Threat Assessment for Safe Navigation in Environments with Uncertainty in Predictability

by
Georges Salim Aoude
Bachelor of Engineering in Computer Engineering
McGill University, 2005
Master of Science, Aeronautics and Astronautics
Massachusetts Institute of Technology, 2007
Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
August 2011

Author .....
Department of Aeronautics and Astronautics
August 1, 2011

Certified by .....
Jonathan P. How
R. C. Maclaurin Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by .....
Emilio Frazzoli
Associate Professor of Aeronautics and Astronautics

Certified by .....
John J. Leonard
Professor of Mechanical and Ocean Engineering

Accepted by .....
Eytan H. Modiano
Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Threat Assessment for Safe Navigation in Environments with Uncertainty in Predictability

by

Georges Salim Aoude

Submitted to the Department of Aeronautics and Astronautics
on August 1, 2011, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

This thesis develops threat assessment algorithms to improve the safety of the decision making of autonomous and human-operated vehicles navigating in dynamic and uncertain environments, where the source of uncertainty is in the predictability of the nearby vehicles' future trajectories.

The first part of the thesis introduces two classes of algorithms to classify drivers behaviors at roads intersections based on Support Vector Machines (SVM) and Hidden Markov Models (HMM). These algorithms are successfully validated using a large real-world intersection dataset, and can be used as part of future driver assistance systems. They are also compared to three popular traditional methods, and the results show significant and consistent improvements with the developed algorithms.

The second part of the thesis presents an efficient trajectory prediction algorithm that has been developed to improve the performance of future collision avoidance and detection systems. The proposed approach, RR-GP, combines the Rapidly-exploring Random Trees (RRT) based algorithm, RRT-Reach, with mixtures of Gaussian Processes (GP) to compute dynamically feasible paths, in real-time, while embedding the flexibility of GP's nonparametric Bayesian model. RR-GP efficiently approximates the reachability sets of surrounding vehicles, and is shown in simulation and on naturalistic data to improve the performance over two standard GP-based algorithms.

The third part introduces new path planning algorithms that build upon the tools that have been previously introduced in this thesis. The focus is on safe autonomous navigation in the presence of other vehicles with uncertain motion patterns. First, it presents a new threat assessment module (TAM) that combines the RRT-Reach algorithm with an SVM-based intention predictor, to develop a threat-aware path planner. The strengths of this approach are demonstrated through simulation and experiments performed in the MIT RAVEN testbed. Second, another novel path planning technique is developed by integrating the RR-GP trajectory prediction algorithm with a state-of-the-art chance-constrained RRT planner. This framework provides several theoretical guarantees on the probabilistic satisfaction of collision avoidance constraints. Extensive simulation results show that the resulting approach

can be used in real-time to efficiently and accurately execute safe paths.

The last part of the thesis considers the decision-making problem for a human-driven vehicle crossing a road intersection in the presence of other, potentially errant, drivers. The proposed approach uses the TAM framework to compute the threat level in real-time, and provides the driver with a warning signal and the best escape maneuver through the intersection. Experimental results with small autonomous and human-driven vehicles in the RAVEN testbed demonstrate that this approach can be successfully used in real-time to minimize the risk of collision in urban-like environments.

Thesis Supervisor: Jonathan P. How
Title: R. C. Maclaurin Professor of Aeronautics and Astronautics

Acknowledgments

First of all, I would like to thank my advisor Prof. Jonathan How for his guidance and support over the years of this thesis. His pursuit of excellence in research and his constant desire to push the limits were very inspiring. Being his student was a great learning experience at many different levels. Many thanks also go to the other committee members Prof. Emilio Frazzoli and Prof. John Leonard. Prof. Frazzoli was a tremendous source of motivation and inspiration. He provided me with a wealth of insights and ideas that led to significant advances in my research. Prof. Leonard was a great source of inputs, particularly during the Ford project. His feedback on the thesis also improved the quality of the document. My thesis readers, Dr. Karl Iagnemma and Dr. Luca Bertuccelli generously provided many suggestions that enhanced the quality of the thesis.

I would like to thank my collaborators Brandon Luders, Vishnu Desaraju, Josh Joseph, Dan Levine, and Kenneth Lee. Brandon was very helpful with the integration of the trajectory prediction algorithm with his state-of-the-art CC-RRT planner, and the testing of the resulting approach. Collaborating with him also led to several other successful research efforts. Working with Vishnu Desaraju on behavior classification was very enjoyable. His subtle humour made the long hours spent on the project feel much shorter. I would also like to thank him for his generosity to proofread the thesis and provide me with very valuable feedback. Josh Joseph introduced me to the world of Gaussian Processes, and never hesitated to give his feedback and inputs on the subject. Thank you for being such an amazing collaborator.

Several people have also contributed to shaping my research experience at MIT. In particular, I would like to thank Dr. Tom Pilutti, Dr. Wassim Najm, Dr. Luke Fletcher, Prof. Nicholas Roy, Sertac Karaman, Alborz Geramifard, Tuna Toksoz, Dr. Yoshi Kuwata, Dr. Carl Nehme, Dr. Chun Sang Teo, Dr. Zahi Karam, and Dr. Ali Shoeb. Special credit also goes to the many undergraduate students I supervised over the years, especially Lauren Stephens, Ben Switala, and Alejandro Dos Reis for their remarkable contributions.

Thanks also to all my former and current labmates in the Aerospace Controls Laboratory. In particular, I would like to thank Vishnu Desaraju, Brandon Luders, Alborz Geramifard, Sameera Ponda, Josh Redding, Buddy Michini, Kemal Ure, Tuna Toksoz, Kenneth Lee, Dan Levine, Luke Johnson, Mark Cutler, Jim McGrew, Karl Kulling, Andrew Whitten, Frank Fan, Andrew Kopeikin, Dr. Aditya Undurti, Dr. Chun Sang Teo, Dr. Louis Breger, Dr. Yoshiaki Kuwata, and Dr. Han-Lim Choi.

Special thanks to Kathryn Fischer, Barbara Lechner, Beth Marois, Marie Stuppard, Brían O'Conaill, and Quentin Alexander for their outstanding administrative help.

I am very grateful to the great friends I met at MIT. Samir Mikati, Hazem Zureiqat, Jeff Soto, Mary-Irene Alexandrakis, Carl Nehme, Sherif Kassatly, Dana Najjar, Salman Aldukheil, Zeina Saab, Ali Shoeb, Zahi Karam, John Boghossian, Alice Nawfal, Ghassan Fayyad, Cyril Koniski, Bahjat Dagher, Jad Mezher, Bernard Khoury, Lana Awad, Gaylee Saliba, Houssan Zebian, Ghassan Wakim, Anwar Ghauche, Toufic Gebran, the friends from the Greek Club, and all the current and past Lebanese Club members: you made my PhD years very memorable, and I will always cherish our friendship.

Carine Abi Akar, a heartfelt thank you for your support and encouragement, especially over the last several months. You were truly a great source of motivation; I am blessed to have you in my life. You made the PhD years a joyful experience.

Finally, I would like to acknowledge my dearest family: Mom, Dad, and my two sisters, Zeina and Georgette. I cannot thank you enough for all the love and support you have showed me. I would not have done it without you. Dad, I will always remember your sacrifices and generosity towards me; Mom, I will never forget your long trips to visit me when I needed you the most; and Sisters, I am very appreciative of your continuous faith in me that always motivated me to do my best. I am also very grateful to my uncle Metropolitan Elias for all his prayers and encouragement. Lastly, and most importantly, I would like to thank God for giving me this opportunity and the strength to complete it.

I dedicate this thesis to my wonderful parents, Salim and Samar.

This research was funded in part by the Ford-MIT Alliance whose initial funding led to the seedling of the ideas in the thesis, and the continued funding of Scientific Systems Company, Inc. (SSCI) under Grants #N68335-09-C-0472 and #N68335-09-C-0590. The author also gratefully acknowledges funding from Le Fonds Québécois de la Recherche sur la Nature et les Technologies (FQRNT) Graduate Award.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and Motivation

Safety in transportation is becoming increasingly important because of the growing volume of traffic both on the roads and in the air [1, 2].

Taking advantage of recent improvements in sensors such as infrared, cameras, and radars, active safety features such as lane departure warning, adaptive cruise control, and blind-spot warning, have become widely deployed in automobiles to reduce road accidents [3, 4]. These active safety systems assess the level of perceived threat and take a range of actions from simple warnings to full interventions to minimize the risk or severity of collisions [5]. But road safety remains a serious public economical and health issue [6]. In 2008, road accidents in the United States caused 37,261 fatalities and 2.35 million injuries, leading to an estimated societal cost of 230 billion US dollars [7]. In the aviation field, traffic alert and collision avoidance systems (TCAS) have significantly reduced mid-air collisions for commercial airlines over the last thirty years [8]. More recently, Automatic Dependent Surveillance-Broadcast (ADS-B) technology has been developed to improve aircraft situational awareness by enabling procedures not available with current TCAS systems, such as receiving detailed weather reports and traffic information of all surrounding aircraft [9]. However, conflict detection and collision avoidance still present several challenges for aviation safety [10, 11]. For example, in Naval Aviation, mid-air collisions are still among the top-five causes

for Class-A accidents, ones that involve fatalities, destroyed aircraft, or two million dollars or more of property damage [12]. Furthermore, as unmanned aerial vehicles (UAV) will be sharing the airspace with commercial and military aircraft, new detect, sense and avoid (DSA) technologies must be developed [13]. Similarly, autonomous cars will be navigating urban roads and interacting with other drivers and pedestrians, requiring new capabilities for anticipating and avoiding accidents [14, 15].

To tackle these challenges, both manned and autonomous vehicles (cars and aircraft) will require improved situational awareness to enable better collision detection and avoidance strategies through enhanced communication and trajectory prediction capabilities [16]. While developing communication technologies is not the focus of this thesis, it is an active area of research in both ground and air domains [17, 18]. The other major challenge that these systems must address is the uncertainty in predicting the intents and future trajectories of the surrounding vehicles [19–22].

Consider for example the difficult driving task of negotiating a traffic intersection safely. An estimated 45 percent of injury crashes and 22 percent of roadway fatalities in the US are intersection-related [23]. A main contributing factor in these accidents is the driver's inability to correctly assess the danger involved in such situations [24]. Therefore, future advanced driver assistance systems (ADAS) must include new risk assessment algorithms for intersections, one of the most dangerous road scenarios due to the difficulty in predicting incoming drivers' trajectories [25].

Similar problems must be addressed for the Next Generation Air Transportation System (NextGen). This initiative aims to increase the safety and capacity of air transport operations. To achieve these goals, new conflict detection algorithms must be developed to allow tighter separation distances, which will require improved trajectory and intent prediction algorithms [26].

To address these challenges, several new algorithms have been developed in this thesis, and their contributions are summarized in the following section.

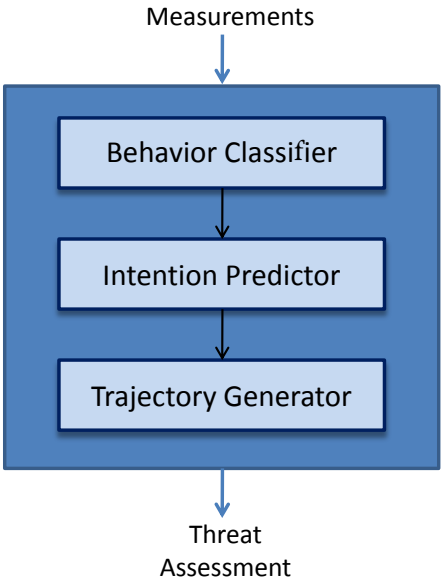


Figure 1-1: Threat Assessment Architecture.

1.2 Outline and Summary of Contributions

This thesis develops threat assessment algorithms to improve the safety of the decision making process in dynamic and uncertain environments. The focus is on the uncertainty in predicting the other vehicles' intents and trajectories. To deal with different sources and types of uncertainties in prediction, the thesis introduces the threat assessment architecture (Figure 1-1). Several algorithms implementing the components of the architecture are introduced and demonstrated through simulations, hardware experiments and real traffic data testing and validation. The contributions of each chapter in this thesis are summarized below.

- Chapter 2 develops solution approaches for the behavior classification component, tailored for road driving. More specifically, the main contribution of this chapter is two novel algorithms that model the behavior of drivers at road intersections using Support Vector Machines (SVM) and Hidden Markov Models (HMM) [27, 28]. The merits of these algorithms are demonstrated through real-traffic validation using a large naturalistic intersection dataset collected through the US Department of Transportation Cooperative Intersection Collision Avoidance System for Violations (CICAS-V) initiative. Their performances are also compared to those of three popular traditional methods, and the results show

significant improvements with the developed algorithms in terms of classification accuracy and false warning rates.

- Chapter 3 presents a solution approach that combines the intention prediction and trajectory generator components of the threat assessment framework. The main contribution is an efficient trajectory prediction algorithm that has been developed to improve the performance of future collision avoidance and detection systems [29]. The main idea is to embed the inferred intention information of surrounding agents into their estimated reachability sets to obtain a probabilistic description of their future paths. More specifically, the proposed approach combines the recently developed rapidly-exploring random trees (RRT) based algorithm, RRT-Reach [30], with mixtures of Gaussian Processes. The resulting approach, denoted as RR-GP, has RRT-Reach's benefits of computing trajectories that are dynamically feasible by construction, and the flexibility of GP's nonparametric Bayesian model, therefore efficiently approximating the reachability sets of surrounding vehicles. A demonstrative simulation on a car-like vehicle illustrates the advantages of the RR-GP approach, and highlights its advantages over two other GP-based algorithms. Another contribution of this chapter is the validation of the RR-GP approach on real-traffic data from the CICAS-V project.

- Chapter 4 develops path planning algorithms for autonomous vehicles navigating in unpredictable and dynamic environments. These algorithms embeds two different implementations of the threat assessment architecture to extend the capabilities of well established path planners in order to handle the uncertainty in predictability.

  The first contribution is a new threat-aware planning approach that is applied to the closed-loop RRT (CL-RRT) path planning framework [31], used by the MIT team in the 2007 DARPA Grand Challenge [32]. The novelty is in the threat assessment module that consists of an intention predictor and a threat assessor, which augments the host vehicle's path planner with a real-time threat

value representing the risks posed by the estimated intentions of other vehicles. The strengths of this approach are demonstrated through simulation and experiments performed in the RAVEN testbed facilities.

Another contribution is the integration of RR-GP algorithm (See Chapter 3) in a state-of-the-art probabilistic path planning approach (CC-RRT), to extend its formulation to handle dynamic objects with uncertain intents [33]. CC-RRT is a state-of-the-art chance-constrained path planning approach based on rapidly-exploring random trees [34]. CC-RRT explicitly incorporates probabilistic constraint satisfaction in its formulation while maintaining the computational benefits of sampling-based algorithms. With RR-GP embedded in the CC-RRT framework, theoretical guarantees are demonstrated for linear systems, though the extension to nonlinear systems is also considered. Extensive simulation results show that the resulting approach can be used in real-time to efficiently and accurately execute safe paths.

- Chapter 5 presents an implementation of the threat assessment framework (Figure 1-1) for the decision-making problem of a human-driven vehicle crossing a road intersection in the presence of other, potentially errant, drivers. The main contribution is a novel threat assessment module, which combines an intention predictor based on support vector machines with an efficient threat assessor using RRT-Reach, a novel sampling-based reachability algorithm [30]. This module warns the host driver with the computed threat level and the corresponding best escape maneuver through the intersection. Through experimental results with small autonomous and human-driven vehicles in the Aerospace Controls Laboratory RAVEN's testbed, we demonstrate that this threat assessment module can be used in real-time to minimize the risk of collision in urban-like environments. This chapter also includes another game-theoretic formulation that extends RRT-Reach to a dual exploration-pursuit mode that considers the worst-case scenario of errant drivers that are capable of causing intentional collisions [35].

Chapter 2

Behavior Classification

The ability to classify driver behavior lays the foundation for more advanced driver assistance systems. In particular, improving safety at intersections has been identified as a high priority due to the large number of intersection related fatalities. This chapter focuses on developing algorithms for estimating driver behavior at road intersections and validating them on real traffic data [27, 28]. It introduces two classes of algorithms that can classify drivers as compliant or violating. They are based on *i*) Support Vector Machines (SVM) and *ii*) Hidden Markov Models (HMM), which are two very popular machine learning approaches that have been used successfully for classification in multiple disciplines. However, existing work has not explored the benefits of applying these techniques to the problem of driver behavior classification at intersections. The developed algorithms are successfully validated using naturalistic intersection data collected in Christiansburg, VA, through the US Department of Transportation Cooperative Intersection Collision Avoidance System for Violations (CICAS-V) initiative. Their performances are also compared to those of three traditional methods, and the results show significant improvements with the new algorithms.

2.1 Introduction

The field of road safety and safe driving has witnessed rapid advances due to improvements in sensing and computation technologies. Active safety features like anti-lock braking systems and adaptive cruise control have been widely deployed in automobiles to reduce road accidents [3]. However, the US Department of Transportation (DOT) still classifies road safety as "a serious and national public health issue." In 2008, road accidents in the US caused 37,261 fatalities and about 2.35 million injuries. A particularly challenging driving task is negotiating a traffic intersection safely; an estimated 45 percent of injury crashes and 22 percent of roadway fatalities in the US are intersection-related [23]. A main contributing factor in these accidents is the driver's inability to correctly assess and/or observe the danger involved in such situations [24]. This data suggests that driver assistance or warning systems may have an appropriate role in reducing the number of accidents, improving the safety and efficiency of human-driven ground transportation systems. Such systems typically augment the driver's situational awareness, and can also act as collision mitigation systems [30].

Research on intersection decision support systems has become quite active in both academia and the automotive industry. In the US, the federal DOT, in conjunction with the California, Minnesota, and Virginia DOTs and several US research universities, is sponsoring the Intersection Decision Support (IDS) project [24, 36], and more recently the Cooperative Intersection Collision Avoidance Systems (CICAS) project [37]. In Europe, the InterSafe project was created by the European Commission to increase safety at intersections. The partners in the InterSafe project include European vehicle manufacturers and research institutes [25]. Both projects try to explore the requirements, tradeoffs, and technologies required to create an intersection collision avoidance system, and demonstrate its applicability on selected dangerous scenarios [24, 25].

This research is focused on developing algorithms that infer driver behaviors at road intersections, and validating them using naturalistic data. The resulting algo-

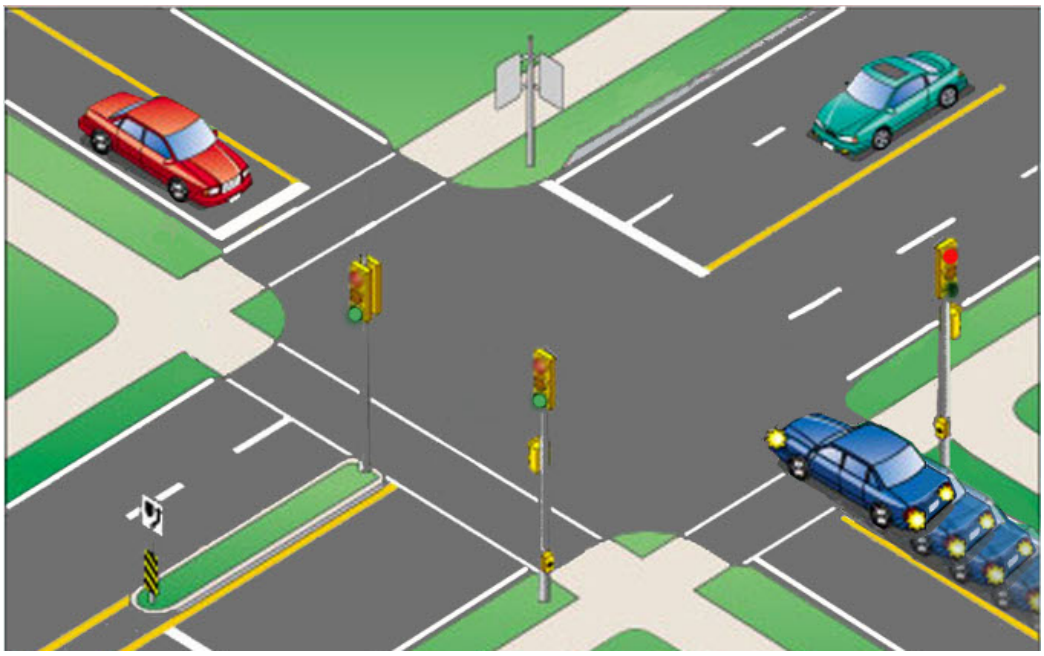


Figure 2-1: Red light violation at signalized intersection. Adapted from www.drivingschool.ca.

rithms can be applied to either vehicle-based systems or infrastructure based systems. Inferring driver intentions has been the subject of extensive research. For example, Ref. [38] introduced a mind-tracking approach that extracts the similarity of the driver data to several virtual drivers created probabilistically using a cognitive model. Ref. [39] used graphical models and Hidden Markov Models to create and train models of different driver maneuvers using experimental driving data.

More specifically, the modeling of behavior at intersections has been studied using different statistical models [40–46]. These studies showed that the stopping behavior depends on several factors including driver profile (e.g., age and perception reaction time), yellow-onset kinematic and geometric parameters (e.g., vehicle speed and distance to intersection). Ref. [42] developed red light running predictors based on estimating the time to arrival at intersections and the different stop and go maneuvers. It used speed measurements at two discrete point sensors. But the performance of their approach is limited by the complexity of the multidimensional optimization problem that must be solved. Closely related to the focus of this chapter is the work presented in Refs. [45, 46]. Ref. [45] discusses the use of time to intersection

(TTI) and its advantages over time to collision (TTC) for intersection safety systems. Ref. [46] developed different warning algorithms for signalized and stop intersections based on the required deceleration parameter (RDP), TTI, and speed-distance regression models. These algorithms will be used as a baseline for comparison in our work since they have been widely used in the driver behavior classification literature, and have also been validated on a large traffic dataset [47]. Note, however, that those authors only consider simple relationships between the driving parameters, while the algorithms developed in this work have the flexibility to combine many parameters in the same model.

This chapter develops two novel classes of algorithms based on distinct branches of classification in machine learning to model driver behaviors at signalized intersections. It also successfully validates these algorithms on a large naturalistic dataset. First, it describes the driver behavior inference problem and the different factors involved in the decision making. Then, it introduces the two classes of algorithms, a discriminative approach based on Support Vector Machines (SVM), and a generative approach based on Hidden Markov Models (HMM), along with the traditional approaches that they are compared to. Next, it describes the implementation process of the different algorithms. Finally, it evaluates their performance on intersection data collected in Christiansburg, VA as part of the DOT Cooperative Intersection Collision Avoidance System for Violations (CICAS-V) initiative [37].

2.2 Problem Statement

Consider an intersection controlled by a traffic signal as shown in Fig. 2-1. As a vehicle approaches the intersection, the objective is to predict from a set of observations whether the driver will stop safely if the signal indicates to do so. Drivers who do not stop before the stop bar are considered to be *violators*, while those who do stop are considered to be *compliant*. Naturally, drivers behave differently, and the variation in the resulting observations must be taken into account in the classification process.

The ability to classify drivers lays the foundation for more advanced driver assis-

Table 2.1: Classification categories

	Classification: Compliant	Classification: Violating
Actual: Compliant	True Negative	False Positive
Actual: Violating	False Negative	True Positive

tance systems. In particular, these systems would be able to warn drivers of their own potential violations as well as detect other potential violators approaching the intersection. Integrating the classifier into a driver assistance system imposes performance constraints that balance violator detection accuracy with driver annoyance.

This requirement can be encoded in terms of signal detection theory (SDT), which provides a framework for evaluating decisions made in uncertain situations [48]. Table 2.1 shows the mapping between classifier output and the SDT categories. To meet this performance constraint, the classifier must maximize the number of true positives (to correctly identify violators) while maintaining a low ratio of false positives (to minimize driver annoyance).

An underlying assumption for this classification is the availability of communication or sensing infrastructure to provide the observations needed to classify the driver's behavior and enable the detection of traffic signal phase. Vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication systems would provide exactly this functionality and are an active area of research [18, 49]. Alternatively, onboard sensors could be used to make these observations, especially when warning drivers of their own impending violations [14, 15].

While several scenarios could be considered for this problem, this work focuses on the case consisting of one host vehicle and several target vehicles. The goal is to is to warn the host vehicle when any of the target vehicles is predicted not to comply with the traffic lights. To further specify the problem, the following assumptions are made:

- The host vehicle has the right of way and is compliant. Only the target vehicles that do not have the right of way are considered in the problem; the other ones

(i.e., with right of way) are ignored. In other words, the focus is on warning compliant drivers from the danger created by other potentially violating drivers. An implicit assumption is the existence of V2V and V2I systems to detect the traffic signal phase, and to share position, speed, and acceleration information among vehicles.

- The host vehicle is warned at t_{warn} only when a target vehicle is classified as violating. Fig. 2-2 illustrates the different warning-related variables. t_{warn} corresponds to the time when a target vehicle's estimated time to arrive at the intersection, also known as time to intersection (TTI) [50], reaches TTI_{min} seconds, or when the distance of a target vehicle to the intersection equals d_{min} meters, whichever condition happens first. The time and distance thresholds are chosen such that the host driver has enough time to react to the warning. A detailed analysis of the choice of TTI_{min} and d_{min} is presented in Section 2.5.2.

- The target vehicles are tracked as early as possible, but their classification as violating or compliant is based on measurements taken in the T_{w} time window (Fig. 2-2). Different values of T_{w} are analyzed in the developed algorithms; larger T_{w} brings a longer measurement "memory" at the expense of an additional computation requirement. A large T_{w} might also include irrelevant measurements when the vehicle is very far from the intersection. Finally, note that a target vehicle that stops in or before the T_{w} window is directly labeled as compliant.

2.3 Algorithms

Classifying human drivers is a very complex task because of the various nuances and peculiarities of human behaviors [51]. Researchers have shown that the state of a vehicle driver lies in some high dimensional feature space [52].

Basic classification is traditionally performed by identifying simple relationships or trends in data that define each class. This includes using techniques such as model

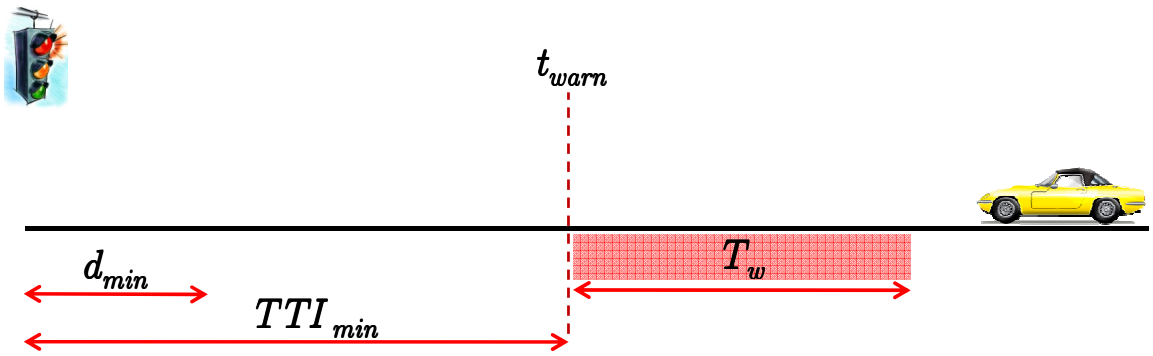


Figure 2-2: Target vehicle approaching intersection. Classification is performed in the T_{w} time window, and warning is potentially sent to host vehicle at t_{warn} measured as the expected time to reach the intersection. In some cases the time corresponding to d_{min} can be larger than TTI_{min}.

fitting and regression to identify classification criteria [53]. However, by only considering simple relationships, these approaches are limited in their ability to accurately classify complex data where the classes may be defined by a variety of factors. To overcome this limitation, two approaches to classification have been developed in the machine learning community.

Discriminative approaches, such as support vector machines (SVM), are typically used in binary classification problems, which makes them appropriate for the classification of compliant vs. violating drivers. SVMs have several useful theoretical and practical characteristics [54]. We highlight two of them: 1) training SVMs involves an optimization problem of a convex function, thus the optimal solution is a global one (i.e., no local optima), 2) the upper bound on the generalization error does not depend on the dimensionality of the problem.

Classification is often also performed using generative approaches, such as HMMs, to model the underlying patterns in a set of observations and explicitly compute the probability of observing a set of outputs for a given model [55]. HMMs are well suited to the classification of dynamic systems [39], such as a vehicle approaching an intersection. The states of the HMM define different behavioral modes based on observations, and the transitions between these states captures the temporal relationship between observations.

2.3.1 Traditional Methods

This section presents three notable techniques from the literature for classifying drivers based on simple relationships between observations: static *TTI* [50], static *RDP* [45], and speed-distance regression [46]. These algorithms are widely used and provide a baseline for the performance analysis in Section 5.5.4.

Static TTI

One of the most intuitive approaches to classification is to use the vehicle's time to intersection (TTI). It is thought that this type of temporal property is what humans use to decide whether a vehicle will be compliant [56]. The vehicle's TTI is defined simply as

$$TTI = \frac{r}{v} \tag{2.1}$$

where v is the vehicle's current speed and r is its distance to the stop line. For this classifier, the TTI value is computed when the vehicle's deceleration crosses some predefined threshold, indicating the onset of braking. Then the driver is classified as a violator if $TTI < TTI_{\text{req}}$, where TTI_{req} is the time given for a driver to stop safely after the onset of braking [50]. This static parameter can be adjusted to change how conservative the algorithm is in its classifications.

Static RDP

Other work has used a classifier based on a required deceleration parameter (RDP) [45, 46]. This parameter gives the deceleration needed for the vehicle to stop safely given its current distance and speed. It is defined as

$$RDP = \frac{v^2}{2rg} \tag{2.2}$$

where r and v are as defined above and g is the gravitational acceleration constant. For a given RDP threshold, RDP_{warn}, a warning distance is computed as

$$r_{\text{warn}} = \frac{v^2}{2RDP_{\text{warn}}} \tag{2.3}$$

The vehicle is then classified as a violator if at any time $r < r_{\text{warn}}$, i.e., if its required deceleration is greater than the selected RDP threshold.

Speed-Distance Regression

Another more complex classification strategy is based on fitting regression curves [46, 57]. The approach takes a set of speed and distance measurements from vehicles that are known to be compliant, discretizes the speeds, and collects the distance measurements at each speed into a set of bins based on percentiles. A regression curve of the form

$$r_{\text{warn}} = av^b + c \tag{2.4}$$

is then fit for each bin. These speed-distance regression (SDR) curves attempt to identify relationships between speed and distance that can discriminate between compliant and violating drivers. For a given curve (corresponding to a certain percentile of compliant driver trajectories), the vehicle is classified as a violator if at any time $r < r_{\text{warn}}$, i.e., if it is closer to the stop line than expected for a compliant vehicle at its current speed. Selecting a curve corresponding to higher percentile bins yields a more conservative classifier.

The version of this algorithm in Ref. [46] includes two additional layers that declare a vehicle to be compliant if its deceleration is below some fixed threshold (e.g., due to braking) or if its velocity is below some fixed threshold (e.g., to permit rolling stops). However, only the deceleration layer is considered here as rolling stops are typically not associated with signalized intersections.

Figure 2-3: The SVM-BF Architecture.

2.3.2 SVM-BF

The first developed algorithm, denoted as SVM-BF, combines SVM and Bayesian filtering. The core of the algorithm is the SVM, a popular supervised machine learning technique based on the margin-maximization principle [54]. SVM has been successfully applied to several applications including text categorization, bioinformatics and database marketing [58]. It has also been used recently in the active safety research, including lane departure warning systems [51] and driver distraction detection algorithms [59]. This work develops a novel architecture that combines SVM with a Bayesian filter (BF) that enables it to perform well on the driver behavior classification problem. The following sections introduce the architecture of the SVM-BF algorithm, and provide additional theoretical and practical details about each of its components.

SVM-BF Architecture

The architecture of the SVM-BF algorithm is shown in Fig. 2-3. At the beginning of each measurement cycle inside the T_{w} window, the SVM module (Section 2.3.2) extracts the relevant features from the sensor observations. It then outputs a single classification (violator vs. compliant) per cycle to the Bayesian filter (BF) component (Section 2.3.2). Then, at the end of the T_{w} window, i.e., at time t_{warn}, the BF uses the current and previous SVM outputs to estimate the probability that the driver is compliant. Using a threshold detector, the SVM-BF outputs a final classification at t_{warn} specifying whether the driver is estimated as violator or compliant. To speed up the convergence of the BF, a discount function is added to the SVM-BF designed

to de-emphasize earlier classifications in T_{w}, and therefore put more weight on the measurements of the vehicle that are closer to t_{warn}.

SVM Component

This section gives a brief introduction to SVMs and their implementation in the SVM-BF framework. The reader is encouraged to refer to Ref. [60] for a detailed description of SVM.

