VEHICLE DYNAMICS MODELING AND MOTION PLANNING WITH PREDICTABLE TIMING AND VELOCITY

Ty V. Nguyen

Computer Engineering Program
Graduate School of UNIST

Vehicle Dynamics Modeling and Motion Planning with Predictable Timing and Velocity

A thesis
submitted to the Graduate School of UNIST
in partial fulfillment of the
requirements for the degree of
Master of Science

Ty V. Nguyen

2016/06/15
Approved by

[illegible]

Academic Advisor
Tsz-Chiu Au

Vehicle Dynamics Modeling and Motion Planning with Predictable Timing and Velocity

Ty V. Nguyen

This certifies that the thesis of Ty V. Nguyen is approved.

2016/06/15

[illegible]

Thesis Supervisor: Dr. Tsz-Chiu Au

[illegible]

Thesis Committee Member #1: Dr. Sungju Hwang

[illegible]

Thesis Committee Member #2: Dr. Jun Moon

Abstract

Intelligent transportation systems and autonomous vehicles can improve how we drive, save energy and avoid traffic accidents. However, we still need to make more effort on the process of interdisciplinary research to turn these systems into reality. My work, aimed to contribute to this process, consists of two parts. In the first part, I address a longitudinal motion planning problem in which a vehicle aims to arrive at a given position on a road *segmented* based on various driving conditions at a given time and velocity. I show that it is possible to fully describe the set of all reachable arrival configurations using a table of closed-form equations, under a simplified model of vehicles with linear acceleration. Then I devise a sampling-based algorithm to solve this motion planning problem, using the table to check whether a feasible plan exist. The simulation results showed that the proposed sampling-based algorithm with heuristics has a higher probability of success than the simple random sampling approach. After that, I will discuss how to use the feasible set on real vehicles. In this part, the task of planning on a segment is executed by using the bisection method, which utilizes the dynamics model of a vehicle. This model, however, is not always available and empirically expensive to obtain. Therefore, in the second part, I focus on the problem of learning a vehicle dynamics model. I introduce an instance-based learning method to learn a performance model automatically, and compare it with the artificial neural network and matrix factorization methods. Furthermore, an exploration strategy called plan-based exploration, based on planning using the reference model is given to speed up the learning process. Our experimental results demonstrated that the instance-based learning method coupled with the plan-based exploration strategy has the fastest learning rate.

Contents

Contents v

List of Figures vii

I. Introduction 1

II. Related Works 4

III. Definitions 8

3.1 Longitudinal Motion Planning Problems . . . 8

3.2 Feasible Sets of Arrival Configurations . . . 9

3.3 Performance Models . . . 9

IV. Validating the Feasibility of an Arrival Configuration 12

4.1 Feasible Sets of Arrival Configurations . . . 12

4.2 Sampling-based Validation Procedure . . . 19

4.3 Experiments . . . 19

4.4 The Use of Feasible Sets on Real Vehicles . . . 20

V. Vehicle Dynamics Modeling 22

5.1 Learning Algorithms . . . 22

5.1.1 Artificial Neural Networks . . . 22

5.1.2 Matrix Factorization with Gradient Descent . . . 23

5.1.3 Instance-based Learning . . . 24

5.2 Plan-based Exploration Strategy . . . 25

5.3 Setpoint Schedule Generation . . . 27

5.3.1 Bisection Method . . . 27

5.3.2 Dynamic Programming . . . 28

5.4 Experiments . . . 30

VI. Conclusions 35

References 37

List of Figures

1.1 A car runs over a small hill in order to arrive at the destination on the other side of the hill. 2

3.1 Stable time and stable distance for a particular vehicle under a particular environment. The light color means longer time and distance, as indicated in the bars beside the graphs. 10
3.2 An example of the velocity response of a PID controller. Overshooting occurs when the vehicle decelerates to a velocity that is close to zero. 11

4.1 The four canonical velocity functions. 13
4.2 The feasible set $F = (F_{\mathsf{UU}} \cup F_{\mathsf{UD}} \cup F_{\mathsf{DU}} \cup F_{\mathsf{DD}})$. 14
4.3 Special values that dictates the boundary of F. 15
4.4 The upper boundaries and the lower boundaries of the feasible sets in seven different cases. 17
4.5 The equations in Fig. 4.4. 18
4.6 Success rates of the sampling algorithms. 20

5.1 A performance model represented by two artificial neural networks 23
5.2 The car and the platform used in the experiments 27
5.3 RMSEs of stable time versus training time 33
5.4 RMSEs of stable distance versus training time 33
5.5 RMSEs of execution time versus training distance. 34

CHAPTER I

Introduction

Over the last decade we have witnessed tremendous progress in autonomous vehicle research, as demonstrated in the DARPA Urban Challenge in 2007 [10]. The technology of flying drones also becomes mature enough to be considered civilian use such as package delivery.[1] These stories suggest that more and more autonomous robots will take part in our daily life. When autonomous robots are common, new applications will emerge in conjunction with innovations that will transform current practices. For example, a team of IkeaBots can coordinate to assemble a piece of furniture [26]. By exploiting the precise control of autonomous vehicles, Dresner and Stone proposed a new intersection control protocol called Autonomous Intersection Management (AIM), which coordinates vehicles to enter an intersection in unison [15]. In these new applications, robots are required to move to positions with precision in time and velocity. In general, for a team of robots to collaborate effectively, precision in time and velocity often plays a pivotal role. In the first part of this work, I present motion planning algorithms for moving an autonomous vehicle along a segmented path with *guarantees* of their arrival time and velocity at the destination. This longitudinal motion is fundamental in a number of multi-robot systems. Au et al. [5] considered the problem of controlling an autonomous vehicle to arrive at a specific position on a road at a given time and velocity. They showed that the decision problem (a.k.a. the validation problem) is tractable by giving a simple procedure to check whether an arrival time and an arrival velocity are feasible. The efficiency of the validation procedure is crucial in AIM, since an early detection of the infeasibility of a given arrival time and velocity can prevent vehicles from claiming space and time

[1] http://www.amazon.com/b?node=8037720011

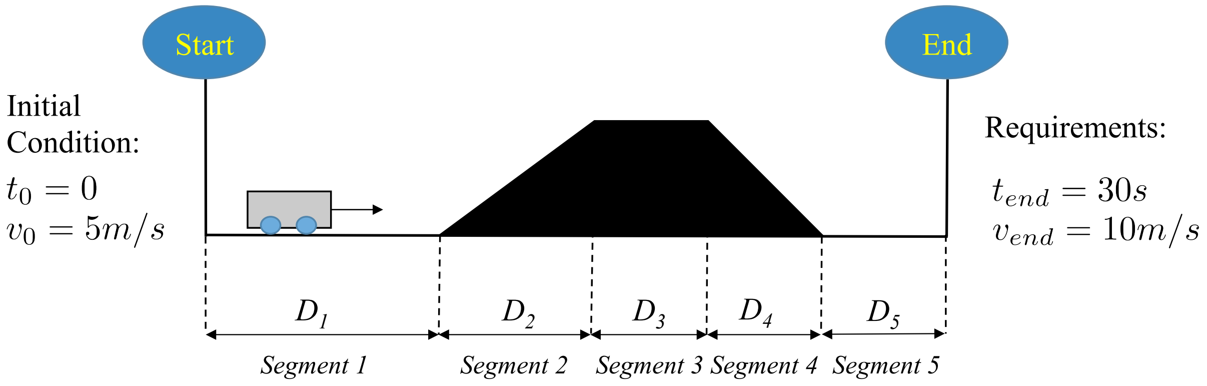


Figure 1.1: A car runs over a small hill in order to arrive at the destination on the other side of the hill.

in an intersection that they cannot utilize, and that in turn increases the efficiency of an autonomous intersection [36]. However, these work assumed that the road condition is constant, and the interaction between the vehicle and its environment will never change. Nonetheless, this *homogeneity* assumption does not always hold. For example, a mountain car needs to go uphill and downhill from time to time, and different road segments can have different road resistance and speed limits (Figure 1.1). Likewise, the air resistance and gravity acting on a flying drone can vary as it flies up and down. Therefore, I want to relax the homogeneity assumption and develop motion planning algorithms with guarantees on arrival time and velocity in spite of non-uniform conditions in a path with multiple road segments. In order to solve this problem, I first derive a *complete* set of equations describing the set of *all* possible arrival configurations given an initial configuration. These equations enables me to come up with a sampling-based algorithm to check the feasibility of a pair of arrival time and velocity on a multi-segmented path. The experimental results showed that utilizing our proposed heuristics can improve the efficiency of the sampling algorithm. I will also discuss how to apply this approach to real vehicles which do not have a linear acceleration.

In generating a setpoint schedule on one road segment, I utilize the bisection method. This method in turn adopts knowledge about acceleration and deceleration of a vehicle which can be represented as a performance model. Moreover, vehicle performance models are especially important in designing traffic system including geometric elements of road intersections, signal plans of traffic lights and the width of road lanes. They are also crucial in crash simulation and estimating fuel consumption and emission [8]. This leads to a demand to devise an efficient and convenient approach to build such models. HoIver, building a performance model is empiricly tough as a vehicle will run in a variety of road conditions and we could not obtain the performance model for running on every possible road. Therefore, this is necessary to employ some machine learning techniques to conduct vehicle performance profiling automatically. In the next part of this work, I investigates exploration strategies to enhance the learning process of the performance model of a vehicle. A learning approach is introduced to integrate with an exploration strategy. I conducted experiments to evaluate approaches and find out which approach has a faster learning rate. In addition, I introduce a dynamic programming approach to utilize performance models to do planning.

This work is organized as follows. After presenting the related works in Chapter II, I give the definition of a performance model and a longitudinal motion planning problem that utilizes a performance model in planning in Chapter III. Chapter IV gives the details of the validation process including the description of feasible set and a sample-based approach. Chapter V describes the exploration strategies and a learning approach. Finally, I conclude this paper in Chapter VI.

CHAPTER II

Related Works

Some complete motion planning algorithms have been proposed in the early days of the field. For example, Lozano-Peŕez and Wesley [32] presented an algorithm for planning collision-free paths among polyhedral obstacles. Donald [14] described the first known implementation of a complete algorithm for the full six degree of freedom Movers' problem. Nevertheless, these early algorithms are usually unsuitable for practical applications due to their poor computational complexity. More recent work focuses on planning in limited domains or exploiting the problem structure. Švestka and Vleugels [46] gave an exact motion planning algorithm for tractor-trailer robot in the absence of obstacles. Halperin [19] concerns with the geometric algorithms for robust primitives for complete motion planning. Varadhan and Manocha [44] proposed a complete motion planner on star-shaped roadmaps. Varadhan et al. [43] described a complete algorithm for motion planning of translating polyhedral robots in 3D. Most of these planners concern with checking whether a robot can arrive at a position with a correct final orientation. However, since these algorithms aims to solve motion planning problems in general, they run in exponential time in the worst cases.

Probabilistic roadmap methods (PRM) [21, 24] and rapidly-exploring random trees [29, 30] are both widely used, sampling-based algorithms. While these algorithms are probabilistically complete under very general conditions [29], they are actually incomplete algorithms because there is no guarantee that they find a solution if one exists in a finite amount of time. However, some extensions can turn them into complete algorithms. Hirsch and Halperin [20] proposed a hybrid motion planner that generates complete solutions with PRM. Zhang et al. [48] proposed another hybrid approach for complete motion planning based on PRM using approximate cell decomposition. Nonetheless, these modified algorithms will suffer from inefficiency due to their completeness.

Longitudinal control of autonomous/semi-autonomous vehicles has been widely studied since the 1960's, in particular in platooning in automated highway systems [18, 41, 45]. These studies mainly focus on car following in a platoon [40], but our approach is more suitable for point following [9]. Most work on motion planning for autonomous vehicles (e.g., [17]) has treated the arrival time and velocity requirements as secondary. But finding optimal arrival times and velocities is an important issue in some applications [5, 36]. Au and Stone had studied the longitudinal control problem on real vehicles [4], but they made use of the homogeneity assumption as in [5]—the maximum and minimum accelerations remain constant all the time.

Some previous works on motion planning focus exclusively one-dimensional trajectories. For example, Bobrow et al. [7] dealt with the minimum-time manipulator control problem, which is about controlling a robot manipulator to move along a specified path in a time-optimal manner, subject to the actuator torque constraints. Kunz and Stilman [28] dealt with a similar problem using the same approach with a path preprocessing step. In contrast, we investigate the validation problem, which could be much *harder* than the optimization problem because proving that there is no feasible trajectory for an arbitrarily given arrival time and velocity requires to check not just the optimal trajectory but all possible trajectories.

Some general-purpose motion planners are capable of satisfying *both* the arrival time and arrival velocity requirements. For example, Johnson and Hauser [22] presented a polynomial-time, complete planner that computes collision-free, time-optimal, longitudinal control sequences for meeting arrival time and velocity requirements, via the computation of the reachable sets in the path-velocity-time space. Johnson and Hauser [23] improved their previous work by allowing non-rectangular obstacles in position-velocity-time space. Their problem is significantly different from ours as it mainly focuses on computing a time-optimal trajectory that avoids collisions. Unlike our work, [22] relies on the homogeneity assumption.

