_{\text{enter}}|$, $|\mathcal{A}_{\text{exit}}|$, and λ_{spawn} , each of which represents a different scenario:

1. $|\mathcal{A}_{\text{enter}}| = 30$, $|\mathcal{A}_{\text{exit}}| = 0$, $\lambda_{\text{spawn}} = 8$: large fleet entering a lot
2. $|\mathcal{A}_{\text{enter}}| = 15$, $|\mathcal{A}_{\text{exit}}| = 15$, $\lambda_{\text{spawn}} = 8$: high-density traffic
3. $|\mathcal{A}_{\text{enter}}| = 15$, $|\mathcal{A}_{\text{exit}}| = 15$, $\lambda_{\text{spawn}} = 12$: medium-density traffic

4. $|\mathcal{A}_{\text{enter}}| = 10$, $|\mathcal{A}_{\text{exit}}| = 20$, $\lambda_{\text{spawn}} = 8$: high-density traffic with more vehicles already in the lot
5. $|\mathcal{A}_{\text{enter}}| = 10$, $|\mathcal{A}_{\text{exit}}| = 20$, $\lambda_{\text{spawn}} = 12$: medium-density traffic scenario with more vehicles already in the lot

Note that all of these scenarios have λ_{spawn} smaller than in the recorded-agent experiments. This is because we are trying to simulate higher-traffic scenarios where the spot assignment algorithms have to make more difficult decisions about where to park.

In Figure 6.5, we vary λ_{spawn} while holding $|\mathcal{A}_{\text{enter}}|$ and $|\mathcal{A}_{\text{exit}}|$ constant at 30 and 10, respectively. This experiment is intended to isolate the effect of one hyperparameter, in this case the average spawn interval.

Results

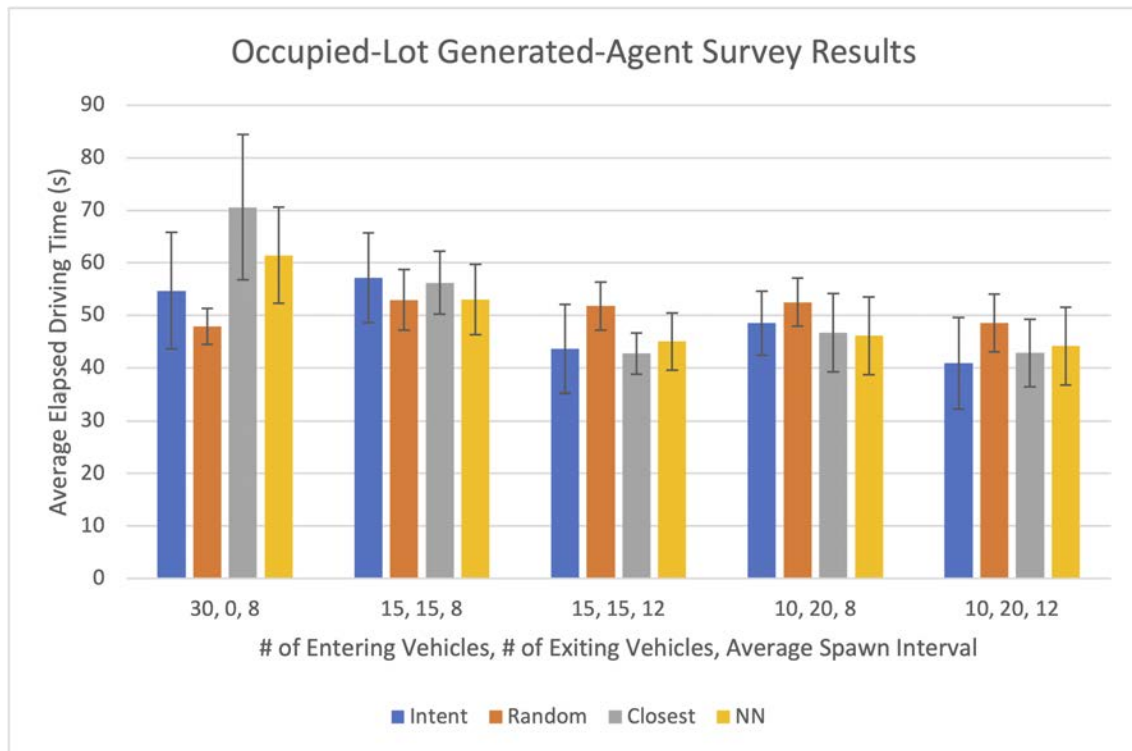


Figure 6.4: Generated-agent hyperparameter survey

Random performs similarly across all experiments because its propensity to distribute vehicles evenly in the lot makes it relatively immune to increased traffic density. An interesting result is that this property makes Random the best-performing algorithm in some scenarios. This is because there are not many spots available close to the entrance, which can