Given a set of binary labeled training data $\{\boldsymbol{x}_i, y_i\}$, where $i = 1, \ldots, N_{\mathrm{tr}}, y_i \in \{-1, 1\}, \boldsymbol{x}_i \in \Re^d$, N_{tr} is the number of training vectors, and d is the size of the input vector, a new test vector $\boldsymbol{z}$ is classified into one class ($y = 1$) or the other ($y = -1$), by evaluating the following decision function

$$D(\boldsymbol{z}) = \mathrm{sgn}\left[\sum_{i=1}^{N_{\mathrm{tr}}} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{z}) + B\right]. \tag{2.5}$$

$K(\boldsymbol{x}_i, \boldsymbol{x}_j)$, known as the kernel function, is the inner product between the mapped pairs of points in the feature space, and B is the bias term [60]. $\boldsymbol{\alpha}$ is the *argmax* of the following optimization problem

$$\max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}) = \sum_{i=1}^{N_{\mathrm{tr}}} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N_{\mathrm{tr}}} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{2.6}$$

subject to the constraints

$$\sum_{i=1}^{N_{\mathrm{tr}}} \alpha_i y_i = 0 \quad \alpha_i \geq 0, \tag{2.7}$$

Appropriate kernel selection and feature choice are essential to obtaining satisfactory results using SVM. Based on experimenting with different kernel functions and several combinations of features, the best results for this problem were obtained using the Gaussian radial basis function and combining the following three features: range to intersection, speed, and longitudinal acceleration. At each measurement cycle, the output of the SVM block is a classification $y = +1$ (compliant) or $y = -1$ (violator).

This output is then fed into the Bayesian Filtering component (Section 2.3.2) which uses additional logic before making a final classification.

Bayesian Filtering Component

The Bayesian filter (BF) component views the outputs of the SVM component as samples of a random variable $y \in \{\text{violator}, \text{compliant}\}$, that is controlled by a parameter θ such that

$$p(y = \text{compliant}|\theta) = \theta. \tag{2.8}$$

The parameter θ is unknown. It represents the probability that the driver belongs to the compliant class. The role of the BF component is to compute the expected value of θ given a sequence of previous outputs from the SVM component.

To infer the value of the hidden variable, a standard Bayesian formulation is used [61]. We choose a *beta* distribution prior for θ, which is a function of some hyperparameters a and b,

$$beta(\theta|a, b) = \frac{\Gamma(a+b)}{\Gamma(a)+\Gamma(b)}\theta^{a-1}(1-\theta)^{b-1}, \tag{2.9}$$

where $\Gamma(x)$ is the gamma function [61]. The values of a and b have an intuitive interpretation; they represent the initial "confidence" given for each class, respectively. In other words, they reflect the number of observations corresponding for each behavior, which were accumulated in previous measurement cycles.

Given a sequence of SVM outputs $\boldsymbol{y} = [y_1, \ldots, y_N]$, the posterior distribution of θ, $p(\theta|\boldsymbol{y})$, is computed by multiplying the *beta* distribution prior by the binomial likelihood function given by

$$bin(m|N, \theta) = \binom{N}{m}\theta^m(1-\theta)^{N-m} \tag{2.10}$$

where m and l represent the number of SVM outputs corresponding to $y = \text{compliant}$ and $y = \text{violator}$, respectively. Variable N is the total number of SVM classifications,

$N = m + l$. By normalizing the resulting function, we obtain

$$p(\theta|\boldsymbol{y}) = \frac{\Gamma(m+a+l+b)}{\Gamma(m+a)+\Gamma(l+b)}\theta^{m+a-1}(1-\theta)^{l+b-1} \tag{2.11}$$

The expected value of θ given the sequence $\boldsymbol{y}$, which is the output of the Bayesian filter component, can then be simply expressed as

$$E(\theta|\boldsymbol{y}) = \int_0^1 \theta p(\theta|\boldsymbol{y})d\theta = \frac{m+a}{m+a+l+b} \tag{2.12}$$

Discount Function

To improve the accuracy of the expected value computed in Equation 2.12, earlier classifications in the T_w window should be given less weight compared to later ones. The following discount function achieves the desired purpose:

$$d_k = C^{N-k}, \quad \text{with} \quad d_0 = C^N \tag{2.13}$$

where $k = 1 \ldots N$ is the index of the SVM output in the T_w window, N represents the index of the last output in T_w, i.e., at time t_warn, and C is a constant discount factor, $0 < C \leq 1$, used to discount exponentially the weight of the output at time k. Note that $C = 1$ is equivalent to no discounting. The value of C affects the performance of the SVM-BF significantly. Section 2.5.4 investigates different values for C in the search for the best combination of the SVM-BF parameters. The variables m and l also need to be indexed by k, where m_k and l_k are the binary outputs of SVM at step k, and $m_k + l_k = 1$. Given these changes, Equation 2.12 can be rewritten as

$$E(\theta|\boldsymbol{y}) = \frac{\sum_{k=1}^{N} d_k m_k + d_0 a}{\sum_{k=1}^{N} d_k m_k + d_0 a + \sum_{k=1}^{N} d_k l_k + d_0 b} \tag{2.14}$$

where a and b are the same hyperparameters defined in Equation 2.9.

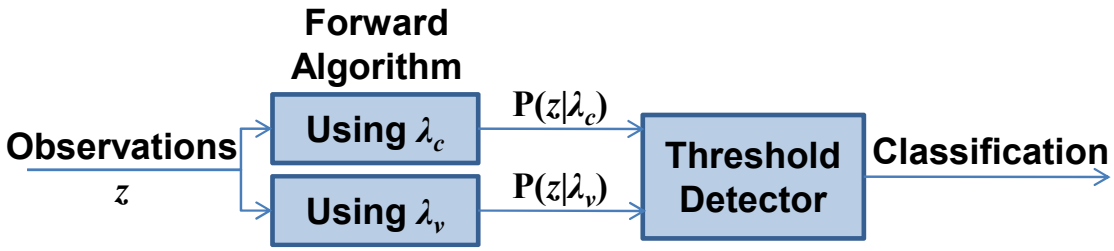


Figure 2-4: The HMM-Based Classification Architecture.

Threshold Detector

Given $E(\theta|\boldsymbol{y})$, the SVM-BF algorithm outputs the final classification based on the threshold detector specified value τ_S. The driver is classified as compliant if $E(\theta|\boldsymbol{y}) > \tau_S$, otherwise it is classified as violating. A large threshold value τ_S is equivalent to a more conservative algorithm (catching more violators) but at the expense of an increased number of wrong warnings (i.e., false positives). The choice of the value of τ_S is analyzed in Section 2.5.4.

Sliding Window

An extension to the original SVM-BF algorithm is the introduction of a sliding window over the features, which proved to be essential in improving the performance of the SVM-BF on the road traffic data. To elaborate, each feature consists of the means and variances of the last K different measurements. This change replaces the individual measurements (range, velocity, and acceleration) with their means and variances computed over the window. This addition indirectly adds time dependency to the sequence of outputs of the SVM component without affecting computation times, thus improving the SVM-BF model. The choice of the value of K is analyzed in Section 2.5.4.

2.3.3 HMM-based

An alternative approach is based on the idea of learning generative models from a set of observations. Hidden Markov Models (HMMs) have been used extensively to de-

velop such models in many fields, including speech recognition [55], and part-of-speech tagging [62]. The application of HMMs to isolated word detection is particularly relevant to the task of driver classification. In isolated word detection, one HMM is generated for each word in the vocabulary and new words are tested against these models to identify the maximum likelihood model for each test word [55]. HMMs have also been used to recognize different driver behavior, such as turning and braking [39]. This motivates the use of HMMs in this work to detect patterns that characterize compliant and violating behaviors.

HMM-based Architecture

Suppose two sets of observations are available: one known to be from compliant drivers and the other from violators. Each set of observations can be considered an emission sequence produced by an HMM modeling vehicle behavior. Using the EM algorithm (Section 2.3.3), two models, λ_c and λ_v, are learned from the compliant driver and violator training data, respectively. Then, given a new sequence of observations, $\boldsymbol{z}$, the forward algorithm (Section 2.3.3) is used with λ_c and λ_v to estimate the probability that the driver is compliant. As in the SVM-BF algorithm, a threshold detector (Section 2.3.3) uses this result to output a final classification, labeling the driver as either violating or compliant. Again, this classification occurs at t_{warn} based on the observations from the T_{w} window. Fig. 2-4 summarizes this architecture.

HMMs and the Forward Algorithm

This section provides a brief introduction to HMMs and the forward algorithm as a technique for determining how well a model fits a set of observations. Please refer to Ref. [55] for additional details on HMMs and the forward algorithm.

An HMM $\lambda(\boldsymbol{T}, \boldsymbol{t}, \boldsymbol{e})$ consists of a set of n discrete states and a set of observations at each state, as shown in Fig. 2-5. At any given time k, the system being modeled will be in one of these states, $q_k = s_i$, and the transition probability matrix $\boldsymbol{T}$ gives the probability of transitioning to any other state at the next timestep, $q_{k+1} = s_j$.

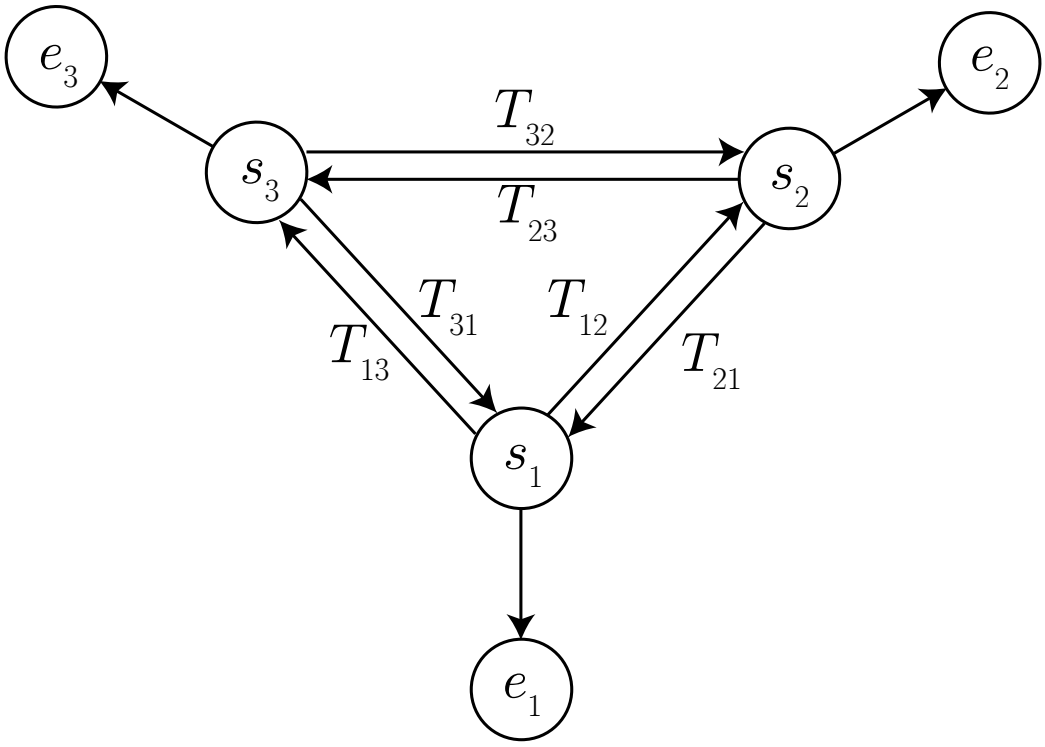


Figure 2-5: Basic HMM $\lambda(\boldsymbol{T}, \boldsymbol{t}, \boldsymbol{e})$ with $n = 3$ states, transition probabilities T_{ij}, and emissions e_i.

Specifically,

$$T_{ij} = P(q_{k+l} = s_j \mid q_k = s_i). \tag{2.15}$$

The probability of the system starting in each state is given by the initial state distribution $\boldsymbol{t}$, where $t_i = P(q_1 = s_i)$. Due to these probabilistic transitions, the current state is typically not known. Instead, a set of observations are assumed to be available. The probability of a state s_i emitting a certain observation z_k is given by $e_i(z_k)$. The emission distribution for each type of observation is assumed to be Gaussian with unique mean (μ_i) and variance (σ_i^2) for every state s_i. This design decision ensures that each state corresponds to one specific mode of driving, which is characterized by a set of observations normally distributed around some typical values (specified by the means and variances).

A common task with HMMs is determining how well a given model $\lambda(\boldsymbol{T}, \boldsymbol{t}, \boldsymbol{e})$ fits a sequence of observations $\boldsymbol{x} = x_1, \ldots, x_K$. This can be quantified as the probability of observing $\boldsymbol{x}$ given λ, $P(\boldsymbol{x} \mid \lambda)$. The forward algorithm is an efficient method for

computing this probability [55], and is defined as follows. Let $\alpha_i(k)$ be given by

$$\alpha_i(k) = P(x_1, \ldots, x_k, q_k = s_i \mid \lambda), \tag{2.16}$$

which is the probability of observing the partial sequence $x_1, \ldots, x_k$, and having the current state q_k at time k equal to s_i, given the model λ. Then the forward algorithm is initialized using the initial state distribution $\boldsymbol{t}$

$$\alpha_i(1) = t_i e_i(x_1), \quad i = 1, \ldots, n. \tag{2.17}$$

The probability of each subsequent partial sequence of observations for $k = 1, \ldots, K-1$ is given by

$$\alpha_j(k+1) = \left[\sum_{i=1}^{n} \alpha_i(k) T_{ij}\right] e_j(x_{k+1}), \quad i = 1, \ldots, n. \tag{2.18}$$

Upon termination at $k = K$, the algorithm returns the desired probability,

$$P(\boldsymbol{x} \mid \lambda) = \sum_{i=1}^{n} \alpha_i(K). \tag{2.19}$$

EM Algorithm for HMMs

These observations can also be used to learn an HMM that captures the behavior of the underlying system. A standard technique for doing so, the Expectation-Maximization (EM) algorithm, is summarized below. The complete algorithm is detailed in Ref. [63].

Given a set of N observation sequences (training data), $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, the EM algorithm computes the maximum-likelihood estimates of the HMM parameters, i.e.,

$$\lambda^*(\boldsymbol{T}, \boldsymbol{t}, \boldsymbol{e}) = \arg\max_{\lambda} P(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N | \lambda(\boldsymbol{T}, \boldsymbol{t}, \boldsymbol{e})). \tag{2.20}$$

To do so, it uses the forward algorithm, as defined earlier, as well as the backward

algorithm [55], which is defined similarly to the forward algorithm. Let

$$\beta_i(k) = P(x_{k+1}, \ldots, x_K \mid q_k = s_i, \lambda), \tag{2.21}$$

be the probability of observing the rest of the partial sequence of observations at time k, for $k \leq K$. Then the backward algorithm follows as

$$\beta_i(K) = 1 \tag{2.22}$$

$$\beta_j(k) = \sum_{j=1}^{n} T_{ij} e_j(x_{k+1}) \beta_i(k+1). \tag{2.23}$$

Using the terms $\alpha_i(k)$ from the forward algorithm and $\beta_i(k)$ from the backward algorithm, the probability of being in state s_i at time k given the observations $\boldsymbol{x}$ is given by

$$\gamma_i(k) = P(q_k = s_i \mid \boldsymbol{x}, \lambda) = \frac{\alpha_i(k)\beta_i(k)}{\sum_{i=1}^{n} \alpha_i(k)\beta_i(k)}. \tag{2.24}$$

Then the probability of being in state s_i at time k and s_j at time $k+1$ is given by

$$\begin{aligned} \xi_{ij}(k) &= P(q_k = s_i, q_{k+1} = s_j \mid \boldsymbol{x}, \lambda) \\ &= \frac{\alpha_i(k) T_{ij} e_j(x_{k+1}) \beta_j(k+1)}{\sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i(k) T_{ij} e_j(x_{k+1}) \beta_j(k+1)}. \end{aligned} \tag{2.25}$$

From these terms, the parameters of an updated HMM $\overline{\lambda}$ are computed with the

following update equations

$$t_i = \gamma_i(1), \tag{2.26}$$

$$T_{ij} = \frac{\sum_{k=1}^{K-1} \xi_{ij}(k)}{\sum_{k=1}^{K-1} \gamma_i(k)}, \tag{2.27}$$

$$\mu_i = \frac{\sum_{k=1}^{K} \gamma_i(k) x_k}{\sum_{k=1}^{K} \gamma_i(k)}, \tag{2.28}$$

$$\sigma_i = \frac{\sum_{k=1}^{K} \gamma_i(k)(x_k - \mu_i)^2}{\sum_{k=1}^{K} \gamma_i(k)}. \tag{2.29}$$

These maximum-likelihood estimates reflect the relative frequencies of the state transitions and emissions in the training data.

Repeating this procedure with λ replaced by $\overline{\lambda}$ is guaranteed to converge to a local maximum [63], i.e., as the number of iterations increases, $P(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N|\overline{\lambda}) - P(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N|\lambda) \to 0$. The resulting λ is the maximum likelihood model $\lambda^*(\boldsymbol{T}, \boldsymbol{t}, \boldsymbol{e})$. Since the EM algorithm is only guaranteed to converge to a local maximum, several sets of random initializations can be tested to reduce the effects of local maxima on the final model parameters.

As with the choice of features in the SVM, the observations used for the HMM can have a dramatic impact on its performance. After testing several combinations of observations, the following five parameters were identified to give the best results in terms of high detection accuracy and low false positive rates: range to intersection, speed, longitudinal acceleration, TTI, and RDP. In addition, the observations can be normalized to remove any bias introduced by differences in the order of magnitude of the observations.

It is interesting to note that TTI and RDP were not included in the SVM feature set since they did not a bring noticeable advantage for the SVM classification results. One explanation is the RBF kernel used in the SVM maps the feature space to an infinite dimensional space and is therefore flexible enough to learn the variations in its features (range, speed, and acceleration) without the need for additional derived ones. On the other hand, the HMM algorithm learns the transition matrix directly

from the features, and therefore could benefit from adding extra ones.

Threshold Detector

Using the EM algorithm, two models, λ_c and λ_v, are learned from the compliant driver and violator training data, respectively. Then, given a new sequence of observations, $\boldsymbol{z}$, the forward algorithm [55] is used with λ_c and λ_v to find the posterior probability of observing that sequence given each model, $P(\boldsymbol{z} \mid \lambda_c)$ and $P(\boldsymbol{z} \mid \lambda_v)$. The prior over the models is assumed to be uniform, $P(\lambda_c) = P(\lambda_v) = 0.5$, since nothing is known beforehand about whether the driver is compliant or violating. Then the likelihood ratio

$$\frac{P(\boldsymbol{z}, \lambda_c)}{P(\boldsymbol{z}, \lambda_v)} = \frac{P(\boldsymbol{z} \mid \lambda_c)}{P(\boldsymbol{z} \mid \lambda_v)} > e^{-\tau_{\mathrm{H}}} \tag{2.30}$$

determines whether the driver is more likely to be compliant or violate the stop bar and assigns the corresponding classification. Note that this ratio is typically computed using log probabilities, which introduces the e term in Equation 2.30. The threshold τ_{H} can be selected to adjust the conservatism of the classifier and is discussed in greater detail in Section 2.5.5.

Since states have one emission distribution per observation, each state in the HMM represents a coupling between specific ranges of values for each observation. It is this coupling and the transitions between different coupled ranges that allows the HMM-based classifier to distinguish between compliant drivers and violators.[1]

2.4 Data Collection and Filtering

The roadside data used in this chapter was collected as part of the Cooperative Intersection Collision Avoidance System for Violations (CICAS-V) project [37]. The

[1] In practice, the probability of the state self-transition in the HMM was seen to be much higher than the probability of transitioning to another state, which means that keeping additional state history should not have a noticeable impact on the model. It also reinforces the Markovian assumption.

Figure 2-6: Satellite image of the Peppers Ferry intersection (US 460 and Peppers Ferry Rd, Christiansburg, Montgomery, Virginia 24073) taken from Google Earth. CICAS-V data from vehicles at Peppers Ferry intersection were used to test the algorithms presented in this chapter.

CICAS-V project collected data on over 5,500,000 approaches across three intersections. In this work, the data from the Peppers Ferry intersection at U.S. 460 Business and Peppers Ferry Rd in Christiansburg, VA (See Fig. 2-6) was used to evaluate the algorithms, providing a total of 3,018,456 car approaches. The method of collection is detailed in Ref. [64]. A brief description is given below.

At the Peppers Ferry intersection, a custom data acquisition system was installed to monitor real-time vehicle approaches. This system included four radar units which identified vehicles, measured vehicle speed, range, and lateral position at a rate of 20 Hz beginning approximately 150 m away from the intersection, a GPS antenna to record the current time, four video cameras to record each of the four approaches, and a phase-sniffer to record the signal phase of the traffic light. These devices collected data on drivers who were unaware of the experiment as they moved through

the intersection. The information from these units then underwent post-processing including smoothing and filtering to remove noise such as erroneous radar returns. In addition, the Geometric Intersection Description (GID), a detailed plot of the intersection accurate to within 30 cm, was used to derive new values such as acceleration, lane id, and a unique identifier for each vehicle. This information on each of the car approaches was then uploaded onto a SQL database [65], which was used to obtain the data analyzed in this work.

The data was further processed at the Aerospace Controls laboratory at MIT for the purposes of this research. Microsoft SQL Server 2008 Developer Edition was used to filter individual trajectories from the data collected in the CICAS-V project. To maintain tractable offline runtimes for the learning phases of the algorithms, the first 300,000 trajectories out of the 3,018,456 car approaches were extracted. They were classified as compliant or violating based on whether they committed a traffic light violation. Violating behaviors included drivers that committed a traffic violation at the intersection, defined as crossing over the stop bar after the presentation of the red light and continuing into the intersection for at least 3 m within 500 ms. Compliant behaviors included vehicles that stopped before the crossbar at the yellow or red light. Out of the extracted trajectories, 1,673 violating and 13,724 compliant trajectories were found and then used in the classification algorithms.

2.5 Implementation

This section highlights the several decisions made in implementing the different algorithms introduced in Section 2.3. First, Section 2.5.1 describes the training and testing procedures used for data validation, and the rationale that motivates them. It also introduces an analysis tool that is frequently used to compare algorithm performance against parameter choice. Then, Section 2.5.2 discusses the parameters that are common to all the algorithms. More specifically, it explains the values of the variables affecting the warning timing and the maximum driver annoyance levels. Finally, Sections 2.5.3, 2.5.4, and 2.5.5 detail the choice of parameters that are specific

to the traditional, SVM-BF, and HMM algorithms, respectively.

2.5.1 Training/Testing Approaches

Using the trajectories selected from the CICAS-V database as described in Section 2.4, the algorithms are tested in pseudo-real-time, i.e., by running them on the trajectories of the database as if the observations of the target vehicle were arriving in real-time. The observations from each trajectory are downsampled from 20 Hz to 10 Hz to reduce the computational load. The training and testing are performed using two different approaches: 1) Basic generalization test (Section 2.5.1), and 2) m-Fold Cross-Validation (Section 2.5.1). Both approaches aim at evaluating the generalization property of the algorithms.

To evaluate the results of these tests, the receiver operation characteristic (ROC) curve is used to display the true positive and false positive rate of each set of algorithm parameters [48]. The curve is generated by varying a parameter of interest (or set of parameters), which is referred to as the Beta parameter in the SDT terminology [48]. Each point on the ROC curve then corresponds to a different value of the Beta parameter. The choice of Beta for each algorithm is detailed in its respective section below.

Basic Generalization Test

The first approach is a straightforward test of generalization. This consists of training the algorithms on a randomly selected subset that is some small fraction p of the data, and testing on the remaining $1 - p$. This approach demonstrates the generalization property (or lack thereof) of the algorithms. This property is essential for any warning algorithm to perform successfully when deployed on driver assistance systems, especially given the number of vehicles encountered in everyday driving. The value of p is chosen to be 0.2. The total number of trajectories used for this approach is 10,000 compliant and 1,000 violating. In other words, 2,000 compliant and 200 violating trajectories are used in the training phase, while the testing phase consists

Table 2.2: Cumulative Population Percentile vs. Driver Response Time [66]

Response Time (s)	Population Percentile
1.0	45%
1.6	80%
2.0	90%

Table 2.3: Minimum TTI ($TTI_{\min}$) and Minimum Distance ($d_{\min}$) Pairs

$TTI_{\min}$ (s)	$d_{\min}$ (m)
1.0	6.25
1.6	10.0
2.0	12.5

of 8,000 compliant and 800 violating trajectories.

m-Fold Cross-Validation

The second approach uses the standard m-fold cross-validation technique for testing generalization [53]. This involves randomly dividing the training set into m disjoints and equally sized parts. The classification algorithm is trained m times while leaving out, each time, a different set for validation. The mean over the m trials estimates the performance of the algorithm in terms of its ability to classify any given new trajectory. The advantage of m-fold cross-validation is that, by cycling through the m parts, all available training data can be used while retaining the ability to test on a disjoint set of test data. A total of 5,000 compliant and 1,000 violating trajectories are used in the m-fold approach with $m = 4$. First, each algorithm is run once on this data with the same ratio of training and testing data, producing a classifier with fixed parameters. This classifier is then tested using the m-fold cross-validation approach.

2.5.2 Shared Parameters

Minimum Time Threshold ($TTI_{\min}$)

For each trajectory, as shown in Fig. 2-2, the final output of the algorithms is given at time t_{warn}, which is computed as

$$t_{\text{warn}} = \min\left(TTI_{\min}, t(d_{\min})\right) \tag{2.31}$$

In other words, t_{warn} corresponds to the time when the estimated remaining time for the target vehicle to arrive to the intersection is $TTI_{\min}$ seconds, or when the distance to the intersection equals $d_{\min}$ meters, whichever happens first.

The choice of $TTI_{\min}$ is important. It represents the amount of time the host vehicle is given to react after being warned that a violating target vehicle is approaching its intersection. Choosing one single mean value for $TTI_{\min}$ provides little information about the performance of the warning algorithms for response times away from the mean. Instead, we base the choice of $TTI_{\min}$ on the cumulative human response time distribution presented in Ref. [66]. This distribution answers the following question: given a specific driver response time, what is the percentage of population that is able to react to a potential collision? The larger $TTI_{\min}$, the bigger the percentage of population to react on time to the warning. But a larger $TTI_{\min}$ is expected to lead to a worse performance of the warning algorithms because the final classification would be given earlier and after fewer measurements. To address this problem, the different algorithms were developed and evaluated for three different values of $TTI_{\min}$ summarized in Table 2.2. They are 1.0 s, 1.6 s, and 2.0 s, corresponding to 45%, 80%, and 90% of the population, respectively. Therefore, the engineer deciding which algorithm to implement has a clearer understanding of the tradeoffs for each choice. Note that the host vehicle is assumed to be at rest or moving with a negligible speed in this analysis. This is typically the case at t_{warn}, the time where it is warned of the target vehicle possible violation.

Minimum Distance Threshold ($d_{\min}$)

The $d_{\min}$ distance plays the role of a safety net. In most intersection approaches, the $TTI_{\min}$ condition happens first. But for some cases where the target vehicle approaches the intersection with a low speed, the $TTI_{\min}$ condition is met too close to the intersection. The $d_{\min}$ condition ensures that such cases are captured, and the warning (if needed) is given with enough time for the driver to react. For $TTI_{\min}$ of 1.6 s, $d_{\min}$ is chosen to be 10 m. This is equivalent to situations where vehicles cross the $d_{\min}$ mark with speeds lower than 6.25 m/s or 22.5 km/h, consistent with the low speed assumption. For $TTI_{\min}$ of 1.0 s and 2.0 s, $d_{\min}$ is scaled to 6.25 m and 12.5 m, respectively. These values are summarized in Table 2.3. Note that in the case of a warning, the driver will have a period of time larger than $TTI_{\min}$ to react, ensuring that the percentage of drivers responding on time to the warning is consistent with Table 2.2 numbers.

Maximum FP Rate

Warning algorithms must take into consideration driver tolerance levels, i.e., they should try to ensure that the rate of false alarms is below a certain "annoyance" level that is acceptable to most drivers. In this work, the maximum false positive rate is chosen to be 5% in accordance with automotive industry recommendations [46]. Therefore, the developed algorithms are designed and tuned under the constraint of keeping false positive rates below 5%, while trying to maximize true positive rates.

2.5.3 Traditional Algorithm Parameters

Static TTI

The static TTI algorithm has two key parameters: the safety threshold TTI_{req} and the deceleration threshold that indicates the onset of braking. The TTI_{req} parameter is a natural choice for controlling how conservative the classifier should be and thus is used as the Beta parameter for the ROC curve analysis. Values for TTI_{req} were selected between 1.0 and 10.0 seconds. The onset of braking is taken to be the time

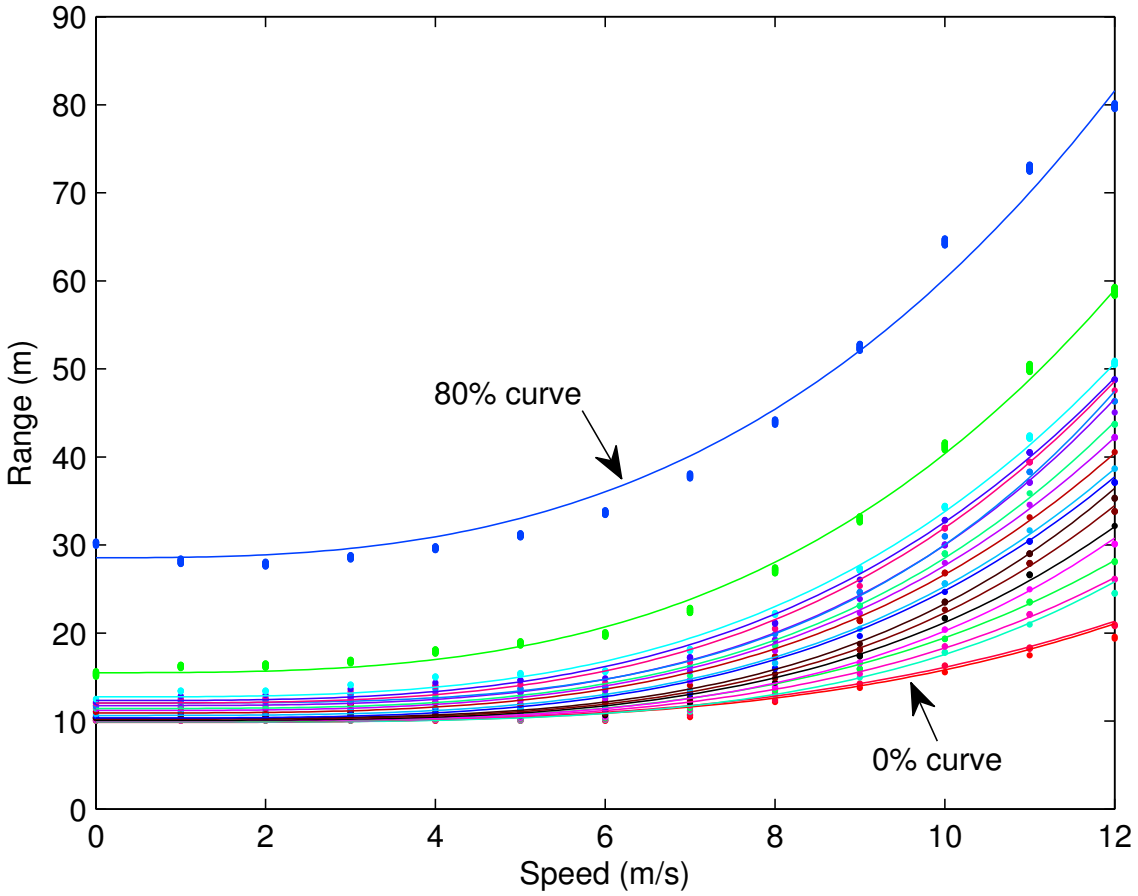


Figure 2-7: Regression curves used as decision thresholds for the SDR algorithm.

at which the vehicle's deceleration is less than $-0.075g$. Earlier work has shown that this is a realistic threshold for identifying brake activation [47]. In the event that a vehicle never crosses this threshold (i.e., does not brake), the classification is instead performed at t_{warn}.

Static RDP

The static *RDP* algorithm is also very straightforward. The only adjustable parameter is the warning threshold RDP_{warn}. So this is taken as the Beta for the ROC curve, with values ranging from $0.01g$ to $3.0g$.

Speed-Distance Regression

For the SDR classifier, several regression curves must first be fit to a set of observations from compliant drivers. Each curve models some percentage of compliant driver behaviors, and the choice of these percentages determines the classification thresholds. Fig. 2-7 shows one such set of curves. The bottom curve corresponds to 0%, that is, none of the measurements from the compliant driver training data were below this curve. The top curve corresponds to the 80^{th} percentile of compliant drivers. The other curves cover the range, with a higher density of curves at lower percentages to allow for more precise threshold selection. Each choice of curve for the classification threshold is treated as a Beta parameter for the ROC analysis.

2.5.4 SVM-BF Parameters

There are four key parameters for the SVM-BF classifier: the T_{w} window size, the discount factor C, the decision threshold τ_{S}, and the sliding window size K (See Section 2.3.2). The threshold variable is selected as the Beta parameter as it was introduced specifically to tune the performance of the algorithm. Models with T_{w} varying from 5 to 15 observations were considered, while C varied from 0.5 to 1.0 and K ranged from 3 to 10 measurements. All combinations of these parameters were tested, and Fig. 2-8 shows the ten combinations that produced the highest rates of true positives while maintaining a false positive rate below 5% for one basic generalization test. The results for this test (see Section 5.5.4) were obtained using the best combination of parameters in Fig. 2-8: $T_{\text{w}} = 15$, $K = 7$, $C = 0.9$, and $\tau_{\text{S}} = 0.9$. The hyperparameters a and b in Equation 2.9 are set both to 0.5, specifying no bias towards either behavior. These values could be changed to reflect a bias towards one driving behavior if the system is given prior knowledge of the target driving history.