Practically, there are many difficulties to obtain an analytical model of a vehicle acceleration due to imperfect vehicle dynamics, noise and the complexity in modeling the environmental characteristics. Therefore, data-driven methods have been serving as an alternative approach to acquire a model of the accelerating behavior, and machine learning is the key to build these models.

Data-driven methods can be divided into two main categories: parametric and nonparametric approaches. These methods aim to provide either a parametric or a nonparametric model of the physical system to fit the training data and base on that model to predict the outcome performance of the system under other conditions. In parametric approaches, the model can fall into either kinematics model or dynamics model categories. To begin with, kinematics models consider the mathematical relationship between acceleration, speed, and distance that the vehicle has traveled. The most basic of kinematics models are the constant acceleration model, linear decay model [16] and dual-regime model [6], to name a few. These models basically attempt to empirically construct mathematical expressions that describe how the vehicle accelerates. However, the determinant factor of the acceleration process—the tractive force provided by the engine and the opposing resistance forces are ignored. For this reason, these kinematics models cannot provide reasonable fitting to field data.

On the contrary, both of the tractive effort and resistance forces which act on the vehicle's body and control the vehicle's motion are taken into account to develop vehicle dynamics models. Rakha et al. was the first to bring forth a constant power model and a variable power model to determine the performance of trucks [37]. Many efforts have been followed by [39] and [6] to calibrate the dynamics models. Although these dynamics models provide a good fit to the field data, it is hard to decide which breaking points are appropriate for different regimes, not to mention that these breaking points are subject to variation as data sets change. Besides, these models need intensive calibration before using them. The downsides of the parametric models are twofold: First, the majority of parametric models only predict the maximum acceleration capabilities of a vehicle. Second, due to the limit of number of parameters, it is hard to estimate highly nonlinear terms and measurement noise. For these reasons, nonparametric estimation can be an appropriate alternative. Indeed, there are a few works on nonparametric estimation. For example, in [47], Kim and Oh adopted an Artificial Neural Network model to predict the next state of the vehicle given the current vehicle state, the current input steering angle of the wheels and the vehicle's velocity. The neural network model is associated with the hybrid learning scheme. In addition, Park et al. introduced a speed prediction algorithm, namely Neural Network Traffic Modeling-Speed Prediction, which is trained with the historical traffic data and capable of predicting the vehicle speed profile by using current traffic information [35]. These works are different from ours as we aim to construct the performance model of the vehicle and introduce exploration strategies to foster this process.

Previous research in modeling acceleration and deceleration of vehicles mainly focuses on determining the values of acceleration and deceleration of vehicles with an assumption that the stable time and stable distance data can be obtained by measurement [1]. R.Akcelik et al. [2] provided a parametric model for regression method to determine the acceleration time and distance of a vehicle based on Sydney data. These approaches however, require a moderate data for training. In this study, we propose a learning approach which can relax this assumption. Our proposed exploration strategies can work with various learning schemes which have been existed in the literature. For example, Al-hasan et al. [3] aims to find the minimum-cost route from the start cell to the destination cell. The route's cost is defined as as the weighted sum of three cost metrics: distance, hazard, and maneuvering. Their learning algorithm can be used to integrate with our exploration strategies by defining the route's cost as time and/or distance and view each discretized velocity as a cell. Gaussian process regression utilized in [31] to estimate the wind map for Unmanned aerial vehicles is another example of learning algorithms that can work with our proposed exploration strategies.

In this work, we represent a performance model associated with a pair of road slope angle and road friction coefficient as two $M \times M$ matrices which represents stable time and stable distance. Therefore, learning this model can be viewed as a matrix completion problem. This problem has been studied by various works such as [11–13,25,27,33]. Yet, our work is not solely a matrix completion problem as we introduce a scheme to reduce the training time needed for planning phase. Moreover, the performance matrix is extremely sparse at the beginning of the training phase and this affects the efficiency of the existing techniques used for solving the matrix completion problem. In the experimental result section, we will evaluate the performance of the matrix fractorization approach compared with that of our proposed approach.

CHAPTER III

Definitions

3.1 Longitudinal Motion Planning Problems

Let us first consider the vehicle in Fig. 1.1. The vehicle is approaching a hill with steep slopes on both sides. At the beginning, the vehicle is located at the starting position at time $t_0 = 0$ and has an *initial velocity* v_0. Our objective is to control the vehicle to reach the destination (the "End" sign) at a given *arrival time* t_{end} and at a given *arrival velocity* v_{end}, subject to the speed limits and the vehicle's acceleration constraints. We can divide the road of the entire journey into five road segments: R_i with length D_i, for $1 \leq i \leq 5$. The vehicle's constraints differ on every road segment due to the varying road conditions such as slope, air resistance and friction coefficients. Let $v_{\mathsf{max},i}$ be the speed limit of the road segment R_i such that the velocity of the vehicle cannot exceed $v_{\mathsf{max},i}$ at any point in R_i. Let $a_{\mathsf{max},i}$ and $a_{\mathsf{min},i}$ be the absolute values of the maximum acceleration and the maximum deceleration, respectively, on R_i, such that the vehicle cannot accelerate more than $a_{\mathsf{max},i}$ or decelerate more than $a_{\mathsf{min},i}$ at any point on R_i.

In general, there are N road segments. We define 1) the *initial configuration* as (t_0, v_0), 2) the *arrival configuration* as $(t_{\mathsf{end}}, v_{\mathsf{end}})$, and 3) the *road segment configuration* of R_i as $(D_i, a_{\mathsf{max},i}, a_{\mathsf{min},i}, v_{\mathsf{max},i})$, for $1 \leq i \leq N$. A *longitudinal motion planning problem* $\mathcal{P}_{\mathsf{valid}}$ is a 3-tuple $\langle (t_0, v_0), (t_{\mathsf{end}}, v_{\mathsf{end}}), \{(D_i, a_{\mathsf{max},i}, a_{\mathsf{min},i}, v_{\mathsf{max},i})\}_{i=1..N} \rangle$, where $t_0 = 0$, $0 \leq v_0 \leq v_{\mathsf{max},1}$, $0 < t_{\mathsf{end}}$, $0 \leq v_{\mathsf{end}} \leq v_{\mathsf{max},N}$, $0 < D_i$, $a_{\mathsf{max},i} \geq 0$, $a_{\mathsf{min},i} \geq 0$, and $0 < v_{\mathsf{max},i}$, for $1 \leq i \leq N$. Our task is to generate a sequence of *control signals* such that if the vehicle follows the sequence exactly, it will reach the destination while satisfying all requirements and constraints. There are many different type of vehicle controllers, but for *velocity-based* controllers, the sequence of control signals is a *velocity function* $v(\cdot)$, such that the controller will set the velocity of the vehicle according to $v(\cdot)$ over time. We are only interested in non-negative velocity functions because we forbid a vehicle to move backward. Let t_i be the time the vehicle arrive at a road segment R_i according to $v(\cdot)$ for $1 \le i < N$, and let t_N be the time the vehicle arrives at the destination. We say $v(\cdot)$ is *feasible* if it satisfies the following constraints:

C1) $v(t_0) = v(0) = v_0$;

C2) $t_N = t_{\mathsf{end}}$ and $v(t_N) = v_{\mathsf{end}}$;

C3) $0 \le v(t) \le v_{\mathsf{max},i}$ for $t_{i-1} \le t < t_i$, $1 \le i < N$ (i.e., the velocity cannot exceed the speed limit of R_i or be negative at any point in R_i);

C4) $\int_{t_{i-1}}^{t_i} v(t)\,dt = D_i$, for $1 \le i < N$ (i.e., the distance traveled in R_i must be D_i); and

C5) $-a_{\mathsf{min},i} \le v'(t^-) \le a_{\mathsf{max},i}$ and $-a_{\mathsf{min},i} \le v'(t^+) \le a_{\mathsf{max},i}$, where $v'(t^-)$ is the left derivative of $v(\cdot)$ at t, $v'(t^+)$ is the right derivative of $v(\cdot)$ at t, $t_{i-1} < t < t_i$, and $1 \le i < N$ (i.e., the acceleration and the deceleration must be within the limitations when moving on R_i).

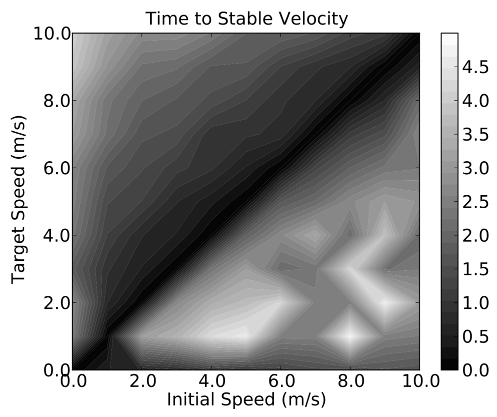
The objective of a longitudinal motion planning problem $\mathcal{P}_{\mathsf{valid}}$ is to check whether a feasible velocity function $v(\cdot)$ exists. Alternatively, $\mathcal{P}_{\mathsf{valid}}$ is called an instance of the *validation problem*, in which we want to *validate* the given arrival configuration $(t_{\mathsf{end}}, v_{\mathsf{end}})$ by checking whether $(t_{\mathsf{end}}, v_{\mathsf{end}})$ is reachable by a feasible velocity function. This problem will be addressed in Chapter IV. Furthermore, we define another longitudinal motion planning problem called $\mathcal{P}_{\mathsf{gen}}$ to generate a setpoint schedule given both initial configuration and arrival configuration. For $\mathcal{P}_{\mathsf{gen}}$, we assume that the arrival configuration is feasible. In Chapter V, we will first propose a learning scheme to learn the performance model of a vehicle and then evaluate it by solving a $\mathcal{P}_{\mathsf{gen}}$ problem on one road segment.

3.2 Feasible Sets of Arrival Configurations

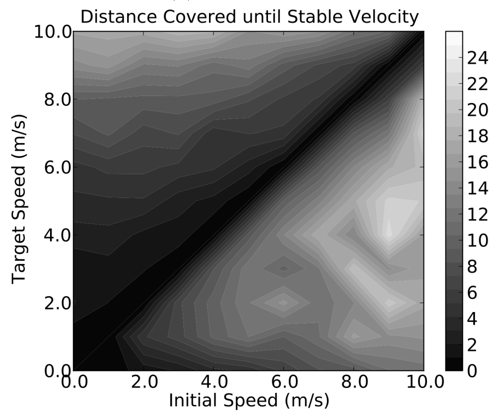
Before presenting the algorithms for the validation problem for any number of road segments in the next chapter, we first consider the case of one road segment: given an initial configuration (t_0, v_0) and a road segment configuration $(D, a_{\mathsf{max}}, a_{\mathsf{min}}, v_{\mathsf{max}})$, we want to find the set F of all feasible arrival configurations. We simply call F a *feasible set*.

3.3 Performance Models

The goal of modeling vehicle performance is to enable long-term planning of vehicle's movement *without* knowing the details of vehicle dynamics and controls. This separation of high-level planning



(a) Stable Time



(b) Stable Distance

Figure 3.1: Stable time and stable distance for a particular vehicle under a particular environment. The light color means longer time and distance, as indicated in the bars beside the graphs.

issues from the concerns of lower-level vehicle controls enables our planning procedures, called *set-point schedulers*, to work with a wide variety of vehicle hardwares with different underlying control mechanisms. Longitudinal control of autonomous vehicles are usually achieved by throttle and braking systems coupled with sensors such as odometers and speedometers using PID-controllers. A setpoint is the target velocity given to the PID-controllers so as to control to vehicle to reach the setpoint. However, due to the complexity of the system, it is often hard to tune the PID gains to achieve a smooth transition after changing the setpoint. For example, if the autonomous vehicle at UT Austin decelerates from 9 m/s to 2 m/s, it will take 4.7 s to stabilize and the stable distance is 19.3 m—-a long stable time and stable distance when compared with acceleration. This problem is due to overshooting, as illustrated in Fig. 3.2, which is an intrinsic characteristics of vehicle dynamics.