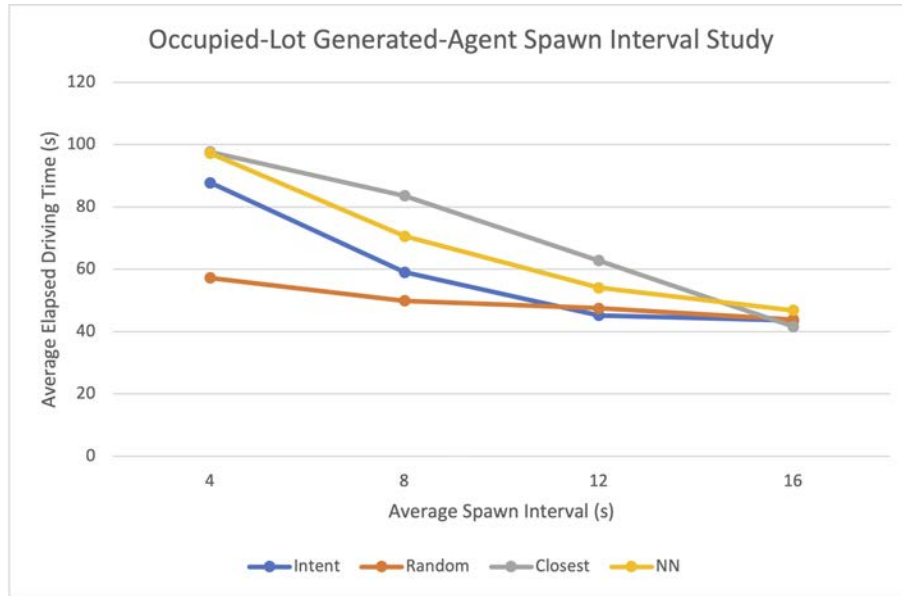


Figure 6.5: Generated-agent spawn interval survey

be seen by examining the scene map in Figure 6.3, so randomly selecting from available spots usually results in parking far from the entrance, further relieving entrance traffic compared to the other algorithms. Overall, Random performs well compared to the other algorithms in high-density scenarios with large amounts of entering vehicles, and poorly elsewhere.

Closest shines when the lot is less dense and there are fewer entering vehicles, since there is less traffic near the entrance. NN does similar to Closest, although it does slightly better when the lot is very dense, as it has more of a tendency to avoid traffic jams. But, similarly to the recorded-agent experiments, in lots with fewer available parking spaces, there is not as much variance in strategy compared to emptier lots, so NN does not do much better than Closest.

Like the other algorithms, Intent performs best in less dense simulations. In very dense lots, it performs better than Closest and NN, as it does a decent job of spreading out the vehicles. However, when the lot is less dense, it ends up with similar results to Closest. This is because, like Closest, it has a tendency towards choosing parking spots quickly and near the entrance.

6.4 Empty-lot Experiments

Empty-lot experiments use neither obstacles nor agents from the dataset, so entering vehicles can park in any spot (as long as there is not an exiting vehicle there). We run the same experiments as the occupied-lot generated-agent scenario for Figures 6.6 and 6.7.