2.5.5 HMM Parameters

There are three key parameters for the HMM-based classifier: the number of states in the HMM, the T_{w} window size, and the decision threshold τ_{H}. As in the previous

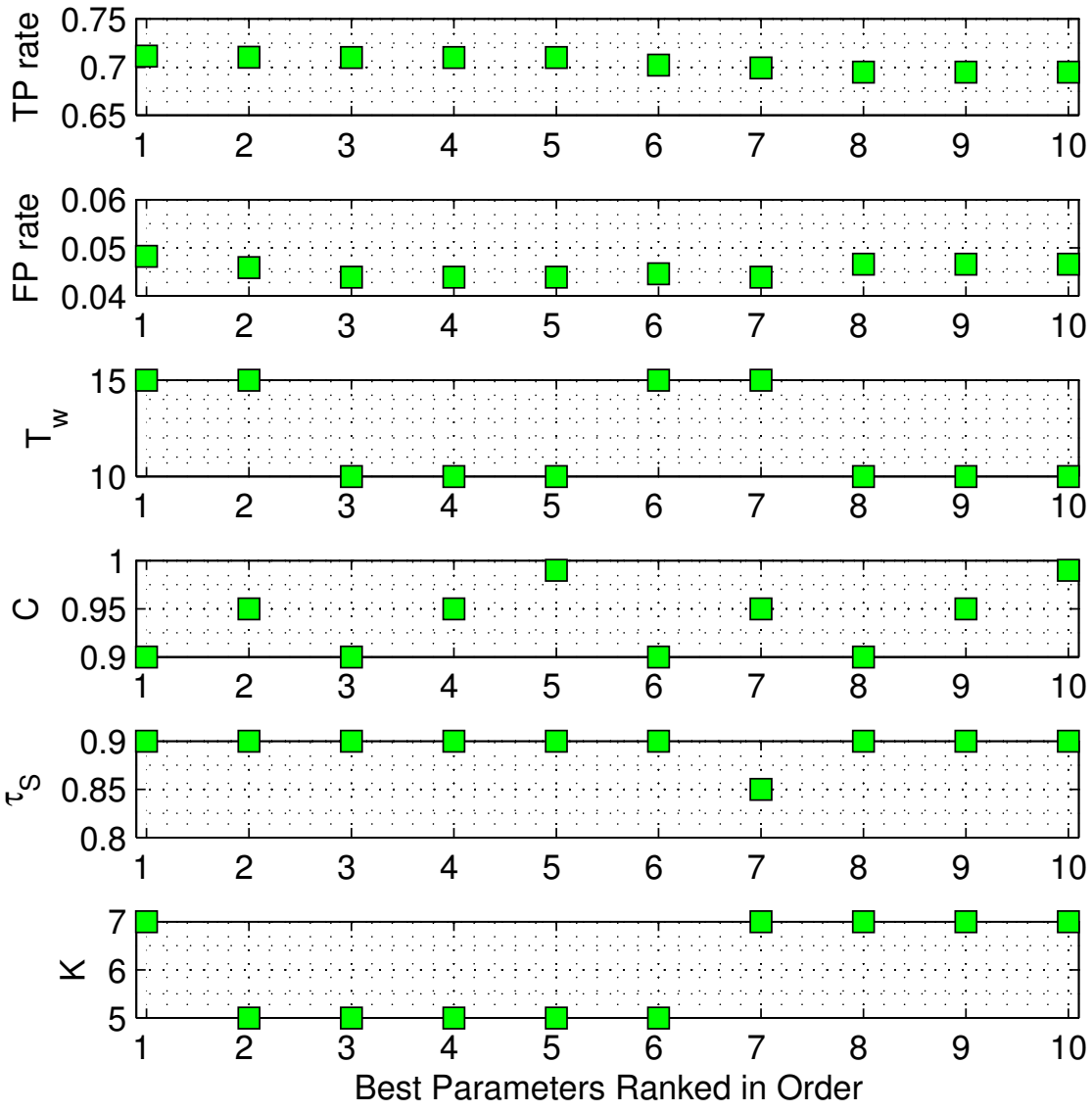


Figure 2-8: Ten best parameter combinations for the SVM-BF classifier with corresponding true positive rates for a basic generalization test with $TTI_{\min} = 1.6$ s and $d_{\min} = 10$ m.

methods, the threshold is selected as the Beta parameter. The number of states determines how many different modes the HMMs can capture, and as a result, the range of behaviors that can be classified accurately. However, increasing the number of states also increases the complexity of the model and the risk of overfitting the training data. Models with between 6 and 15 states were considered, while T_{w} was varied from 10 to 20 observations. All combinations of these parameters were tested, and Fig. 2-9

shows the ten combinations that produced the highest rates of true positives while maintaining a false positive rate below 5% for one basic generalization test. The results for this test (see Section 5.5.4) were obtained using the best combination of parameters in Fig. 2-9: $T_{\mathrm{w}} = 15$, 8 states, and $\tau_{\mathrm{H}} = 54.4$. Recall that τ_{H} defines a threshold on the likelihood ratio (Section 2.3.3) and is distinct from τ_{S}, which is a threshold on the probability of being classified as compliant (Section 2.3.2). Monte Carlo testing was used to learn multiple models for each set of parameters in order to reduce the effects of local minima on the algorithm.

2.6 Results

This section presents the results of the traditional and the developed SVM-BF and HMM-based algorithms described in Section 2.3. The algorithms were implemented in MATLAB using the LIBSVM toolbox [67] for SVM-BF and the PMTK toolkit [68] for the HMM-based classifier. Although these classifiers rely on more complex techniques than the traditional methods, most of the computational complexity is in the offline training phase. In fact, the time complexity of the testing phase of the SVM-BF algorithm is $\mathcal{O}(N_{\mathrm{tr}} T_{\mathrm{w}} d)$ [69], while that of the HMM algorithm is $\mathcal{O}(n^2 T_{\mathrm{w}} d)$ [70]. Note that n, the number of states in the HMM, is typically much smaller than N_{tr}, the number of training vectors in the SVM-BF. However, in practice, N_{tr} is replaced by the number of support vectors, which is significantly smaller, leading to comparable run times for both algorithms.

For online classification of a new trajectory, the computation time for the testing phase must be small. For the results presented below, the SVM-BF algorithm has an average run-time of 5 ms per trajectory evaluation, while the HMM-based classifier averages 2 ms per trajectory evaluation. These tests were run on a 2.5 GHz quad-core computer, but transitioning to an embedded implementation, rather than a MATLAB implementation, would greatly speed up the computation, allowing it to be run on less powerful hardware while retaining these small runtimes.

Figs. 2-10, 2-11, and 2-12 show the ROC curves for the five algorithms described

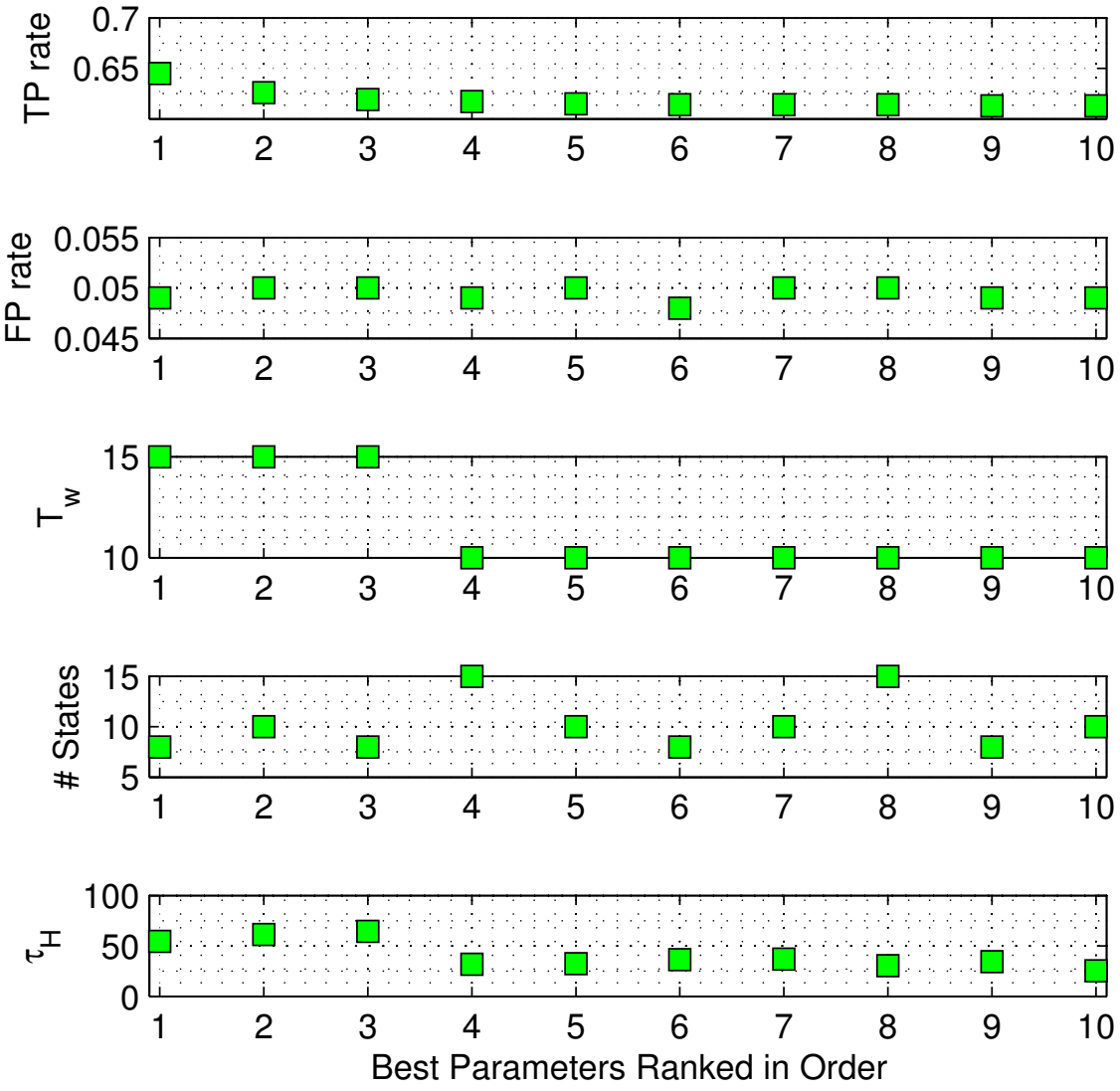


Figure 2-9: Ten best parameter combinations for the HMM-based classifier with corresponding true positive rates for a basic generalization test with $TTI_{\min} = 1.6$ s and $d_{\min} = 10$ m

above. The algorithms were tested with both the basic generalization test and the m-fold cross-validation test (e.g., compare Fig. 2-10(a) and Fig. 2-10(b)). Each test also considered three different decision thresholds: $TTI_{\min} = 1.0$ s, 1.6 s, and 2.0 s (e.g., compare Fig. 2-10(a) vs. Fig. 2-11(a) vs. Fig. 2-12(a)). These values of $TTI_{\min}$ are also coupled with $d_{\min} = 6.25$ m, 10.0 m, and 12.5 m (refer to Section 4.1.2). The parameters used in the SVM-BF and HMM-based algorithms for each test were chosen according to the procedure explained in Section 2.5.1. They are summarized

in Appendix A. The inset in each plot shows the region of interest around a false positive rate of 5%, while the precise true positive rates at this point are detailed in Table 2.4.

As these results illustrate, both the SVM-BF classifier and the HMM-based classifier out-perform the traditional methods by a significant margin in each set of tests. Although there is a difference between the true positive rates of the SVM-BF and HMM-based classifiers, this difference is consistent across tests, typically in the range of 5-7%. In contrast, the performance difference between these new approaches and the leading traditional one (RDP) increases with $TTI_{\min}$. For example, in the basic generalization tests, the SVM-BF algorithm has roughly 10% higher true positive rates than the static RDP algorithm for a $TTI_{\min}$ of 1.0 s. However, as this decision time is increased to 1.6 s and 2.0 s, the difference between SVM-BF and static RDP grows to nearly 18% and 21%, respectively (See Table 2.4). The m-fold tests exhibit a similar pattern. This reflects the fact that making a decision earlier, i.e., with a larger $TTI_{\min}$, is a more challenging problem. The ability to perform well with larger $TTI_{\min}$ is of practical importance in order to provide an earlier warning to the driver.

As Figs. 2-10, 2-11, and 2-12 illustrate, the SVM-BF algorithm produces the highest rate of true positives while keeping the false positive rate at 5%. This follows from the fact that the SVM algorithm uses convex optimization to perform accurate binary classification. With the radial basis kernel, it is able to capture the nuances characterizing each class of driver behavior. Note that ROC curves for SVM-BF are densely computed for false positive rates less than 5%, as this is the area of interest. The HMM-based classifier also yields a higher rate of true positives than the traditional methods because it is a rich model that couples observations into modes (states) that characterize driver behavior. The state transitions also capture the time dependencies that are inherent in the evolution of driver behavior while approaching an intersection. However, the HMM-based classifier does not perform as well as the SVM-BF classifier as it is trying to fit general models to two sets of behaviors that may include a few drastic outliers. In contrast, the SVM is only trying to find the separating boundary or hyperplane between these two models.

There are also several interesting trends in the traditional methods. The *RDP*-based classifier appears to be much more accurate than the other two methods across all the test combinations. Since *RDP* can be thought of as an effort-based parameter, it provides richer information than the observations used in the other two traditional algorithms.

Although the *TTI*-based classifier may resemble how humans classify approaching vehicles, the results also show that it does not perform well. This is in part due to the fact that the decision is made at the onset of braking. In general, the onset of braking occurs well before $TTI_{\min}$, so the challenge of correctly classifying a vehicle is comparable to classifying a vehicle with a large $TTI_{\min}$. This is reflected in the nearly identical *TTI* curves in Figs. 2-10, 2-11, and 2-12.

The SDR classifier exhibits a unique behavior as $TTI_{\min}$ is varied. For low $TTI_{\min}$, it outperforms the *TTI* algorithm. However, as $TTI_{\min}$ increases, performance drops severely and is unable to reach a false positive rate of 5% for any choice of the threshold curve (e.g., compare Fig. 2-10(a) and Fig. 2-12(b)). Thus the SDR algorithm does not yield a true positive rate for $TTI_{\min}$ of 1.6 s and 2.0 s, as indicated in Table 2.4.

In much of the existing literature, classification is performed when the driver arrives at the intersection, i.e., with $TTI_{\min} = 0.0$ s. This is the case in Ref. [46], which shows very accurate classification using the SDR algorithm. Since this is a less challenging scenario, most other algorithms, including the SVM-BF and HMM-based classifiers, would also see a substantial performance improvement. However, since the objective is to provide an early warning, $TTI_{\min}$ must be selected to be larger than the typical driver response time, as described in Section 2.5.2.

The general trends observed with these traditional methods are also consistent with the behaviors observed in previous work [46]. This ensures the new algorithms developed in this work are evaluated accurately against the baseline performance set by these traditional methods.

The basic generalization and m-fold cross-validation tests show the consistent improvement the SMV-BF and HMM-based algorithms provide in classifying a new set of trajectories. This validates the two approaches and demonstrates the flexibility

Table 2.4: True positive rates for each algorithm for 5% false positives

Generalization Tests			
$TTI_{\min}$	**Algorithm**	**TP** (%) (Basic)	**TP** (%) (m-Fold)
1.0 s	SVM	83.4	85.4
	HMM	77.3	80.0
	RDP	73.1	71.1
	TTI	29.0	30.0
	SDR	64.8	61.0
1.6 s	SVM	71.1	76.5
	HMM	64.5	65.2
	RDP	53.5	52.3
	TTI	31.4	30.6
	SDR	-	-
2.0 s	SVM	65.2	66.1
	HMM	58.6	61.6
	RDP	44.6	44.9
	TTI	32.3	31.5
	SDR	-	-

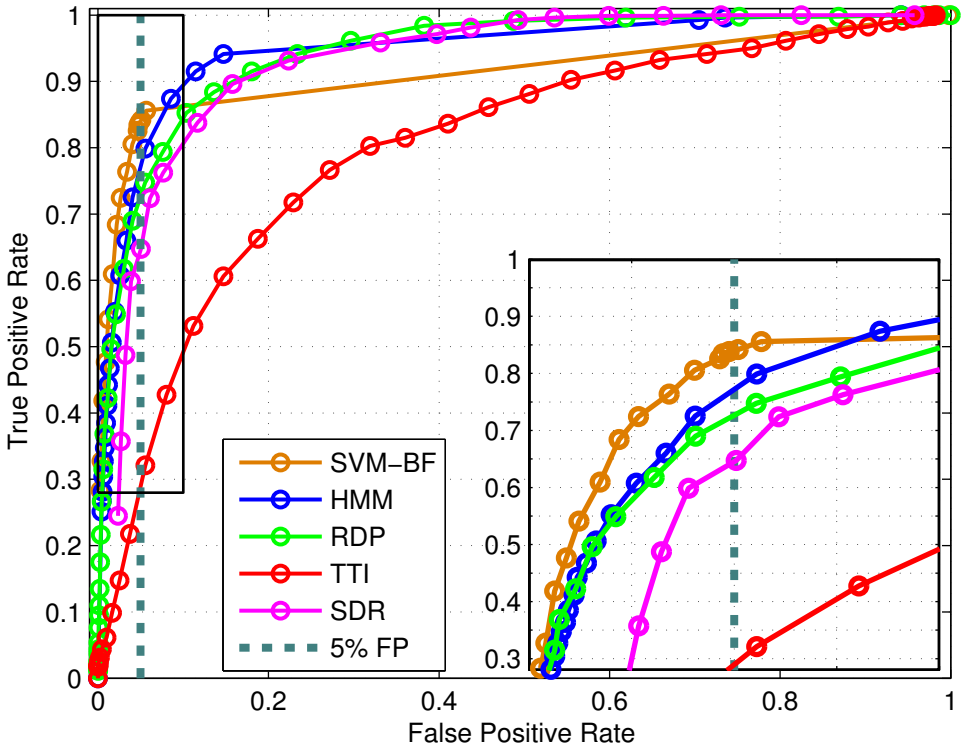
embedded in their design. Furthermore, the ROC analysis and testing of multiple $TTI_{\min}$ values gives the designer or engineer the flexibility to select the parameters that best match the requirements for any real-world implementation of these algorithms.

It is worth nothing that the bias introduced due to training and testing on the same intersection is limited due to 1) the large number of trajectories and equivalently drivers included, and 2) an expected similar trend of how drivers approach intersections independently of intersection geometry or location.

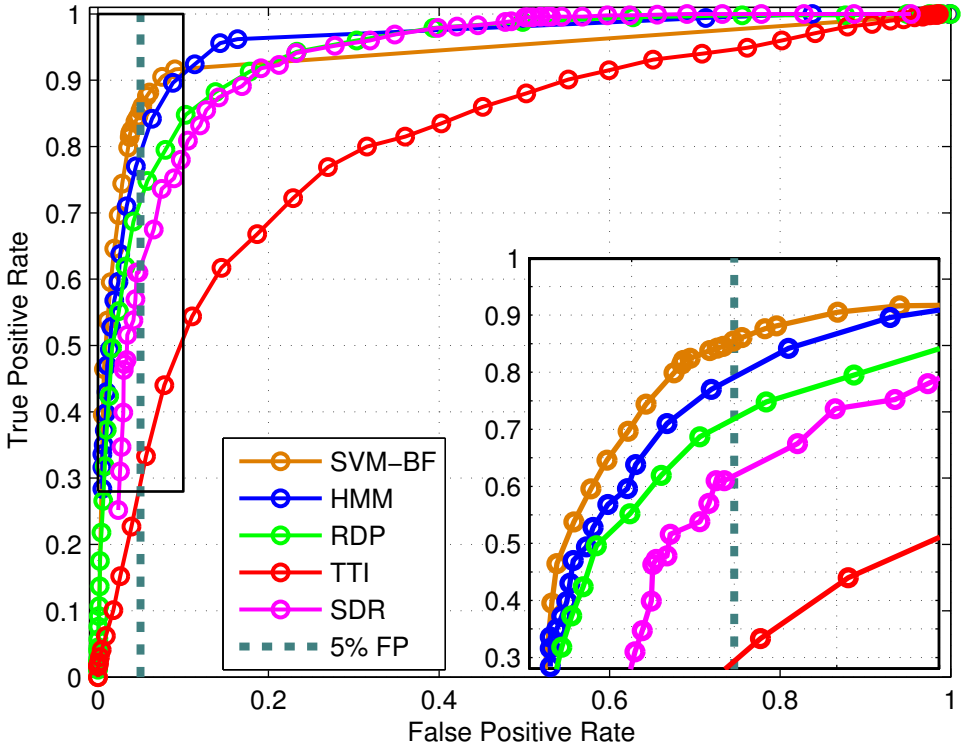
2.7 Conclusion

This chapter introduced two new approaches for classifying driver behaviors at road intersections. The first one, denoted SVM-BF, combines a Support Vector Machines classifier with a Bayesian filter, to discriminate between compliant drivers and violators based on vehicle speed, acceleration and distance to intersection. The second one, an HMM-based classifier, uses the EM algorithm to develop two distinct Hidden Markov Models for compliant and violating behaviors. To optimize safety while respecting driving acceptance levels, the algorithms were designed to maximize true positive rates while keeping false alarm rates below the 5% threshold. The two algorithms were successfully validated on more than 10,000 intersection approaches collected in Christiansburg, VA as part of the US Department of Transportation CICAS-V initiative. The performance of the algorithms were also compared to three popular traditional approaches consisting of a TTI-based, an RDP-based, and an SDR-based algorithm. The results of several generalization tests showed consistent and significant improvements with the developed algorithms, ranging from a minimum of 10% increase in true positive rates to more than 20% increase when issuing a warning 1 and 2 seconds in advance, respectively.

For real application, the choice would be the SVM-BF algorithm because it was the top performer algorithm for all the performed tests. In practice, the algorithm could be implemented in the vehicle as part of a driver assistance system, or/and on future intelligent intersection infrastructure. A driver assistance system would have different set parameters for the SVM-BF algorithm each associated with a different number of intersection classes (e.g., urban vs. rural). When the vehicle approaches an intersection, the system will find the most likely intersection class and then perform the classification. If the SVM-BF is is implemented on the intersection infrastructure, the parameters will be learnt from previously observed intersection approaches, and could also include different subsets corresponding to more specific characteristics (e.g., day vs. night).

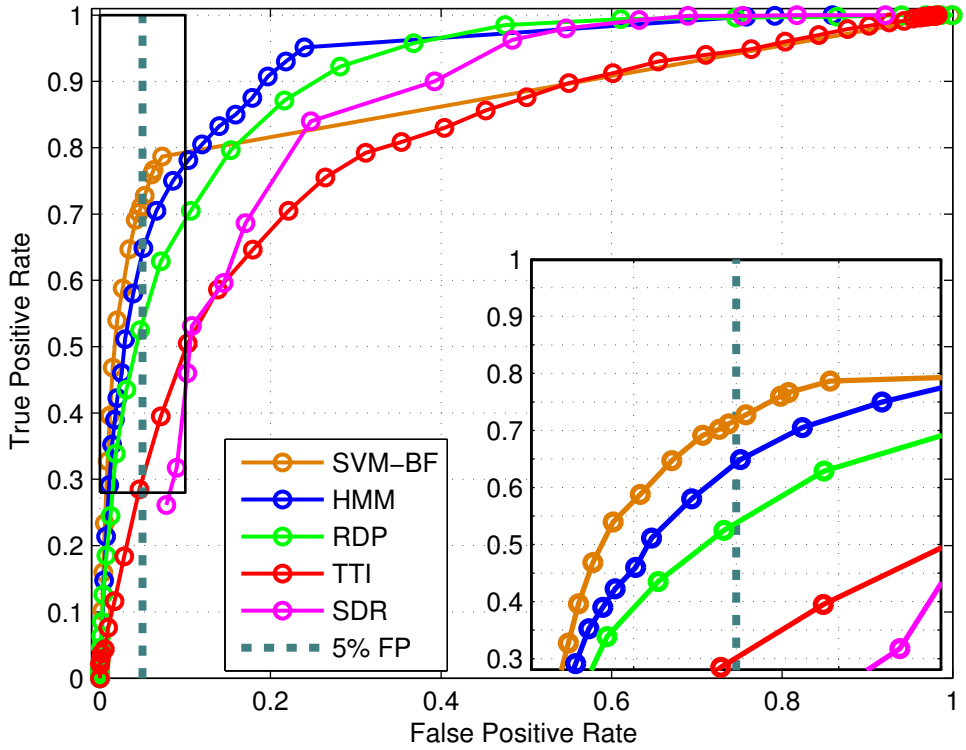


(a) ROC curves for 10000/1000 at $TTI_{\min} = 1.0\ s$

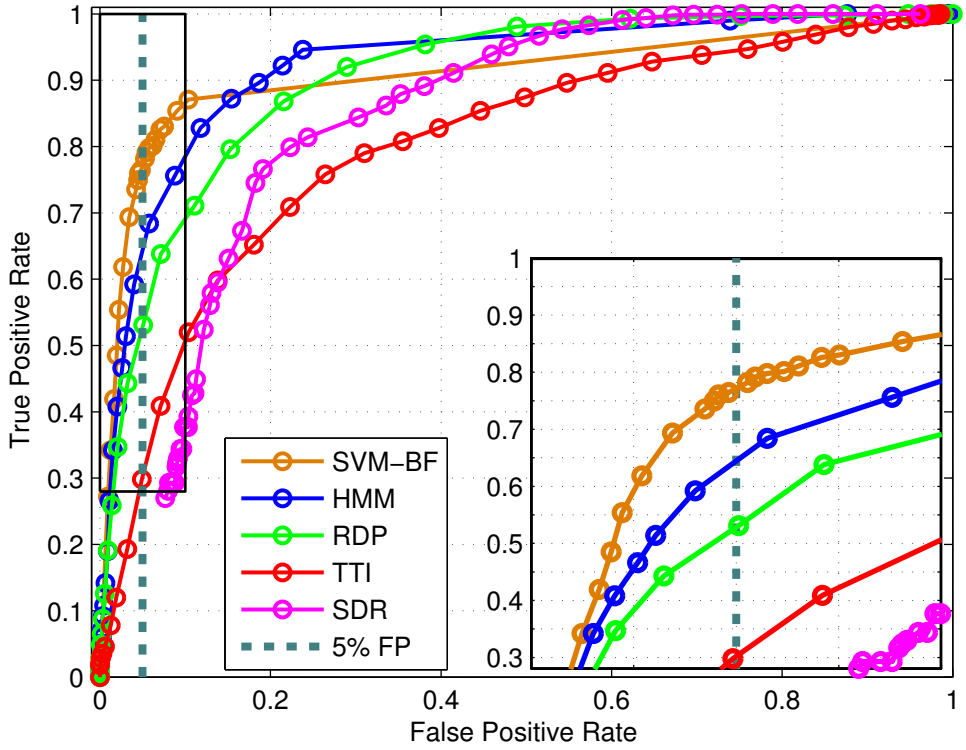


(b) ROC curves for 5000/1000 m-fold at $TTI_{\min} = 1.0\ s$

Figure 2-10: ROC curves for all five algorithms at $TTI_{\min} = 1.0\ s$ with insets showing area of interest around 5% false positives

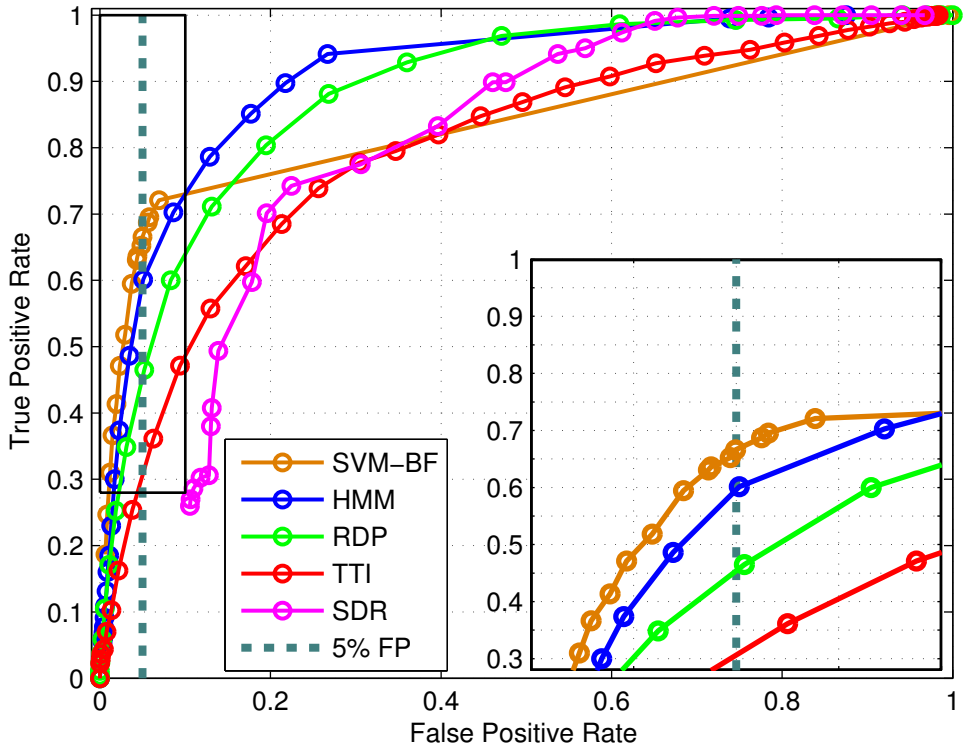


(a) ROC curves for 10000/1000 at $TTI_{\min} = 1.6\ s$

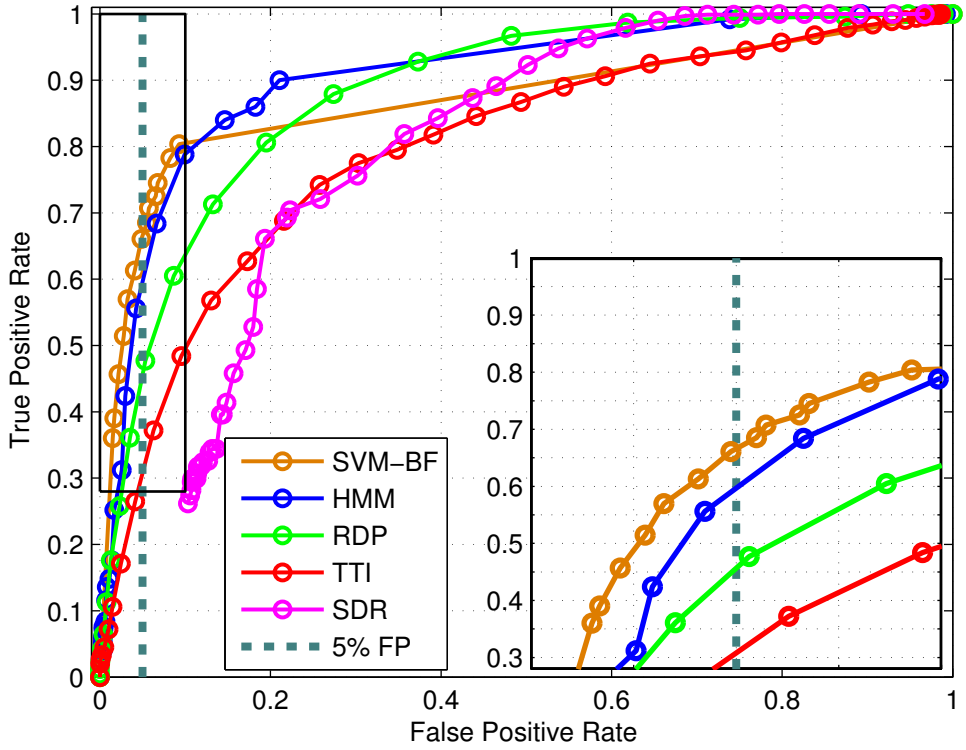


(b) ROC curves for 5000/1000 m-fold at $TTI_{\min} = 1.6\ s$

Figure 2-11: ROC curves for all five algorithms at $TTI_{\min} = 1.6\ s$ with insets showing area of interest around 5% false positives



(a) ROC curves for 10000/1000 at $TTI_{\min} = 2.0\ s$



(b) ROC curves for 5000/1000 m-fold at $TTI_{\min} = 2.0\ s$

Figure 2-12: ROC curves for all five algorithms at $TTI_{\min} = 2.0\ s$ with insets showing area of interest around 5% false positives

Chapter 3

Trajectory Prediction

This chapter presents an efficient trajectory prediction algorithm [29, 33] that has been developed to improve the performance of future collision avoidance and detection systems. The main idea is to embed the inferred intention information of surrounding agents into their estimated reachability sets to obtain a probabilistic description of their future paths. More specifically, the proposed approach combines the recently developed RRT-Reach algorithm [30] and mixtures of Gaussian Processes. The resulting approach, denoted as RR-GP, has RRT-Reach's benefits of computing trajectories that are dynamically feasible by construction, and the flexibility of GP's nonparametric Bayesian model, therefore efficiently approximating the reachability sets of surrounding vehicles. A demonstrative simulation on a car-like vehicle illustrates the advantages of the RR-GP approach, and highlights its performance by comparing it to two other GP-based algorithms. Another contribution of this chapter is the validation of the RR-GP approach on real-traffic data from the CICAS-V project.