Fortunately, for planning purpose it is *not* necessary to take every detail of the vehicular behavior into account. Given the current velocity v and a setpoint $\hat{v}$, the setpoint scheduler only needs to know how long the PID controllers will take to stabilize the velocity of the vehicle at $\hat{v}$ after setting the setpoint

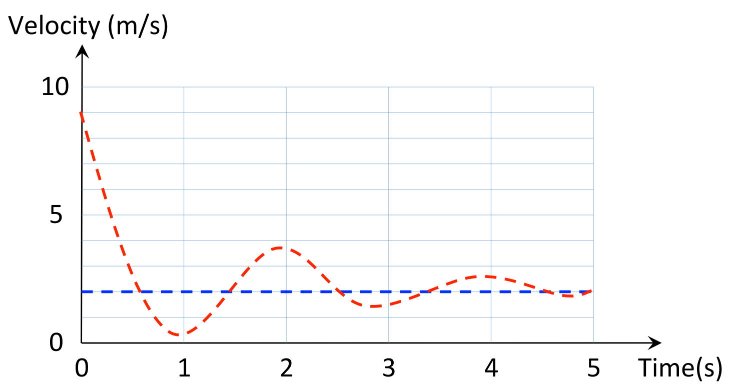


Figure 3.2: An example of the velocity response of a PID controller. Overshooting occurs when the vehicle decelerates to a velocity that is close to zero.

to $\hat{v}$, and how much the vehicle will move before its velocity is stabilized. Thus our approach relies on the estimation of two functions T^{stable} and D^{stable}, where $T^{\text{stable}}(v,\hat{v})$ is the time the vehicle takes to stabilize at $\hat{v}$ and $D^{\text{stable}}(v,\hat{v})$ is the distance the vehicle travels after setting the setpoint to $\hat{v}$ for a period of $T^{\text{stable}}(v,\hat{v})$. We call $T^{\text{stable}}(v,\hat{v})$ and $D^{\text{stable}}(v,\hat{v})$ the *stable time* and the *stable distance*, respectively. The *performance model* of the vehicle is the pair $(T^{\text{stable}}, D^{\text{stable}})$. Fig. 3.1 shows that the performance model in a table format. In this table, each row represents stable time and stable distance for the vehicle to settle from changing a discretized velocity (initial speed) to a target velocity (target speed). Because vehicle speeds are discretized, our table-type performance model cannot fully captured the continous velocity changing. Yet, the error can be reduced with a finner discretization.

The performance model is built via an empirical performance profiling of the PID controllers for the brake and throttle actuators of a vehicle. Our previous work assumed this profiling is given [5]. This work will discuss in detail how this profiling can be achieved by machine learning techniques.

CHAPTER IV

Validating the Feasibility of an Arrival Configuration

4.1 Feasible Sets of Arrival Configurations

Some previous motion planning algorithms are based on the computation of feasible sets for autonomous vehicles [22], robot arms [34], and humanoid robots [42]. However, feasible sets are often too complicated to be computed exactly. Some works consider meeting both time and velocity objectives simultaneously, such as [22], which computes a reachable set that denotes the range of reachable velocities and positions after the vehicle travels for a specific time in longitudinal motion. The reachable set, however, cannot be used to solve our motion planning problem because we do not know how much time a vehicle should spend in each road segment. As we shall see, the feasible sets in this paper, due to their non-convexity and special cases, are far more complicated than the reachable sets in [22].

Previously, Au and Stone have provided a validation procedure for problems with exactly one road segment [5]. However, their algorithm can only check whether the arrival configuration is a member of the feasible set. It is not sufficient for our algorithms, which rely on operations such as checking whether two feasible sets of adjacent road segments intersect. Thus, we want to find not just one but *all* members in F. Nonetheless, Au and Stone provided a hint for us to construct a feasible set [5]. Fig. 4.1 showed four velocity functions: $\bar{v}_{UU}(t;v_{\mathsf{int}})$, $\bar{v}_{DU}(t;v_{\mathsf{int}})$, $\bar{v}_{UD}(t;v_{\mathsf{int}})$, and $\bar{v}_{DD}(t;v_{\mathsf{int}})$. We will call them *canonical* velocity functions. $\bar{v}_{UD}(t;v_{\mathsf{int}})$ is also called an "up-down" velocity function, which

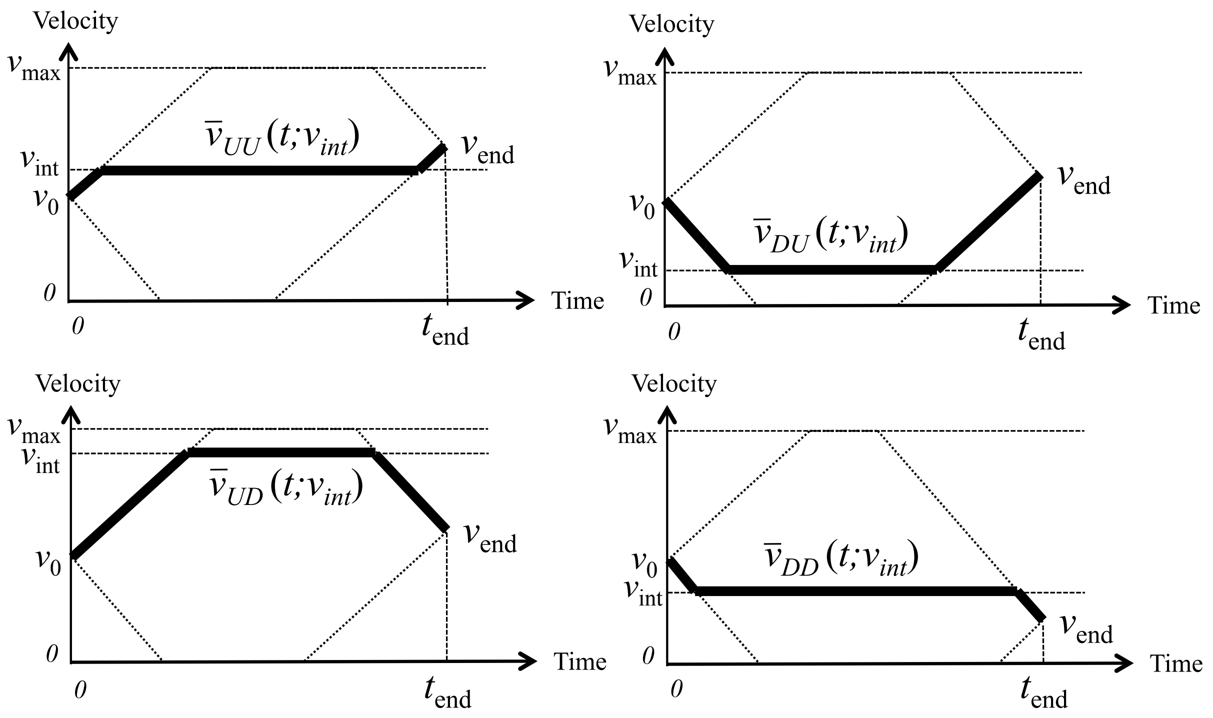


Figure 4.1: The four canonical velocity functions.

instructs the vehicle to immediately accelerate to an *intermediate velocity* v_{int} at t_0 using the maximum acceleration a_{max}, and then maintain the velocity at v_{int} until the last moment at which the vehicle can decelerate using a_{min} to reach the destination at the given t_{end} and v_{end}. Likewise, each of $\bar{v}_{UU}(t;v_{\mathsf{int}})$, $\bar{v}_{DU}(t;v_{\mathsf{int}})$, and $\bar{v}_{DD}(t;v_{\mathsf{int}})$ has one parameter—the intermediate velocity v_{int}—and will instruct the vehicle to accelerate or decelerate to v_{int} and maintain the speed as long as possible. These canonical functions are significant due to Theorem 1.

Theorem 1 *If a feasible velocity function $v(\cdot)$ exists for a validation problem with one road segment only, there exists a canonical velocity function $\bar{v}(\cdot;v_{\mathsf{int}})$ for some intermediate velocity v_{int} such that $\bar{v}(\cdot;v_{\mathsf{int}})$ is also feasible.*

An informal proof of Theorem 1 is given in [5]. The theorem implies that there is no need to check all possible feasible velocity functions in the validation problem—it is sufficient to check whether one of the four canonical velocity functions exists and is feasible. This result significantly reduces the complexity of the validation problem, as discussed in [5].

It turns out that Theorem 1 can also help us to identify some interesting structures in the feasible set. Let F_{UU} be the feasible set of arrival configurations that are reachable by using $\bar{v}_{UU}(t;v_{\mathsf{int}})$ only. Similarly, let F_{UD}, F_{DU}, and F_{DD} be the feasible sets using $\bar{v}_{UD}(t;v_{\mathsf{int}})$, $\bar{v}_{DU}(t;v_{\mathsf{int}})$, and $\bar{v}_{DD}(t;v_{\mathsf{int}})$, respectively. Theorem 1 infers that F is the union of the four feasible sets: $F = (F_{\mathsf{UU}} \cup F_{\mathsf{UD}} \cup F_{\mathsf{DU}} \cup F_{\mathsf{DD}})$.

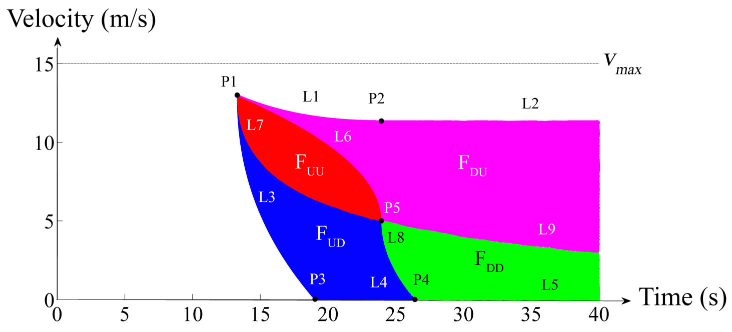


Figure 4.2: The feasible set $F = (F_{\mathsf{UU}} \cup F_{\mathsf{UD}} \cup F_{\mathsf{DU}} \cup F_{\mathsf{DD}})$.

Fig. 4.2 is a time-velocity diagram showing the feasible set F when $v_0 = 5m/s$, $t_{\mathsf{end}} = 40s$, $D = 120m$, $a_{\mathsf{min}} = 1m/s^2$, $a_{\mathsf{max}} = 0.6m/s^2$, and $v_{\mathsf{max}} = 15m/s$. As can be seen, F can be divided into four regions: F_{UU}, F_{UD}, F_{DU}, and F_{DD}. All four regions meet at a point P5, which is reachable by a constant velocity function $v(t) = 5$ for 24s. P5 exists as long as $v_0 > 0$; when $v_0 = 0$, the vehicle cannot start to decelerate as negative velocity is prohibited, and F_{DU} and F_{DD} do not exist. Line L6, L7, L8 and L9, which radiate from P5, are sets of arrival configurations shared by the adjacent feasible sets. F_{UU}, F_{UD}, F_{DU}, and F_{DD} are overlapped only on these lines. For example, $F_{\mathsf{UU}} \bigcap F_{\mathsf{DU}} = L6$. The interiors of F_{UU}, F_{UD}, F_{DU}, and F_{DD}, together with L6, L7, L8, and L9, form a subdivision of F.

We are interested in the boundary of F, which encloses all sets of feasible configurations. The fact that the boundary of F is a combination of some boundaries of F_{UU}, F_{UD}, F_{DU}, and F_{DD} can be used to deduce the closed-form expressions of the set of equations describing the boundary of F. However, the shape of F depends on the initial configuration and the road segment configuration, and in some cases F is infinite. It is necessary to enumerate all possible cases in which the set of equations differ from each other. Let consider the following four areas that are visualized in Fig. 4.3:

- $Area_L$ is the distance the vehicle travels from v_0 with a deceleration of a_{min} until a complete stop;
- $Area_R$ is the distance the vehicle travels from a complete stop with an acceleration of a_{max} until it hits the speed limit v_{max};
- $Area_U$ is the distance the vehicle travels from v_0 with an acceleration of a_{max} until it hits the speed limit v_{max}; and
- $Area_Q$ is the distance the vehicle travels from v_{max} with a deceleration of a_{min} until a complete stop.

It turns out that these values play a critical role in identifying different cases as their combinations serve as interval values that shape the set of all feasible points of each *canonical* velocity function. For example, if the road distance D is less that $Area_L$, there is no feasible velocity function $\bar{v}_{DU}(t; v_{\mathsf{int}})$ that can end up with the arrival velocity $v_{\mathsf{end}} = 0$. Similarly, $Area_U$ is the minimum value of the road segment such that there exists a velocity function $\bar{v}_{UD}(t; v_{\mathsf{int}})$ ending up with the arrival velocity $v_{\mathsf{end}} = v_{\mathsf{max}}$. Likewise, $Area_L + Area_R$ is the lower bound condition for the existence of $\bar{v}_{UD}(t; 0)$ ending at

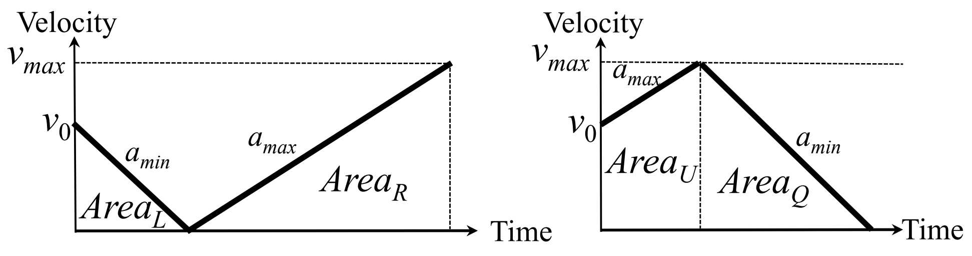


Figure 4.3: Special values that dictates the boundary of F.