Results

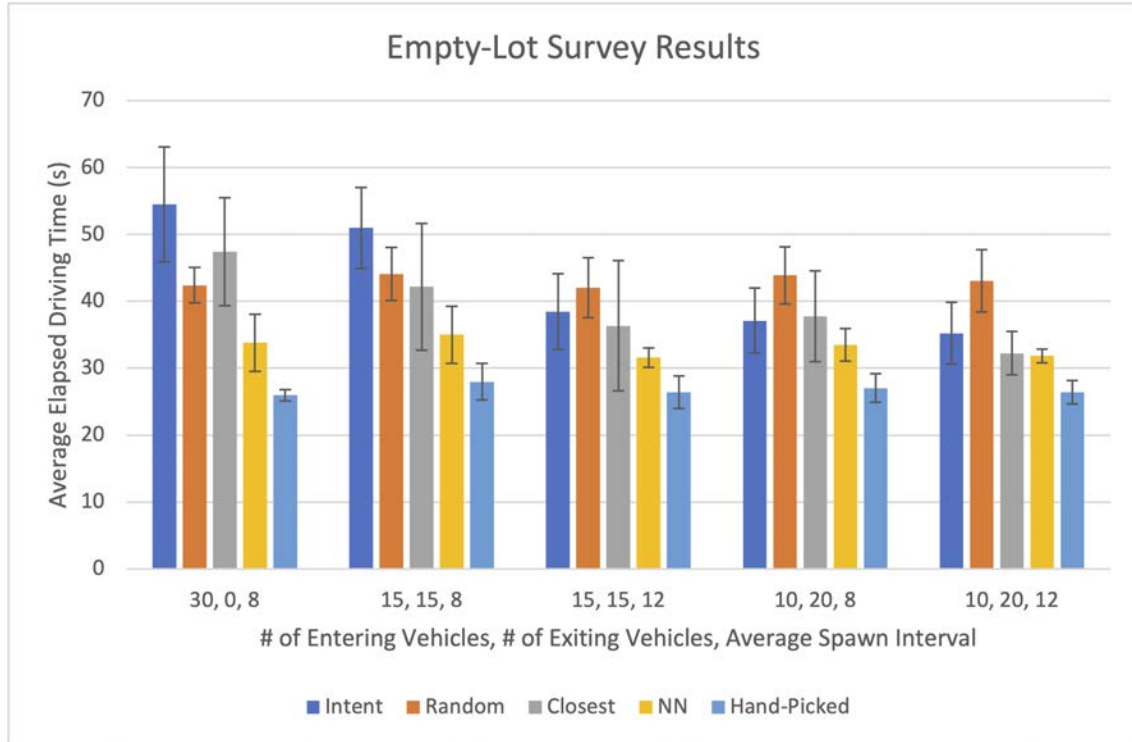


Figure 6.6: Empty-lot hyperparameter survey

Many of the takeaways are the same as the generated-agent scenario. Random performs similarly across simulation parameters because of its tendency to distribute the vehicles evenly around the lot. Closest performs better as the lot density and number of entering vehicles decreases. Intent has similar results to Closest, though it struggles even more in high-density situations because with more spots to choose from near the entrance, the intent prediction model’s greediness in selecting nearby spots creates large traffic jams.

The empty lot scenario is where NN excels. It distributes vehicles mostly evenly in the aisles of the lot, preventing a buildup of traffic near the entrance and ensuring vehicles spend less time waiting for other vehicles to park. Hand-Picked also does well, as it was engineered for this specific scenario, and is mostly invariant to the number of exiting vehicles and spawn interval.

6.5 Analysis

Through our experiments, it is clear that applying our single-vehicle planning and control framework greatly improves elapsed driving times in real parking lots, and applying intel-