3.1 Introduction

Future collision avoidance (CA) and conflict detection (CD) systems must handle increasingly complex scenarios including both unmanned and manned vehicles interacting in uncertain and possibly hostile environments. A key challenge is inferring

threats posed by the surrounding agents through prediction of their future motion. For example, future advanced driver assistance systems (ADAS) could include new risk assessment algorithms for intersections, one of the most dangerous road scenarios due to the difficulty in predicting incoming drivers' trajectories [25]. Another example is the Next Generation Air Transportation System (NextGen), which has the goal of increasing the safety and capacity of air transport operations but will require the development of new conflict detection algorithms that can tolerate tighter separation distances and an increasing number of aircraft flying in the National Air Space [26].

A key component for such threat assessment systems is a trajectory prediction algorithm (TPA) that estimates the future states of the *target vehicles*; these vehicles move in the vicinity of the *host vehicle* that has the threat assessment system onboard. The output of the TPA along with the knowledge of the current path of the host vehicle are fed into a threat assessment algorithm, which computes an estimate of the threat incurred by the host vehicle if it follows its intended path. A collision warning system uses the output of the threat assessment algorithm to warn the host vehicle about the risk of its path, while a collision mitigation system takes over the control of the vehicle and apply an escape maneuver that reduces the threat of collision.

A major challenge in trajectory prediction algorithms is caused by the different sources of uncertainties in dynamic environments. They can be classified into two different categories of uncertainties: 1) environment sensing (ES), and 2) environment predictability (EP) [22]. While the ES type deals with the uncertainty due to the imperfection of sensor measurements or incomplete knowledge of the environment configuration, the EP type is concerned with the uncertainty in the future state of the environment. The focus of this work is on the EP uncertainty. Even with the assumption of the existence of perfect sensors and complete knowledge of the current state environment, predicting long-term trajectories of mobile agents remains very difficult. For instance, when approaching a road intersection, an agent could follow different sets of paths corresponding to different underlying intents (e.g., right turns versus straight paths), in addition to traversing its paths with varying speeds and headings. Modelling these variations is impossible when only relying on perfect state

information.

Modelling the evolution of target vehicles can be classified into three main categories: 1) worst-case, 2) pattern-based, and 3) dynamic-based approaches. In the worst-case approach, the target vehicle is assumed to be actively trying to collide with the host vehicle [35, 71, 72]. The predicted trajectory of the target vehicle is the solution of a differential game where the target vehicle is modelled as a pursuer and the host vehicle as an evader. This solution is conservative so it is typically limited to short time-horizons in collision warning/mitigation problems to keep the level of false positives below a reasonable threshold [10]. In the pattern based approach, the vehicles are assumed to move according to typical patterns across the environment. These patterns are learned by observing the targets in the environment. There are two main techniques that fall under this category: a) discrete state-space, and b) clustering based [21]. In the discrete state-space technique, the motion model is typically based on Markov chains: the object state evolves from one state to another according to a learned transition probability [73]. In the clustering based technique, previously observed trajectories are grouped into different clusters. Each cluster is then represented by one trajectory prototype [74]. Given a partial path, prediction is then performed by finding the most likely cluster, or computing a probability distribution over the the different clusters. Both pattern based techniques have been popular in solving long-term prediction problems for mobile agents [21]. Finally, the dynamic based approach predicts the motion of the vehicle by typically estimating the current state of the vehicle and propagating it in the future assuming a fixed mode of operation. This prediction typically uses a continuous Bayes filters such as the Kalman Filter or its variations [75]. A popular extension in the target tracking literature is the Interacting Multiple Model Kalman Filter (IMM-KF). It uses available observations to update a bank of Kalman Filters and then find the filter closest to the current mode of operation of the vehicle [76]. Since they are only based on the dynamic model of the vehicle and its current state, dynamic based approaches are mainly suitable for short-term motion prediction. They typically perform poorly in long term prediction of trajectories due to their inability to model future changes in

control inputs or take into consideration other external factors such as obstacles.

Since the focus of this work is on long-term prediction of typical trajectories of vehicles navigating in cluttered environments, a clustering-based approach is followed. Although learning a dynamic obstacle motion pattern model from training data reduces the need for expert knowledge, there still exists a need to choose a class of models for the motion patterns. If the choice is a model class that is too simple, such as linear trajectories, the variety of motion patterns encountered would not be captured. On the other hand, if the pick is an extremely sophisticated model class with many parameters, large amounts of data would be needed, which can be costly or impossible to collect in real world domains [77]. Existing work showed that a specific clustering-based technique which is a Bayesian nonparametric approach to modeling motion patterns is well-suited to modeling dynamic obstacles with unknown motion patterns [77, 78]. This nonparametric model, a mixture of Gaussian process (GP), generalizes well from small amounts of data and allows the model to capture complex trajectories as more data is observed. This work augments the GP predictions with a new approach based on closed-loop rapidly-exploring random trees (CL-RRT) [31] to generate dynamically feasible and collision-free trajectories that improve the accuracy of trajectory prediction results.

Section 3.2 first describes the problem statement. Then, Section 3.3 reviews of the motion model [77], which is used in Section 3.4 with the CL-RRT algorithm to as the main components of the RR-GP algorithm. Finally, results showing the performance of RR-GP are presented in Section 3.5 in a simulated environment with a human-driven target vehicle.

3.2 Problem Statement

The focus of this work is on the problem of predicting the position of a target vehicle moving according to a set of motion patterns in a complex environment cluttered with obstacles. The goal is to develop an efficient mechanism for prediction that is suitable for real-time applications while achieving high long-term prediction accuracy. There

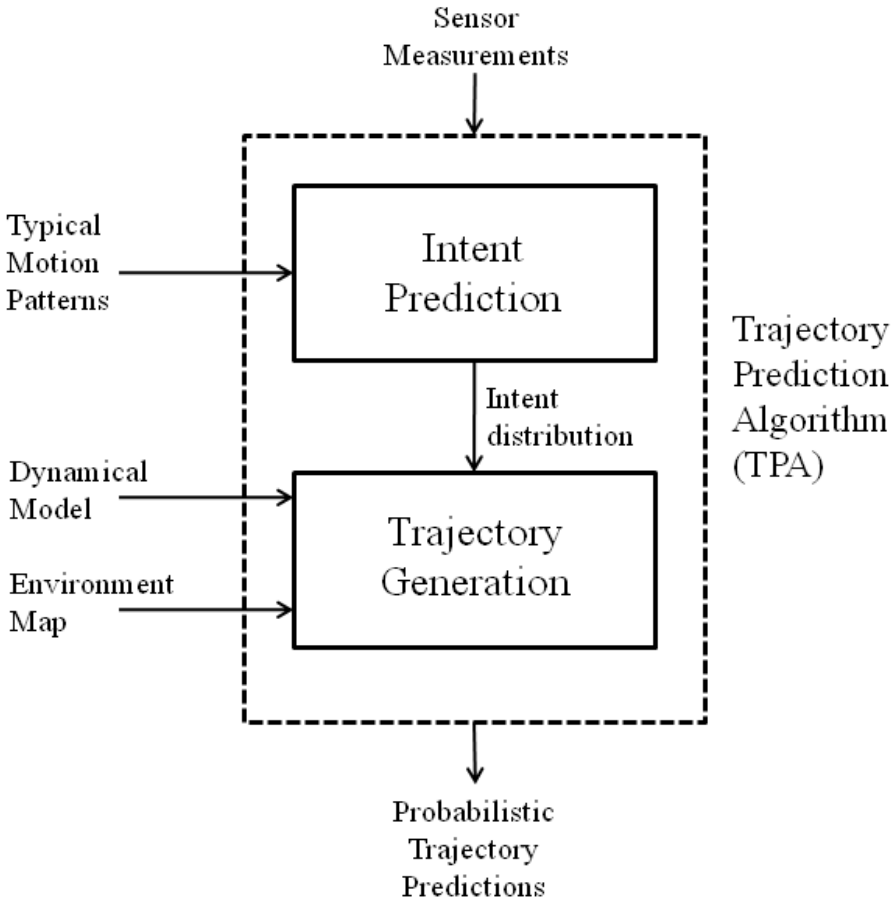


Figure 3-1: Architecture of the Trajectory Prediction Algorithm (TPA).

are two main potential applications for the developed algorithm. The first is improved navigation for autonomous vehicles where the prediction algorithm would extend the collision avoidance capabilities of a probabilistic path planner. The second potential application is better threat assessment systems for driver collision warning detection. In fact, a main motivation for this work is the road intersection threat assessment problem [30]; the trajectory prediction algorithm would improve the estimation of threat of dangerous vehicles approaching an intersection.

Figure 3-1 shows the high level architecture of the proposed solution. The input of the Trajectory Prediction Algorithm (TPA) is sensor measurements of the target vehicle, and its output is a distribution over future trajectories of the target vehicle. The first component of the TPA is the Intention Predictor that uses the current and previous sensor measurements, along with a set of typical motion patterns, to pro-

duce an intent distribution which is given to the trajectory generation component. Using the intent information, a dynamic model of the target vehicle, and a map of the environment, the trajectory generator outputs a set of predicted trajectories with different probabilities of occurrence. To limit the scope of the problem to uncertainty in environment predictability (EP), the sensors measurements are assumed to be noise-free despite the motion pattern representation being robust to noisy measurements. Additionally, the map of the environment, typically a road network, is assumed to be available a priori. The next section introduces the motion model used in the intent predictor.

3.3 Motion Model

The motion model formulation is based on earlier work by Joseph et al. [77–79]. It is reproduced in this section for the reader's convenience.

3.3.1 Motion Pattern

A *motion pattern* is defined as a mapping from positions to a distribution over velocities. Given an target vehicle current position (x_t, y_t) and a trajectory derivative $(\frac{\Delta x_t}{\Delta t}, \frac{\Delta y_t}{\Delta t})$, at time t, its predicted next position (x_{t+1}, y_{t+1}) at time $t+\Delta t$ is expressed as $(x_t + \frac{\Delta x_t}{\Delta t}\Delta t, y_t + \frac{\Delta y_t}{\Delta t}\Delta t)$. Thus, modeling trajectory derivatives is equivalent to modeling trajectories. But it has several advantages including 1) being unaffected to the lengths and discretizations of the trajectories, and 2) allowing clustering of trajectories sharing same characteristics represented by their flow fields, i.e., trajectory derivatives, as for example trajectories starting from different locations and ending in a single one.

3.3.2 Mixtures of Motion Patterns

A finite mixture model specifies a prior probability $p(b_1), p(b_2), \ldots, p(b_M)$ for each of its M motion patterns $b_1, b_2, \ldots, b_M$. Then, under the mixture model, the probability

of the i^{th} observed trajectory t^i is expressed as

$$p(t^i) = \sum_{j=1}^{M} p(b_j)p(t^i|b_j). \tag{3.1}$$

Note that throughout this chapter t without a superscript refers to the time, while t with a superscript refers to a trajectory, where the superscript is an index of the trajectory among all trajectories belonging to a specific motion pattern. Also, since we are interested in vehicles traveling along a known road network we can assume the number of motion patterns, M, is known *a priori*.

The *motion model* is defined as a mixture of weighted motion patterns [77, 78]. Each motion pattern is modeled as a pair of Gaussian pocesses that map (x, y) positions to distributions over velocities $\frac{\Delta x}{\Delta t}$ and $\frac{\Delta y}{\Delta t}$. The weight on each motion pattern is its probability of occurrence. This work assumes that the trajectory in the training data are labeled with their corresponding motion pattern. This is a reasonable assumption for several domains of interests (e.g., road intersections) where the road network is known and the different set of patterns is distinguishable during a preprocessing step. A clustering technique could also be used to compute the number of Gaussian Processes using Dirichlet Processes [77] or K-clustering along with the Expectation-Maximization (EM) algorithm [80].

The probability of trajectory t^i given motion pattern b_j, $p(t^i|b_j)$, is specified by the choice of motion model. It represents the distribution over trajectories for a specific mobility pattern.

A simple example of distribution over trajectories is that of a linear model with Gaussian noise, but this approach cannot capture the dynamics of the variety expected in this work. Another common approach are discrete Markov models, but these are not well suited to model mobile agents in the types of real-world domains of interest here [79, 80]. In particular, they have an inherent limitation related to the decision of the state discretization. To capture the variety of trajectories that might be encountered, an expressive representation (a fine discretization) is necessary, resulting in a model that requires a large amount of training data. In real-world domains where

collecting a large data set is costly, or impossible, these models are prone to over-fitting. A coarser discretization can be used to prevent over-fitting, but then the approach is unable to accurately capture the agent's dynamics. This works uses Gaussian processes (GP) as the model for motion patterns. Although GP's have a significant mathematical and computational cost, they provide a natural balancing between generalization in regions with sparse data and preventing under-fitting in regions of dense data [81]. Related work has also highlighted the advantages of using Gaussian Processes over traditional discretion-based techniques [80, 82, 83].

3.3.3 Gaussian Process Motion Patterns

An agent trajectory is observed as a set of discrete measurements along its continuous path through space. These measurements are interpolated using a Gaussian process (GP) that places a distribution over functions [81]. Gaussian process non-parametric models are extremely robust to unaligned, noisy measurements. Furthermore, they are well-suited for modeling the continuous paths underlying potentially non-uniform time-series samples of the agent's trajectories.

Two quantities must be computed inside the summation of Eq. (3.1) to determine the likelihood of each behavior, $p(b_j)$. The first term is the prior probability of motion pattern b_j. After observing an agent's trajectory t^i, the posterior probability of the motion pattern is

$$p(b_j|t^i) \propto p(t^i|b_j)p(b_j) \tag{3.2}$$

where $p(t^i|b_j)$ is the probability of trajectory t^i under motion pattern b_j. This distribution, $p(t^i|b_j)$, is computed by

$$p(t^i|b_j) = \prod_{t=0}^{L^i} p\left(\frac{\Delta x_t}{\Delta t}\middle| x^i_{0:t}, y^i_{0:t}, \{t^k : z_k = j\}, \theta^{GP}_{x,j}\right) \cdot p\left(\frac{\Delta y_t}{\Delta t}\middle| x^i_{0:t}, y^i_{0:t}, \{t^k : z_k = j\}, \theta^{GP}_{y,j}\right) \tag{3.3}$$

where L^i is the length of trajectory i, z_k indicates the motion pattern trajectory t^k is assigned to and $\theta^{GP}_{x,j}$ and $\theta^{GP}_{y,j}$ are the hyperparameters of the Gaussian process for motion pattern b_j.

A set of mean and covariance functions specifies each GP motion pattern. The mean functions are written as $E[\frac{\Delta x}{\Delta t}] = \mu_x(x, y)$ and $E[\frac{\Delta y}{\Delta t}] = \mu_y(x, y)$, and they are both implicitly initialized to zero everywhere (for all x and y) by the choice of parametrization of the covariance function. This encodes the prior bias that the target vehicle is expected to stay in the same place if no any additional knowledge is given. The covariance function of the x-direction is denoted by $K_x(x, y, x', y')$, which expresses the correlation between trajectory derivatives at two positions, (x, y) and (x', y'). Given locations $(x_1, y_1, .., x_k, y_k)$, the corresponding trajectory derivatives $(\frac{\Delta x_1}{\Delta t}, .., \frac{\Delta x_k}{\Delta t})$ are jointly distributed according to a Gaussian with mean $\{\mu_x(x_1, y_1), .., \mu_x(x_k, y_k)\}$ and covariance Σ, where the $\Sigma_{ij} = K_x(x_i, y_i, x_j, y_j)$. This work uses the standard squared exponential covariance function

$$K_x(x, y, x', y') = \sigma_x^2 \exp\left(-\frac{(x - x')^2}{2{w_x}^2} - \frac{(y - y')^2}{2{w_y}^2}\right) + \sigma_n^2 \delta(x, y, x', y') \tag{3.4}$$

where $\delta(x, y, x', y') = 1$ if $x = x'$ and $y = y'$ and zero otherwise. The exponential term above encodes that similar trajectories should make similar predictions and the length-scale parameters w_x and w_y normalize for the scale of the data. The σ_n-term represents within-point variation (e.g., due to measurement or process noise); the ratio of σ_n and σ_x weights the relative effects of noise and influences from nearby points. $\theta_{x,j}^{GP}$ refers to the set of hyperparameters σ_x, σ_n, w_x, and w_y associated with motion pattern b_j (each motion pattern has a separate set of hyperparameters).

For a GP over trajectory derivatives trained with tuples $(x_k, y_k, \frac{\Delta x_k}{\Delta t})$, the predictive distribution over the trajectory derivative $\frac{\Delta x}{\Delta t}^*$ for a new point (x^*, y^*) is given by

$$\begin{aligned} \mu_{\frac{\Delta x}{\Delta t}^*} &= K_x(x^*,y^*,X,Y) K_x(X,Y,X,Y)^{-1} \frac{\Delta X}{\Delta t} \\ \sigma^2_{\frac{\Delta x}{\Delta t}^*} &= K_x(x^*,y^*,X,Y) K_x(X,Y,X,Y)^{-1} K_x(X,Y,x^*,y^*) \end{aligned} \tag{3.5}$$

where the expression $K_x(X, Y, X, Y)$ is shorthand for the covariance matrix Σ with terms $\Sigma_{ij} = K_x(x_i, y_i, x_j, y_j)$, with $\{X, Y\}$ representing the previous trajectory points.

The equations for $\frac{\Delta y}{\Delta t}^*$ are equivalent to those above, using the covariance K_y.

3.3.4 Estimating Future Trajectories

As summarized in Section 3.3.3, the Gaussian process motion model places a Gaussian distribution over trajectory derivatives $(\frac{\Delta x}{\Delta t}, \frac{\Delta y}{\Delta t})$ for every position (x, y). While this provides a distribution over the agent's next location, for longer term predictions this distribution over next position must be used to produce a distribution over the future trajectories. Unfortunately, the distribution over future trajectories can be computed in closed form, so instead, the approach is to draw trajectory samples and use these sampled trajectories as a trajectory distribution.

To sample a trajectory from a current starting location (x_0, y_0), first a trajectory derivative $(\frac{\Delta x_0}{\Delta t}, \frac{\Delta y_0}{\Delta t})$ is sampled to calculate the agent's next location (x_1, y_1). Starting from (x_1, y_1), the trajectory derivative $(\frac{\Delta x_1}{\Delta t}, \frac{\Delta y_1}{\Delta t})$ is sampled to estimate its next location (x_2, y_2). The process is then repeated until the trajectory is of the desired length L. The entire sampling procedure is then repeated from the current location (x_0, y_0) multiple times to obtain a set of future trajectories the agent may take. Given a current location (x_t, y_t) and a given motion pattern b_j, sampling K time steps in the future is done using

$$
\begin{aligned}
&p(x_{t+K}, y_{t+K} | x_t, y_t, b_j) \\
&= \prod_{k=0}^{K-1} p(x_{t+k+1}, y_{t+k+1} | x_{t+k}, y_{t+k}, b_j) \\
&= \prod_{k=0}^{K-1} p\left(\frac{\Delta x_{t+k+1}}{\Delta t}, \frac{\Delta y_{t+k+1}}{\Delta t} \middle| x_{t+k}, y_{t+k}, b_j \right) \\
&= \prod_{k=0}^{K-1} p\left(\frac{\Delta x_{t+k+1}}{\Delta t} \middle| x_{t+k}, y_{t+k}, b_j \right) p\left(\frac{\Delta y_{t+k+1}}{\Delta t} \middle| x_{t+k}, y_{t+k}, b_j \right) \\
&= \prod_{k=0}^{K-1} \mathcal{N}\left(x_{t+k+1}; \mu_{j, \frac{\Delta x_{t+k+1}}{\Delta t}}, \sigma^2_{j, \frac{\Delta x_{t+k+1}}{\Delta t}} \right) \mathcal{N}\left(y_{t+k+1}; \mu_{j, \frac{\Delta y_{t+k+1}}{\Delta t}}, \sigma^2_{j, \frac{\Delta y_{t+k+1}}{\Delta t}} \right) \qquad (3.6)
\end{aligned}
$$

where the parameters from the Gaussian distribution are calculated using Eq. (3.5). When this process is done online, the trajectory's motion pattern b_j will not be known

directly. Given the past observed trajectory $(x_0, y_0), ..., (x_t, y_t)$, the distribution can be calculated K time steps in the future by combining Eq. (3.1) and Eq. (3.6). Formally,

$$
\begin{aligned}
& p(x_{t+K}, y_{t+K} | x_{0:t}, y_{0:t}) \\
&= \sum_{j=1}^{M} p(x_{t+K}, y_{t+K} | x_{0:t}, y_{0:t}, b_j) p(b_j | x_{0:t}, y_{0:t}) && (3.7) \\
&= \sum_{j=1}^{M} p(x_{t+K}, y_{t+K} | x_t, y_t, b_j) p(b_j | x_{0:t}, y_{0:t}) && (3.8)
\end{aligned}
$$

where $p(b_j | x_{0:t}, y_{0:t})$ is the probability of motion pattern b_j given the observed portion of the trajectory and $p(x_{t+K}, y_{t+K} | x_t, y_t, b_j)$ is given by Eq. (3.6). The progression from Eq. (3.7) to Eq. (3.8) is based on the assumption that, given b_j, the trajectory's history provides no additional information about the future location of the agent. Please refer to the authors' previous work [77, 79] for further discussion and analysis of the motion model implementation and performance.

3.4 RR-GP Trajectory Prediction Algorithm

Section 3.3 outlined the approach of using GP mixtures to model mobility patterns and its benefits over other models. However, in practice, GPs suffer from two interconnected shortcomings: their high computational cost and their inability to embed static feasibility or vehicle dynamical constraints. Since GPs are based on statistical learning, they are unable to model prior knowledge of road boundaries, static obstacle location, or dynamic feasibility constraints (e.g., minimum turning radius). Very dense training data may alleviate this feasibility problem by capturing, in great detail, the environment configuration and physical limitations of the vehicle. Unfortunately, the computation time for predicting future trajectories using the resulting GPs would suffer significantly, rendering the motion model unusable for real-time applications.

To handle both of these problems simultaneously, this section introduces a specific implementation of the TPA architecture (Section 3.2), denoted as RR-GP (See Figure 3-2). This novel approach augments RRT-Reach, a reachability based method

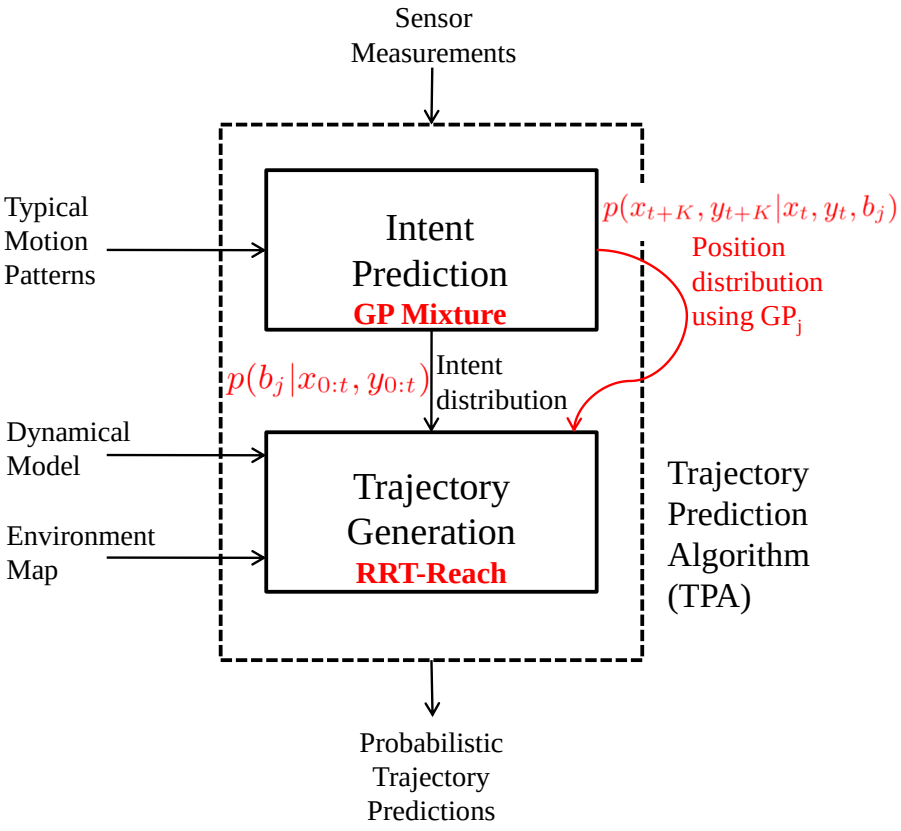


Figure 3-2: RR-GP High Level Architecture.

which creates dense, feasible trajectories from sparse samples, with the GP mixture model. RRT-Reach is an extension of the closed-loop rapidly-exploring random tree (CL-RRT) algorithm to compute reachable sets of moving objects in real-time [32]. RRT-Reach was also successfully used in a novel threat assessment module for driver assistance systems at intersections [30]. Refer to Chapter 4 for more details about the original RRT-Reach algorithm.

RR-GP is based on two main components: 1) a rapidly-exploring random tree (RRT), and 2) a GP-based mobility model. For each GP-based pattern b_j, $j \in \{1, \ldots, M\}$ as defined in Section 3.3, and the current position of the target vehicle, the RR-GP uses an RRT-based technique to grow a separate tree of trajectories that follows b_j while guaranteeing dynamical feasibility and collision avoidance. More specifically, it is based on the closed-loop RRT (CL-RRT) algorithm [31], which grows a tree by randomly sampling points in the environment and simulating dynamically

feasible trajectory towards them. CL-RRT was successfully used by the MIT team in the 2007 DARPA Grand Challenge [31]. CL-RRT samples inputs to a controller rather than the vehicle itself, allowing the generation of smoother trajectories more efficiently than traditional RRT algorithms. Unlike the original CL-RRT approach, the RR-GP tree is not used to create paths leading to a goal location, but instead each tree is grown toward regions corresponding to a learned mobility pattern b_j of the target vehicle (See Figure 3-3). RR-GP is also different from the RRT-Reach algorithm in that it does not approximate the complete reachability set of target vehicle. RRT-Reach is a more conservative approach that could be useful when no information regarding typical patterns is provided. Finally, note that throughout this chapter b_j is sometimes referred to as an intent, since each mobility pattern corresponds to a different intentional motion pattern.

3.4.1 Single Tree RR-GP Algorithm

In RR-GP, it is assumed that the target vehicle has car-like dynamics, more specifically a bicycle dynamical model [84]. Another assumption is that the parameters of the model are available but its inputs are unknown; they represent the commands that the target vehicle applies to follow trajectories generated by RR-GP. The target vehicle inputs are approximated by the outputs of a pure-pursuit (PP) low level controller [85] that computes a sequence of commands towards samples generated from the GP. Note that the RR-GP algorithm can be easily modified to handle other dynamical models and low level controllers. Another interesting observation is that the unmodeled process noise in the dynamical model is indirectly considered in both the GP and RRT-Reach components. In the GP component, σ_n in Equation 3.4 could model the process noise, while the closed-loop RRT in RRT-Reach is robust to unmodelled process noise [31].

Every time Algorithm 1 is called, a tree denoted as $\mathcal{T}_{GP}$ is initialized with a root node at the current target vehicle position $(x(t), y(t))$. Variable t_{GP}, which is used for time bookkeeping in the expansion mechanism of $\mathcal{T}_{GP}$, is also initialized to $t + \Delta t$, where t is the current time, and Δt is the GP sampling time interval. Variable gp is

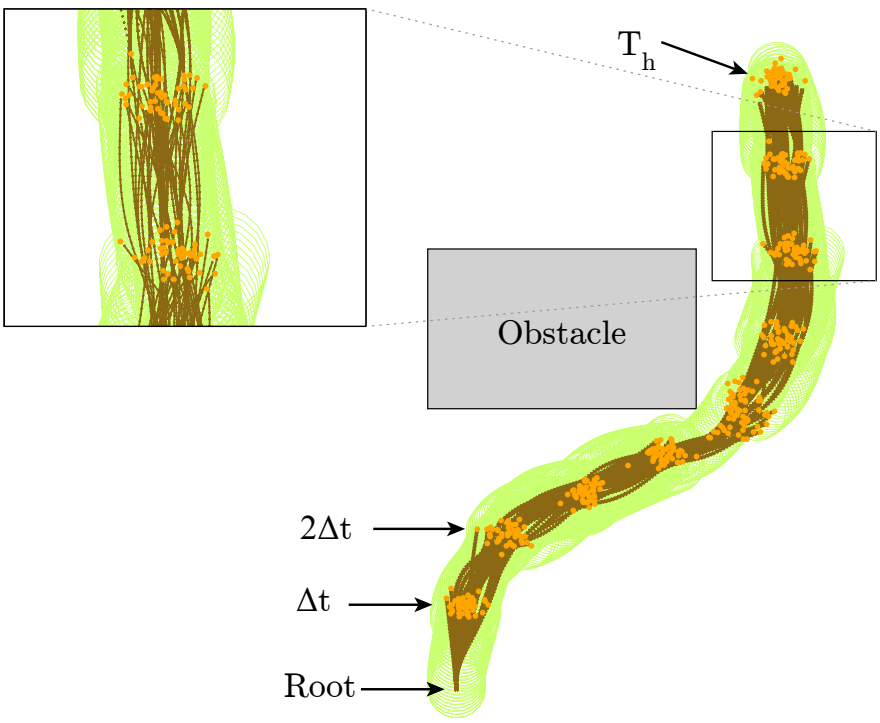


Figure 3-3: Simple RR-GP Illustration ($M = 1$). RR-GP grows a tree (in brown) using GP samples (orange dots) sampled at Δt intervals for a given motion pattern or intent b. The green circle represents the actual size of the target vehicle. The resulting tree provides a distribution of predicted trajectories of the target at $\delta t \ll \Delta t$ increments.

initialized to the learned GP mobility pattern b. Finally, variable K specifying the number of time steps in the GP sampling is initialized to 1. At each step, only nodes belonging to the time bucket $t_{GP} - \Delta t$ are eligible to be expanded. They correspond to the nodes added at to the previous t_{GP}. Initially, the root node is the only node added to time bucket corresponding to initial time t. To grow $\mathcal{T}_{GP}$, first a sample x_{samp} is taken from the environment (line 3) at time $t_{GP} + \Delta t$, or equivalently K time steps in the future using Eq. (3.6). The nodes nearest to this sample, which were added in the previous step (i.e., belonging to time bucket $t_{GP} - \Delta t$), are identified for tree expansion in terms of some distance heuristics (line 4). The algorithm attempts to connect the nearest node to the sample using a straight line reference path; this path is then tracked using the propagation function of a PP controller for steering and a Proportional-Integral controller for tracking the velocity obtained from the GP

Algorithm 1 RR-GP, Single Tree Expansion

1: Initialize tree $\mathcal{T}_{GP}$ with node at $(x(t), y(t))$; $gp \leftarrow$ GP motion pattern b; $t_{GP} \leftarrow t + \Delta t$; $K \leftarrow 1$; $n_{\text{success}} \leftarrow 0$; $n_{\text{infeas}} \leftarrow 0$
2: **while** $t_{GP} - t \leq T_h$ **do**
3: Take sample x_{samp} from gp, K time steps in the future using Eq. (3.6), and variance heuristics if necessary
4: Among nodes added at $t_{GP} - \Delta t$, identify N nearest to x_{samp} using distance heuristics
5: **for** each nearest node, in the sorted order **do**
6: Extend $\mathcal{T}_{GP}$ from nearest node using propagation function until it reaches x_{samp}
7: **if** propagated portion is collision free **then**
8: Add sample to $\mathcal{T}_{GP}$ and create intermediate nodes as appropriate **break**
9: Increment successful connection count n_{success}
10: **else**
11: Increment infeasible connection count n_{infeas} and if limit is reached **goto** line 18
12: **end if**
13: **end for**
14: **if** n_{success} reached desired target **then**
15: $t_{GP} \leftarrow t_{GP} + \Delta t$; $K \leftarrow K + 1$; $n_{\text{success}} \leftarrow 0$; $n_{\text{infeas}} \leftarrow 0$
16: **end if**
17: **end while**
18: **return** $\mathcal{T}_{GP}$

prediction [31]. The actual resulting path is dynamically feasible and is checked for collisions. Since the $\mathcal{T}_{GP}$ is trying to generate typical trajectories that the target vehicle would follow, only a simulated trajectory that reaches the sample without a collision is kept, and its corresponding node is added to the tree (line 8). Intermediate nodes may be inserted occasionally to help faster future expansion. The new node and intermediate nodes are added to the current t_{GP} time bucket. When the total number of successful connections n_{success} is reached, t_{GP} is incremented by Δt and K is incremented by 1 (line 15).