$v_{\text{end}} = v_{\text{max}}$, and $Area_U + Area_Q$ is the lower bound condition for the existence of $\bar{v}_{UD}(t; v_{\text{max}})$ ending at $v_{\text{end}} = 0$.

We need to identify all possible relationships between D, $Area_L$, $Area_U$, $Area_L + Area_R$ and $Area_U + Area_Q$. In general, the number of possible ways to compare the 5 values is $5! = 120$. Fortunately, from Fig. 4.3 we identify some additional relationships among those four values: 1) $Area_L \leq Area_L + Area_R$; 2) $Area_U \leq Area_U + Area_Q$; 3) $Area_L \leq Area_Q \leq Area_U + Area_Q$; and 4) $Area_U \leq Area_R \leq Area_L + Area_R$. Basically, these relationships imply that both $Area_L$ and $Area_U$ are less than or equal to $Area_L + Area_R$ and $Area_U + Area_Q$. This helps us to reduce the number of different cases to seven:

- Case 1: $D \leq Area_L$ and $D \leq Area_U$;
- Case 2: $D \leq Area_L$ and $D \geq Area_U$;
- Case 3: $D \geq Area_L$ and $D \leq Area_U$;
- Case 4: $D \geq \max\{Area_L,\ Area_U\}$ and $D \leq \min\{Area_L + Area_R,\ Area_U + Area_Q\}$;
- Case 5: $D \geq Area_L + Area_R$ and $D \leq Area_U + Area_Q$;
- Case 6: $D \leq Area_L + Area_R$ and $D \geq Area_U + Area_Q$; and
- Case 7: $D \geq \max\{Area_L + Area_R,\ Area_U + Area_Q\}$.

We derived the equations for the upper bound $\Omega^{\text{upper}}(t)$ and the lower bound $\Omega^{\text{lower}}(t)$ of the feasible set in these seven cases. These equations, shown in Fig. 4.4 and 4.5, are inferred from an analysis of the aforementioned canonical velocity functions, but due to the lack of space we omit the analysis. Given an initial configuration (t_0, v_0) and a road segment's configuration $(D, a_{\text{max}}, a_{\text{min}}, v_{\text{max}})$, we can determine which case it belongs to, and then find the equations of $\Omega^{\text{upper}}(t)$ and $\Omega^{\text{lower}}(t)$ in the tables in Fig. 4.4 and 4.5. To see how to use the tables, take a look at the example in Fig. 4.2. We have $Area_L = \frac{v_0^2}{2a_{\text{min}}} = 12.5m$; $Area_R = \frac{v_{\text{max}}^2}{2a_{\text{max}}} = 187.5m$; $Area_Q = \frac{v_{\text{max}}^2}{2a_{\text{min}}} = 112.5m$; and $Area_U = \frac{v_{\text{max}}^2 - v_0^2}{2v_{\text{max}}} = 166.7m$. Hence this is Case 3 since $D \geq Area_L$ and $D \leq Area_U$. The equation of the upper bound is

$$\Omega^{\text{upper}}(t) = \begin{cases} \text{undefined} & \text{if } t < 13.3 \\ g_1(t) & \text{if } 13.3 \leq t \leq 23.9 \\ 11.4 & \text{if } 23.9 \leq t \end{cases}$$

where $g_1(t) = 5 - t + \sqrt{1.6t^2 - 16t + 384}$. This corresponds to L1 and L2 in Fig.4.2. We also have

$$\Omega^{\text{lower}}(t) = \begin{cases} \text{undefined} & \text{if } t < 13.3 \\ g_2(t) & \text{if } 13.3 \leq t \leq 19.1 \\ 0 & \text{if } 19.1 \leq t \end{cases}$$

where $g_2(t) = 5 + 0.6t - \sqrt{0.96t^2 + 16t - 384}$. This corresponds to L3 and L4 $\cup$ L5 in Fig. 4.2. All feasible configurations are enclosed between $\Omega^{\text{upper}}(t)$ and $\Omega^{\text{lower}}(t)$.

Case 1: $D \leq Area_L$ **and** $D \leq Area_U$

t	$t < t_2^d$	$t_2^d \leq t \leq t_1^d$	$t_1^d < t$
$\Omega^{\text{upper}}(t)$		$h_1^d(t)$	

t	$t < t_2^d$	$t_2^d \leq t \leq t_1^d$	$t_1^d < t$
$\Omega^{\text{lower}}(t)$		$g_1^u(t)$	

Case 2: $D \leq Area_L$ **and** $D \geq Area_U$

t	$t < t_2^u$	$t_2^u \leq t \leq t_3^d$	$t_3^d \leq t \leq t_1^d$	$t_1^d < t$
$\Omega^{\text{upper}}(t)$		v_{max}	$h_1^d(t)$	

t	$t < t_2^u$	$t_2^u \leq t \leq t_1^u$	$t_1^u \leq t \leq t_1^d$	$t_1^d < t$
$\Omega^{\text{lower}}(t)$		$g_2^u(t)$	$g_1^u(t)$	

Case 3: $D \geq Area_L$ **and** $D \leq Area_U$

t	$t < t_2^d$	$t_2^d \leq t \leq t_5^d$	$t_5^d \leq t$
$\Omega^{\text{upper}}(t)$		$h_1^d(t)$	$h_2^d(t)$

t	$t < t_2^d$	$t_2^d \leq t \leq t_4^u$	$t_4^u \leq t$
$\Omega^{\text{lower}}(t)$		$g_1^u(t)$	0

Case 4: $D \geq \max\{Area_L,\ Area_U\}$ **and** $D \leq \min\left\{Area_L + Area_R,\ Area_U + Area_Q\right\}$

t	$t < t_2^u$	$t_2^u \leq t \leq t_3^d$	$t_3^d \leq t \leq t_5^d$	$t_5^d \leq t$
$\Omega^{\text{upper}}(t)$		v_{max}	$h_1^d(t)$	$h_2^d(t)$

t	$t < t_2^u$	$t_2^u \leq t \leq t_1^u$	$t_1^u \leq t \leq t_4^u$	$t_4^u \leq t$
$\Omega^{\text{lower}}(t)$		$g_2^u(t)$	$g_1^u(t)$	0

Case 5: $D \geq Area_L + Area_R$ **and** $D \leq Area_U + Area_Q$

t	$t < t_2^u$	$t_2^u \leq t$
$\Omega^{\text{upper}}(t)$		v_{max}

t	$t < t_2^u$	$t_2^u \leq t \leq t_1^u$	$t_1^u \leq t \leq t_4^u$	$t_4^u \leq t$
$\Omega^{\text{lower}}(t)$		$g_2^u(t)$	$g_1^u(t)$	0

Case 6: $D \leq Area_L + Area_R$ **and** $D \geq Area_U + Area_Q$

t	$t < t_2^u$	$t_2^u \leq t \leq t_3^d$	$t_3^d \leq t \leq t_5^d$	$t_5^d \leq t$
$\Omega^{\text{upper}}(t)$		v_{max}	$h_1^d(t)$	$h_2^d(t)$

t	$t < t_2^u$	$t_2^u \leq t \leq t_5^u$	$t_5^u \leq t$
$\Omega^{\text{lower}}(t)$		$g_2^u(t)$	0

Case 7: $D \geq \max\left\{Area_L + Area_R,\ Area_U + Area_Q\right\}$

t	$t < t_2^u$	$t_2^u \leq t$
$\Omega^{\text{upper}}(t)$		v_{max}

t	$t < t_2^u$	$t_2^u \leq t \leq t_5^u$	$t_5^u \leq t$
$\Omega^{\text{lower}}(t)$		$g_2^u(t)$	0

Figure 4.4: The upper boundaries and the lower boundaries of the feasible sets in seven different cases.

$$h_1^d(t) = v_0 - a_{\text{min}} t + \sqrt{(a_{\text{min}} + a_{\text{max}})(a_{\text{min}} t^2 - 2 v_0 t + 2D)}$$

$$h_2^d(t) = \sqrt{\left(\frac{a_{\text{max}}}{a_{\text{min}}}\right)\left(2 a_{\text{min}} D - v_0^2\right)}$$

$$g_1^u(t) = v_0 + a_{\text{max}} t - \sqrt{(a_{\text{min}} + a_{\text{max}})(a_{\text{max}} t^2 + 2 v_0 t - 2D)}$$

$$g_2^u(t) = v_{\text{max}} - \sqrt{\left(\frac{a_{\text{min}}}{a_{\text{max}}}\right)\left[2 a_{\text{max}} v_{\text{max}} t - (v_{\text{max}} - v_0)^2 - 2 a_{\text{max}} D\right]}$$

$$t_1^d = \frac{v_0 - \sqrt{v_0^2 - 2 a_{\text{min}} D}}{a_{\text{min}}}$$

$$t_2^d = \frac{-v_0 + \sqrt{v_0^2 + 2 a_{\text{max}} D}}{a_{\text{max}}}$$

$$t_3^d = \frac{v_0}{a_{\text{min}}} + \frac{v_{\text{max}}}{a_{\text{max}}} - \sqrt{\left(\frac{a_{\text{min}} + a_{\text{max}}}{a_{\text{min}}^2 a_{\text{max}}^2}\right)\left(a_{\text{max}} v_0^2 + a_{\text{min}} (v_{\text{max}})^2 - 2 a_{\text{min}} a_{\text{max}} D\right)}$$

$$t_4^d = \frac{2 a_{\text{max}} D - (v_{\text{max}} - v_0)^2}{2 a_{\text{max}} v_0}$$

$$t_5^d = \frac{v_0}{a_{\text{min}}} + \sqrt{\frac{2 a_{\text{min}} D - v_0^2}{a_{\text{min}} a_{\text{max}}}}$$

$$t_1^u = \left(\frac{1}{a_{\text{min}}} + \frac{1}{a_{\text{max}}}\right) v_{\text{max}} - \frac{v_0}{a_{\text{max}}} - \sqrt{\frac{(a_{\text{min}} + a_{\text{max}})(v_{\text{max}})^2 - a_{\text{min}} v_0^2 - 2 a_{\text{min}} a_{\text{max}} D}{a_{\text{min}}^2 a_{\text{max}}}}$$

$$t_2^u = \frac{(v_{\text{max}} - v_0)^2 + 2 a_{\text{max}} D}{2 a_{\text{max}} v_{\text{max}}}$$

$$t_3^u = \frac{v_0^2 + 2 a_{\text{min}} D}{2 a_{\text{min}} v_0}$$

$$t_4^u = -\frac{v_0}{a_{\text{max}}} + \sqrt{\left(\frac{a_{\text{min}} + a_{\text{max}}}{a_{\text{min}} a_{\text{max}}^2}\right)\left(v_0^2 + 2 a_{\text{max}} D\right)}$$

$$t_5^u = \left(\frac{1}{a_{\text{min}}} + \frac{1}{a_{\text{max}}}\right)\left(\frac{v_{\text{max}}}{2}\right) - \frac{v_0}{v_{\text{max}}} + \frac{v_0^2 + 2 a_{\text{max}} D}{2 a_{\text{max}} v_{\text{max}}}$$

Figure 4.5: The equations in Fig. 4.4.

4.2 Sampling-based Validation Procedure

In this section, we propose a *sampling-based* validation procedure for a multi-segment path. A naïve sampling algorithm is the one that chooses a point randomly from a feasible set to be used as the initial configurations for the next road segment, until it reaches the last segment. Then it checks whether $(t_{\mathsf{end}}, v_{\mathsf{end}})$ is in the last feasible set. The naïve sampling algorithm is fast, but does not work when there is a large number of road segments. Hence we modify the naïve sampling algorithm to take advantage of our knowledge about the feasible set. The new M-sample algorithm randomly draws M points rather than one point from the set of feasible sets. These M points will become the initial configuration for the next road segment. Then another M points are chosen in the next road segment. At first glance, this strategy is not different from running the naïve sampling algorithm M times. But there is a key difference: the M-sample algorithm allows us to employ some heuristics to spread out the sample points in feasible sets so as to maximum the coverage of the feasible sets. One heuristic we studied is to choose M points that maximizes a distance function among the points to extend the coverage of the union of feasible sets generated from these sample points. At each road segment, we will randomly choose ten sets of M points from the set of feasible sets. Then we will select the set that has the maximum sum of the Euclidean distances of all pairs of points in the set, to be the initial configurations for the next road segments.