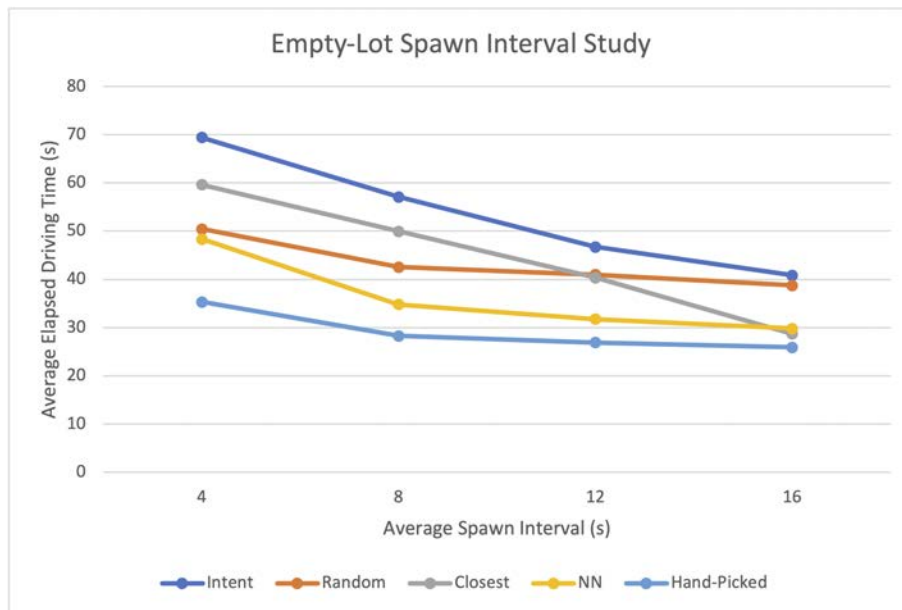


Figure 6.7: Empty-lot spawn interval survey

ligent fleet-level strategies can reduce those times even further. We observe that each spot selection algorithm we developed performs well under certain circumstances: Random for very dense lots, Intent and Closest for fuller lots with lower traffic flow, and NN for emptier lots. We also present a set of heuristics for fleet parking decision making learned from the experiments:

- In lots with low traffic flow (i.e. large λ_{spawn}), parking in the closest spot possible is very efficient.
- In lots with high traffic flow, care must be taken to avoid a buildup of vehicles near the entrance.
- To avoid traffic jams, maximize cars parking in parallel, either in different aisles or spaced out enough in the same aisle to avoid a collision.

Chapter 7

Conclusion and Future Work

In this work, we present a framework for autonomous vehicle control in parking lots as well as strategies for autonomous fleet management through improved vehicle spot assignment.

First, we formulate the problem of minimizing time spent parking for fleets of autonomous vehicles. We also introduce the DLP dataset, which provides a complete parking lot map and hours of annotated data of vehicles.

Then, we introduce our single-vehicle planning and control framework. We divide each vehicle’s trajectory into two phases: cruising through the aisles of the lot and performing a parking maneuver to enter or exit a spot. For cruising, we use A* search for path planning and the Stanley controller for path following. For maneuvering, we plan the trajectory using an optimization problem formulation and follow it with MPC.

After, we discuss our multi-vehicle simulator design, which manages large numbers of vehicles. Factors include the initial condition of the lot, vehicle spawning methods, and occupancy management. We also introduce our “task profile” behavior abstraction for vehicles.

Next, we present various vehicle assignment strategies for improving fleet parking time. Decentralized methods include using pre-recorded data for spot selection and using a driver intent-based model. Centralized methods include random spot selection, closest spot selection, a neural network-based algorithm, and a heuristic-driven hand-picked form of spot selection.

Finally, we run experiments to test the efficacy of our framework in improving parking time. We demonstrate significant time savings compared to human driving and show the efficiency of certain fleet-level algorithms in different parking scenarios, such as neural network-based spot selection in empty lots. We also present some general suggestions for fleet management—for instance, parking in the closest spot to the entrance is efficient when the lot has low traffic flow, while ensuring vehicles are spread out should be the primary priority in busier lots.

We identify some areas for future work with this simulation toolkit:

1. Transitioning from simulation-based experiments to real-world experiments with autonomous vehicles.

2. Increased research with a mixed-autonomy environment, where human-driven vehicles interact with autonomous vehicles.
3. Development of an efficient electric vehicle charging solution that maximizes charger utilization through autonomous vehicle control (expanded on in Appendix B).

We believe our work will help accelerate autonomous parking control research by providing a versatile framework for future researchers and tools for modeling driving control in parking lots.