Several heuristics have been used in Algorithm 1. RR-GP keeps tracks of the total number of unsuccessful connections n_{infeas} at each iteration. When n_{infeas} reaches some predetermined first threshold, the variance of the GP for the current iteration is temporary grown to capture a broader class of paths (line 3). This heuristic is typically useful to generate feasible trajectories when GP samples are close to obsta-

cles. For example, in Figure 3-3, variance of gp was increased at $t_{GP} = t + 5\Delta t$. But if n_{infeas} reaches a second and final predetermined threshold, RR-GP "gives up" on growing the tree, and returns $\mathcal{T}_{GP}$. This situation usually happens when the mobility pattern b has a low likelihood and is generating a large number of GP samples in infeasible areas of the environment. The nearest node selection chooses between a cost-to-go metric and a path optimization metric that is based on estimated total path length [86]. Nodes that result in an infeasible trajectory are marked as unsafe and avoided in future node selections. These heuristics, along with the time bucket logic, facilitate efficient feasible trajectory generation in RR-GP.

The nodes and edges of the resulting RR-GP are post-processed to produce a fine time-parametrized distribution of the target vehicle future positions. Since the RR-GP tree is grown at a higher rate compared to the original GP learning phase, the resulting distribution is generated at $\delta t << \Delta t$ sec increments δt is the low-level controller rate. The result is a significant improvement of the accuracy of the prediction without a deterioration of the computation times (Section 3.5).

3.4.2 Multi-Tree RR-GP Algorithm

This section introduces the Multi-Tree RR-GP Algorithm that extends Algorithm 1 to handle multiple motion patterns for the target vehicle. A key feature of the Multi-Tree RR-GP is its computation of the probability of each single tree based on previous measurements of the target vehicle, which removes infeasible patterns and results in a better prediction than traditional GP approaches (Section 3.5).

The length of the prediction problem is T seconds, and the prediction time horizon is T_h seconds. The value of T_h is problem specific, and depends on the time length of the training data. For example, in a threat assessment problem for road intersections, T_h will typically be in the order of 3 to 6 seconds [30]. The RR-GP algorithm updates its measurement of the target vehicle every dt seconds. This value is chosen such that it ensures that the inner loop (lines 6-9) of the Algorithm 2 reaches completion before the next measurement update. Finally, the time period of the low-level controller is equal to δt seconds. A low δt signifies a higher precision of the predicted trajectories,

Algorithm 2 RR-GP, Multi-Tree Trajectory Prediction

1: Inputs: GP motion pattern b_j; $p(b_j(0))$ $\forall j \in [1, \ldots, M]$
2: $t \leftarrow 0$
3: **while** $t < T$ **do**
4: Measure target vehicle position $(x(t), y(t))$
5: Update probability of each motion pattern $p(b_j(t)|x_{0:t}, y_{0:t})$ using Eq. (3.2)
6: **for** each motion pattern b_j **do**
7: Grow a single $\mathcal{T}_{GP}^j$ tree rooted at $(x(t), y(t))$ using b_j (Algorithm 1)
8: Using $\mathcal{T}_{GP}^j$, compute means and variances of predicted distribution $(\hat{x}_j(\tau), \hat{y}_j(\tau))$, $\forall \tau \in [t + \delta t, t + 2\delta t, \ldots, t + T_h]$
9: **end for**
10: Adjust probability of each motion pattern using dynamic feasibility check (Algorithm 3)
11: Propagate updated probabilities backwards and recompute $p(\hat{x}(\tau), \hat{y}(\tau))$ $\forall \tau \in [0, \delta t, \ldots, t]$ if any motion pattern probability was updated
12: $p(\hat{x}(\tau), \hat{y}(\tau)) \leftarrow \sum_j p(\hat{x}_j(\tau), \hat{y}_j(\tau)) \times p(b_j(t)|x_{0:t}, y_{0:t})$ $\forall \tau \in [t+\delta t, t+2\delta t, \ldots, t+T_h]$ using Eq. (3.8)
13: $t \leftarrow t + dt$
14: **end while**

but can slightly affect computation times. A suitable range of δt is 0.02 s to 0.1 s for the problems of interest.

The input of the Algorithm 2 is a set of GP motion patterns and the initial probability distribution over the motion patterns. This prior probability of each motion pattern is proportional to the number of of trajectories belonging to it. In line 4, the position of the target vehicle is measured. Then, the probability that the vehicle trajectory belong to each of the M motion patterns is updated using Eq. (3.2). Recall that the measurements are assumed to be noise-free. For each motion pattern (in parallel), line 7 grows a single-tree RR-GP rooted at the current position of the target vehicle using Algorithm 1. Then, in line 8, the means and variances of the predicted positions of the target vehicle $(\hat{x}(t), \hat{y}(t))$ are computed for each time step τ where $\tau \in [t + \delta t, t + 2\delta t, \ldots, t + T_h]$ using position and time information of the nodes and edges of the single-tree output. This process can be parallelized for the different motion patterns since there is no information sharing between the growth operation of the different single RR-GP trees.

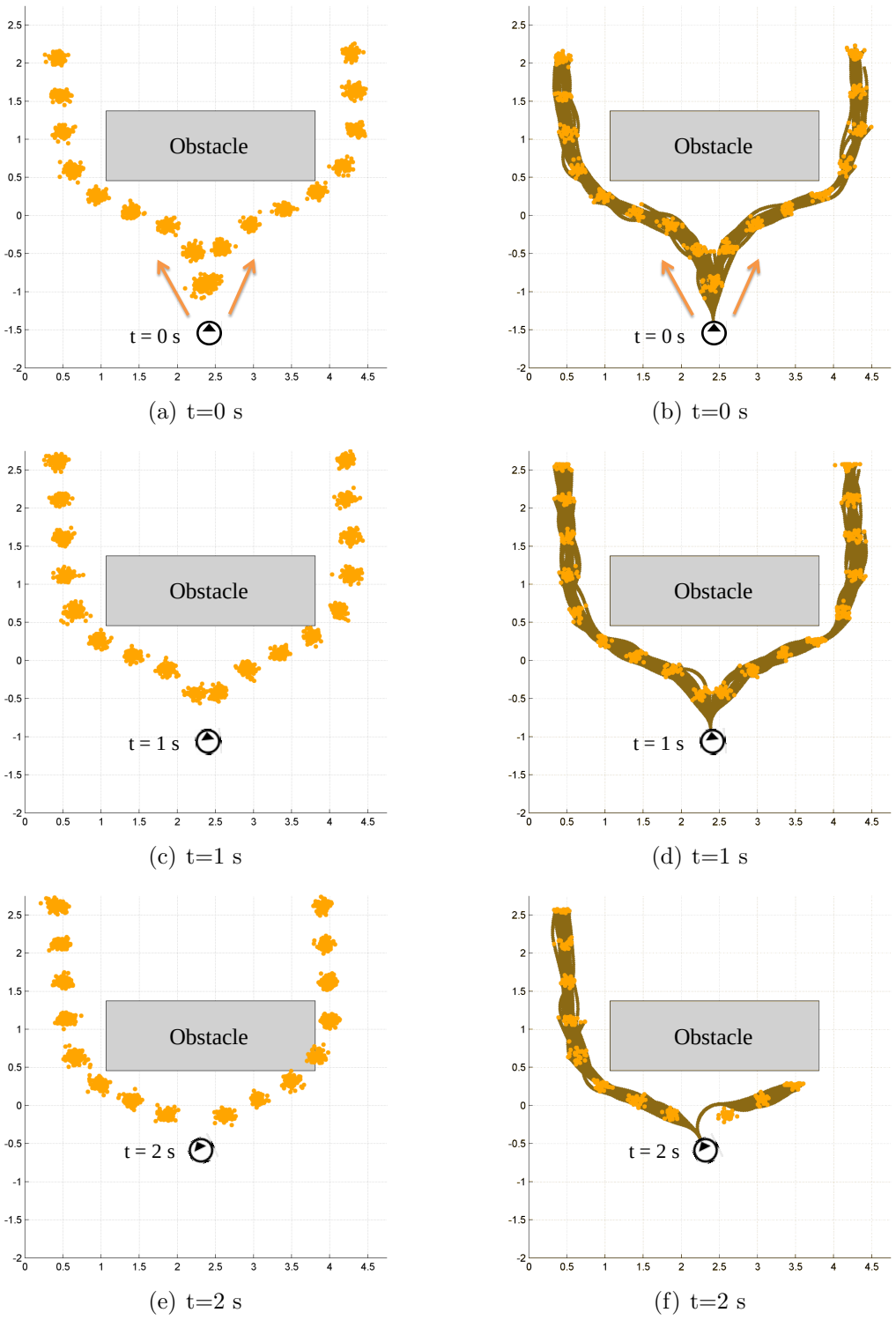
Algorithm 3 Dynamic Feasibility Adjustment

1: Inputs: GP motion pattern b_j; $\mathcal{T}_{GP}^{j}$ trees $\quad \forall j \in [1, \ldots, M]$
2: **for** each motion pattern b_j **do**
3: **if** $\mathcal{T}_{GP}^{j}$ tree ended with infeasibility condition **then**
4: $\tilde{p}(b_j(t)) \leftarrow 0$
5: **else**
6: $\tilde{p}(b_j(t)) \leftarrow p(b_j(t))$
7: **end if**
8: **end for**
9: **for** each motion pattern b_j **do**
10: **if** $\sum_j \tilde{p}(b_j(t) > 0$ **then**
11: $p(b_j(t)) \leftarrow \frac{\tilde{p}(b_j(t))}{\sum_j \tilde{p}(b_j(t)}$
12: **end if**
13: **end for**

In line 10, the probability of each motion pattern is adjusted using Algorithm 3, which zeros the probability of each pattern that ended in an infeasible region (line 4). This modification helps the RR-GP algorithm converge in practice to the more likely patterns faster by incorporating the dynamic feasibility into the probability computation of the motion patterns (See Section 3.5). The dynamic infeasibility for each motion pattern b_j is detected in line 3 of Algorithm 3 by checking if n_{infeas} reached its limit while during the growth of $\mathcal{T}_{GP}^{j}$ tree. Line 11 of Algorithm 3, recomputes the probability for each motion pattern to reflect the changes in Line 4. This probability update is key to an earlier and therefore better prediction of the intent of the target vehicle, leading to better trajectory prediction results (see Section 3.5.2). In the event where all RR-GP trees end with an infeasible condition, the probability values are not adjusted in order to conserve the second axiom of probability i.e., sum of the intent probabilities equals to one. This infeasibility case should rarely happen since the assumption is that the target vehicle actually follows one of the typical patterns. Then, if any of the motion pattern probabilities was altered, line 11 of Algorithm 2 recomputes the probability distribution of the positions of the target vehicle $(\hat{x}(t), \hat{y}(t))$ for all time $\forall \tau \in [0, \delta t, 2\delta t, \ldots, t]$. This step is called the backward propagation (BP), as it propagates the effects of the updated likelihoods to the previously computed position distributions.

Finally, Line 12 of Algorithm 2 combines the position prediction from the different single-tree RR-GP output into one distribution by incorporating the updated motion pattern probabilities b_j into the position distribution of the target vehicle. This computation is performed for all time τ where $\tau \in [t+\delta t, t+2\delta t, \ldots, t+T_h]$, resulting in a probabilistic distribution of the future trajectories of the target vehicle that is based on a mixture of GPs.

Figure 3-4 demonstrates the dynamic feasibility and collision avoidance features of the RR-GP approach on a simple example consisting of two motion patterns belonging to left and right paths around a single obstacle. Training and testing procedures of RR-GP scenarios are explained in detail in Section 3.5. In this illustration, the test trajectory belongs to the left motion pattern. At time $t = 0$ s (Figure 3-4(b)) the target vehicle is pointing upwards and the RR-GP outputs two trees using Algorithm 2, one for each GP motion pattern. At current time $t = 1$ s (Figure 3-4(d)) the vehicle has moved upwards and slightly rotated left. As in every step, RR-GP updates the likelihoods (Figure 3-7) of each motion pattern using Equations 3.2 and 3.3, and generates two new dynamically feasible trees. Figure 3-4(d) shows the updated RR-GP output trees after receiving this second measurement. Simply due to the forward movement and slight left rotation, RR-GP is starting to find more feasible left trajectories than right trajectories (this can be seen by comparing the trajectories as they pass the bottom left and bottom right corner of the obstacle). Finally, at time $t = 2$ s (Figure 3-4(f)), the vehicle has more clearly turned to the left; the RR-GP algorithm returns an incomplete right tree reflecting that all connections to grow the tree further along the right motion pattern failed. This trajectory did, in fact, go left as detected by RR-GP. It is also worth noticing that the GP samples were not all infeasible at $t = 2$ s (Figure 3-4(e)). Furthermore, simple interpolation techniques would not have been able to detect dynamic infeasibilities, highlighting the importance of the dynamic model embedded in the RRT-Reach component of the RR-GP algorithm. Note that Figures 3-4(b), 3-4(d), and 3-4(f) show how the predicted left trees are dynamically feasible and only grown to collision free regions which is key to better predictions.



(a) t=0 s (b) t=0 s

(c) t=1 s (d) t=1 s

(e) t=2 s (f) t=2 s

Figure 3-4: Snapshots of the GP samples (left column) and dual-tree output (right column) of the RR-GP algorithm on a test trajectory following the left GP motion pattern. They illustrate RR-GP's dynamic feasibility and collision avoidance features.

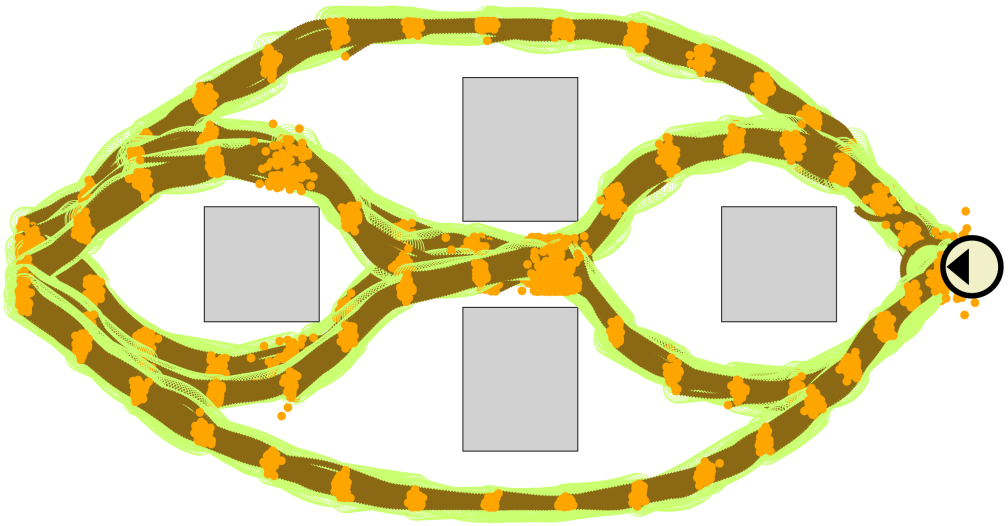


Figure 3-5: Multiple Tree RR-GP Illustration. RR-GP grows six trees corresponding to six different motion patterns of the target vehicle. A likelihood value is also computed for each tree. The trees and the likelihood values are updated in real-time with every new observation of the position of the target vehicle.

Figure 3-5 shows the trajectories generated by RR-GP in a scenario involving six different motion patterns. One pattern consists of the vehicle navigating above all obstacles from the top. Another intent corresponds to navigating above the first obstacle, then in between the middle obstacles, and finally below the last obstacle. The mobility patterns were learned using synthetic training trajectories, and used in Section 4.2.4 in the testing of a probabilistic motion planner enhanced with the RR-GP algorithm to plan safe paths in uncertain environments.

The next section demonstrates the Multi-Tree RR-GP Algorithm on a simplified example that highlights the benefits of the developed approach. It also provides a detailed comparison of the algorithm performance versus two other GP-based techniques.

3.5 Simulated Environment with Human-Operated Target Vehicle

To highlight the advantages of the RR-GP algorithm discussed in Section 3.4, Algorithm 2 is applied on an example scenario consisting of a single target vehicle traveling in an environment with a single fixed obstacle. Recall that our purpose behind augmenting the motion model's predictions with the RRT component was to both overcome the performance loss due to having to use sparse training data for real-time planning as well as only allowing feasible trajectory predictions. The goal of the example is to compare the performances of the RR-GP approach against two baseline algorithms; they both use GP mixtures, but while one is given sparse training data (Sparse-GP), the other one is fed with dense training data (Dense-GP). The performance metrics are both accuracy and computation time. The intuition is that augmenting the GP mixture model with the RRT-Reach algorithm should achieve similar runtime to the sparse GP mixture while maintaining (or even exceeding) the performance of the dense GP mixture. Note that the RR-GP algorithm is trained with the identical data as Sparse-GP, and is therefore equally "sparse".

3.5.1 Setup

The trajectories were manually generated by driving a car-like vehicle in a simulated urban environment described in Ref. [32]. The vehicle uses the iRobot Create software platform [87] with a skid-steered vehicle modified in software to emulate the traditional automotive steering obeying the standard bicycle model

$$\begin{aligned} \dot{x} &= v\cos(\theta), \quad \dot{y} = v\sin(\theta), \\ \dot{\theta} &= \frac{v}{L}\tan(\delta), \quad \dot{v} = a, \end{aligned} \tag{3.9}$$

where (x, y) is the rear axle position, v is the forward speed, θ is the heading, L is the wheelbase equal to 0.33 m, a is the forward acceleration, and δ is the steering angle (positive counter-clockwise). The state of the vehicle is $s = (x, y, \theta, v) \in \mathcal{S}$, while the

input is $u = (\delta, a) \in \mathcal{U}$, including the constraints $a_{\min} \leq a \leq a_{\max}$ and $|\delta| \leq \delta_{\max}$, where $a_{\min} = -0.7$ m/s^2, $a_{\max} = 0.4$ m/s^2, and $\delta_{\max} = 0.6$ rad. The vehicle follows two motion patterns, i.e., $M = 2$ in Eq. 3.1. Starting from its initial location, the vehicle is either driven to the left of the obstacle or to its right. A total of 30 (15 left, 15 right) trajectories were driven, and data were collected at 50 Hz (Figure 3-6).

Given the training trajectories, two GP motion patterns were learned according to Eqs. (3.2), (3.3), (3.4), and (3.5). Both the RR-GP and Sparse-GP use data that were downsampled to 1 Hz. On the other hand, the Dense-GP was trained with 2 Hz data. A higher rate could have been also used, but doubling the density of the data is sufficient to highlight the differences between the algorithms, given the dimensions of the scenario and the maximum speed of the vehicle.

The test data consists of 90 withheld trajectories (45 left, 45 right) generated in the same manner as the training data. In testing, the algorithms received simulated measurements from the test trajectories at one second intervals, i.e., $dt = 1s$ in Algorithm 2. For each time step, Sparse-GP, Dense-GP, and RR-GP are run using the current state of the target vehicle over a time horizon $T_h = 8s$.

In the RR-GP implementation, the control time step is $\delta t = 0.1$ s, while the GP samples are produced at $\Delta t = 1$ s. The limits of successful and infeasible connections per Δt (lines 9,11 Algorithm 1) are 30 and 150, respectively. We follow Ref. [19] in calculating prediction error as the root mean square (RMS) difference between the true position (x, y) and mean predicted position $(\hat{x}, \hat{y})$. The mean is computed using Eq. 3.8 for the Sparse-GP and Dense-GP techniques, and the multi-tree probabilistic distribution for the RR-GP approach. The predictions errors are averaged for all the 90 trajectories at each time step, and summarized in the following section.

3.5.2 Simulation Results

This section presents the simulation results of the RR-GP algorithm that were obtained in our scenario and compares its performance in terms of the prediction accuracy and computation time with both Sparse-GP and Dense-GP. To highlight one of the key features of the RR-GP algorithm, two variations are implemented: 1) RR-GP

Figure 3-6: Training trajectories generated in the simulated road environment according to two motion patterns. The black circle (middle) bottom represents the target vehicle, and the arrow inside the circle represents its heading. The orange arrows point in the direction of each pattern (left and right).

without the backward propagation feature (BP), and the full RR-GP which includes it. RR-GP w/o BP is identical to Algorithm 2, except Line 11 (the BP step) is removed.

Motion Pattern Probabilities

Figure 3-7 shows the average probability (over 90 trajectories) of the correct motion pattern given the observed portion of path followed by the target vehicle. It is computed in RR-GP (w/ BP), Sparse-GP (1Hz), and Dense-GP (2Hz) as function of time. In intervals of $dt = 1$ s, each algorithm receives a new measurement of the target vehicle, it updates the probability of the different motion patterns. While Sparse-GP and Dense-GP only use Eq. (3.2) to update these probabilities, RR-GP algorithm embeds additional logic for dynamic feasibility (See Algorithm 3). The motion pattern that the target vehicle actually follows is labelled "correct". Figure 3-7 shows the the average values of this probability for the different algorithms. Table

Table 3.1: Average Probability of Correct Motion Pattern (over 90 tests)

Alg. \ Time(s))	0	1	2	3
RR-GP	0.922	0.935	1.0	1.0
GP(1Hz)	0.5	0.633	0.995	1.0
GP(2Hz)	0.5	0.665	0.988	1.0

3.1 summarizes these probabilities for the different algorithms for the first three time steps. Note that average probability corresponding to RR-GP w/o BP is the same as RR-GP w/ BP.

At time $t = 0$ s, Sparse-GP and Dense-GP's likelihoods are based on the size of the training data of each motion pattern. Since they are equal, the probability of the correct motion pattern is 0.5. On the other hand, RR-GP algorithm using its collision check and backward propagation (Lines 10-11 of Algorithm 2) is able to improve its "guess" of the correct motion pattern from 0.5 to more than 0.92. At time $t = 1$ s, using its observation of the previous target position, the three algorithms have improved their results for the correct motion pattern likelihood. But RR-GP probability of 0.93 is still significantly better than both Sparse-GP and Dense-GP, which are 0.63 and 0.66, respectively. After three seconds have elapsed, i.e., at $t = 3$ s, the probability of the correct motion pattern for all algorithms have approached 1.0, after they have received three observations and therefore updated their likelihoods three times. It is worth noticing that the differences between the Sparse-GP and Dense-GP plots are not significant; the main advantage of the Dense-GP over Sparse-GP is in the prediction accuracy as illustrated in the next section.

Prediction Errors

Figure 3-8 shows the performance at different times of the scenario of the four algorithms, Sparse-GP, Dense-GP, RR-GP w/o BP, and the full RR-GP (w/ BP), in terms of the RMS of the prediction error between the true value and the predicted

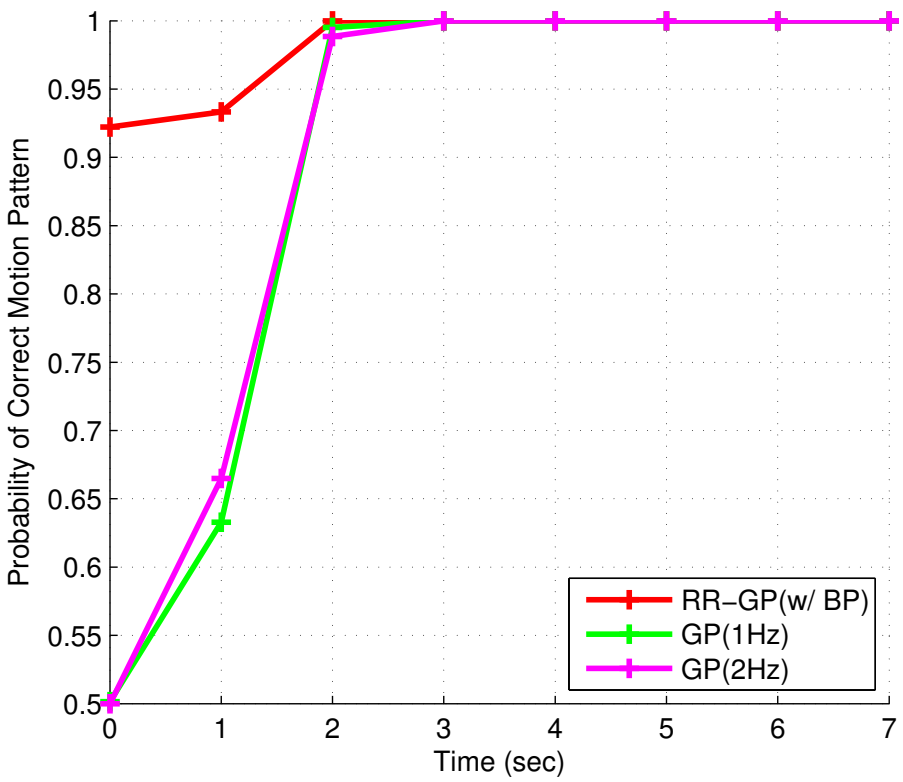


Figure 3-7: Average probability (over 90 trajectories) of the correct motion pattern for RR-GP (w/ BP), Sparse-GP (1Hz), and Dense-GP (2Hz) algorithms as function of time. For example, the values at $t = 1$ s represent the probability of the correct motion patterns after the target vehicle has actually moved for one second on its path.

mean position of the target vehicle.

At the start of each test, when time $t = 0$ s (Figure 3-8(a)), the four algorithms are initialized with the same likelihood values for each motion pattern. This is seen in Sparse-GP and Dense-GP plots which are almost identical. RR-GP (w/o BP) has also a similar performance from $t = 0$ s until $t = 4$ s, because no dynamic infeasibilities or collisions with obstacles happen in this time range. Starting $t = 5$ s, the prediction of RR-GP (w/o BP) improves significantly compared to the GP algorithms since the algorithm is able to detect infeasibility of the wrong pattern, and therefore adjust the trajectory prediction. In other words, the target first encounters the obstacle around $t = 5$ s. The full RR-GP algorithm, denoted as RR-GP (w/ BP) in Figure 3-8, displays the best performance. Using the backward propagation feature, its

prediction accuracy shows significant improvement over that of RR-GP (w/o BP), mainly between $t = 1$ s and $t = 5$ s, by embedding the knowledge of the dynamic infeasibility into the probability computation of each pattern, therefore improving the accuracy of the earlier portion of the prediction. This led to a reduction of RMS prediction errors by a factor of more than 2.4 over the GP-only based algorithms at $t = 8$ s.

After one second has elapsed (Figure 3-8(b)), the vehicle has moved to a new position and the likelihood values of each motion pattern have been updated (Figure 3-7). The probability of the correct motion pattern computed using Eq. 3.2 has slightly increased, leading to lower errors for all three algorithms. But as in figure 3-8(a), a trend is seen for the four algorithms; the performance of the RR-GP (w/ BP) algorithm is consistently and significantly better than both Sparse-GP and Dense-GP. Its performance is also better than RR-GP (w/o BP) in the time range prior to the collision detection, i.e., $t = 2\ s$ till $t = 5\ s$. The improvement in the accuracy of the RR-GP algorithms comparatively to Sparse-GP and Dense-GP is mainly due to a better likelihood computation (Figure 3-7) resulting from the likelihood update logic introduced in the RR-GP algorithms. Another interesting observation is that Dense-GP performs slightly better than Sparse-GP especially between $t = 5\ s$ and $t = 8\ s$. In fact, it is 7% better at $t = 8\ s$. This can be explained by a more accurate GP model due to more training data in the GP learning phase.

Generally, after three seconds have elapsed, the probability of the correct motion pattern has approached 1.0 (Figure 3-7), which explains the decreased level of prediction error among the four algorithms (Figure 3-8(c)). The target value has moved to areas where dynamic feasibility and collision checks are not significant due to the negligible weight of the likelihood of the wrong motion pattern prediction. Equation 3.8 then simplifies to $p(x_{t+K}, y_{t+K}|x_t, y_t, b_{j^*})$, where j^* is the index of the correct motion pattern, and thus the prediction accuracy is only related to the position distribution of the correct motion pattern. This explains why Sparse GP and both RR-GP variations show a very similar accuracy, since their position distributions are based on the same sparse data. On the other hand, Dense-GP, due to more dense data, is the best

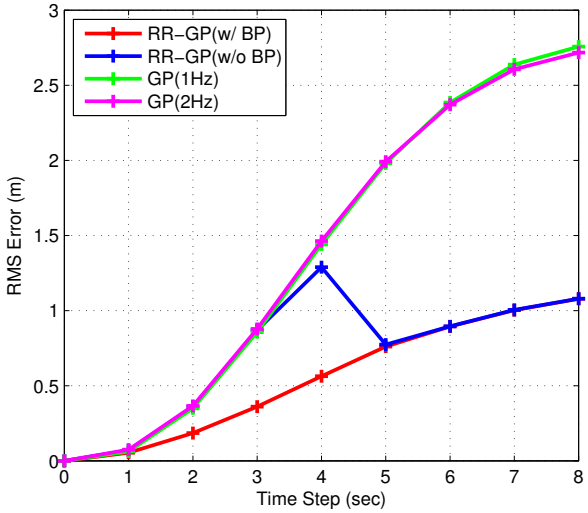
performer as expected with RMS errors 1.5 times smaller than the other algorithms at $t = 8$ s.

To further highlight the performance of RR-GP algorithm, Figure 3-9 presents the box plots [88] of the average difference, over the 90 test trajectories, between the prediction errors of Sparse-GP (1Hz) and the full RR-GP (w /BP). The box plots, also known as the Box-and-Whisker plots, conveniently display the $Q1 = 25^{\text{th}}$ and $Q3 = 75^{\text{th}}$ percentile (the top and bottom of the blue boxes), along with the medium value (the red horizontal bar) of these errors differences. The length of the whisker (dashed black vertical line) is $W = 1.5$, such that data are considered outliers if they are either smaller than smaller than $Q1 - W * (Q3 - Q1)$ or larger than $Q3 + W * (Q3 - Q1)$. These box plots are directly related to the RMS in 3-8. In addition, they provide an insight on the the distribution of these errors of the algorithms.

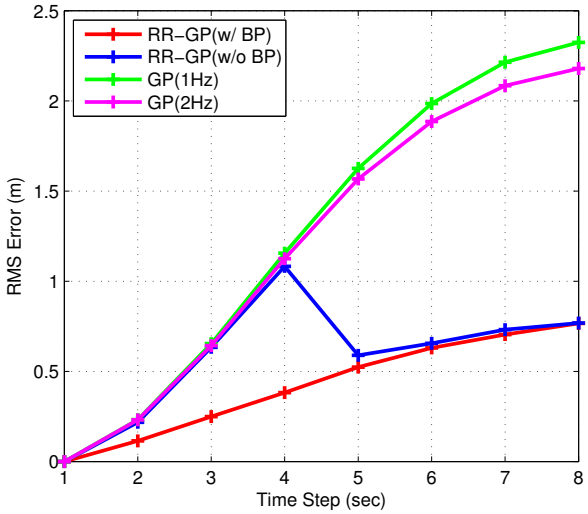
At $t = 0$, Figure 3-9(a) shows the improvement of the RR-GP (w/ BP) over Sparse-GP consistently increase over time to reach a 2.5 m difference at $t = 8$ s. After further analysis of the individual trajectories, we found that the outliers are the test trajectories that were dynamically feasible at $t = 0$ s to follow any of the two motion patterns. In these few cases, RR-GP does not have an advantage over the Sparse-GP, leading to no error difference. But in the majority of the tests, as shown by the whisker lengths and box sizes, RR-GP (w/ BP) significantly reduced the prediction error.

At $t = 1$ s, Figure 3-9(a) presents the error difference after the target vehicle has moved for 1 s. A similar trend to Figure 3-8(b) can be seen. The median of the difference grows with the time steps, but its magnitude slightly decreases, reaching 2.1 m difference at $t = 8$ s. There are fewer outliers, but equivalently, the length of the whiskers are longer to include similar cases described at $t = 0$.

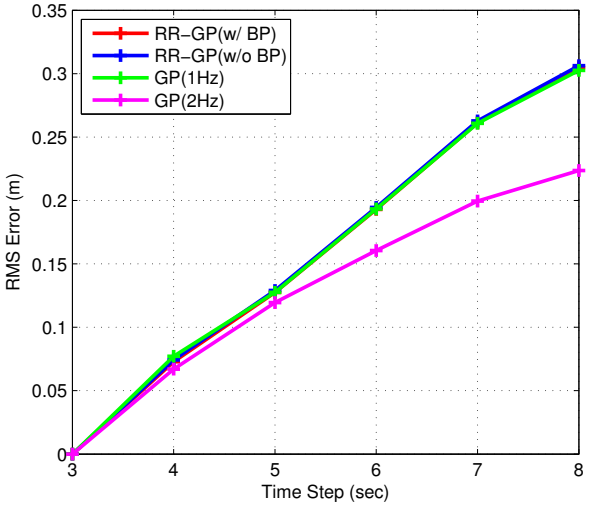
Finally, at $t = 3$ s, the target vehicle has moved for 3 s, and the differences between the RR-GP (w/ BP) and Sparse-GP are not statistically significant for the same reasons discussed previously (See discussion for Figure 3-8(c)).