4.3 Experiments

We conducted a simulation experiment in Python to compare the M-sample algorithm with the naïve sampling algorithm as well as M-sample without the heuristics. First, we consider different numbers of road segments N, from 1 to 31. Second, for each value of N, we generated 1000 random problems that are valid. The problems are created by randomly choosing the values of the parameters from their predefined ranges: $v_0 \in [0,\ 50]$, $D_i \in [10,\ 600]$, $a_{\mathsf{max},i} \in [0.5,\ 6]$, $a_{\mathsf{min},i} \in [0.5,\ 6]$, and $v_{\mathsf{max},i} = 50$. We randomly generated some velocity schedules, and then the ending configuration after executing each velocity schedule will be used as the arrival configuration $(t_{\mathsf{end}}, v_{\mathsf{end}})$. For each random problem, we ran each of the algorithms repeatedly until it finds a solution or a time limit of 0.2 seconds is exceeded. Then we measured the success rate of over 1000 random problems. This evaluation over 1000 random problems is repeated 100 times to obtain the average and 95% confident intervals as appeared in Fig. 4.6. The 95% confident intervals are shown as the small error bars in the figure lines.

From Fig. 4.6 we can see that the M-sample method with heuristics (the M-heuristic lines) generally has a higher success rate than the M-sample method without heuristics (the M-random lines) if the number of sample points is the same. Both M-sample methods have a much higher success rate than the naïve method (the 1-sample line).

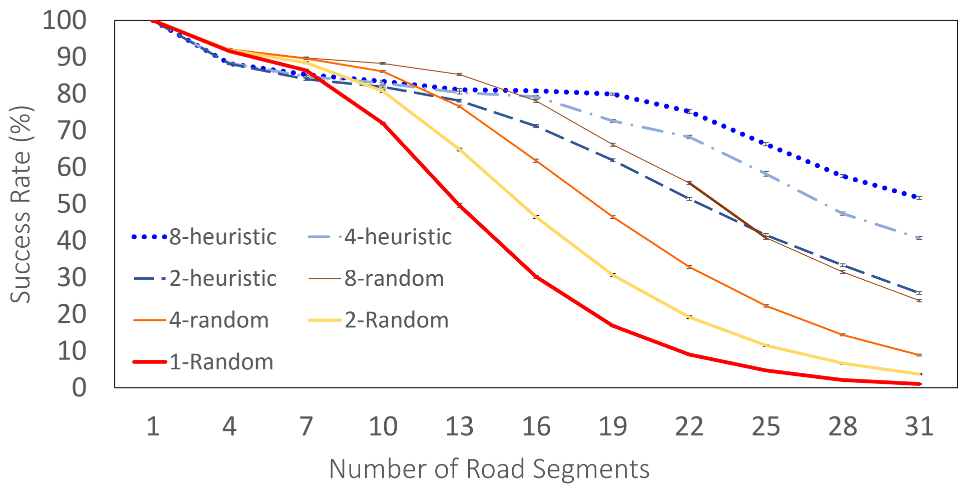


Figure 4.6: Success rates of the sampling algorithms.

4.4 The Use of Feasible Sets on Real Vehicles

We also studied how to utilize feasible sets in the control of a real vehicle. The computation of a feasible set is based on a simplified vehicular model with linear acceleration and deceleration, but the actual behavior of a real vehicle is more complicated than that, and hence the feasible set of a real vehicle is different from the one in Table 4.4. Nonetheless, the actual behavior will only impose additional constraints to the computation of the feasible set, and hence *the feasible set of a real vehicle is a subset of a feasible set in Table 4.4*. Thus, we can use the feasible set to help finding a feasible arrival configuration as well as eliminate arrival configurations that are impossible to reach.

Our experiments were launched on a road consisting two segments, one is flat and the other is a slope. Our goal is to control a physical vehicle to arrive at the top of the slope at certain time and at certain velocity. To do this, we first tried to find a suitable arrival time and velocity at the junction of the two road segments. First, we used Table 4.4 to compute a feasible "superset" at the junction, based on the measurement of maximum acceleration and deceleration. Then we randomly chose a configuration in the superset and then used the bisection method in [4] to check whether there is a setpoint schedule to control the vehicle to arrive at the junction at the chosen configuration. If not, then we chose another configuration in the superset and tested again. Otherwise, we used the bisection method to check whether it is possible to meet the arrival requirement at the top of the slope starting from the chosen configuration. We repeat the process until we find a configuration at the junction that feasible setpoint schedules exist on both road segments. Then we control the vehicle to run according to the two setpoint schedules, and measured the errors in the arrival time and the arrival velocity. We repeated the procedure 32 times with randomly chosen arrival configurations at the top of the slope. The errors in the arrival time and velocity are $-1.42 \pm 0.57s$ and $-0.16 \pm 0.16m/s$, respectively. These errors are probably due to the model error in performance profiling of the vehicle, as well as the sudden slowdown when the vehicle hit the slope at the junction. In the future, we will investigate how to reduce these errors.

CHAPTER V

Vehicle Dynamics Modeling

5.1 Learning Algorithms

In this chapter, I will represent learning schemes to facilitate the modeling of the vehicle dynamics. To begin with, the first section introduces three learning algorithms. In the next section, I will introduce a Plan-based exploration strategy which can be combined with a learning algorithm to make a complete learning scheme. The performance of learning schemes will be evaluated by an experimental result section.

5.1.1 Artificial Neural Networks

According to our problem formulation, a performance model has two functions: T^{stable} and D^{stable}. Our learning task is to estimate these functions by function approximators. One popular function approximator is artificial neural network (ANN). It has been shown that ANN with hidden layers can be used to approximate any functions. One common and well-known algorithm for training ANN with hidden layers is the backpropagation algorithm. Fig. 5.1 shows the ANNs that we used as the performance model. We use two ANNs, one for T^{stable} and the other for D^{stable}, and they work independently. The input nodes are v_0 and v_1, which are the starting velocity and the setpoint, respectively. Both ANNs have a hidden layer with 5 nodes. Given the error in the output nodes, we will use the backpropagation algorithm to adjust the weights on the edges in the ANNs.

The weights of the connections between neurons can be initialized by fitting the reference model.

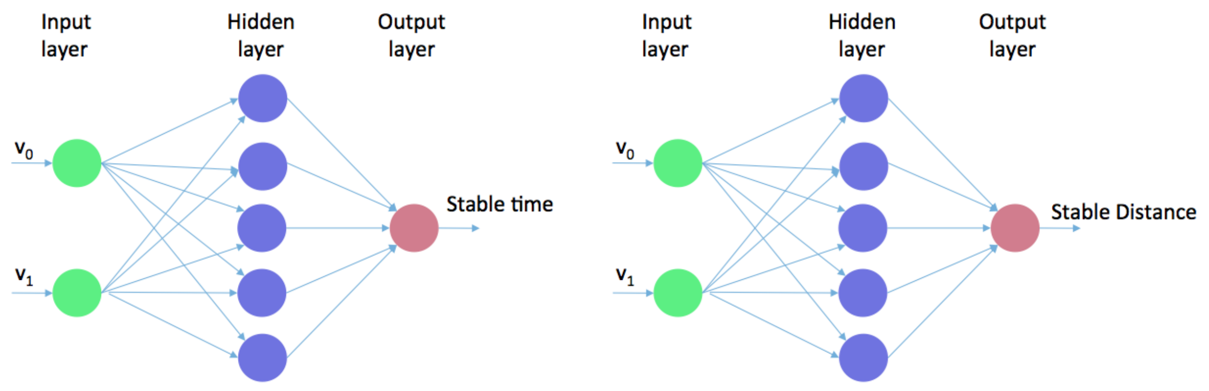


Figure 5.1: A performance model represented by two artificial neural networks

5.1.2 Matrix Factorization with Gradient Descent

As discussed in Chapter II, we represent a performance model as two $M \times M$ matrices whose each row corresponds to an initial velocity and each column relates to a target velocity. I argues that there must be some latent features underlying the interactions between initial and target velocities that determine how long and how far it takes to settle at a new target velocity. Thus, we can adopt recommendation algorithms in solving our learning task. In this work, we favor the matrix factorization algorithm with gradient descent for its efficiency and relatively fast speed in discovering the hidden under the data [**?**]. Let R of size $|U| \times |D|$ be the matrix that contains all the time or distance to change from initial velocities to target velocities and assume that there would be K latent features that need to be discovered. Our task is to figure out two matrices P and Q such that $R \approx P \times Q^T$ and P, Q are $|U| \times K$ and $|D| \times K$ matrices respectively. By doing this, each row of P would indicate the strength of associations between an initial velocity and the features; each row of Q would show the strength of associations between a target velocity and the features. It can be seen that the predicted time or distance of changing from u_i to d_j is the dot product of the two vectors corresponding to u_i and d_j.

$$\hat{r}_{ij} = p_i^T \times q_j = \sum_{k=1}^{K} p_{ik} q_{kj} \tag{V.1}$$

Given the set T of observed elements in the matrix R, we can compute the local difference between an estimated element and its real value as in Equation V.2.

$$e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj})^2 \tag{V.2}$$

Our goal is to find matrices P and Q to minimize these local differences because once we find them, we can fulfill matrix R using equation V.1. The gradient descent method achieves P and Q matrices by finding which direction we should modify the values of p_{ik} and q_{kj} to minimize such local errors. This step is done by partially differentiating e_{ij}^2 with respect to p_{ik} and q_{kj}.

$$\frac{\partial}{\partial p_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(q_{kj}) = -2e_{ij}q_{kj} \tag{V.3}$$

$$\frac{\partial}{\partial q_{kj}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(p_{ik}) = -2e_{ij}p_{ik} \tag{V.4}$$

Based on the obtained gradient, we can formulate the update rules for both p_{ik} and q_{kj} as follows.

$$p'_{ik} = p_{ik} + \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\alpha e_{ij}q_{kj} \tag{V.5}$$

$$q'_{kj} = q_{kj} + \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + 2\alpha e_{ij}p_{ik} \tag{V.6}$$

$$\tag{V.7}$$

where α is a constant that determines the learning rate. In this work, we chose α equal to 0.001. These updates are terminated when the overall error calculated using Equation V.8 less than a constant ε or the improvement in reducing the overall error is small after a number of iterations.

$$\mathbb{E} = \sum_{(u_i, d_j, r_{ij} \in T)} e_{ij}^2 = \sum_{(u_i, d_j, r_{ij} \in T)} (r_{ij} - \sum_{k=1}^{K} p_{ik}q_{kj})^2 \tag{V.8}$$

To avoid overfitting, we add a parameter β to Equation V.2 and modify the squared error as in Equation V.9.

$$e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 + \frac{\beta}{2} \sum_{k=1}^{K} (\|P\|^2 + \|Q\|^2) \tag{V.9}$$

The update rules, therefore, are modified accordingly as in Equation V.12.

$$p'_{ik} = p_{ik} + \alpha(2e_{ij}q_{kj} - \beta p_{ik}) \tag{V.10}$$

$$q'_{kj} = q_{kj} + \alpha(2e_{ij}p_{ik} - \beta q_{kj}) \tag{V.11}$$

$$\tag{V.12}$$

For the experiments, we chose β equal to 0.5 empirically.

5.1.3 Instance-based Learning

In this work, we propose a learning approach that utilizes the performance model on another road while learning the performance model. A reference model is typically the model of a similar vehicle (not need to exactly same) associated with an arbitary road slope and road friction coefficient. In this study, we model $T^{\text{stable}}(v_i, v_j)$ and $D^{\text{stable}}(v_i, v_j)$ as Gaussian distributions and utilize the reference model to obtain the initial parameter values. While utilizing a reference model is optional, it fosters the convergence of the learning process. Let call the reference model T_{ref}^{stable}, T and D are initialized as $T^{\text{stable}}(v_i, v_j) \sim \mathbb{N}(T_{ref}^{\text{stable}}(v_i, v_j), 0)$ and $D^{\text{stable}}(v_i, v_j) \sim \mathbb{N}(D_{ref}^{\text{stable}}(v_i, v_j), 0)$. The second stage is to update the parameters of these gaussian distributions as learning process is executed. Suppose the vehicle has velocity v_i, if we drive it to a new setpoint v_j, we can measure the necessary time t_s for it to be stable and the distance it travels d_s during that time. By doing this, we have explored the performance model and collected a sample of this model. How to schedule a plan to collect sample efficiently will be covered in the next section. We will continue with how to use this sample to update the parameters of the Gaussian distributions. To update every Gaussian distributions with a sample, we utilize multivariate linear regression approach to generate a ”virtual sample” for each Gaussian distribution. The multivariate linear regression models for stable time and stable distance are represented in Fig. V.13. These models are trained with all collected samples (which are practically measured).

$$f(v_i, v_j) = W(v_i, v_j)^T + b_t \tag{V.13}$$

where $f(v_i, v_j)$ is the value of stable time or stable distance After generating ”virtual samples” based on the newly-collected sample, we update parameters for every Gaussian distribution using Equation V.15. For a pair of setpoints (v_i, v_j), the updated mean and standard deviation of the posterior probability at the learning iteration l of $f(v_i, v_j)$ are symbolized as $\mu_{f(v_i,v_j),t}$ and $\sigma_{f(v_i,v_j),t}$ and calculated by the following equations:

$$\mu_{f(v_i,v_j),t} = \frac{\mu_{f(v_i,v_j),0} + \sum_{l=1}^{N} k_l f(v_i, v_j, l)}{1 + Nk_l} \tag{V.14}$$

$$\sigma_{f(v_i,v_j),t} = \sqrt{\frac{(\mu_{f(v_i,v_j),t\cdot} - \mu_{f(v_i,v_j),N})^2 + \sum_{l=1}^{N} k_l (f(v_i, v_j, l) - \mu_{f(v_i,v_j,N)})^2}{1 + Nk_l}} \tag{V.15}$$

Where $\mu_{f(v_i,v_j),0}$ is the initialized mean and equal to the corresponding value of the reference model; f is either f_t or f_d function; $f(v_i, v_j, l)$ is a sample (can be ”virtual” or real) of $f(v_i, v_j)$ at the iteration l. k_l is learning rate which is set to be different with respect to the type of the sample.