Bibliography

- [1] Xuemin Chi et al. *Optimization-Based Motion Planning for Autonomous Parking Considering Dynamic Obstacle: A Hierarchical Framework*. 2022. arXiv: 2210.13112 [cs.R0].
- [2] Roya Firoozi et al. *A Distributed Multi-Robot Coordination Algorithm for Navigation in Tight Environments*. 2020. arXiv: 2006.11492 [cs.R0].
- [3] Peter Hart, Nils Nilsson, and Bertram Raphael. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. DOI: 10.1109/tssc.1968.300136. URL: <https://doi.org/10.1109/tssc.1968.300136>.
- [4] Gabriel M. Hoffmann et al. “Autonomous Automobile Trajectory Tracking for Off-Road Driving: Controller Design, Experimental Validation and Racing”. In: *2007 American Control Conference*. 2007, pp. 2296–2301. DOI: 10.1109/ACC.2007.4282788.
- [5] Jiawei Hou et al. “SUPS: A Simulated Underground Parking Scenario Dataset for Autonomous Driving”. In: *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*. 2022, pp. 2265–2271. DOI: 10.1109/ITSC55140.2022.9922031.
- [6] J. D. Hunter. “Matplotlib: A 2D graphics environment”. In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.
- [7] Yuchao Li, Karl H. Johansson, and Jonas Mårtensson. “A hierarchical control system for smart parking lots with automated vehicles: Improve efficiency by leveraging prediction of human drivers”. In: *2019 18th European Control Conference (ECC)*. 2019, pp. 2675–2681. DOI: 10.23919/ECC.2019.8796055.
- [8] Tong Qin et al. *AVP-SLAM: Semantic Visual Mapping and Localization for Autonomous Vehicles in the Parking Lot*. 2020. arXiv: 2007.01813 [cs.R0].
- [9] Xu Shen, Xiaojing Zhang, and Francesco Borrelli. “Autonomous parking of vehicle fleet in tight environments”. In: *2020 American Control Conference (ACC)*. IEEE. 2020, pp. 3035–3040.
- [10] Xu Shen et al. “ParkPredict+: Multimodal Intent and Motion Prediction for Vehicles in Parking Lots with CNN and Transformer”. In: *arXiv:2204.10777 [cs, eess]* (Apr. 2022). arXiv: 2204.10777 [cs, eess].

- [11] Xiaojing Zhang, Alexander Liniger, and Francesco Borrelli. “Optimization-Based Collision Avoidance”. In: *IEEE Transactions on Control Systems Technology* 29.3 (2021), pp. 972–983. DOI: 10.1109/TCST.2019.2949540.

Appendix A

Rule-Based Collision Avoidance Pseudocode

Here, we present helper functions for the rule-based collision avoidance algorithm introduced in Section 3.3.

Algorithm 4 is executed when a vehicle i is braked with the intent of determining whether the vehicle should stop braking and resume its travel. Input vehicle j is the vehicle that vehicle i is waiting for to resume driving. The algorithm is written as a series of cases where i should resume driving; lines 3-13 are for when j is maneuvering, lines 14-17 are for when j is cruising, and lines 18-19 are to prevent deadlock. $d_{\text{maneuvering}}$ is the distance a maneuvering j must be from i for i to resume driving, and d_{braking} is the distance a non-maneuvering j must be from i for i to resume driving.

Algorithm 5 is executed when a vehicle i is in the cruising phase of its path. Its intent is to determine if i is close enough to the final waypoint of its cruising trajectory, and therefore its assigned parking spot, that it should begin slowing down by applying a lower reference speed. “Close enough” is when i is within $d_{\text{steps_to_end}}$ waypoints of the final waypoint.

Algorithm 6 determines if a vehicle i has “driven past” another vehicle j , which we define as having the vectors formed by each of its rear corners to each of the front corners of j be more than 90 degrees from i ’s heading (lines 2-9). There is also an optional buffer distance d , where i has to be past j by a certain distance in the x-direction (lines 10-16). A visualization of this algorithm is presented in Figure A.1.

Algorithm 7 is one of the conditions for determining if a vehicle i should brake. It returns true if vehicle j is maneuvering and within a certain “box” of i . We define this “box” differently based on if j is in front of i (lines 5-6) or it is more on the side of i (lines 7-8). j is considered in front of i if it is within the cone formed from the direction i is pointing and extending angle $\psi_{\text{maneuvering_ahead}}$ to the left and right. A visualization of a parking box is in Figure A.2.