(a) t = 0 s

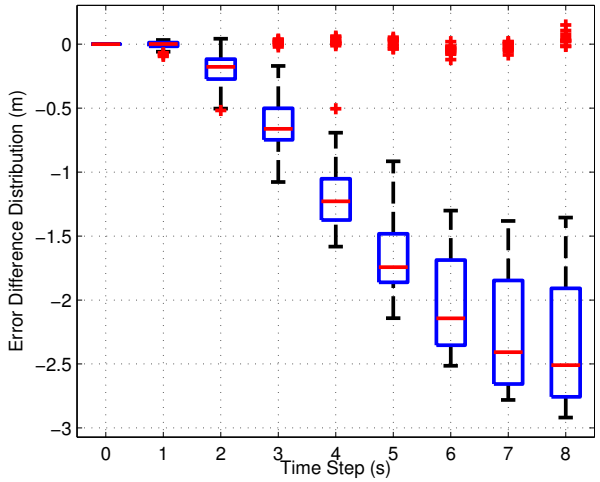


(b) t = 1 s

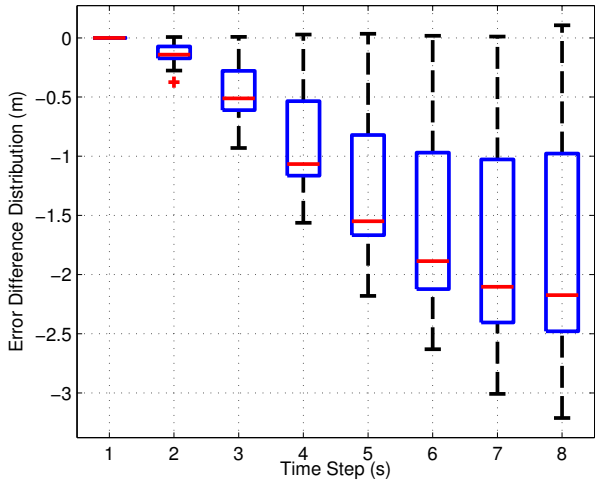


(c) t = 3 s

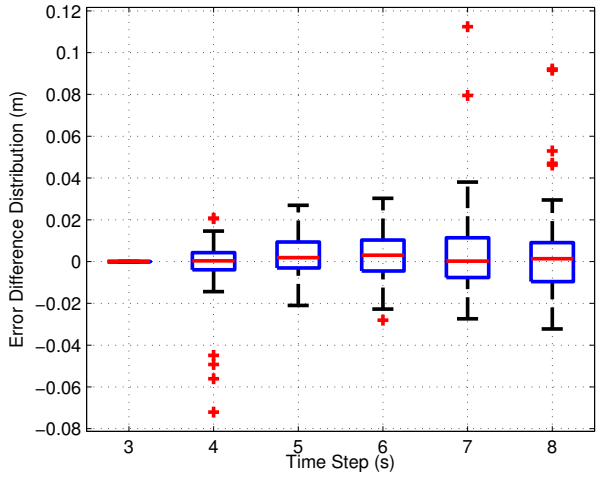
Figure 3-8: Average position prediction errors (over 90 trajectories) for Sparse-GP (1Hz), Dense-GP (2Hz), and the two variations of RR-GP algorithm at different times of the example.



(a) t = 0 s



(b) t = 1 s



(c) t = 3 s

Figure 3-9: Box plots of the difference of prediction errors (for the 90 test trajectories) between Sparse-GP (1Hz) and the full RR-GP algorithm at different times of the example.

Computation Times

Table 3.2 summarizes the average computation times per iteration of the three algorithms over the 90 testing paths. Both RR-GP variations have identical computation times, so only one computation time is shown for RR-GP. Note that the implementation uses the GPML MATLAB toolbox [81], and these tests were run on a 2.5 GHz quad-core computer.

As expected, the Sparse-GP has the lowest computation time, while the RR-GP algorithm ranks second with times well below 1 s, which are suitable for real-time application as shown in Section 4.2.4. Out of the 0.69 s of RR-GP computation time, an average 0.5 s is spent in the RR-GP tree generation while the remaining 0.19 s is used to generate the GP samples. On the other hand, the Dense-GP had an average computation time of roughly 2.3 s, which is significantly worse than the other two approaches.[1] This is expected since GPs scale poorly with the training data size due to the matrix inversion operations in Eq. 3.5. In fact, the time complexity of the GP algorithms is $\mathcal{O}(|X|^3K)$, where $|X|$ is the size of training data points, and K is the number of time step predictions in the future [81]. On the other hand, the time complexity of RR-GP is $\mathcal{O}(|X|^3K + N_{\text{ic}}K)$, where N_{ic} corresponds to the maximum allowed number of infeasible connections per time step (line 11 of Algorithm Algorithm 1). Moreover, the dominant factor in both computational complexities is $|X|^3$, which explains the significant increase in computation time for Dense-GP compared to both Sparse-GP and RR-GP. The lengthy computation times of Dense-GP makes it useless for any typical real-time implementation. Finally, it is worth noting that the computation times of Dense-GP violate the assumption that the GP algorithm can update its prediction of the target vehicle during the $dt = 1$ s measurement cycle (Section 3.5.1). If dt is adjusted to 2.3 s for Dense-GP tests, the prediction accuracy will decrease due to slower and fewer updates of the probabilities of the different motion patterns.

In summary, this simulated environment with a human-driven target vehicle showed

[1] This increase in computation time is even more significant with a denser GP algorithm trained using 5Hz data. Its average computation time per iteration is found to be more than 35 s.

Table 3.2: Average Computation Times Per Iteration (over 90 Trajectories)

Algorithm	Comp. Time (s)
Sparse-GP (1 Hz)	0.19
Dense-GP (2 Hz)	2.32
RR-GP	0.69

that the RR-GP algorithm performed consistently better than the Sparse-GP and Dense-GP in the long-term prediction of the target vehicle. When the approaches compared in this section are used in combination with a planner in this type of domains, we often desire to take actions at frequencies approaching 10 Hz. It is important to highlight that the RR-GP can output predict trajectories at arbitrarily high output frequency, which is significantly higher than both the Sparse-GP (1Hz) and the Dense-GP (2Hz). While we could augment the comparison approaches with some form of interpolation technique, our approach systematically guarantees collision avoidance and dynamic feasibility of the trajectory predictions. Another feature of the RR-GP algorithm is its efficient computation time (Table 3.2), which is low enough to be suitable for real-time implementation in collision warning systems or probabilistic path planners for autonomous systems.

3.6 Validation on Road Traffic Data

This section validates the developed RR-GP algorithm on road traffic data collected through the Cooperative Intersection Collision Avoidance System for Violations (CICAS-V) project [37]. The CICAS-V data consists of a large database of intersection approaches in different US cities. This work, as in Chapter 2, uses trajectories gathered at the Peppers Ferry intersection (U.S. 460 Business and Peppers Ferry Rd in Christiansburg) to verify the developed RR-GP algorithm and compare its performance to two GP-based algorithms.

This section is structured as follows. First, Section 3.6.1 reviews the data collection

and selection process. Then, Section 3.6.1 describes the training procedures for the RR-GP and the two GP-based algorithms. Finally, Section 3.6.3 presents the results of the tests by summarizing prediction accuracy and running times.

3.6.1 Data Collection and Selection

The method of collection of the roadside data is briefly presented in Section 2.4. A detailed description can be found in Ref. [64]. In the Aerospace Controls laboratory, vehicle trajectories collected at the Peppers Ferry intersection were classified as proceeding left, right, or straight based on a combination of the trajectory ID and the lane ID extracted from the CICAS-V database. The final lane of the vehicle was used to determine its final direction. The trajectories were then filtered based on whether the vehicle stopped or not. This resulted in six different classifications on the basis of turning and stopping. This work focused on the non-stopping trajectories, which were further filtered to remove erroneous instances that showed discontinuities in either positions or velocities. Finally, it selected trajectories that were longer than 80 m and lasted more than 8 s. Note that data points further than 90 m from the intersection were ignored. Figure 3-10 shows a satellite image of the Peppers Ferry intersection taken from Google Earth (http://www.google.com/earth/index.html), along with some of the trajectories selected from the CICAS-V database for the validation of the RR-GP and GP-based algorithms.

3.6.2 Training Procedure

This work uses the full RR-GP (Algorithm 2) which showed the best performance in the simulation example results (Section 3.5). RR-GP algorithm is compared to two GP-based approaches, Sparse-GP and Dense-GP, that are trained with different amount of data. Sparse-GP is provided with 45 trajectories in the training phase, equally distributed between the three intersection approaching patterns (left, right, straight). The RR-GP algorithm is given the same number of trajectories as Sparse-GP. On the other Dense-GP is trained with 135 trajectories, also equally divided

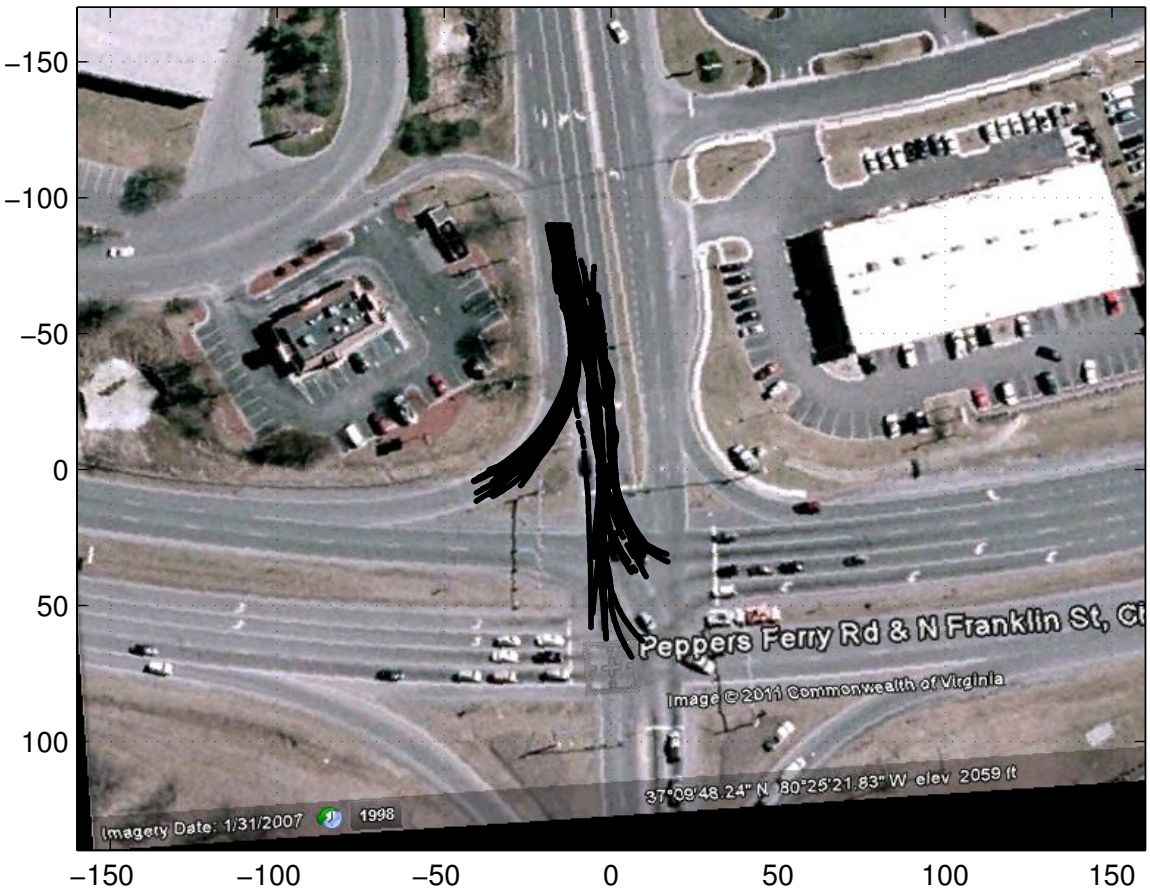


Figure 3-10: Satellite image of the Peppers Ferry intersection adapted from Google Earth. CICAS-V trajectories of vehicles at Pepper's Ferry intersection were used to test the algorithms presented in this section. 45 instances are shown as black lines superimposed on the intersection image.

between the three patterns. The trajectories are all downsampled from the original 20 Hz rate to 1 Hz. Therefore, unlike the simulated example (see Section 3.5.1), the density of training data for Dense-GP is due to the number of training trajectories rather than the rate to which the trajectories are downsampled.

3.6.3 Testing Results

The test data consists of 120 trajectories (40 left, 40 right, 40 straight) selected in the same manner as the training data. In testing, the algorithms received simulated measurements from the test trajectories at one second intervals, i.e., $dt = 1$ s in Algorithm 2. For each time step, Sparse-GP, Dense-GP, and RR-GP are run using

the current state of the target vehicle over a time horizon $T_h = 6$ s.

In the RR-GP implementation, the control time step is $\delta t = 0.1$ s, while the GP samples are produced at $\Delta t = 1$ s. The limits of successful and infeasible connections per Δt (lines 9,11 Algorithm 1) are 15 and 75, respectively. The vehicles are assumed to obey the standard bicycle model (Eq. 5.7). Ideally, the parameters of the model of the each vehicle should be estimated in real-time to obtain the best performance. But since the database does not provide any information about the vehicle parameters, this work uses the values from the MIT Land Rover LR3 car that participated in the 2007 DARPA Grand Challenge [31]. They are $a_{\min} = -6.0$ m/s^2, $a_{\max} = 2.8$ m/s^2, $\delta_{\max} = 0.54$ rad, and $L = 2.9$ m. The RR-GP algorithm also requires the location of the infeasible regions of the environment. These regions were visually estimated using the road geometry in Figure 3-10. They are then approximated using several rectangles with different orientations (See Figure 3-11).

As in Section 3.5.2, the prediction errors are defined as the root mean square (RMS) difference between the true position (x, y) and mean predicted position $(\hat{x}, \hat{y})$. The mean is computed using Eq. 3.8 for the Sparse-GP and Dense-GP techniques, and the multi-tree probabilistic distribution for the RR-GP approach. The predictions errors are averaged for all the 120 trajectories at each time step.

Illustration

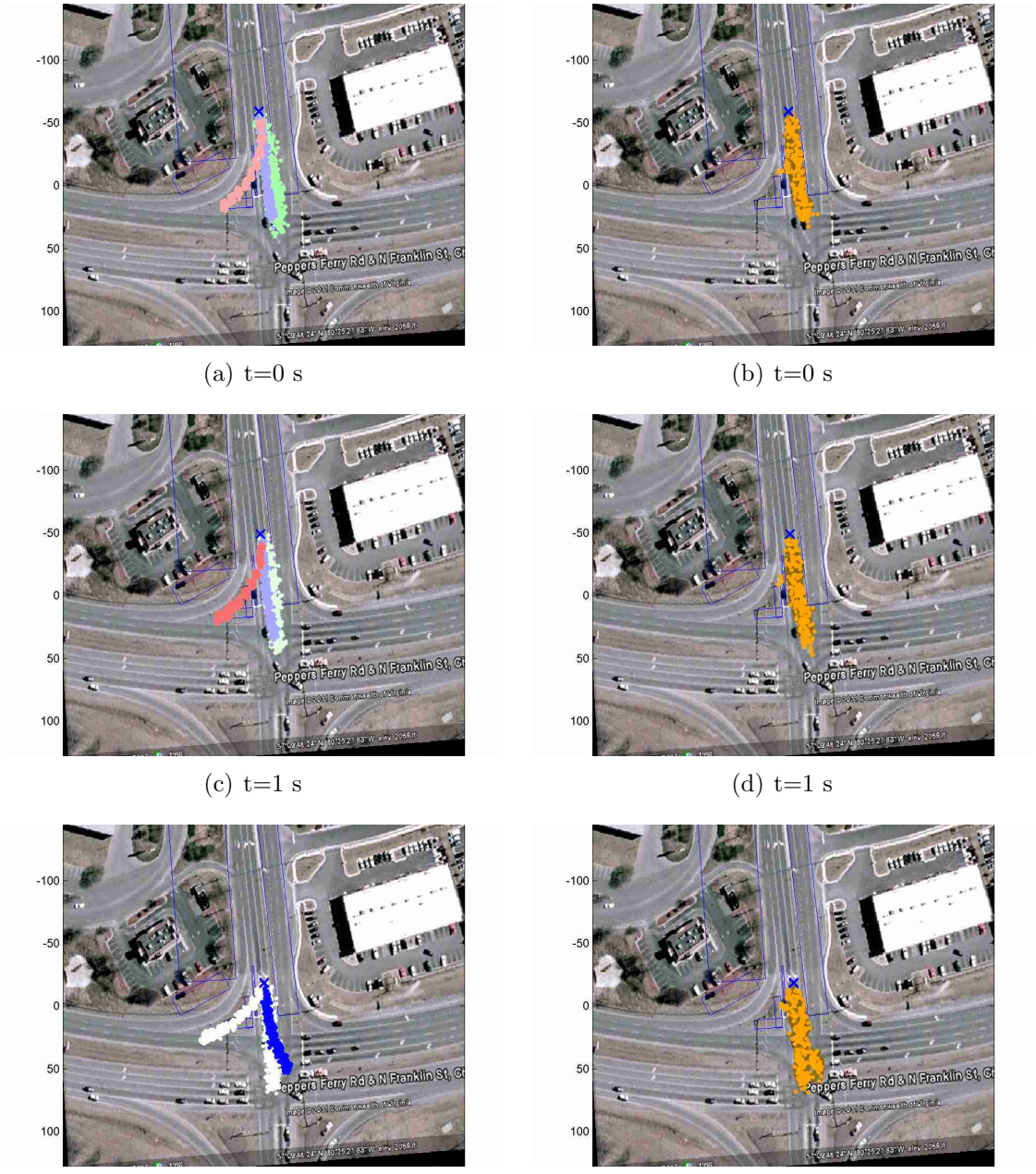
Figure 3-11 illustrates the RR-GP approach on a test trajectory belonging to the left turn motion pattern (See Section 3.6.3). It shows the GP samples (left column), and the multi-tree output (right column). The GP samples are identical to the outputs of Sparse-GP, since RR-GP and Sparse-GP use the same training data. The current position of the vehicle is shown as a blue $\times$. The motion patterns are shown in red (right), green (straight), and blue (left). The probability of each pattern is reflected by the saturation of its color. A saturation of 1 gives a colorful red, green, or blue, while a saturation of 0 turns any color into white. The probabilities of the motion patterns are written as the vector $p_{\text{mp}} = (p_{\text{left}}, p_{\text{right}}, p_{\text{straight}})$. Note that the naming of the pattern is relative to the heading direction of the vehicle.

At time $t = 0$ s, Sparse-GP predicts that the future trajectory of the target vehicle has an equal likelihood of following any of the tree motion patterns (Figure 3-11(a)), i.e., $p_{\text{mp}} = (0.333, 0.333, 0.333)$ as it is also apparent from the similar saturation levels of the colors. On other hand, the RR-GP algorithm is able to detect, using its dynamic feasibility check, that the right pattern is infeasible (Figure 3-11(b)). Using its backward propagation logic, it finds $p_{\text{mp}} = (0.5, 0, 0.5)$, which results in a significantly smaller prediction error. In fact, the true $p_{\text{mp}} = (1, 0, 0)$. It is worth noticing that the left and straight patterns are similar when the vehicle is far from the intersection (as can been seen in the training data in Figure 3-10). At time $t = 1$ s, Sparse-GP prediction evolves to $p_{\text{mp}} = (0.337, 0.558, 0.105)$ slightly favoring the right turn motion pattern, which is the incorrect one. This can be seen in Figure 3-11(c) where the red trajectories have the most saturated color. On the other hand, RR-GP algorithm predicts again that the right motion pattern is infeasible, and estimates that $p_{\text{mp}} = (0.768, 0, 0.232)$, leading to an improved trajectory prediction 3-11(d). It is only at $t = 4$ s that Sparse-GP predicts that the left motion pattern is the most likely one (Figure 3-11(e)) with $p_{\text{mp}} = (0.973, 0.0183, 0.087)$, while RR-GP computes an even better probability estimate, $p_{\text{mp}} = (0.991, 0, 0.009)$ (Figure 3-11(f)).

This section illustrated the RR-GP's advantage over a GP-based algorithm on a single test trajectory. The following section will compute the prediction errors for the entire testing dataset.

Prediction Errors

Figure 3-12 shows the performance at different times of the the Sparse-GP, Dense-GP, full RR-GP algorithm, in terms of the RMS of the prediction error between the true value and the predicted mean position of the target vehicle. After 1 s has elapsed (Figure 3-12(a)), the RR-GP algorithm has the lowest average RMS error. It is 1.6 and 1.8 times smaller than the Dense-GP and Sparse-GP errors at $t = 6$ s, respectively. As explained in detail in Section 3.5.2, the improvement of the RR-GP algorithm is due to the novel likelihood computation that is based on both the dynamic feasibility of the RRT-Reach trees and the back propagation technique. The magnitude of the

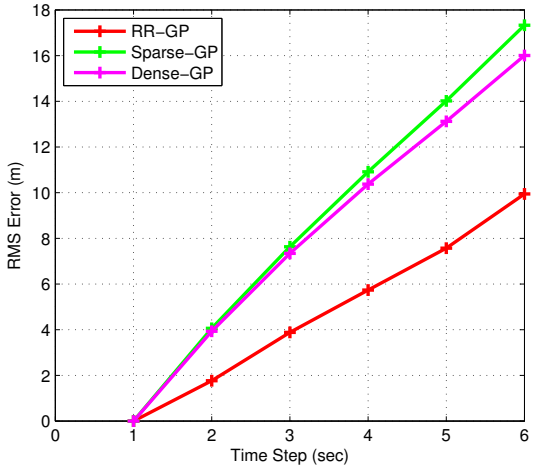


(a) t=0 s (b) t=0 s

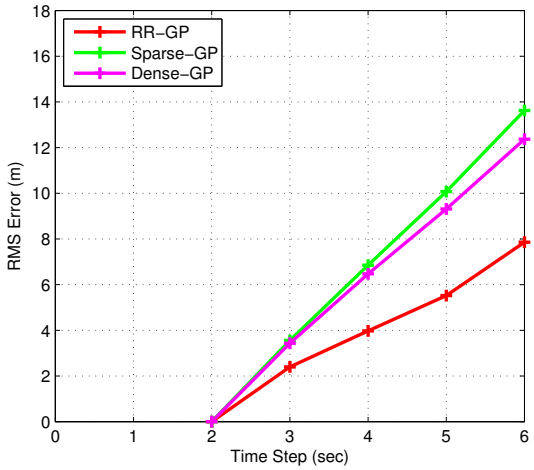
(c) t=1 s (d) t=1 s

(e) t=4 s (f) t=4 s

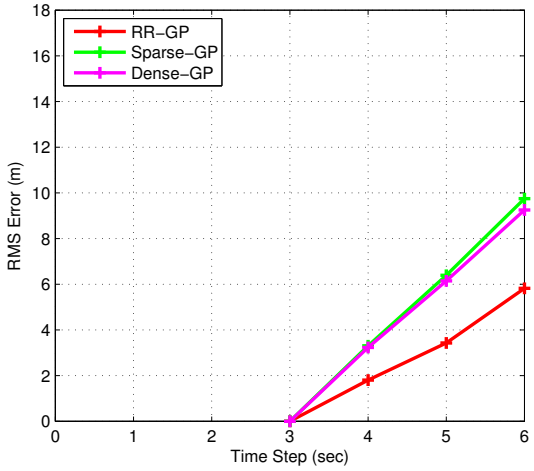
Figure 3-11: Snapshots of the GP samples (left column) and multi-tree output (right column) of the RR-GP algorithm on a vehicle trajectory that follows the left turn motion pattern at the Peppers Ferry intersection. The position of the vehicle is shown as a blue $\times$. Rectangles (with blue borders) represent infeasible regions.



(a) t = 1 s



(b) t = 2 s



(c) t = 3 s

Figure 3-12: Average position prediction errors (over 120 trajectories) for Sparse-GP, Dense-GP, and the full RR-GP algorithm at different times of the real-traffic tests.

Table 3.3: Average Computation Times Per Iteration (over 120 Trajectories)

Algorithm	Comp. Time (s)
Sparse-GP	0.25
Dense-GP	1.07
RR-GP	0.44

RMS errors of the three algorithms decreases after 2 s. It continues to decrease after 3 s, due to the likelihood updates that occur every time a measurement of the target is received. After 3 s has elapsed (Figure 3-12(c)), the gap between the RMS errors of RR-GP algorithm and the GP-based algorithm becomes smaller, but the ratio of the errors does not change significantly. In fact, the RR-GP average RMS error is still 1.6 and 1.7 smaller than the Dense-GP and Sparse-GP errors, respectively, at $t = 6$ s. These results demonstrate that the RR-GP algorithm shows a significantly better prediction accuracy on the road traffic data compared to both Sparse-GP and Dense-GP algorithms.

Computation Times

Table 3.3 summarizes the average computation times per iteration of the three algorithms over the 120 testing paths. As expected, the Sparse-GP has the lowest average computation time, while the RR-GP algorithm ranks second with average times below 0.5 s. Both computation times are suitable for real-time implementation with a $dt = 1$ s measurement cycle. On the other hand, the Dense-GP has an average computation time of slightly over 1 s. This is expected since GPs scale poorly with the training data size due to the matrix inversion operations in Eq. 3.5. Despite longer computation times with Dense-GP, the deterioration in the times is not as apparent as in Section 3.5. This is due to the fact that the real-traffic trajectories are more scattered than the simulated trajectories due their limited number, leading to sparser matrices and thus more efficient matrix inversions. For larger training dataset, it is expected, as in Section 3.5, that the Dense-GP computation times will further in-

crease compared to Sparse-GP and RR-GP. Finally, these results confirm that the RR-GP algorithm running times are suitable for real-time implementation.

3.7 Conclusion

This chapter presented a novel approach to improve the trajectory predictions of agents with uncertain intentions for collision avoidance (CA) and conflict detection (CD) systems. It combined the RRT-Reach algorithm with a Gaussian Process (GP) mixture model. RRT-Reach predicts the trajectories using a sampling-based reachability algorithm that incorporates vehicle and environment constraints. The GP mixture, a nonparametric Bayesian model, captured the distribution over a wide range of possible target agent motion patterns. The resulting RR-GP algorithm, ensures that the predicted trajectories are both feasible by construction and probabilistically weighted to reflect the typical motion patterns of the agents. RR-GP was compared against two GP-based algorithms trained with sparse and dense data, respectively. It was shown to outperform both GP algorithms in prediction accuracy, while still achieving computation times that are suitable for real-time implementation. This was first verified using a simulation example of a human-operated target vehicle. Then, traffic data from intersection approaches collected through the CICAS-V project demonstrated the effectiveness and superiority of the RR-GP approach in accurately predicting future trajectories in a real-world example.

Chapter 4

Safe Motion Planning with Uncertainty in Predictability

This chapter introduces new path planning algorithms that build upon the tools that have been introduced previously in this thesis. The focus is on navigation in environments with uncertainty in predictability, where the autonomous vehicle plans trajectories in in the presence of other vehicles with unknown future intentions / destinations.

To tackle this problem, Section 4.1 proposes a new threat-aware path planning approach that augments existing path planners with an efficient threat estimation of other vehicles in the environment [32]. This computation is performed in a novel threat assessment module that consists of a behavior classifier based on Chapter 2 along with an efficient reachability-based algorithm. The strengths of this approach are demonstrated through simulation and experiments performed in the MIT RAVEN testbed [89].

Section 4.2 extends the reachability component of Section 4.1 to include probabilistic knowledge of the typical motion patterns followed by other vehicles in the environment [33]. This knowledge is computed is based on a trajectory prediction of their motion, computed using the RR-GP algorithm introduced in Chapter 3. RR-GP is integrated with a state-of-the-art probabilistic path planner [34] that explicitly incorporates probabilistic constraint satisfaction in its formulation. Theoretical guar-

antees are demonstrated for linear systems, though the extension to nonlinear systems is also considered. Simulation results show that the resulting approach can be used in real-time to efficiently and accurately execute safe paths.

4.1 Threat-aware Path Planning

This section considers the path planning problem for an autonomous vehicle in an urban environment populated with static obstacles and moving vehicles with uncertain intents. We propose a novel threat assessment module, consisting of an intention predictor and a threat assessor, which augments the host vehicle's path planner with a real-time threat value representing the risks posed by the estimated intentions of other vehicles. This new threat-aware planning approach [32] is applied to the CL-RRT path planning framework, used by the MIT team in the 2007 DARPA Grand Challenge. The strengths of this approach are demonstrated through simulation and experiments performed in the RAVEN testbed facilities.

4.1.1 Introduction

Whether driving on highways or navigating in the middle of a battlefield, intelligent vehicles must be able to quickly and robustly compute motion plans in very uncertain worlds. The sources of this uncertainty may be internal, i.e., incomplete or imperfect knowledge of the vehicle model, or external, i.e., incomplete or imperfect knowledge of the environment, and may be present either in sensing or in predictability [22]. This section addresses problems involving uncertainty in predictability, and in particular the intentions of the other vehicles within our vehicle's world.

To make meaningful predictions of other vehicles' intentions, a smart vehicle should be able to gather information from the environment to build models approximating those intentions. It is not safe to assume that all drivers obey all "rules of the road," as many collisions occur when this is not the case, but classifying all other vehicles as hostile would be overly conservative. The information gathered by our vehicle might include onboard camera images, radar-based measurements of surrounding

objects, or messages intercepted or shared on some communication channels.

The vehicle planner must be able to address the uncertainty in each vehicle's future state and actions, or at a higher level, the behavior / intent governing those actions. Existing path planners typically rely on *a priori* information to generate trajectories, applying reactive maneuvers or replanning as needed to correct the path online and maintain feasibility. However, uncertainty in object predictability is typically not considered explicitly in the global path planner. Instead, moving vehicles are often assumed to follow known trajectories or policies, which the planner reacts to locally. While full probabilistic representation of the environment is possible, such as with POMDPs [90, 91], the corresponding solution techniques are computationally intractible for real-time path planning problems of even modest complexity or dimension.

The main motivation of this work is the set of challenges faced by autonomous vehicles in the 2007 DARPA Grand Challenge (DGC) [16], which involved navigating an outdoor urban environment in the presence of other vehicles while obeying all traffic regulations. One of the main challenges of this race was negotiating traffic intersections, where several vehicles were involved in collisions or near-collisions. Several explanations have been offered for these occurrences, but the one that motivated this work is the inability of the autonomous vehicles to anticipate the intent of other vehicles [16]. With some knowledge of those intentions, the motion planner could incorporate the risk posed by those vehicles when considering potential trajectories, improving safety.

Recent work ([92–94]) has presented different approaches to embed the threat posed by the other vehicles in the planned trajectories of an autonomous vehicle. Ref. [92] presented a hierarchical framework that combines multiple layers of prediction for moving objects to improve their estimated locations. While it demonstrates in simulation for a specific scenario (lane changing example), Ref. [92] does not address more general scenarios, and no hardware validation is provided. Ref. [93] introduced an approach that embeds the traffic situation into the trajectories of an autonomous vehicle to improve their safety. Each traffic participant is represented by a stochastic

reachability set based on a separate Markov chain. While Ref. [93] states that the intensive computations of the reachability sets are preformed offline, it is unclear how these computations could cover all road scenarios and general dynamical models of the vehicles. Finally, Ref. [94] developed a formal hybrid control approach to guarantee safety in the presence of a human-driven vehicle in a roundabout testbed. In the presented experiments, the human operator was instructed to accelerate or decelerate at a specific decision point to force the creation of unsafe sets. While the formal safety guarantees are important, the practical implementation of the developed approach is yet to proved for more general intersection encounter cases, and more involved human models.

This section introduces a new framework for autonomous vehicles which enables path planning algorithms to explicitly incorporate both obstacle uncertainty and the corresponding risk posed to the vehicle. A threat assessment module, consisting of an intention predictor and a threat assessor, augments the host vehicle's path planner with a real-time threat value for each potential trajectory, reflecting the risks posed by the estimated intentions of other vehicles. The strengths of this approach are demonstrated through simulation and experiments performed in a city-like testbed in the MIT RAVEN facility [89].