5.2 Plan-based Exploration Strategy

As mentioned in the previous section, in this section, we investigate the plan to do sampling efficiently to foster the convergence of the learning process. According to the law of large numbers, a convergence of the estimated model is guaranteed as we collect more and more samples. However, the order of samples collected largely affects the learning speed and the estimated performance model. In general, since the performance model $(T^{\mathsf{stable}}, D^{\mathsf{stable}})$ will be used repeatedly for many different motion planning episodes, each of them uses different parts of the model, we want the estimated model to be as *complete* as possible, meaning that we should spread out the samples so that no part of the model will be ignored forever. To facilitate the sampling process, the vehicle employs an exploration strategy, which determines the sequence of samples so as to minimize the time it takes to learn the performance model.

Here we consider exploration strategies that have the following three characteristics: First, the vehicle will sample at the discrete velocity values only. Let $\mathbb{V} = \{n \times d\}_{n=\{0..m\}}$ be the set of discrete velocity values where d is the discretization step and $d \times m$ is the maximum velocity, which is either the speed limit of the road or the top possible velocity of the vehicle. Second, after a vehicle deliberately changes its setpoint to measure the stable time and the stable distance, it will *immediately* start another measurement right away the vehicle is stabilized at the new setpoint. The result is that the ending velocity of a sampling step is always the starting velocity of the next sampling step. This way the vehicle can avoid the idle time between two samples. Third, the vehicle should avoid collecting the same sample again, because doing the same measurement twice is redundant since we assume that the measurement is exact.

Based on these characteristics, an exploration strategy can be considered as a sequence of setpoints $\langle v_0, v_1, v_2, \ldots, v_n \rangle$, where v_0 is the initial velocity of the vehicle, and v_i is the next setpoint after the vehicle stabilizes at v_{i-1}, for $i \geq 1$. The vehicle, starting with the initial velocity v_0, will first set its setpoint at v_1 and then measure the stable time and stable distance when its velocity stabilizes at v_1. After the measurement, it immediately sets its setpoint at v_2 for the second measurement. Then the process continues until the last setpoint v_n. To ensure completeness, this sequence of setpoints should be chosen to exhibit the property that the set of all possible consecutive pairs of setpoints (i.e., $\{(v_i, v_{i+1})\}_{i=0..(n-1)}$) is exactly the set of all possible pairs of *different* velocities in $\mathbb{V}$ (i.e., $\{(v, v') : v, v' \in \mathbb{V} \text{ and } v \neq v'\}$). Thus, under this exploration strategy, the performance model will converge to the true one.

A faster rate of convergence to the true performance model is important because the vehicle may not have enough time to acquire all the measurements before it uses the performance model for motion planning. It will be quite helpful to the motion planner if the vehicle can get fairly accurate performance model early on. We believe that the order of setpoints in an exploration strategy will greatly affect the speed of learning. In particular, we hypothesize that if an exploration strategy collects samples that will be used to generate a setpoint schedule in advance, the estimated performance model can result in less error in solving the planning problem. Based on this hypothesis, we introduce an exploration strategy called Plan-based sampling as following: 1) use a common planner to generate a plan (in our test bed problem, we use the bisection method) to generate a plan that solves the planning problem based on the

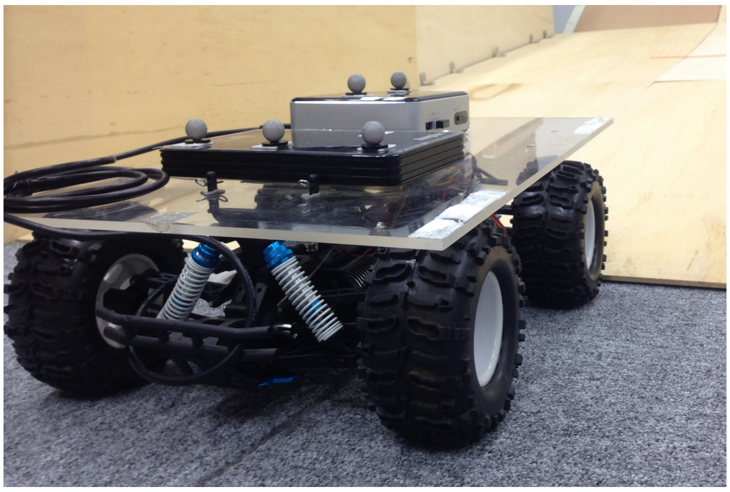


Figure 5.2: The car and the platform used in the experiments

reference model; 2) sample all element in the performance array that captures the changing of setpoints existing in the generated plan, in advance. For example, if the generated plan, respresented by a tuple, is $(t_0,v_0),(t_1,v_1),(t_2,v_2)...(t_n,v_n)$, we would sample elements $(v_0,v_1),(v_1,v_2),...(v_{n-1},v_n)$ with the highest priority. Nevertheless, the generated plan may be not accurate so we also sample elements surrounding these first priority elements. The distance that we sample around these first priority elements depends on the size of the discretized velocities. For our planning problem, a distance d that equal to 2% of total number of discretized velocities provides the best result; 3) sample all other elements with minimum repetition.

5.3 Setpoint Schedule Generation

The previous sections addresses the problem of obtaining the vehicle's performance model. In this section, we introduce two methods to solve the problem $\mathcal{P}_{\mathsf{gen}}$ defined in 3.1.

5.3.1 Bisection Method

This method is first introduced by Au et al. in [4]. Let d the road's length; (t_0,v_0) the initial configuration; (t_{end},v_{end}) the arrival configuration. For every $\hat{v}_{int}$ within the velocity range that we set when obtaining the vehicle's model, we can find a setpoint schedule $\mathbb{S}(v_{int}) =< (t_0,v_0),(t_1,v_{int}),(t_1 +$

$\tau, v_{int}), (t_{end}, v_{end}) >$ where

$$t_1 - t_0 = T^{\mathsf{stable}}(v_0, v_{int}) \tag{V.16}$$

$$t_{end} - t_1 - \tau = T^{\mathsf{stable}}(v_{int}, v_{end}) \tag{V.17}$$

$$\tag{V.18}$$

τ can be computed as $\tau = t_{end} - t_0 - T^{\mathsf{stable}}(v_0, v_{int}) - T^{\mathsf{stable}}(v_{int}, v_{end})$. If the car executes $\mathbb{S}$, its traversal distance can be computed as follows.

$$\mathcal{D}(\mathbb{S}(v_{int})) = D^{\mathsf{stable}}(v_0, v_{int}) + D^{\mathsf{stable}}(v_{int}, v_{end}) + v_{int}\tau \tag{V.19}$$

The bisection method solve $\mathcal{P}_{\mathsf{gen}}$ by searching for v_{int} satisfying $\mathcal{D}(\mathbb{S}(v_{int})) = d$. The procedure starts by letting $\hat{v}_a = v^{min}$, $\hat{v}_b = v^{max}$ and suppose that $\mathcal{D}(\mathbb{S}(\hat{v}_a)) \leq d$ and $\mathcal{D}(\mathbb{S}(\hat{v}_b)) \geq d$. Then, it computes $\hat{v}_c = (\hat{v}_a + \hat{v}_b)/2$ and checks whether $\mathcal{D}(\mathbb{S}(\hat{v}_c))$ is greater than or equal to d. If it is true, set $\hat{v}_b = \hat{v}_c$; otherwise, set $\hat{v}_a = \hat{v}_c$. These steps are repeated until the width of the interval $[\hat{v}_a, \hat{v}_b]$ is smaller than a small threshold, such that $\hat{v}_a \approx \hat{v}_b$. Eventually, we choose $v_{int} = \hat{v}_a$.

5.3.2 Dynamic Programming

This approach first computes the set of all feasible configurations for a road with any given starting velocity v_0. We denote this set by $F(v_0)$, and call F the *feasible set table* of ρ.

Computing the feasible set is not straightforward, because T^{stable} and D^{stable} can be arbitrary functions. We cope with this problem by introducing a dynamic programming algorithm for computing the feasible set, based on a discretization of time, velocity, and distance. Let $\mathbb{V} = \{v_0, v_1, \ldots, v_{m_v}\}$, $\mathbb{T} = \{t_0, t_1, \ldots, t_{m_t}\}$, and $\mathbb{D} = \{d_0, d_1, \ldots, d_{m_d}\}$ be a finite set of velocities, times, and lengths, respectively, so that $\{v \times t : \forall v \in \mathbb{V} \text{ and } \forall t \in \mathbb{T}\} \subseteq \mathbb{D}$.

Let $\mathbb{F}$ be a $|\mathbb{V}| \times |\mathbb{D}|$ table such that $\mathbb{F}(v_0, d)$ is the feasible set if the starting velocity is v_0 and the length of ρ is d. $\mathbb{F}$ can be defined recursively as follows. When $d = 0$,

$$\mathbb{F}(v_0, 0) = \{(0, v_0)\}.$$

When $d > 0$,

$$\mathbb{F}(v_0, d) = \left[\bigcup_{v_{\mathsf{int}} \in V \setminus \{v_0\}} \mathbb{F}^1(v_0, d, v_{\mathsf{int}}) \right] \cup \mathbb{F}^2(v_0, d) \tag{V.20}$$

where

$$\mathbb{F}^1(v_0,d,v_{\mathsf{int}}) = \begin{cases} \textsc{AddTime}(\mathbb{F}(v_{\mathsf{int}}, d - D^{\mathsf{stable}}(v_0,v_{\mathsf{int}})), \\ \qquad\qquad T^{\mathsf{stable}}(v_0,v_{\mathsf{int}})) & \text{if } D^{\mathsf{stable}}(v_0,v_{\mathsf{int}}) \le d, \\ \emptyset & \text{otherwise,} \end{cases}$$

$$\mathbb{F}^2(v_0,d) = \begin{cases} \textsc{AddTime}(\mathbb{F}(v_0, d-\delta d_{v_0}), 1) & \text{if } \delta d_{v_0} \le d, \\ \emptyset & \text{otherwise,} \end{cases}$$

$$\textsc{AddTime}(F,\delta t) = \{\ (t+\delta t, v) : \forall (t,v) \in F\ \},$$

and $\delta d_{v_0} = v_0 \times \delta t$, where δt is the time interval in discretization.

Based on Equation V.20, we can devise a dynamic programming algorithm to compute $\mathbb{F}$.

The algorithm first initializes the entries in the first column ($d = 0$) in $\mathbb{F}$ with $(0, v_0)$ before filling out the rest of the table in ascending order of d. Each entry in the d column depends on some entries in previous columns with smaller d according to Equation V.20. At the end, it returns $F(v_0) = \mathbb{F}(v_0, D)$ for all $v_0 \in \mathbb{V}$. The running time of the algorithm is $O(|\mathbb{T}| \times |\mathbb{V}|^3 \times |\mathbb{D}|)$.

Algorithm 1 Computing the feasible set table of a road.

```
 1: procedure FSetOfEdge($D, T^{\mathsf{stable}}, D^{\mathsf{stable}}$)
 2:     Let $\mathbb{F}$ be a $|\mathbb{V}| \times |\mathbb{D}|$ table.
 3:     $\mathbb{F}(v_0, 0) := \{(0, v_0)\}$ for all $v_0 \in \mathbb{V}$
 4:     for all $d \in \mathbb{D} \setminus \{0\}$ in ascending order do
 5:         for all $v_0 \in V$ do
 6:             Let $\delta d_{v_0} := v_0$ and $\mathbb{F}(v_0, d) := \emptyset$.
 7:             if $\delta d_{v_0} \le d$ then
 8:                 $\mathbb{F}(v,d) := \textsc{AddTime}(\mathbb{F}(v_0, d - \delta d_{v_0}), 1)$
 9:             for all $v_{\mathsf{int}} \in V \setminus \{v_0\}$ do
10:                 if $D^{\mathsf{stable}}(v_0, v_{\mathsf{int}}) \le d$ then
11:                     $\mathbb{F}(v,d) := \mathbb{F}(v,d) \cup \textsc{AddTime}(\mathbb{F}(v_{\mathsf{int}}, d - D^{\mathsf{stable}}(v_0,v_{\mathsf{int}})), T^{\mathsf{stable}}(v_0,v_{\mathsf{int}}))$
        return $\mathbb{F}(\cdot, D)$     // which is equal to $F$
12: procedure AddTime($F, \delta t$)
13:     Let $F' := \emptyset$
14:     for all $(t,v) \in F$ do
15:         $F' := F' \cup \{(t + \delta t, v)\}$
        return $F'$
```

The function AddTime in Algorithm 1 is $O(|\mathbb{T}| \times |\mathbb{V}|)$, because the maximum size of a feasible set is $|\mathbb{T}| \times |\mathbb{V}|$, though in practice the feasible sets are usually much smaller. Since AddTime can be called by at most $|\mathbb{V}|^2 \times |\mathbb{D}|$ times, After computing the feasible set table $\mathbb{F}$, we need to extract a plan π from $\mathbb{F}$ such that π corresponds to a setpoint schedule for a given arrival time and velocity. To facilitate the plan extraction, we record the solutions by adding pointers to $\mathbb{F}$ when we run the dynamic programming algorithm. However, this approach will take a lot of memory space and slow down the algorithm. Fortunately, there is an alternative method that does not require additional memory space.