Algorithm 8 determines which vehicles that a vehicle i will collide with within a certain number of timesteps. It does this by simulating all vehicles’ motion using their planned trajectories (lines 5-9) and comparing their polytopes (lines 10-14). $d_{\text{crash_check}}$ is the radius

Algorithm 4 `should_resume_driving(i : vehicle, j : vehicle)`

```

1:  ▷ Determine if  $i$ , which is braking, should stop braking and continue driving based on
    the state of  $j$ , the vehicle it is waiting for
2:                                     ▷ Constants:  $d_{\text{maneuvering}}$ ,  $d_{\text{braking}}$ 
3: if  $i$  is waiting for a in-parking vehicle, meaning  $j$  is in-parking then
4:   if  $j$  has finished in-parking then
5:     return True
6:   end if
7:   if  $\text{dist}(i, j) > d_{\text{maneuvering}}$  then      ▷  $i$  is far away from  $j$ , then can resume driving
8:     return True
9:   end if
10: else if  $i$  is waiting for an un-parking vehicle, meaning  $j$  is un-parking then
11:   if  $j$  has finished un-parking then
12:     return True
13:   end if
14: else if  $\text{has\_passed}(i, j)$  then              ▷ If  $i$  has driven by  $j$ , can resume driving
15:   return True
16: else if  $\text{dist}(i, j) > d_{\text{braking}}$  and  $\text{dist}(i, j)$  is increasing then
17:   return True
18: else if  $j$  waiting for  $i$  then                ▷ Prevent deadlock
19:   return True
20: end if
21: return False

```

Algorithm 5 `close_to_assigned_spot(i : vehicle)`

```

1:  ▷ Determine if a vehicle is close enough to its assigned parking spot to begin slowing
    down
2:                                     ▷ Constants:  $d_{\text{steps\_to\_end}}$ 
3: return  $i_{\text{target\_idx}} < i_{\text{num\_waypoints}} - d_{\text{steps\_to\_end}}$   ▷ Within  $d_{\text{steps\_to\_end}}$  of end of calculated
    trajectory

```

Algorithm 6 `has_passed(i : vehicle, j : vehicle, d_{buffer} : optional float)`

```

1:                                     ▷ Determine if  $i$  has driven past  $j$ 
2: for  $c_1 \in$  rear corners of  $i$  do
3:   for  $c_2 \in$  front corners of  $j$  do
4:      $\psi_{ij} \leftarrow$  angle formed by vector from  $c_1$  to  $c_2$ 
5:     if  $|\psi^{[i]} - \psi_{ij}| < \frac{\pi}{2}$  then           ▷ Roughly, this is true when  $j$  is in front of  $i$ 
6:       return False
7:     end if
8:   end for
9: end for
10: if  $d_{\text{buffer}}$  is provided then
11:   if  $i$  facing left and  $x_j - x_i < d_{\text{buffer}}$  then           ▷  $j$  is not  $d_{\text{buffer}}$  to the right of  $i$ 
12:     return False
13:   else if  $i$  facing right and  $x_i - x_j < d_{\text{buffer}}$  then           ▷  $j$  is not  $d_{\text{buffer}}$  to the left of  $i$ 
14:     return False
15:   end if
16: end if
17: return True

```

Algorithm 7 `within_parking_box(i : vehicle, j : vehicle)`

```

1:                                     ▷ Determine if  $i$  needs to wait for a parking  $j$  to park before proceeding
2:                                     ▷ Constants:  $\psi_{\text{maneuvering\_ahead}}$ ,  $d_{\text{maneuvering}}$ 
3:  $\psi_{ij} \leftarrow$  angle formed by vector from  $i$  to  $j$ 
4:  $d_{ij} = \text{dist}(i, j)$ 
5: if  $|\psi^{[i]} - \psi_{ij}| < \psi_{\text{maneuvering\_ahead}}$  then           ▷ If  $j$  is roughly in front of  $i$ 
6:   return  $d_{ij} < 2d_{\text{maneuvering}}$ 
7: else                                     ▷ If  $j$  is roughly to the side of  $i$ 
8:   return  $d_{ij} < d_{\text{maneuvering}}$ 
9: end if