4.1.2 Problem Statement

Consider the autonomous system denoted by $\mathcal{HV}$,

$$\dot{s}(t) = f(s(t), u(t), t), \tag{4.1}$$

where s is the state and u is the control input; we have that s is constrained to the state space $\mathcal{S}$, $s \in \mathcal{S}$, while u is contstrained to the control space $\mathcal{U}$, $u \in \mathcal{U}$. Our objective for safe motion planning is to minimize some desired cost functional over the duration of the motion,

$$L = \Psi(s(t_f), t_f) + \int_{t_0}^{t_f} \Gamma(s(t), u(t), t)dt, \tag{4.2}$$

subject to the vector of component-wise constraints

$$\boldsymbol{C}(s(t), u(t), t) \leq \mathbf{0}. \tag{4.3}$$

Of the many constraints $\boldsymbol{C}$ might include, we are interested in those necessitating collision avoidance with N other vehicles, denoted here as $\mathcal{OV}_i$, $i = 1 \dots N$. These can typically be written in the form

$$\min_{t_0 \leq t \leq t_f} \|p_s(t) - p_{r_i}(t)\| > \epsilon \quad \forall i = 1 \dots N, \tag{4.4}$$

where $r_i(t)$ is the state of vehicle i at time t, $p_s(t)$ is the position vector corresponding to $s(t)$, $p_{r_i}(t)$ is the position vector corresponding to $r_i(t)$, ϵ is the minimum allowed distance between the $\mathcal{HV}$ and the $\mathcal{OV}s$, and $t_f = t_0 + T_h$ where T_h is the time horizon of interest. We say that $\mathcal{HV}$ is in collision with $\mathcal{OV}_i$ if the tuple $(p_s(t), p_{r_i}(t))$ enters the closed "collision" set Ω_i,

$$\Omega_i = \{p_s(t) \mid \|p_s(t) - p_{r_i}(t)\| \leq \epsilon\}. \tag{4.5}$$

In uncertain and/or non-cooperative environments, $r_i(t)$ is typically not available for the $\mathcal{HV}$. For $\mathcal{HV}$ to maintain guaranteed safety, some mechanism must be able to identify the input sequence $u(t)$ which yields a feasible path for all possible realizations of $r_i(t)$, $i = 1 \dots N$. In the event that no such path exists, the same mechanism should select $u(t)$ in order to minimize some threat level. Here we define the threat level $\mathcal{T}_i$ for vehicle i as inversely proportional to t_{c_i}, the earliest possible time of collision between $\mathcal{HV}$ and $\mathcal{OV}_i$:

$$\mathcal{T}_i = \frac{1}{t_{c_i}}, \quad t_{c_i} = \inf\{t \mid (p_s(t), p_{r_i}(t)) \in \Omega_i\}. \tag{4.6}$$

Equation (4.4) thus cannot typically be guaranteed at all times, so this constraint is instead converted to a penalty in the objective function. We define the new cost

functional

$$J = L + w \cdot \sup_{i \in 1 \ldots N} \mathcal{T}_i, \tag{4.7}$$

where L is defined in (4.2), $\mathcal{T}_i$ is the threat level defined in (5.3), and w is a user-selected weighting factor representing the tradeoff between safety and optimality. The constraints (4.4) are then removed from (4.3), yielding

$$\bar{\boldsymbol{C}}(s(t), u(t), t) \leq \mathbf{0}. \tag{4.8}$$

Problem Definition (Threat-Aware Dynamic Motion Planning): Given a state space $\mathcal{S}$, a control space $\mathcal{U}$, an initial state $s_0 \in \mathcal{S}$ and a final state $s_f \in \mathcal{S}$, compute the control input sequence $u(t)$, $t \in [0, t_f]$, $t_f \in [0, \infty)$ that minimizes (4.7) subject to the constraints (4.1) and (4.8).

4.1.3 Threat-Aware Planner

Several global path planners, including sampling-based planners, use the philosophy that collision detection should be done in a "black box" which decouples the host vehicle's path planning from any specific geometric or kinematic obstacle models [84]. This work proposes that threat assessment should be done in a similar "black box" framework, providing planners with a quantitative threat metric to identify safer paths during trajectory generation. This threat assessment considers the unknown intentions of moving $\mathcal{OV}s$, which may present a collision risk for the $\mathcal{HV}$.

High-level TAM Architecture

The proposed architecture features a threat assessment module (TAM, Figure 4-1), consisting of an intention predictor (IP) and a threat assessor (TA). The IP uses observations taken by the $\mathcal{HV}$ of the $\mathcal{OV}s$ to make predictions of their intentions. Instead of using low-level reasoning to predict future *trajectories*, the IP uses higher-level logic and learning to provide the planner with predicted future *intentions*, key to earlier and more accurate prediction of future collisions. The TA then converts

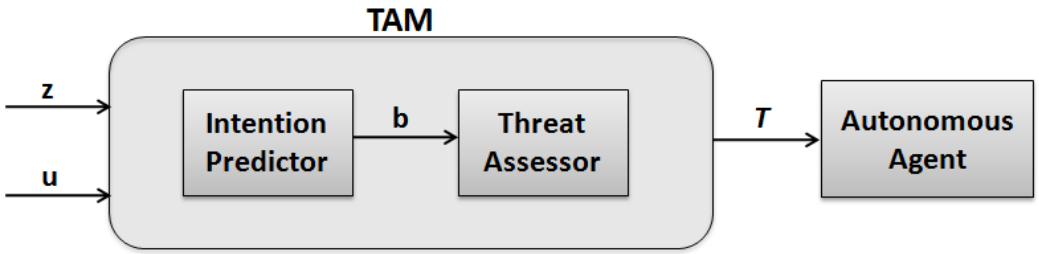


Figure 4-1: High-level architecture of the TAM. The inputs of the TAM are the measurement history z of the $\mathcal{OV}s$ and the candidate control sequence u generated by the $\mathcal{HV}$ planner. The IP computes the intention vector b for each $\mathcal{OV}$, used by the TA to identify the threat level $\mathcal{T}$.

this prediction into a set of potential paths for the $\mathcal{OV}$, and returns a threat value that a candidate trajectory $u(t)$ (or equivalently, $s(t)$), $t \in [t_0, t_f]$ for $\mathcal{HV}$ incurs in the vicinity of these potential paths. As a result, the planner generates paths that are threat-aware by construction, thus improving the vehicle's reactive behavior.

Several distinct representations have been proposed to model intentions for humans and autonomous systems; here we adopt the Ecological Recogniser architecture introduced by [95]. Using the language of [95], the $\mathcal{HV}$ is the recognising agent, while each $\mathcal{OV}$ is an intending agent. We follow the abstract definition that these intentions are directly responsible for the actions executed by the intending agents. Intentions may be desired plans (e.g., following a straight path) or other high level intentional states (e.g., not following the rules of the road). Let the M-vector $\boldsymbol{b}_i$ denote the intention vector of $\mathcal{OV}_i$; the j^{th} entry b_i^j corresponds to the belief $\mathcal{OV}_i$ is operating under the jth intention. The cost functional (4.7) is then modified to embed the intention information,

$$J = L + w \cdot \sup_i \sum_j b_i^j \mathcal{T}_{ij}, \tag{4.9}$$

where $\mathcal{T}_{ij}$ is the threat value of the trajectories followed by $\mathcal{OV}_i$ as a realization of its j^{th} intention; note that $i \in \{1 \dots N\}$ and $j \in \{1 \dots M\}$.

Agent Model

Each agent is modeled using the standard bicycle model

$$\begin{aligned} \dot{x} &= v\cos(\theta), \quad & \dot{y} &= v\sin(\theta), \\ \dot{\theta} &= \frac{v}{L}\tan(\delta), \quad & \dot{v} &= a, \end{aligned} \tag{4.10}$$

where (x, y) is the rear axle position, v is the forward speed, θ is the heading, L is the wheelbase, a is the forward acceleration, and δ is the steering angle (positive counter-clockwise). The state of the vehicle is $s = (x, y, \theta, v) \in \mathcal{S}$, while the input is $u = (\delta, a) \in \mathcal{U}$, including the constraints $a_{\min} \leq a \leq a_{\max}$ and $|\delta| \leq \delta_{\max}$. Each admissible control sequence $u : [0, t_f] \mapsto \mathcal{U}$ is piecewise constant.

In the work that follows, we make several assumptions to constrain the focus on the uncertainty in the prediction of obstacles or other agents. We assume that the current states and models of the $\mathcal{HV}$ and the $\mathcal{OV}s$ are perfectly known by the $\mathcal{HV}$ within a detection radius (Section 4.1.5), and that all sensor measurements are noise-free. Additionally, the path planner is provided with a complete map of the environment *a priori*, excluding any dynamic obstacles or agents.

4.1.4 Developed Algorithms

This section describes the implementation of the IP and TA components of the TAM and the resulting threat-aware path planner, using closed-loop rapidly exploring random trees (CL-RRT), into which they are embedded.

Intention Predictor (IP)

The Ecological Recognizer architecture [95] that the IP adopts has two key components: a pattern matcher and a reasoning module. The pattern matcher is a classifier that is trained offline to recognise different possible intentions by observing agent states, and operates online by giving a continuous estimate of the intentions over time. The reasoning module filters these estimates of intentions, along with knowledge of previous encounters with the agent, to give a final estimate of the intention

vector $\boldsymbol{b}$.

Here we are interested in estimating the intentions of human vehicles in urban environments, specifically near intersections. This Ecological Recognizer implementation is based on an approach previously developed and demonstrated in simulation [96] for classifying agent intentions using support vector machines (SVM) and Bayesian filtering (BF). The human driver intention classification problem is very complex, due to the various nuances of human behaviors. SVM is suitable for the IP because it has been shown to be both robust and efficient for classification problems [54], such as lane-change detection [51].

The IP design consists of an SVM combined with a Bayesian filter that uses the SVM outputs over a specific time period to compute the intention vector $\boldsymbol{b}$. SVM-BF also includes a threshold detector, such that the final vector $\boldsymbol{b}$ is a unit basis vector specifying the most likely intention according to the threshold value [96]. Based on experimenting with different combinations of kernel functions and features, the best results were obtained using the Gaussian radial basis function and combining the following three features: 1) the *relative distance* between the $\mathcal{OV}_i$ and the entrance of the intersection it is approaching, 2) the *speed* of the $\mathcal{OV}_i$, and 3) the *longitudinal acceleration* of the $\mathcal{OV}_i$. Note that the SVM-BF algorithm is only activated when the distance between the $\mathcal{OV}_i$ and the $\mathcal{HV}$ is within some detection radius, and both vehicles are approaching the same intersection.

Threat Assessor (TA)

The threat-aware CL-RRT planner (Algorithm 7) incorporates the computation of a threat value for each candidate trajectory it generates, given knowledge of the current $\mathcal{OV}$ states. We propose an approach, detailed in Algorithm 4, that combines a fast sampling-based reachability method with intention prediction information provided by the IP to efficiently estimate the threat level. The threat assessor has a finite time horizon, limiting the $\mathcal{HV}$ lookahead horizon for possible future $\mathcal{OV}$ paths, in order to focus computation on imminent threats. The choice of horizon length is domain-specific, but should allow the $\mathcal{HV}$ sufficient time to react in a dynamic environment.

Algorithm 4 Intention-based Threat Assessment

1: $\mathcal{T}_i \leftarrow 0 \quad \forall i$
2: **for** each agent $\mathcal{OV}_i$ **do**
3: *Reach-Set* $\leftarrow$ Compute-Intention-Reachability ($\boldsymbol{b}_i$)
4: **for** each intention e_j with probability b_i^j **do**
5: $\mathcal{T}_i \leftarrow \mathcal{T}_i + b_i^j \cdot$ Compute-Threat(*Reach-Set*,$\mathcal{OV}_i, e_j$)
6: **end for**
7: **end for**
8: **return** $\max_{i=1\ldots N} \mathcal{T}_i$

Algorithm 5 Compute-Threat (*Reach-Set*, $\mathcal{OV}$, e)

1: **for** each path $path_k$ in Reach-Set ending in region of intention e **do**
2: $t_k \leftarrow$ compute earliest time of collision of $path_k$ with $\mathcal{HV}$ within time horizon T_h
3: $\mathcal{T}_k \leftarrow \frac{1}{t_k}$
4: **end for**
5: **return** $\max_k \mathcal{T}_k$

The Threat-Assessor algorithm (Algorithm 4) begins by calling the Compute-Intention-Reachability (CIR) subroutine, discussed below, to create the reachability set for each $\mathcal{OV}_i$. The resulting sets are biased based on the perceived intentions of the $\mathcal{OV}s$. For each intention e_j of each $\mathcal{OV}_i$, Algorithm 5 is called to compute the threat incurred by the $\mathcal{HV}$ trajectory in the region reached by the paths corresponding to e_j. Note that the earliest time to collision is converted into a threat value using (5.3) (line 3 of Algorithm 5). This value is weighted by the probability b_i^j provided by the IP. Finally, the threat is computed as the maximum value of all threats created by each $\mathcal{OV}_i$ (line 8 of Algorithm 4).

The CIR algorithm (sometimes referred to as the RRT-Reach algorithm) is also based on the CL-RRT algorithm, but uses the resulting tree only for simulation of the different possible paths of a $\mathcal{OV}$ (which is not controlled by the operator). No best path is selected, but rather the entire RRT-Reach tree is biased towards regions corresponding to the learned intentions of the $\mathcal{OV}s$. This biasing is achieved by devoting a portion of the tree samples to regions corresponding to the intention e_j, with remaining samples being taken uniformly throughout the environment. The algorithm also includes time-parametrization extensions (introduced in [30, 35]) that

Algorithm 6 Compute-Intention-Reachability

```
1: Measure current vehicle state and environment
2: repeat
3:    Take biased sample for input to controller using bias from intention probability
      vector b
4:    repeat
5:       Update time range in time heuristics
6:       Find list of nearest neighbors using time range
7:       Sort list using distance heuristics
8:       for each sorted node do
9:          Call propagation function
10:         if propagated portion is collision free then
11:            Add sample to Tree break
12:         end if
13:      end for
14:   until timestamp reaches time horizon and no collision free portion was found
15: until time limit for growing tree is reached
16: return Tree
```

tailor the RRT algorithm to the efficient computation of intended paths by the $\mathcal{OV}s$.

- First, since we are interested in both approximating the vehicle's fixed-horizon reachability set and checking for collision between moving vehicles, a "timestamp" has been explicitly added to the state of the vehicle to track the time along each generated trajectory. While propagating, if the timestamp reaches the time horizon t_f, where $t_f = t_0 + T_h$, the propagation is interrupted and the current portion of the trajectory is checked for feasibility. Also, when searching for nearest neighbors, the algorithm skips any node with a timestamp already equal to t_f.

- Second, a time-based heuristic is introduced in the nearest neighbor selection; only neighbors with a timestamp lying in a specified time range are eligible to be considered in the nearest neighbor calculation. This time range is initialized to $[t_{\text{root}}, t_{\text{root}} + t_{\text{increment}}]$, and the k-nearest neighbors inside this time range are considered for feasibility check. If none of them leads to the creation of a feasible path, the time range is increased to $[t_{\text{root}} + t_{\text{increment}}, t_{\text{root}} + 2 \times t_{\text{increment}}]$, and so on, until a feasible path is found, or the time range reaches t_f, in which

case the sample is ignored, and a new sample is taken. Simulation results have suggested that the use of such heuristics can result in better approximation of the natural expected paths of the $\mathcal{OV}s$.

- Finally, we note that RRT-Reach does not include a completeness guarantee on the reachability set; there may be some feasible trajectories which are not included when work on constructing the reachability set is completed. However, the problem of computing the full reachability set in real-time is computationally intensive when subject to complex dynamics and complex, dynamic environments. The RRT-Reach algorithm is designed to rapidly approximate the reachability set and improve the approximation with more available time, regardless of the current problem complexity.

Threat-Aware CL-RRT Planner

In this section, we present a global threat-aware planner which builds on the CL-RRT implementation developed for MIT's Talos vehicle in the 2007 DGC competition [31]. The CL-RRT algorithm extends the rapidly-exploring random tree (RRT) algorithm [84, 97], which grows a tree of dynamically feasible trajectories by randomly sampling points toward which the tree is extended. The CL-RRT algorithm adds a path-tracking control loop in the vehicle prediction model, such that sampling takes place in the reference input space rather than in the vehicle input space. The algorithm thus maintains the exploration bias of traditional RRT algorithms, while allowing for generation of smooth trajectories more efficiently. Furthermore, because the RRT algorithm is sampling-based, the quality of the planning tree and resulting paths are scalable with the available computational resources. Please refer to Ref. [31] for a detailed description of the CL-RRT planner, and Ref. [14] for an explanation of the planner integration in the Talos system architecture.

The proposed threat-aware planner is shown in Algorithm 7. The main addition is the threat computation in lines 6 and 10. This computation calls Algorithm 4 to efficiently calculate the threat of colliding with the possible paths of the other $\mathcal{OV}s$

(Section 5.4.2). Whereas the original CL-RRT heuristic identifies nearest nodes based on path duration (in the Dubins sense), we extend this heuristic to also include the threat value (line 6). In line 10, the threat along the edge to a new node is computed and stored at the new node. The best path is then chosen in accordance with the cost defined in (4.9) (line 15 of Algorithm 7), which includes both the total time to the goal and the threat along the trajectory. These modifications embed the threat computation in the CL-RRT trajectory generation, leading to the selection of safer paths which are threat-aware by construction.

Another related addition is the use of threat propagation logic. First, the root node is initialized to have zero threat. When adding a new node, if the path from the selected node (initially the root) to the new node incurs a non-zero threat (computed using Algorithm 4), then an intermediate node is created at the location of the earliest collision, and both the intermediate node and the new node inherit the computed threat value. Note that if a node with a non-zero threat value is chosen to be expanded, the new node inherits that value and no new threat computation is required. The rationale is that the threat value does not decrease along an edge connecting a node with non-zero threat to its child (recall (5.3)). A useful property of the threat heuristic is that it is admissible: it underestimates the "true" cost. This is due to the sampling-based nature of the CIR algorithm, which underapproximates the earliest time of collision, as well as the assumption that children nodes inherit the threat value of their parent if the value is non-zero. Before choosing a best path, the threat values of the tree are updated from the bottom up (i.e., starting from the leafs) by reassigning each parent's threat value to be the minimum of the threat of its children, and so on, until the root is reached.

4.1.5 Experimental Results

This section presents experimental results which validate the effectiveness of the threat-aware CL-RRT algorithm, augmented with the TAM, in enabling an autonomous vehicle to successfully identify and avoid an errant human-driven vehicle under real-world uncertainty. These results focus on the challenging task of avoiding collisions

Algorithm 7 Threat-Aware CL-RRT

```
1: repeat
2:    Measure current vehicle states and environment
3:    Propagate states by computation time limit
4:    repeat
5:       Generate sample for the input to controller
6:       Sort nodes in tree using threat and time heuristics
7:       for each sorted node do
8:          Form controller input by drawing line from node to sample, then propagate

9:          if propagated portion is collision free then
10:            Compute threat of propagated portion to tree break
11:         end if
12:      end for
13:   until time limit is reached
14:   Choose best path and repropagate from current states
15:   if repropagated trajectory is infeasible then
16:      Remove infeasible portion from tree and go to line 15
17:   end if
18:   Send best path to controller
19: until vehicle reaches target
```

at intersections, in particular with errant drivers who do not observe a stop sign and enter the intersection out of turn.

First, a brief overview of the experimental infrastructure is provided. Next, we present a subset of hardware results showing the interaction between an errant human driver and an autonomous vehicle at intersections; additional experiments can be viewed at http://hdl.handle.net/1721.1/64737 [98] or at http://acl.mit.edu/IROS10TAM.mp4.

Hardware

Hardware demonstrations were performed in the Real-time indoor Autonomous Vehicle test ENvironment (RAVEN) [89], a testbed which uses motion-capture cameras to provide high-fidelity vehicle state data. We have constructed a representative road network within the RAVEN testbed, including many intersection types and road signs, to emulate a realistic driving environment (Figure 4-2).

All vehicles in this experiment use the iRobot Create platform [87], a skid-steered

vehicle, modified with a software wrapper to emulate traditional automotive steering (5.7). Each experiment consists of 1 or 2 autonomous vehicles and 1 human-driven vehicle. The autonomous vehicles are controlled through an off-board wrapper which receives waypoints from the threat-aware CL-RRT software (See following section) and steers using pure pursuit [99]. The human-driven vehicle is controlled through a wireless steering wheel [100], including acceleration and brake pedals. A human driver may steer their vehicle through simple visual inspection, or via an onboard dual-camera (Figure 4-2) interface which emulates the first-person driving perspective (Figure 5-4). This interface includes a real-time "GPS" display, which guides the driver through an intended sequence of waypoints.

Software

The planning software consists of two primary modules, discussed below. The navigator module accesses an abstract representation of the road network and selects sequences of waypoints, yielding the current goal location for the vehicle. The CL-RRT motion planning module then uses local knowledge of the environment (e.g., obstacles, terrain, and other agents), including the TAM, to plan dynamically feasible paths to this goal.

The navigator module leverages a sparse representation of the RAVEN road network (Figure 4-4), constructed using the Route Network Data File (RNDF) specification from the 2007 DGC [101]. The RNDF provides a sparse representation of the traversible road network, including the location, size, and connectivity of lane segments and "zones" (e.g., parking lots). Intersections in this road network are defined implicitly via the exit-entrance connections between lane/zone endpoints. The diagram at bottom-right in Figure 4-3 shows the RNDF for our RAVEN road network, constructed from a simple .txt file. Intended as a sparse representation of the traversible road network, the RNDF embeds details about road segments in the form of lane count, individual lane width, and interlane divider types.

As the RNDF represents the traversability graph, an A^* implementation is used to select the shortest-distance waypoint path in the graph between the current and

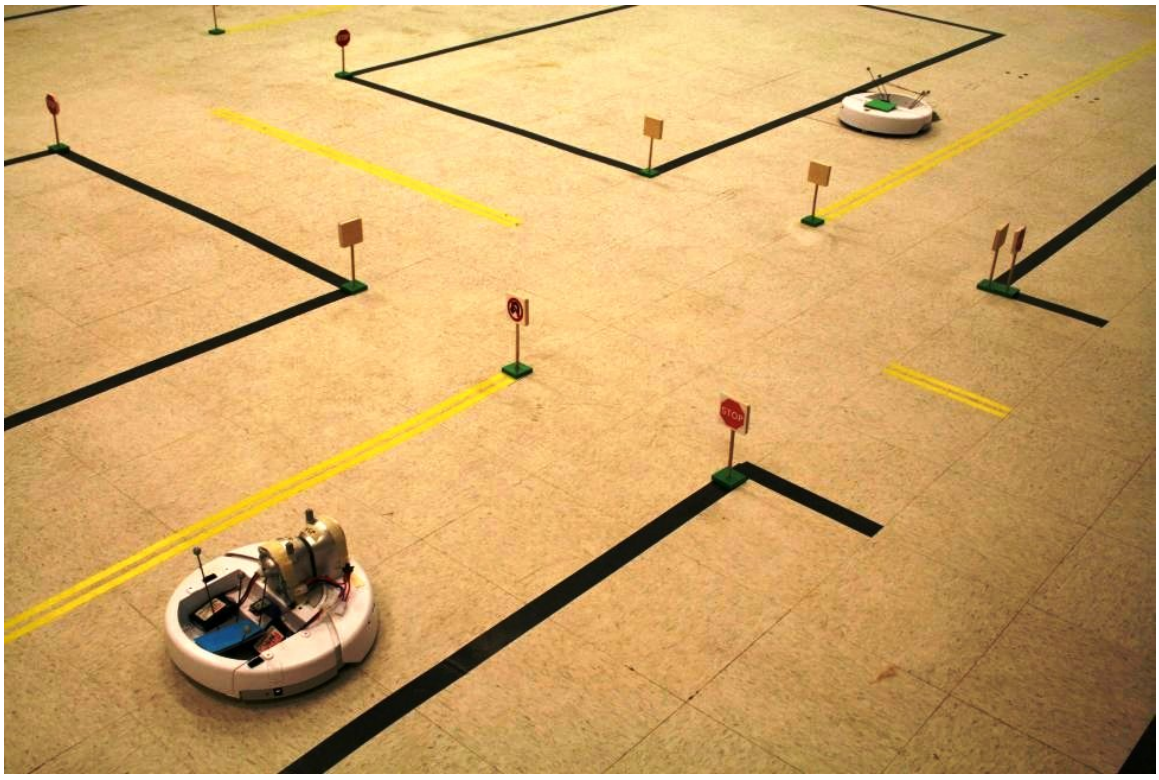


Figure 4-2: The road network used for experiments. In this image, both a human-driven vehicle (front-left) and autonomous vehicle (back-right) are approaching a four-way, stop-sign intersection.

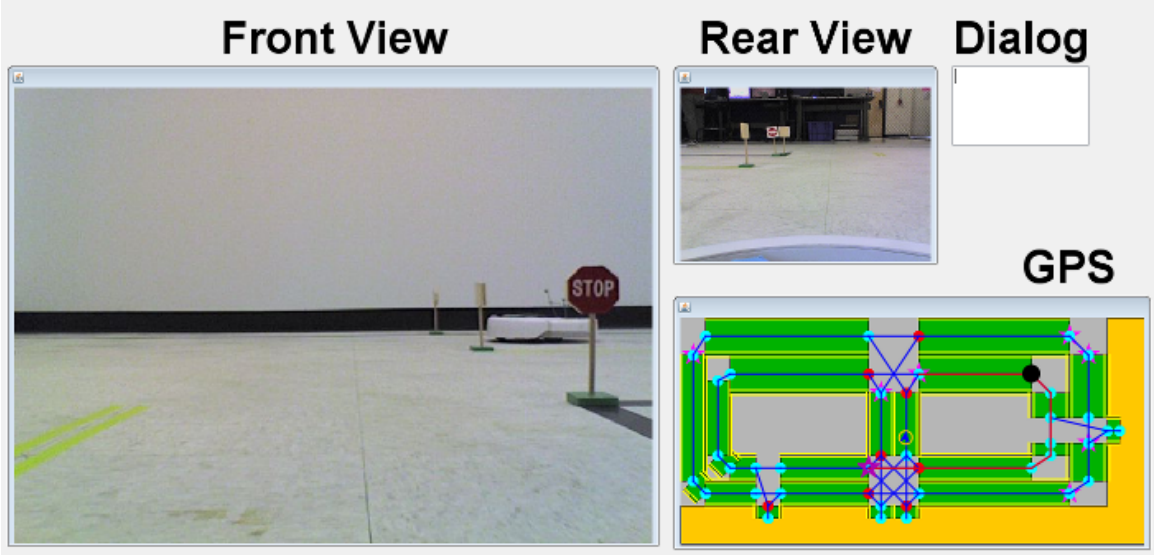


Figure 4-3: Human interface for the human-driven vehicle, including real-time navigation and visual feedback.

desired waypoints, respecting lane directionality constraints [14]. As the vehicle (whether human or autonomous) approaches each intermediate or desired waypoint, the navigator selects the next waypoint on the list as the new target. An arbitrary list of desired waypoints is provided for each vehicle; the ultimate objective is to observe the vehicles' interactions at intersections. The selection of this desired waypoint need not be structured; for example, the vehicle could randomly sample a waypoint in the network, plan a route, and repeat the process upon successfully arriving. The navigator may also access a Mission Data File (MDF), another adaptation from the DGC. The MDF specifies an ordered list of checkpoints the vehicle must visit.

The threat-aware CL-RRT algorithm (Algorithm 7) is implemented as an offboard, multi-threaded Java application, designed to support arbitrary vehicle and environment models in both simulation and hardware. The CIR algorithm [30] is embedded in this offboard planner as a separate thread which runs asynchronously for each $\mathcal{OV}$. Over fixed time intervals (2s), the algorithm grows a new reachability tree based on the current $\mathcal{OV}$ position in RAVEN. At the end of each time interval, the reachability tree is relayed to the host RRT thread, which then relays it to the TA algorithm (Section 5.4.2). Figure 4-4 shows the planner-view display of the developed application, while Figure 4-2 is a picture of the RAVEN testbed used for the experiments. Finally, Figure 4-3 shows the human interface for the human-driver vehicle that includes a real-time GP-like navigator and visual feedback (front and back views).

Results

The results below demonstrate the threat-aware closed-loop RRT algorithm in the context of stopping at intersections. In each experiment, one human-driven vehicle and one or two autonomous vehicles are navigating through the RAVEN road network simultaneously. The human driver may choose to not stop at an intersection and instead enter it out of turn, violating the rules of the road. We assume that all intersections have stop signs in each direction, but this assumption could be modified in the RNDF to allow other configurations. The human driver's intent is classified as either good or errant, based on whether or not they correctly approach the intersections.

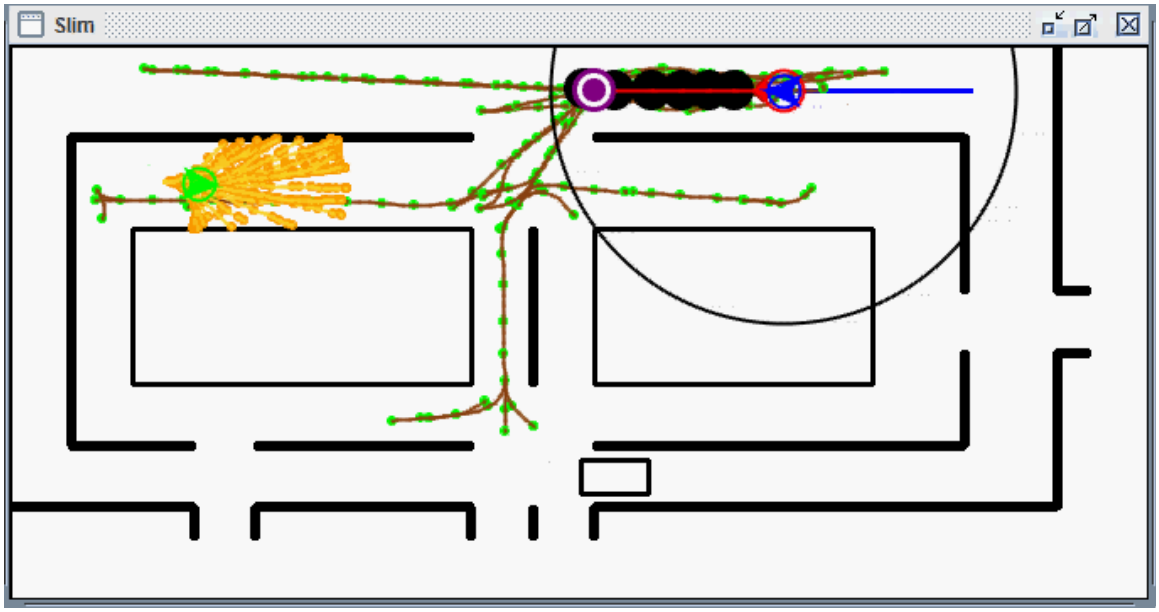
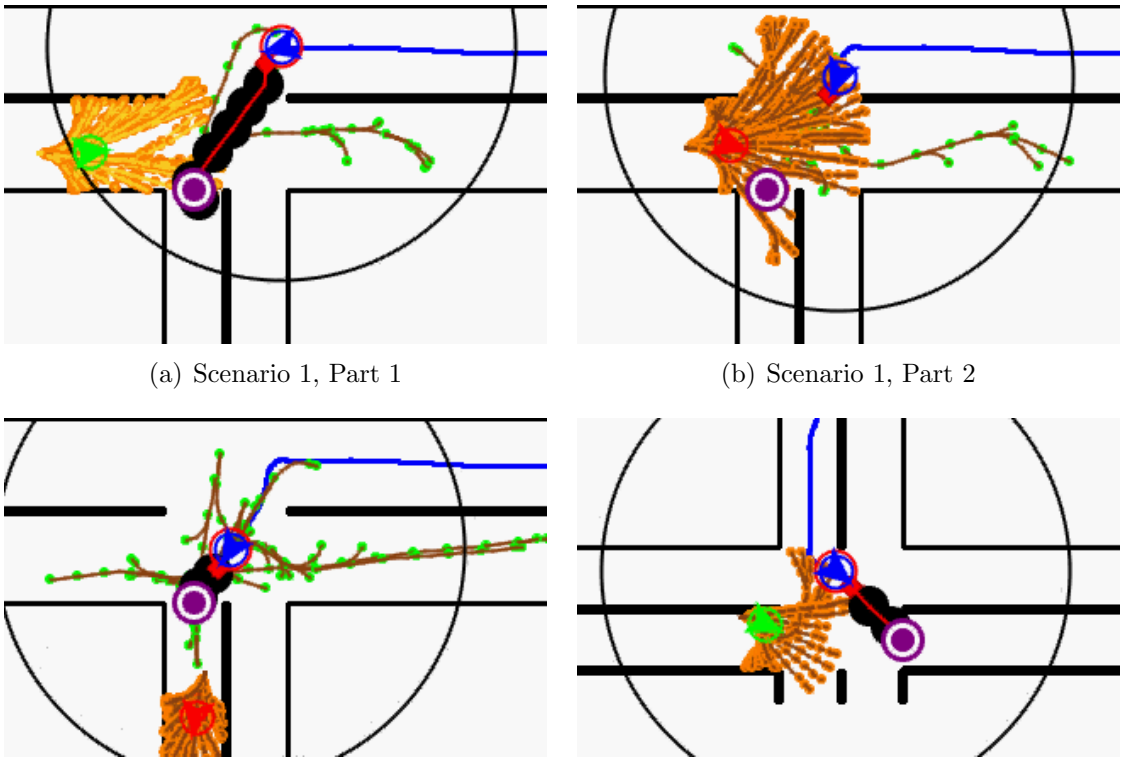


Figure 4-4: Planner-view of the RAVEN road network for the host vehicle (blue chevron, path history in blue). The host uses the CL-RRT tree (brown edges, green nodes) to identify the best known path (red edges, black nodes) to the current waypoint (purple). This path should avoid all obstacles (black boxes), the human-driven vehicle (green circled chevron) and, if active, their reachability tree (yellow edges, orange nodes). The other autonomous vehicle is represented as the car-sized rectangle at bottom-center.

The objective of the autonomous vehicle(s) is then to not only avoid all collisions, but also minimize the risk of collision with an errant human driver in intersections. In the experiments, the time horizon of the RRT-Reach is 6 s. The parameters in the agent models are: $a_{\min} = -0.7$ m/s^2, $a_{\max} = 0.4$ m/s^2, and $\delta_{\max} = 0.6$ rad. The maximum velocity of the vehicles is also set to 0.4 m/s to increase the interaction time between the vehicles at intersection encounters.

The human-driven vehicle is perceived to be a threat if three conditions are satisfied: (1) the IP module has classified the drivers' behavior as errant; (2) the errant driver is within the host vehicle's detection radius (Figure 4-4); and (3) both vehicles are approaching the same intersection. In this case, the autonomous vehicle is to avoid that vehicle's reachability tree constructed by the CIR algorithm; otherwise the reachability tree is not an active constraint.