Suppose we want to find trajectory of a feasible point $(t_{\mathsf{end}}, v_{\mathsf{end}})$ with a starting velocity v_0 and road distance d. Let define a desired setpoint schedule of this configuration as $\mathbb{S}(v_0, v_{\mathsf{end}}, t_{\mathsf{end}}, d)$ and consider an arbitrary point $(T^{\mathsf{stable}}[v_0][v_{\mathsf{int}}], v_{\mathsf{int}})$. It is obvious that

$$\mathbb{S}(v_0, v_{\mathsf{end}}, t_{\mathsf{end}}, d) = \mathbb{S}(v_0, v_{\mathsf{int}}, T^{\mathsf{stable}}[v_0][v_{\mathsf{int}}], D[v_0][v_{\mathsf{int}}]) + \mathbb{S}(vint, v_{\mathsf{end}}, \Delta_t, \Delta_d) \tag{V.21}$$

as long as $\Delta_t + T^{\mathsf{stable}}[v_0][v_{\mathsf{int}}] = t_{\mathsf{end}}$, $\Delta_d + D^{\mathsf{stable}}[v_0][v_{\mathsf{int}}] = d$ with $0 \leq \Delta_t, \Delta_d$ and there exists a trajectory $\mathbb{S}(vint, v_{\mathsf{end}}, \Delta_t, \Delta_d)$. This recursive relationship tells us how to find a desired trajectory based on searching for intermediate points $(T^{\mathsf{stable}}[v_0][v_{\mathsf{int}}], v_{\mathsf{int}})$.

Algorithm 2 Extract trajectory of a feasible point

```
1: procedure EXTRACTTRAJECTORY($v_0, D, t_{end}, v_{end}, T^{\mathsf{stable}}, D^{\mathsf{stable}}, \mathbb{F}$)
2:     Let traj be an array storing intermediate points
3:     Append $(0, v_0)$ to traj                              ▷ First setpoint of trajectory
4:     ADDPOINTS(traj, $t_{end}, v_{end}, T^{\mathsf{stable}}, D^{\mathsf{stable}}, \mathbb{F}$)
5:     return traj
6: procedure ADDPOINTS(&traj, $t, d, v_{end}, T^{\mathsf{stable}}, D^{\mathsf{stable}}, \mathbb{F}$)
7:     if $t \leq 0$ then                                      ▷ The base case
8:         return
9:     for all $v_{\mathsf{int}} \in V$ do
10:        Let $\delta_d = d - D^{\mathsf{stable}}(v_0, v_{\mathsf{int}})$
11:        Let $\delta_t = t - T^{\mathsf{stable}}(v_0, v_{\mathsf{int}})$
12:        if $0 \leq \delta_d, \delta_t$ and $(\delta_t, v_{end}) \in \mathbb{F}(v_{\mathsf{int}}, \delta_d)$ then
13:            Let $t_l$ equal time value of last element of traj
14:            Let $P_{int} = (T^{\mathsf{stable}}(v_0, v_{\mathsf{int}}) + t_l, v_{\mathsf{int}})$
15:            Append $P_{int}$ to traj    ▷ next intermediate point which becomes the starting point of the
     next iteration as following:
16:            ADDPOINTS(traj, $\delta_t, \delta_d, v_{end}, v_{\mathsf{int}}, T^{\mathsf{stable}}, D^{\mathsf{stable}}, \mathbb{F}$) ▷ recursive iteration with new target
     distance, new running time and new starting velocity
```

5.4 Experiments

In this section, we conduct simulation experiments to evaluate the learning methods and the exploration strategy. The evaluation process involves two steps: First, we performed these learning and exploration strategies to estimate vehicle's performance models and calculate the root mean squared error of these models with regard to the true performance model. Second, we used these estimated models with a planner to generate set-point schedules and execute them to evaluate the errors in planning planning. The true models used in these experiments are based on a physical car. To generate a true model, we carry out the following procedure: 1) set up a car and a road specified by a road slope angle and road friction coefficient 2) drive the car with different initial setpoints and target setpoints to measure stable times and stable distances to fullfil the model. In this study, we discretized velocity values from 0 to 1.6 (m/s) with 0.2 (m/s) interval 3) with each sampling, we drive the car 5 times to get the maximum stable time and then measure the distance that the car travels within this time. We carried out this procedure on two type of roads and use a least squared regression method to generalize true models in 360 different settings of road slope angles and friction coefficients. Road slope angles are discretized from -38 degrees to 40 degrees with 2 degrees interval (relative to the horizontal axis) while road coefficients are discretized from 0.1 to 0.9 with 0.1 interval. Figure 5.2 shows the car and the road setup that we used to generate the data. For each parameter setting, our goal is to learn a performance model when the vehicle runs on the slope.

To begin with the first evaluation phase, we implemented our ANNs using ByBrain [38], a machine learning library for Python. The ANNs in this experiment are two feedforword neural networks that use the sigmoid activation function, as shown in Fig. 5.1. Both ANNs consist of two input nodes, which correspond to the starting velocity and the target velocity. We examined the effects of the number of nodes at hidden layer and the number of hidden layers, by allowing the number of nodes at hidden layer to vary between 2 and 10 and the number of hidden layers to vary between 1 and 10. After a thorough evaluation, we settled with one hidden layer and five nodes. As discussed in Section 5.1, the ANNs are pre-trained in the same environment as the one generated the reference model. The ANNs are updated by using the backpropagation function in PyBrain.

The matrix factorization with gradient descent's implementation is straightforward while the implementation of the instance-based learning is based on the update rule as described in Section 5.1. The learning process starts with a performance model of the reference road, and updates the performance model by updating the Gaussian distribution's parameters.

In each experiment, we randomly chose a set of parameters of the reference road as well as the target road we intend to learn. Then we used the reference model to initialize the parameters for both instance-based learning approach and ANNs aroach. After that we started the training stage. We considered two exploration strategies, one is the Plan-based exploration strategy in Section 5.2 and the other is a random strategy that randomly chooses the next setpoint in the learning process. We applied both strategies to learning algorithms. The vehicle used the exploration strategies to try out different setpoints, collect the stable times and stable distances, and update the performance models. The training stage is ended up after a number of explorations or the squared error of performance models is smaller than some number.

We completed the first evaluation phase by computing the root mean squared errors during the learning process against true models as shown in Equation V.22.

$$RMSE = \frac{1}{U \times D}\sqrt{\sum_{i=1}^{U}\sum_{j=1}^{D}(r_{ij} - \hat{r}_{ij})^2} \tag{V.22}$$

where R and $\hat{R}$ are $|U| \times |D|$ matrices; R is the true model and $\hat{R}$ is an intermediate model obtained by the learning process. This allows us to see how quickly the learning approaches can reduce the model error. We plotted the RMSEs as the performance models evolved during the learning process over time. The result is shown in Fig. 5.3 and 5.4. Notice that each data point in the figures is an average of the 300 RMSEs of the 300 trials, and the error bars represent the 95% confident intervals. As can be seen, the instance-based approach outperformed both the ANN approach and the matrix factorization approach in terms of the learning rates in both the stable time and the stable distance. The matrix factorization approach shows the worst performance in this evaluation phase. Meanwhile, the Plan-based exploration strategy is inferior to the random exploration strategy as its convergence speed is lower than that of the random exploration with the same learning approach. This can explained as following: the Plan-based scheme first samples unevenly and focuses on elements related to the plan so is possibly loose to capture the whole model at the beginning. However, this exploration scheme has an advantage that is presented in the second evaluation phase.

In the second evaluation phase, we used intermediate models generated during the training stage to make plans and assessed these plans using the simulation platform. The purpose of this phase is to figure out which learning and strategy approach is appropriate for our set-point scheduling problem. In order to assess an intermediate performance model with a given an problem configuration, we implemented bisection method to generate a set-point schedule and executed it in the simulation platform under a corresponding set of parameters. Each experiment includes the following steps: a) randomly choose a road with a set of parameters as the first evaluation phase and a reference performance model b) do learning performance models for this road with different learning methods and strategies to generate intermediate models c) randomly generate 30 different set-point schedule problems and use bisection method to generate 30 corresponding plans for each intermediate model d) execute those plans and observe the errors in arrival time, velocity and distance.

We did the above experiment 50 times and reported the result. Since we treated arrival velocity as the first priority, the corresponding errors are small so we omitted here. In Fig. 5.5 which demonstrates the error in arrival time, we also eradicated two lines presenting the performance of the random sampling strategy with the ANN and matrix factorization approaches as its convergence speed is very slow. As can be seen, the Plan-based exploration strategy outperformed the random exploration strategy and our proposed learning scheme is superior to the other learning approaches. In addition, the performance of the matrix factorization approach is better than that of the ANN approach.