```

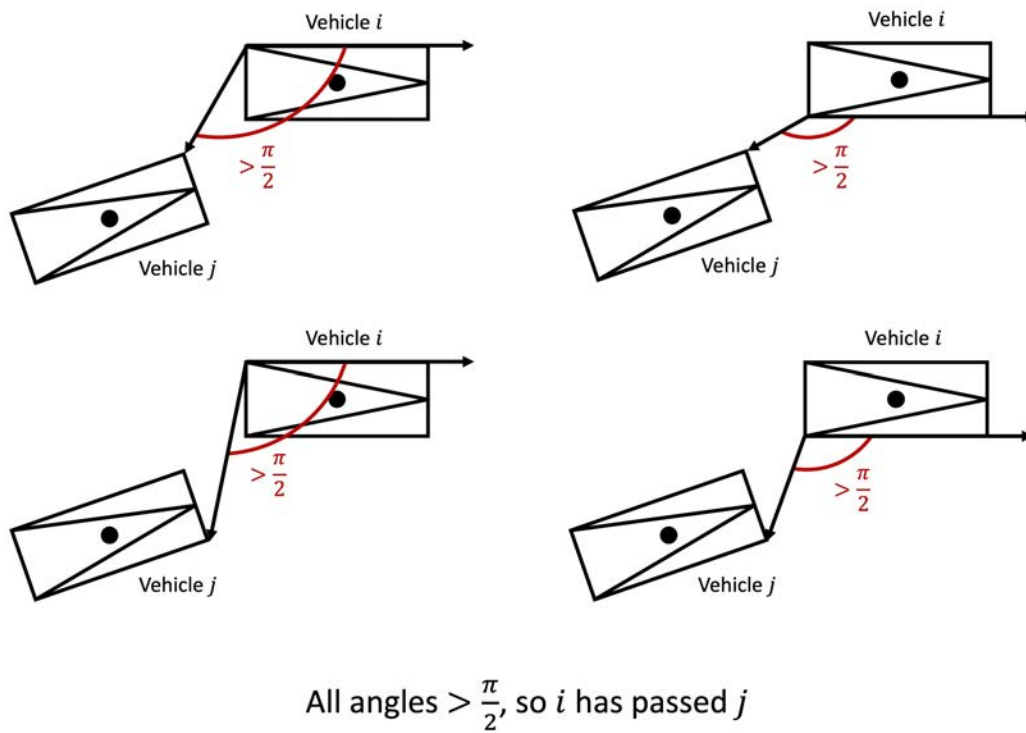


Figure A.1: Visualization of Algorithm 6, where vehicle i has passed vehicle j

in which we check for possible collisions, and $t_{\text{look_ahead}}$ is the number of timesteps we simulate forward to check for collisions.

Algorithm 9 determines priority when two vehicles i and j may collide. It does this using the angle formed by the vector between the two vehicle centers, since this can be used as a measure of which vehicle has gone further past the other and should therefore have priority. A visualization of this algorithm is presented in Figure A.3.

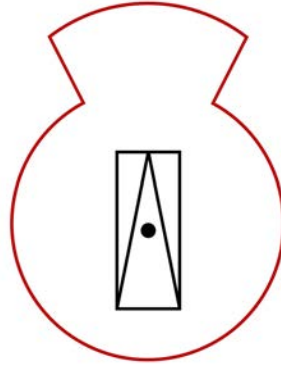


Figure A.2: A vehicle with its parking box (in red)

Algorithm 8 `will_crash_with(i : vehicle)`

```

1:   ▷ Determine all vehicles  $i$  will crash with within  $t_{\text{look\_ahead}}$  timesteps if all vehicles
   continue on their current trajectories
2:   ▷ Constants:  $d_{\text{crash\_check}}$ ,  $t_{\text{look\_ahead}}$ 
3:  $s_{\text{crash}} \leftarrow \emptyset$ 
4:  $s_{\text{nearby}} \leftarrow$  vehicles within  $d_{\text{crash\_check}}$  of  $i$ 
5: for  $t = 1, 2, \dots, t_{\text{look\_ahead}}$  do
6:   Simulate  $i$  moving forward one time step
7:   for  $j$  in  $s_{\text{nearby}}$  do
8:     Simulate  $j$  moving forward one time step
9:   end for
10:  for  $j$  in  $s_{\text{nearby}}$  do
11:    if  $i$  and  $j$  intersect then
12:       $s_{\text{crash}} \leftarrow s_{\text{crash}} \cup j$    ▷ Denote if  $j$  would collide with  $i$ 
13:    end if
14:  end for
15: end for
16: return  $s_{\text{crash}}$ 