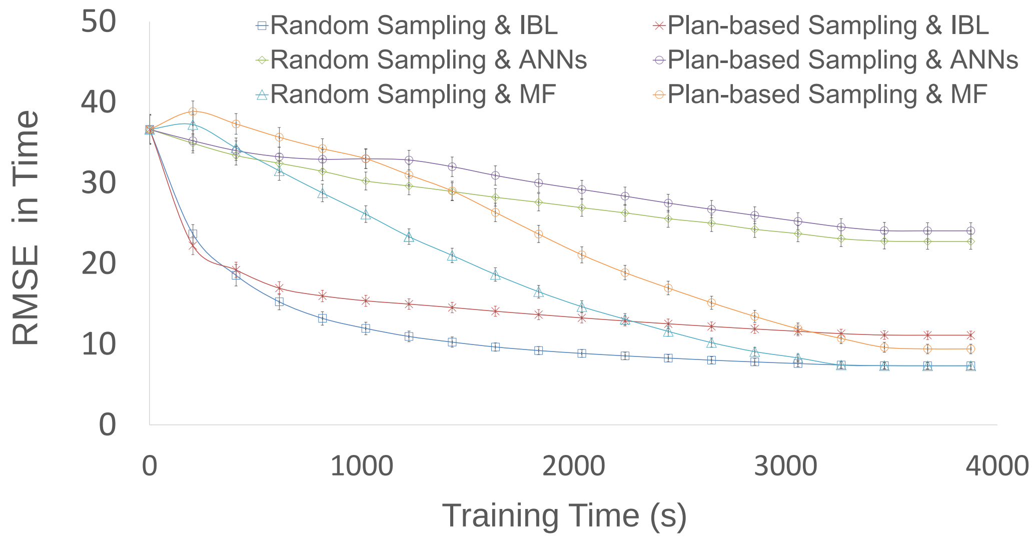


Figure 5.3: RMSEs of stable time versus training time

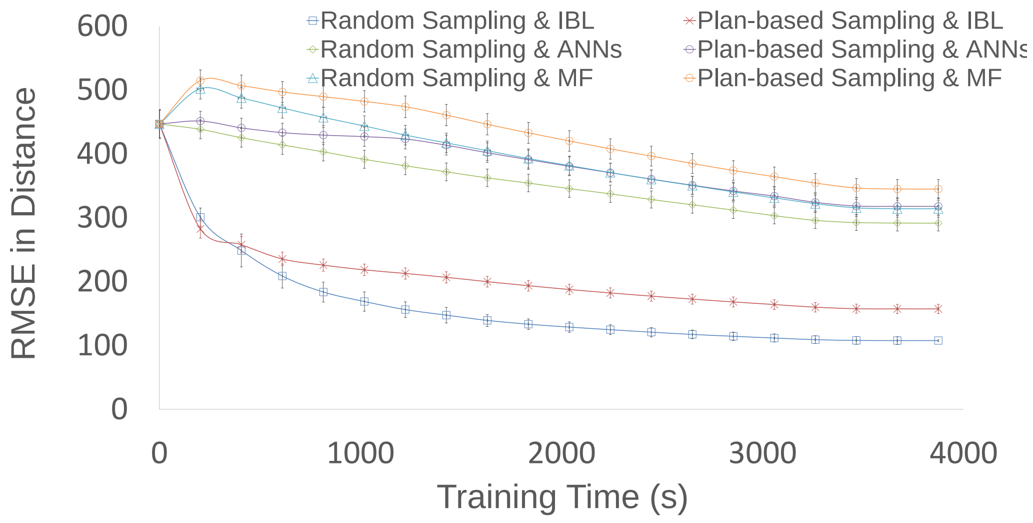


Figure 5.4: RMSEs of stable distance versus training time

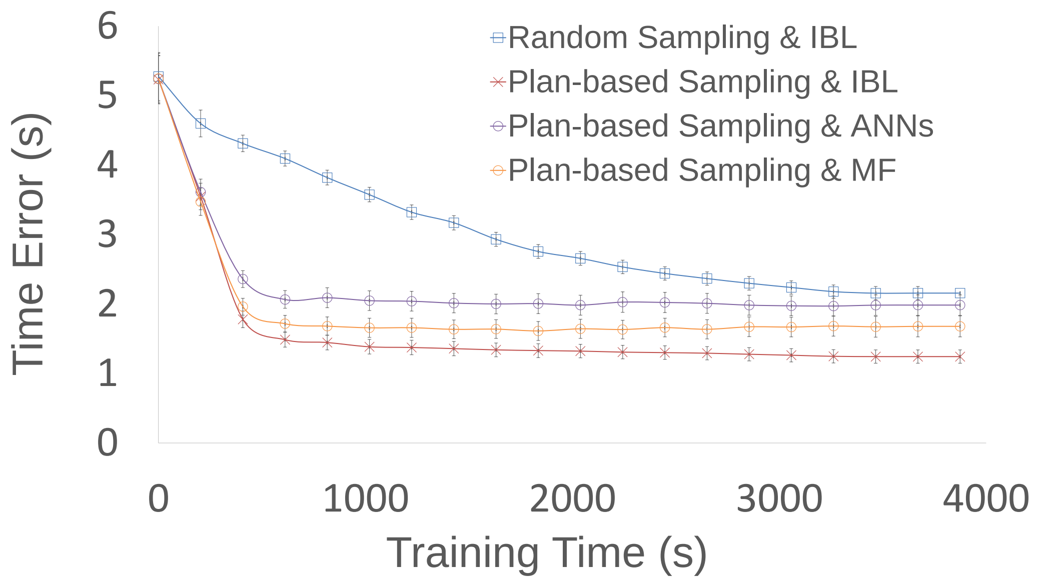


Figure 5.5: RMSEs of execution time versus training distance.

CHAPTER VI

Conclusions

In this work, I first addressed the problem of controlling a robot to arrive at a position on a segmented road at a given time and velocity. I showed that it is possible to completely describe the feasible sets of arrival configurations by closed form equations for any parameters in the problem. The computation of feasible sets can provide a means to solve other path planning problems. Moreover, I devised a sampling-based algorithm to solve the validation problem. The experimental results demonstrated the effectiveness of a heuristics for the sampling-based algorithm to cope with a large number of road segments. The first part of my work is ended with a demonstration on how to use the feasible set in controlling a real vehicle which may not satisfy the vehicle model assumptions. One of possible extensions to this part is to utilize feasible set description to solve the optimization problem in arrival time and velocity.

In the second half of my work, I focused on learning a behavior-based performance model of a vehicle with non-linear control. Instance-based learning approach is suitable for this task because the behavior of a vehicle is quite similar on different roads. I presented a novel instance-based learning approach to learn a performance model of a vehicle's controller. The approach directly adapts the performance model of the reference road for a different road, using an update rule. In essence, my instance-based learning approach is quite similar to transfer learning, through the source domain and the target domain are different in terms of the road settings only. I compared this approach to artificial neural networks that are pre-trained on the same reference road and to the matrix factorization method. An exploration strategy based on the principle of least effort was proposed to speed up the learning process. According to our experiments, both learning schemes can eventually learn the true performance model given enough samples. However, my proposed learning approach outperformed the others and my plan-based exploration strategy significantly facilitates the learning process when doing planning using the obtained models. In the future, I intend to remove the assumption that all measurements are exact and investigate how to select the best performance model as the reference model.

References

[1] Rahmi Akçelik and Mark Besley. Acceleration and deceleration models. In *23rd Conference of Australian Institutes of Transport Research (CAITR 2001), Monash University, Melbourne, Australia*, pages 10–12, 2001. 6

[2] Rahmi Akçelik and DC Biggs. Acceleration profile models for vehicles in road traffic. *Transportation Science*, 21(1):36–54, 1987. 6

[3] S Al-Hasan and G Vachtsevanos. Intelligent route planning for fast autonomous vehicles operating in a large natural terrain. *Robotics and Autonomous Systems*, 40(1):1–24, 2002. 6

[4] Tsz-Chiu Au, Michael Quinlan, and Peter Stone. Setpoint scheduling for autonomous vehicle controllers. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2055–2060, 2012. 5, 20, 27

[5] Tsz-Chiu Au and Peter Stone. Motion planning algorithms for autonomous intersection management. In *AAAI 2010 Workshop on Bridging The Gap Between Task And Motion Planning (BTAMP)*, 2010. 1, 5, 11, 12, 13

[6] G. H. Bham and R. F. Benekohal. Development, evaluation, and comparison of acceleration models. In *81st Annual Meeting of the Transportation Research Board, Washington, DC*, 2002. 6

[7] J. E. Bobrow, S. Dubowsky, and J. S. Gibson. Time-optimal control of robotic manipulators along specified paths. *International Journal of Robotics Research*, 4(3):3–17, 1985. 5

[8] Vuk Bogdanović, Nenad Ruškić, Zoran Papić, and Milan Simeunović. The research of vehicle acceleration at signalized intersections. *PROMET-Traffic&Transportation*, 25(1):33–42, 2013. 2

[9] F. Broqua. Impact of automatic and semi-automatic vehicle longitudinal control on motorway traffic. In *Proceedings of the Intelligent Vehicles '92 Symposium*, pages 144–147, 1992. 5

[10] Martin Buehler, Karl Iagnemma, and Sanjiv Singh. *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*. Springer, 2009. 1

[11] J. Cai, E. Candès, and Z. Shen. A Singular Value Thresholding Algorithm for Matrix Completion. *SIAM Journal on Optimization*, 20(4):1956–1982, January 2010. 7

[12] E. J. Candes and Y. Plan. Matrix Completion With Noise. *Proceedings of the IEEE*, 98(6):925–936, June 2010. 7

[13] Emmanuel J. Candès and Benjamin Recht. Exact Matrix Completion via Convex Optimization. *Foundations of Computational Mathematics*, 9(6):717–772, April 2009. 7

[14] Bruce R. Donald. Motion planning with six degrees of freedom. Technical Report AI-TR-791, MIT Artificial Intelligence Lab, 1984. 4

[15] Kurt Dresner and Peter Stone. A multiagent approach to autonomous intersection management. *Journal of Artificial Intelligence Research (JAIR)*, March 2008. 1

[16] Donald R. Drew. Traffic flow theory and control. Technical report, 1968. 6

[17] Emilio Frazzoli. *Robust Hybrid Control for Autonomous Vehicle Motion Planning*. PhD thesis, Massachusetts Institute of Technology, 2001. 5

[18] D.N. Godbole and J. Lygeros. Longitudinal control of the lead car of a platoon. *IEEE Transactions on Vehicular Technology*, 43(4):1125–1135, 1994. 5

[19] Dan Halperin. Robust geometric computing in motion. *International Journal of Robotics Research*, 21(3):219–232, 2002. 4

[20] Shai Hirsch and Dan Halperin. Hybrid motion planning: Coordinating two discs moving among polygonal obstacles in the plane. In *Algorithmic Foundations of Robotics V*, pages 239–255. Springer, 2004. 4

[21] David Hsu, Jean-Paul Delahaye, and Hanna Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. *International Journal of Robotics Research*, 25(7):627–643, 2006. 4

[22] Jeff Johnson and Kris Hauser. Optimal acceleration-bounded trajectory planning in dynamic environments along a specified path. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2035–2041, 2012. 5, 12

[23] Jeff Johnson and Kris Hauser. Optimal longitudinal control planning with moving obstacles. In *IEEE Intelligent Vehicles Symposium*, pages 605–611, 2013. 5

[24] Lydia E. Kavaki, Petr Švestka, Jean-Claude Latombe, and Mark H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4), 1996. 4

[25] R. H. Keshavan, S. Oh, and A. Montanari. Matrix completion from a few entries. In *IEEE International Symposium on Information Theory, 2009. ISIT 2009*, pages 324–328, June 2009. 7

[26] Ross A. Knepper, Todd Layton, John Romanishin, and Daniela Rus. Ikeabot: An autonomous multi-robot coordinated furniture assembly system. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2013. 1

[27] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(8):30–37, 2009. 7

[28] Tobias Kunz and Mike Stilman. Time-optimal trajectory generation for path following with bounded acceleration and velocity. In *Proceedings of Robotics: Science and Systems*, 2012. 5

[29] Steven M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report TR 98-11, Computer Science Dept, Iowa State University, 1998. 4

[30] Steven M. LaValle and Jr. James J. Kuffner. Rapidly-exploring random trees: progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2000. 4

[31] Nicholas RJ Lawrance and Salah Sukkarieh. Autonomous exploration of a wind field with a gliding aircraft. *Journal of Guidance, Control, and Dynamics*, 34(3):719–733, 2011. 6

[32] Tomás Lozano-Pérez and Michael A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Journal of the ACM*, 1979. 4

[33] Shiqian Ma, Donald Goldfarb, and Lifeng Chen. Fixed point and Bregman iterative methods for matrix rank minimization. *Mathematical Programming*, 128(1-2):321–353, September 2009. 7

[34] Troy McMahon, Shawna Thomas, and Nancy M. Amato. Sampling based motion planning with reachable volumes: Application to manipulators and closed chain systems. In *IEEE/RSJ International conference on Intelligent Robots and Systems*, pages 3705–3712, 2014. 12

[35] Jungme Park, Dai Li, Yi L. Murphey, Johannes Kristinsson, Ryan McGee, Ming Kuang, and Tony Phillips. Real time vehicle speed prediction using a neural network traffic model. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 2991–2996. IEEE, 2011. 6

[36] Michael Quinlan, Tsz-Chiu Au, Jesse Zhu, Nicolae Stiurca, and Peter Stone. Bringing simulation to life: A mixed reality autonomous intersection. In *IEEE/RSJ International conference on Intelligent Robots and Systems*, 2010. 2, 5

[37] Hesham Rakha, Ivana Lucic, Sergio Henrique Demarchi, José Reynaldo Setti, and Michel Van Aerde. Vehicle dynamics model for predicting maximum truck acceleration levels. *Journal of transportation engineering*, 127(5):418–425, 2001. 6

[38] Tom Schaul, Justin Bayer, Daan Wierstra, Yi Sun, Martin Felder, Frank Sehnke, Thomas Rückstieß, and Jürgen Schmidhuber. Pybrain. *Journal of Machine Learning Research*, 11:743–746, 2010. 31

[39] John Searle. Equations for Speed, Time and Distance for Vehicles Under Maximum Acceleration. Technical report, SAE Technical Paper, 1999. 6

[40] Shahab Sheikholeslam and Charles A. Desoer. Longitudinal control of a platoon of vehicles. In *American Control Conference*, pages 291–296, 1990. 5

[41] S.E. Shladover, C.A. Desoer, J.K. Hedrick, M. Tomizuka, J. Walrand, W.-B. Zhang, D.H. McMahon, H. Peng, S. Sheikholeslam, and N. McKeown. Automated vehicle control developments in the path program. *IEEE Transactions on Vehicular Technology*, 40(1):114–130, 1991. 5

[42] Nikolaus Vahrenkamp, Dmitry Berenson, Tamim Asfour, and James Kuffner Rüdiger Dillmann. Humanoid motion planning for dual-arm manipulation and re-grasping tasks. In *IEEE/RSJ International conference on Intelligent Robots and Systems*, pages 2464–2470, 2009. 12

[43] Gokul Varadhan, Shankar Krishnan, T. V. N. Sriram, and Dinesh Manocha. A simple algorithm for complete motion planning of translating polyhedral robots. *International Journal of Robotics Research*, 24(11):983–995, 2005. 4

[44] Gokul Varadhan and Dinesh Manocha. Star-shaped roadmaps—a deterministic sampling approach for complete motion planning. In *Robotics: Science and Systems*, 2005. 4

[45] Pravin Varaiya. Smart cars on smart roads: Problems of control. *IEEE Transactions on Automatic Control*, 38(2), 1993. 5

[46] Petr Švestka and Jules Vleugels. Exact motion planning for tractor-trailer robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2445–2450, 1995. 4

[47] Young Uk Yim and Se-Young Oh. Modeling of vehicle dynamics from real vehicle measurements using a neural network with two-stage hybrid learning for accurate long-term prediction. *Vehicular Technology, IEEE Transactions on*, 53(4):1076–1084, 2004. 6

[48] Liangjun Zhang, Young J. Kim, and Dinesh Manocha. A hybrid approach for complete motion planning. In *IEEE/RSJ International conference on Intelligent Robots and Systems*, 2007. 4