```

Algorithm 9 `should_go_before(i : vehicle, j : vehicle)`

- 1: \triangleright Used to determine which of i or j should be forced to brake based on which vehicle has gone further past the other
 - 2: $\psi_{ij} \leftarrow$ heading angle formed by vector from i to j
 - 3: $\psi_{ji} \leftarrow$ heading angle formed by vector from j to i
 - 4: $\psi_{i\text{diff}} = \psi_{ij} - \psi_i$, adjusted by factors of 2π so $|\psi_{i\text{diff}}| < \pi$
 - 5: $\psi_{j\text{diff}} = \psi_{ji} - \psi_j$, adjusted by factors of 2π so $|\psi_{j\text{diff}}| < \pi$
 - 6: **return** $|\psi_{i\text{diff}}| > |\psi_{j\text{diff}}|$
-

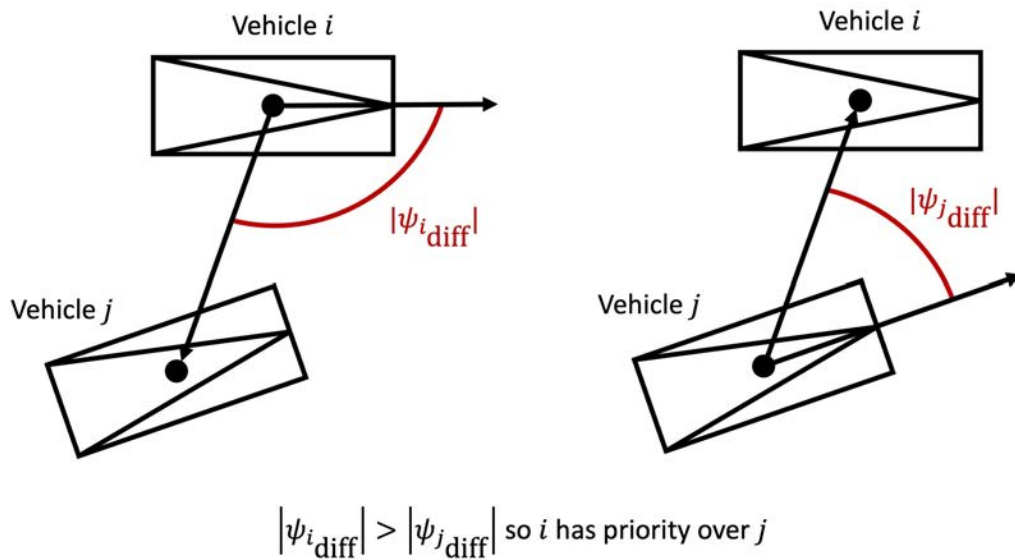


Figure A.3: Visualization of Algorithm 9, where vehicle i has priority over vehicle j

Appendix B

Electric Vehicle Charging Optimization

Thanks to their relative lack of emissions, electric vehicles are the future of transportation. However, one of the major bottlenecks with mass electric vehicle adoption is the inconvenience of charging. There are relatively few electric vehicle charging stations, so consumers cannot count on find a charging station in short order if their vehicle runs out of charge or they go on a long road trip.

One way to rectify this is to include more electric vehicle chargers in parking lots, allowing consumers to charge their cars as they shop or dine. However, when applied to lots of vehicles, this results in a very inefficient charging process, since vehicles cannot be moved until their owner leaves the parking lot. For instance, a vehicle could reach full charge an hour before its owner finishes dinner, wasting an hour of charge that could have been provided to another vehicle.

A solution to this problem is to use autonomous capabilities to automatically “swap” a fully-charged vehicle with a vehicle that needs charge when necessary. We can easily implement this idea in our autonomous control framework. In particular, the notion of “swapping” vehicles meshes well with our task profile design; in order to swap two parked vehicles, they would each perform the tasks [UNPARK, CRUISE, PARK], where the vehicle that needs charge would park in a charging spot and the fully-charged vehicle would park in non-charging spot. Using our simulator, focus could be turned to optimizing this process to maximize charging utilization.