Over the course of this experiment set, 20 intersection encounters occurred: 10



(a) Scenario 1, Part 1

(b) Scenario 1, Part 2

(c) Scenario 1, Part 3

(d) Scenario 2

Figure 4-5: Zoomed-in display view of two interactions between an autonomous vehicle (detection radius in black) and a human-driven vehicle in an intersection. Figures (a)-(c) show snapshots of a scenario in which the human-driven vehicle is classified as errant; in Figure (d), the human-driven vehicle is classified as good. Note that the reachability tree darkens if the driver is perceived to be a threat.

with one autonomous vehicle in the network, and 10 with two autonomous vehicles in the world. Of these encounters, only one resulted in a collision; in this case, the human-driven vehicle aggressively approached the autonomous vehicle, which had already stopped to avoid it.

Figure 4-5 demonstrates two particular scenarios in which the human-driven vehicle interacts with an autonomous vehicle at an intersection. In Scenario 1 (Figure 4-5(a)-(c)), both vehicles are attempting to enter the same lane, with the human-driven vehicle approaching from the left and making a right turn, and the autonomous vehicle approaching from the right and making a left turn. This scenario is known as a Left Turn Into Path (LTIP) [102]. The autonomous vehicle arrives first and, observing (through the IP) that the human driver is decelerating (therefore classified as a

good driver), plans a path through the intersection (Figure 4-5(a)). After entering the intersection, however, the IP module detects the human driver accelerating, as if it will enter the intersection out of turn. The IP module classifies this behavior as errant, and thus the TA module seeks to minimize the corresponding threat level $\mathcal{T}$. Though the autonomous vehicle cannot avoid the reachability tree's outer reaches, it does avoid the inner nodes (where the time to collision is lower) by quickly coming to a stop (Figure 4-5(b)). Once the errant driver is no longer a threat, the autonomous agent finishes crossing the intersection (Figure 4-5(c)). Figures 4-6 and 4-7 show snapshots of the sequence of events through a video camera view, a planner-view and a navigator-view of the RAVEN road network and the vehicles.

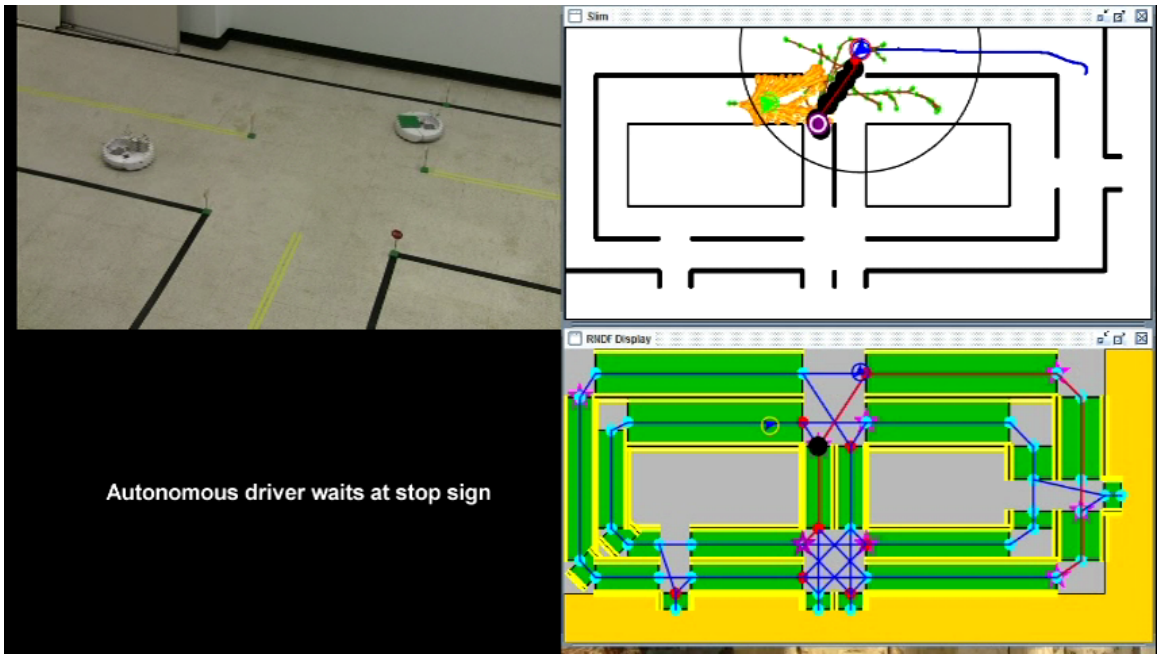
In Scenario 2 (Figure 4-5(d)), the autonomous driver is making a left turn, while the human-driven vehicle is crossing the intersection (from left to right in the figure). Here, the human-driven vehicle has correctly come to a stop at the intersection, and the IP module has classified it as a good driver. Unlike Figure 4-5(b), the autonomous driver then proceeds through the intersection, perceiving that the human driver is likely to obey the rules.

Another set of scenarios involves two autonomous vehicles navigating in the road network vehicles using the threat-aware CL-RRT in the presence of a possibly errant human-driven vehicle. The behavior of the human-driven vehicle and the objective of the autonomous vehicles are identical to the one-autonomous experiment case. Furthermore, for simplicity, the autonomous vehicles follow the rules of the road, and have safe intents, so they did not have to run the threat assessment on each other. They avoid collision (if they are on the same lane) by not coming to within 0.5 car-lengths of each other. Figure 4-8(a) shows a snapshot of the video at the beginning of the two-autonomous vehicle experiment, while Figure 4-8(b) captures a video shot of the three vehicles.

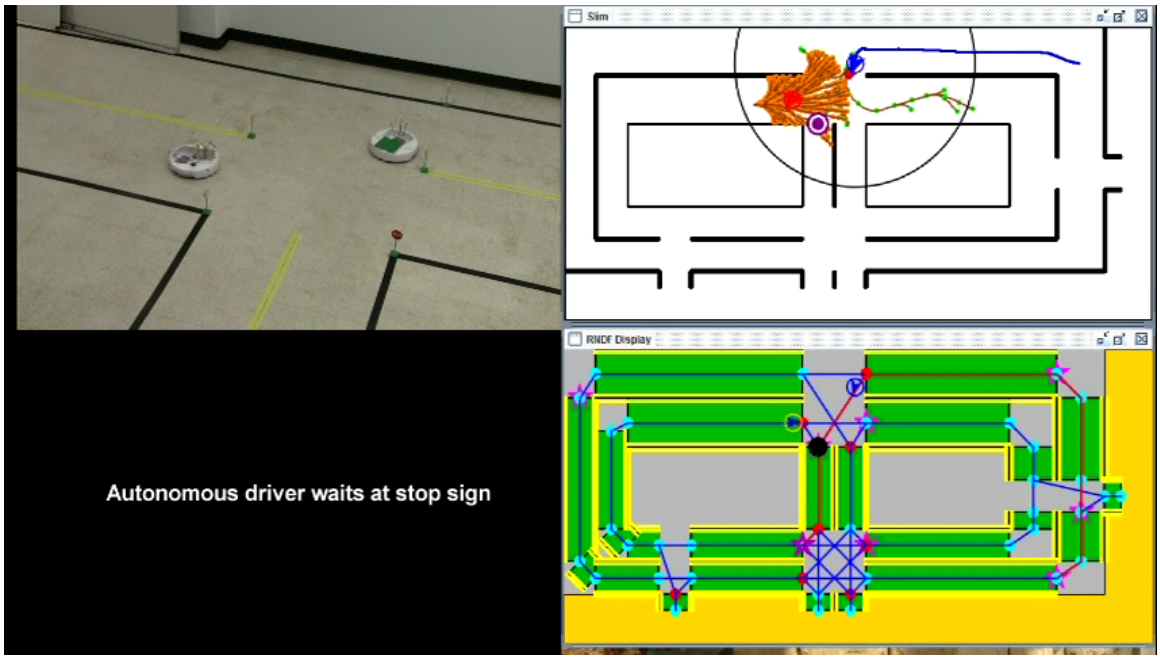
Figure 4-9 demonstrates a particular scenario involving one of the autonomous vehicles and the human driven-vehicle. It is known as the Left Turn Across Path / Opposite Direction (LTAP/OD). It is one of the most dangerous and fatal intersection encounter [23, 102]. The autonomous vehicle is approaching from the right, and

arrives first to the intersection. Before proceeding into the intersection, it detects (through the IP) that the human-driven vehicle approaching from the left is not decelerating (to stop at the intersection), and therefore plans a stopping maneuver (through the TA) to minimize the threat level $\mathcal{T}$ (Figure 4-9(a)). Once the errant driver is no longer a threat, the autonomous agent finishes crossing the intersection (Figure 4-9(b)).

The described scenarios along with additional experiments can be viewed at http://hdl.handle.net/1721.1/64737 [98] or at http://acl.mit.edu/IROS10TAM.mp4.

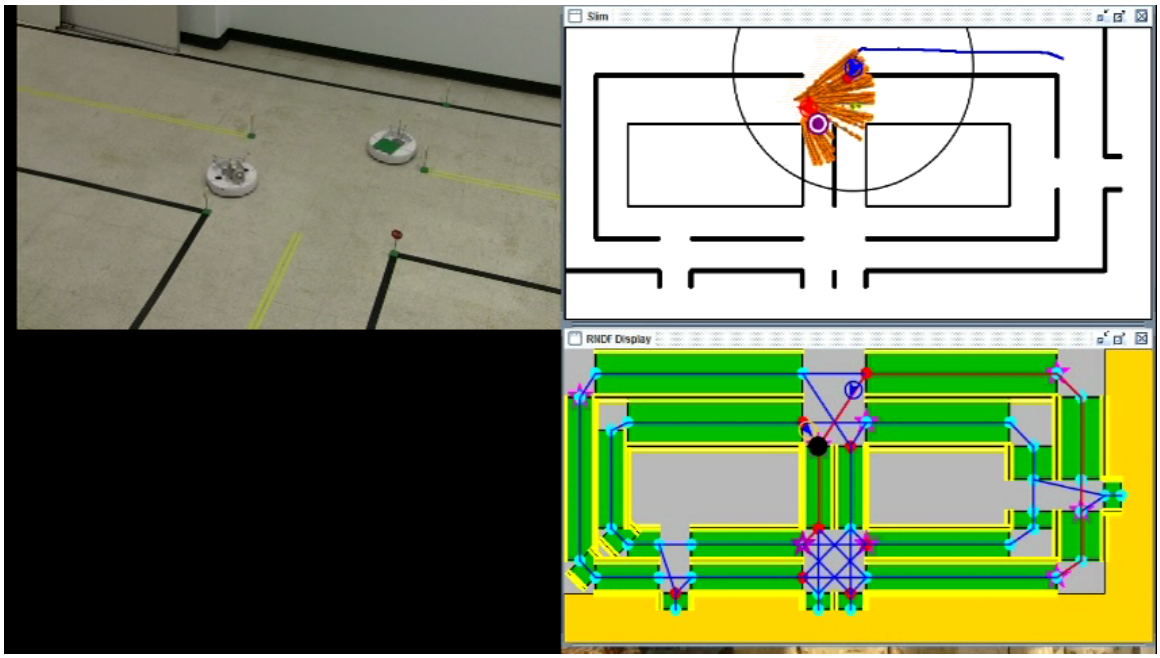


(a) Scenario 1, Part 1

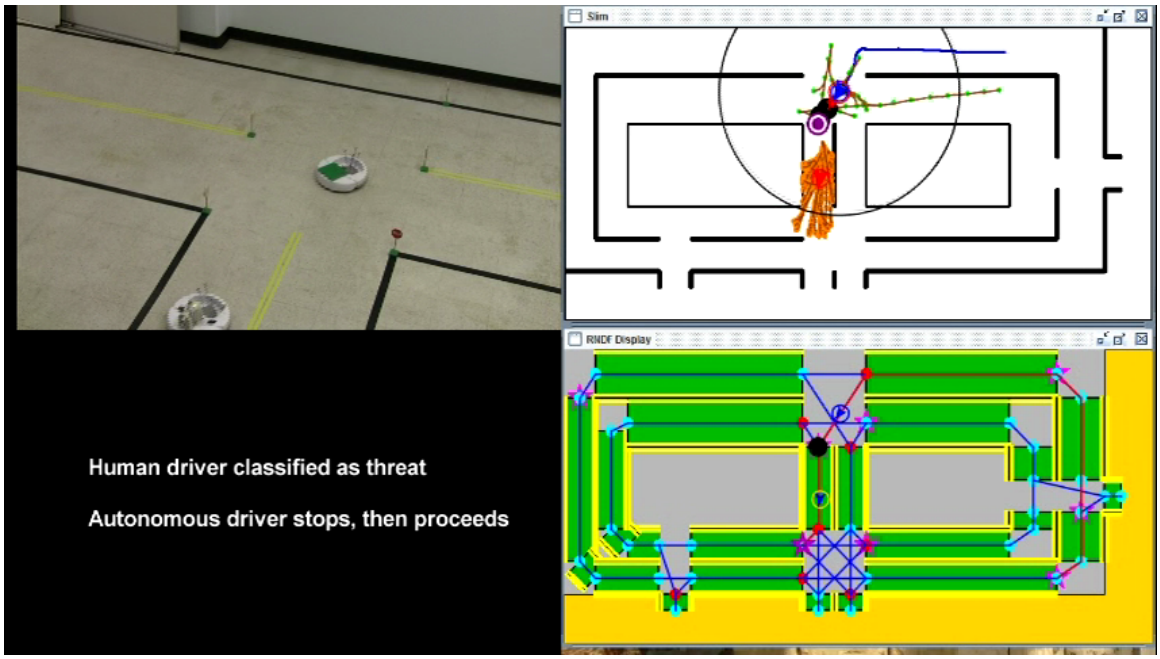


(b) Scenario 1, Part 2

Figure 4-6: RAVEN road network view of an interaction between an autonomous vehicle and a human-driven vehicle in an intersection. A video camera a view (upper left), a message box (lower left), a planner-view (upper right), and a navigator-view (lower right) illustrate the Left Turn into Path (LTIP) intersection encounter where the human-driven vehicle disobeyed the precedence rules, and the autonomous vehicle avoided the accident using the threat-aware CL-RRT planner.

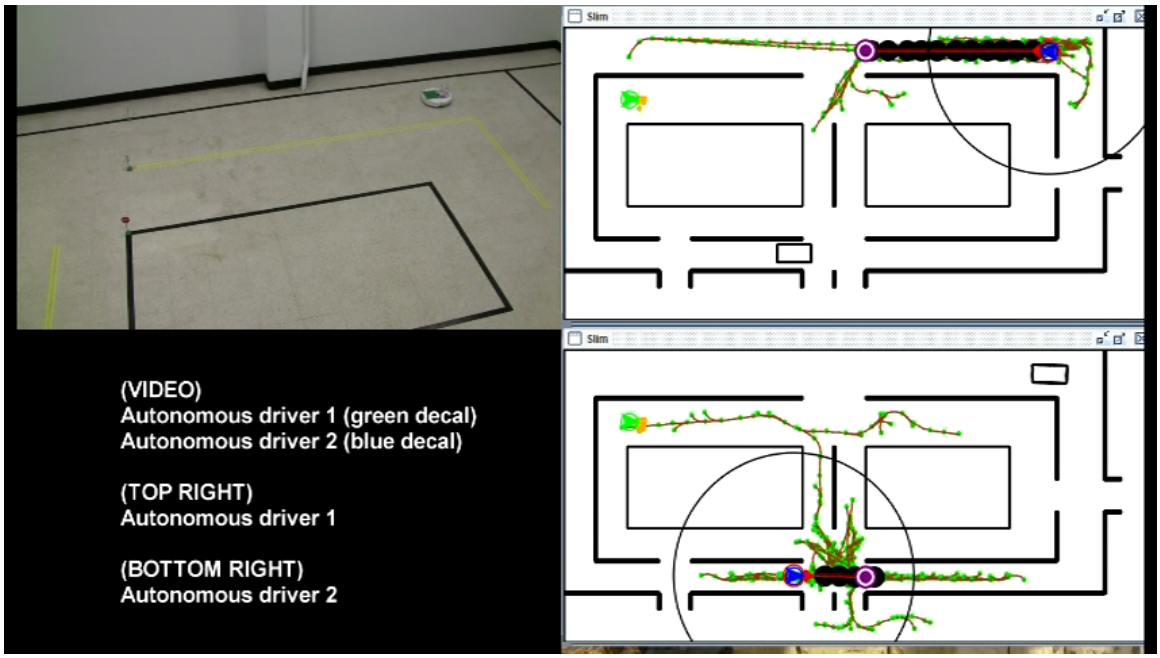


(a) Scenario 1, Part 3

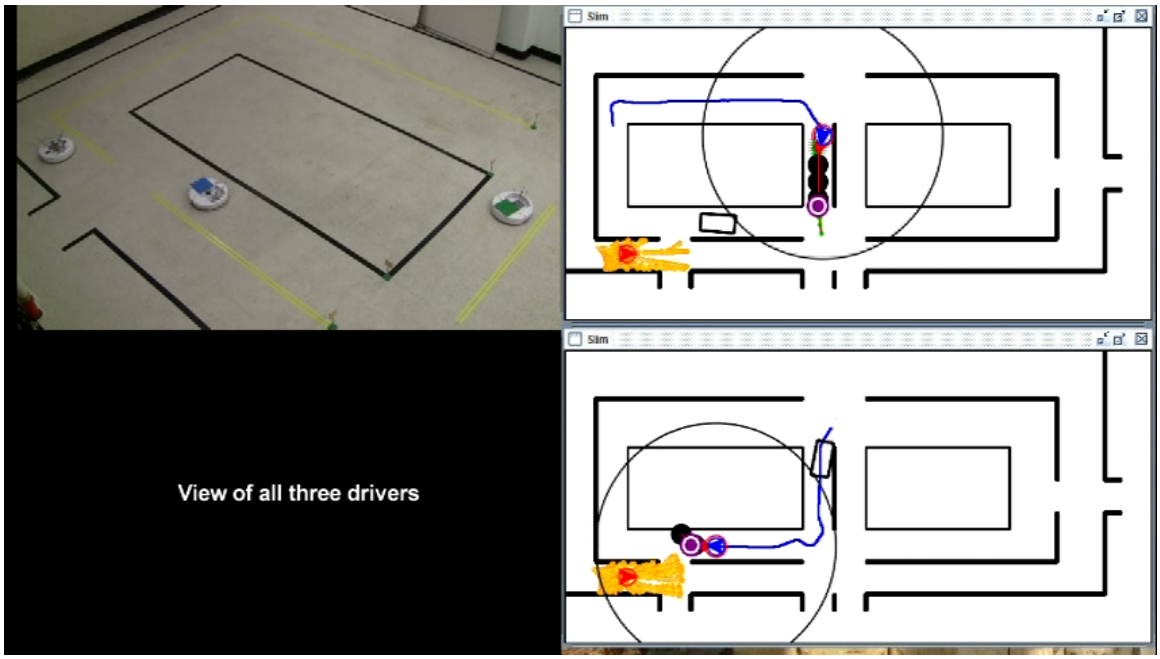


(b) Scenario 1, Part 4

Figure 4-7: RAVEN road network view of an interaction between an autonomous vehicle and a human-driven vehicle in an intersection. A video camera a view (upper left), a message box (lower left), a planner-view (upper right), and a navigator-view (lower right) illustrate the Left Turn into Path (LTIP) intersection encounter where the human-driven vehicle disobeyed the precedence rules, and the autonomous vehicle avoided the accident using the threat-aware CL-RRT planner.



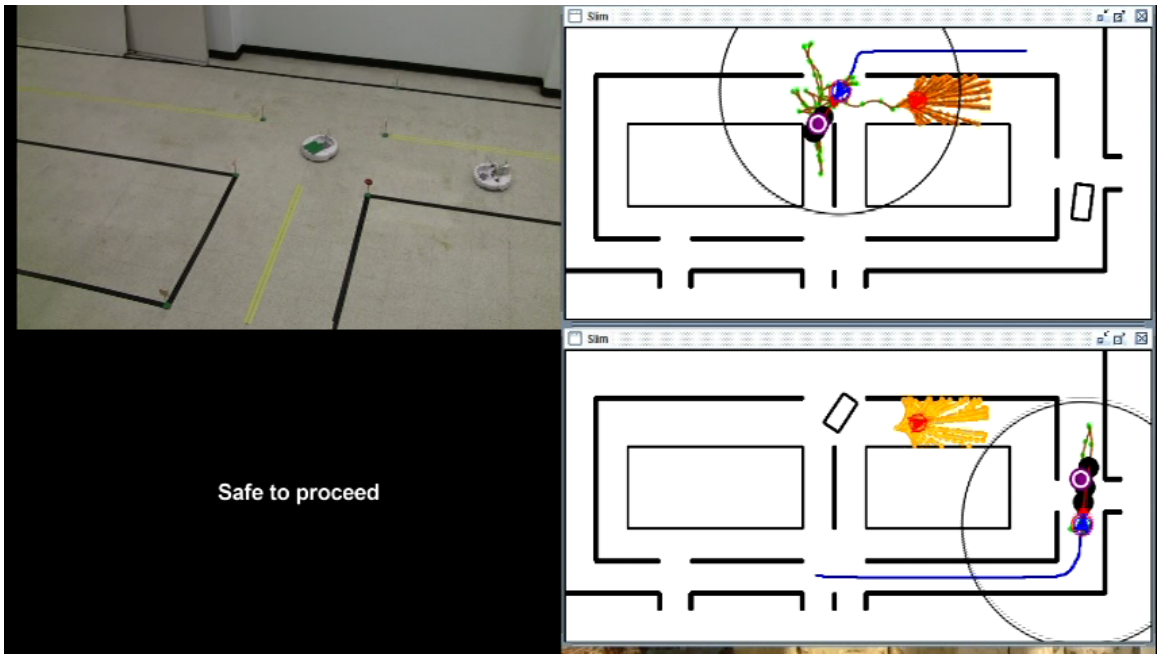
(a) Initial Configuration



(b) View of all three vehicles

Figure 4-8: RAVEN road network view of for the two autonomous and one human-driven vehicle experiment. A video camera a view (upper left), a message box (lower left), a planner-view for autonomous vehicle 1 (upper right), a planner-view for autonomous vehicle 2 (lower right). In each planner view, the other autonomous vehicle is represented as a car-sized rectangle, while the human-driven vehicle is shown as a green circle chevron.

(a) Part 1



(b) Part 2

Figure 4-9: RAVEN road network view of an interaction between an autonomous vehicle and a human-driven vehicle at an intersection in the three vehicle experiment. Autonomous vehicle 1 avoids the threat of the human-driven at an as an LTAP/OD encounter.

4.1.6 Conclusion

This section introduced a new approach for path planning in urban environments subject to uncertainty in the intentions of other vehicles. This work combines an intention predictor (IP) and a threat assessor (TA) to create a threat assessment module (TAM), embeddable in path planning algorithms, which runs efficiently and is suitable for real-time implementation. The IP uses high-level logic based on the Ecological Recogniser, while the TA uses sampling-based techniques to efficiently compute reachable future paths for surrounding vehicles. Using TAM applied to the CL-RRT planner, an autonomous vehicle can successfully identify and avoid an errant driver in realistic intersection scenarios, as demonstrated successfully through hardware results.

4.2 Probabilistically Safe Collision Avoidance

This section presents a real-time path planning algorithm which can guarantee probabilistic feasibility for autonomous robots subject to process noise and an uncertain environment, including dynamic obstacles with uncertain motion patterns [33, 103]. The algorithm incorporates chance-constrained rapidly-exploring random trees (CC-RRT), using chance constraints to explicitly incorporate probabilistic constraint satisfaction in the formulation while maintaining the computational benefits of sampling-based algorithms. The key contribution of the work is the integration of RR-GP path prediction algorithm (Chapter 3). With RR-GP embedded in the CC-RRT framework, theoretical guarantees can be demonstrated for linear systems, subject to Gaussian uncertainty though the extension to nonlinear systems is also considered. Simulation results show that the resulting approach can be used in real-time to efficiently and accurately execute safe paths.

4.2.1 Introduction

To operate safely in stochastic environments, it is crucial for agents to be able to plan in real time in the presence of uncertainty. Indeed, the stochasticity of such environments often precludes the guaranteed existence of safe, collision-free paths. Instead, this work considers *probabilistically safe planning*, in which paths must be able to satisfy all constraints with a user-mandated minimum probability.

A major challenge in utilizing trajectory prediction algorithms is addressing the multiple sources of uncertainty in the environment, often classified between environment sensing (ES) and environment predictability (EP) [22]. Under this partition, ES uncertainties are attributable to imperfect sensor measurements and/or incomplete knowledge of the environment, while EP uncertainties, the focus of this work, consider the typically limited knowledge of the future state of the environment. While probabilistic planning frameworks can readily admit dynamic obstacles, such objects are often very difficult to model in real-world domains. For example, for a car to reliably traverse a busy intersection, it must have some understanding of how other

vehicles typically cross that intersection. Even with the assumption of the existence of perfect sensors and complete knowledge of the current state environment, predicting long-term trajectories of other mobile agents remains a very difficult problem.

In Chapter 3, we surveyed several existing techniques for long-term trajectory prediction, and concluded that pattern-based approaches, despite some existing limitations, are typically the most suitable solution. There are two main techniques that fall into this category: a) discrete state-space, and b) clustering-based [21]. In discrete state-space techniques, the motion model is typically based on learned Markov chains, while clustering-based techniques group previously-observed trajectories into clusters which are each represented by a trajectory prototype.

Both classes of pattern-based approaches have been popular in solving long-term prediction problems for mobile agents [21, 82]. However, discrete state-space techniques can suffer from over-fitting or under-fitting problems due to space discretization issues, while this is not the case for clustering-based techniques. In our previous work [77, 78], we presented a Bayesian nonparametric approach to modeling motion patterns that is well-suited to modeling dynamic obstacles with unknown motion patterns. This nonparametric model, a mixture of Gaussian process (GP), generalizes well from small amounts of data and allows the model to capture complex trajectories as more data is seen. However, in practice, GPs suffer from two interconnected shortcomings: their high computational cost and their inability to embed static feasibility or vehicle dynamical constraints. To handle both of these problems simultaneously, recent work introduced the RR-GP algorithm, a trajectory prediction solution using Bayesian nonparametric reachability trees [29]. Its benefits on both prediction accuracy and computation time over the original GP algorithm make it well suited for real-time applications in prediction problems with poorly understood trajectory patterns. The algorithm presented in this section has similarities with [82], which uses Gaussian processes to model moving obstacles in an RRT path planner. However, unlike this work, [82] relies solely on Gaussian processes, which can lead to less precise prediction, especially when available data is sparse. RR-GP also embeds dynamic feasibility and prior knowledge of the environment, leading to better results

than GP-only approaches. Finally, the planner in [82] uses heuristics to assess the safety of generated paths, while the planner developed in this section uses a principled approach to achieve probabilistic safety.

This section presents a real-time path planning framework which guarantees probabilistic feasibility for autonomous agents subject to both process noise and an uncertain environment, particularly dynamic obstacles with uncertain motion patterns. The planning framework chosen for this work is chance-constrained RRTs (CC-RRT) [34], based on the chance constraint formulation of [104] for linear systems subject to Gaussian uncertainty. CC-RRT uses chance constraints to evaluate the risk of constraint violation at each timestep, embedding uncertainty directly within the planner. This approach maintains the benefits of sampling-based algorithms, particularly the fast identification of feasible solutions for complex motion planning problems. While several apparoaches have been previously proposed for path planning with probabilistic constraints, our approach does not require the use of MILP/SOCP optimizations [104, 105] or particle-based approximations [106–108], each of which can severely limit real-time applicability.

The CC-RRT algorithm has been demonstrated to effectively and efficiently guarantee probabilistic feasibility, even in the presence of dynamic obstacles, as long as the future state distributions are known [34]. This is effectively integrated here with the RR-GP algorithm, which can provide a likelihood and state distribution for each possible behavior of a dynamic obstacle at each future timestep. The CC-RRT formulation is revisited for the case of dynamic obstacles modelled via RR-GP, showing that probabilistic feasibility can still be guaranteed. Simulation results demonstrate the effectiveness of this approach in enabling agents to avoid dynamic threats with high likelihood.

4.2.2 Problem Statement

Consider a discrete-time linear time-invariant (LTI) system with process noise,

$$x_{t+1} = Ax_t + Bu_t + w_t, \tag{4.11}$$

$$x_0 \sim \mathcal{N}(\hat{x}_0, P_{x_0}), \tag{4.12}$$

$$w_t \sim \mathcal{N}(0, P_{w_t}), \tag{4.13}$$

where $x_t \in \mathbb{R}^{n_x}$ is the state vector, $u_t \in \mathbb{R}^{n_u}$ is the input vector, and $w_t \in \mathbb{R}^{n_x}$ is a disturbance vector acting on the system; $\mathcal{N}(\hat{a}, P_a)$ represents a random variable whose probability distribution is Gaussian with mean $\hat{a}$ and covariance P_a. The i.i.d. random variables w_t are unknown at current and future time steps, but have the known probability distribution (4.13) ($P_{w_t} \equiv P_w \ \forall \ t$).

There are also constraints acting on the system state and input. These constraints are assumed to take the form

$$x_t \in \mathcal{X}_t \equiv \mathcal{X} - \mathcal{X}_{t1} - \cdots - \mathcal{X}_{tB}, \tag{4.14}$$

$$u_t \in \mathcal{U}, \tag{4.15}$$

where $\mathcal{X}, \mathcal{X}_{t1}, \ldots, \mathcal{X}_{tB} \subset \mathbb{R}^{n_x}$ are convex polyhedra, $\mathcal{U} \subset \mathbb{R}^{n_u}$, and the $-$ operator denotes set subtraction. The set $\mathcal{X}$ defines a set of time-invariant convex constraints acting on the state, while $\mathcal{X}_{t1}, \ldots, \mathcal{X}_{tB}$ represent B convex obstacles to be avoided. For each obstacle, the shape and orientation are assumed to be known, while the placement is uncertain. This is represented as

$$\mathcal{X}_{tj} = \mathcal{X}_j^0 + c_{tj}, \ \forall \ j \in \mathbb{Z}_{1,B}, \ \forall \ t, \tag{4.16}$$

$$c_{tj} \sim p(c_{tj}) \ \forall \ j \in \mathbb{Z}_{1,B}, \ \forall \ t, \tag{4.17}$$

where the $+$ operator denotes set translation and $\mathbb{Z}_{a,b}$ represents the set of integers between a and b inclusive. In this model, $\mathcal{X}_j^0 \subset \mathbb{R}^{n_x}$ is a convex polyhedron of known,

fixed shape, while $c_{tj} \in \mathbb{R}^{n_x}$ is a possibly time-varying translation, represented by the probability distribution $p(c_{tj})$. This can be used to represent dynamic obstacles whose future state distributions are known, as is done in Section 4.2.4.

The primary objective of the planning problem is to reach the goal region $\mathcal{X}_{\text{goal}} \subset \mathbb{R}^{n_x}$ in minimum time, while ensuring the constraints (4.14)-(4.15) are satisfied at each time step $t \in \{0, \ldots, t_{\text{goal}}\}$ *with probability of at least* p_{safe}. In practice, since there is uncertainty in the state, we assume it is sufficient for the distribution mean to reach the goal region $\mathcal{X}_{\text{goal}}$. A secondary objective may be to avoid some undesirable behaviors, such as proximity to constraint boundaries.

Note that we are assessing probabilistic feasibility of the constraints at each time step, rather than over the entire path. Because the uncertainty at each timestep is correlated, due to the dynamics (4.11)-(4.13), one *cannot* approximate the probability of path feasibility by assuming independence and multiplying the probabilities of feasibility at each timestep. Instead, assessment of path feasibility requires the evaluation of a complex nested integral, necessitating the use of approximate solutions even under the assumption of Gaussian uncertainty. While such approximations do exist [108, 109], they require significant computation for most problems of interest. For the applications being considered, real-time identification of feasible paths is paramount; thus in this work we focus on feasibility at each timestep, using the efficient assessment of risk afforded by CC-RRT [34]. Because the constraints are made more restrictive as the lower bound on probabilistic feasibility is increased, a positive correlation is expected between feasibility at each timestep and feasibility of the whole path; this is shown to be the case in Section 5.5.4.

Motion Pattern

We define a *motion pattern* as a mapping from locations to a distribution over velocities (trajectory derivatives).[1] Given an agent's current position (x_t, y_t) and a trajectory derivative $(\frac{\Delta x_t}{\Delta t}, \frac{\Delta y_t}{\Delta t})$, its predicted next position (x_{t+1}, y_{t+1}) is $(x_t + \frac{\Delta x_t}{\Delta t}\Delta t, y_t +$

[1]The choice of Δt determines the scales we can expect to predict an agent's next position well, making the trajectory derivative more useful than instantaneous velocity.

$\frac{\Delta y_t}{\Delta t}\Delta t$). Thus, modeling trajectory derivatives is equivalent to modeling trajectories. In addition to being blind to the lengths and discretizations of the trajectories, modeling motion patterns as flow fields rather than single paths also allows us to group trajectories sharing key characteristics: for example, a single motion pattern can capture all the paths that an agent might take from different starting points to a single ending location.

Mixtures of Motion Patterns

The finite mixture model[2] defines a distribution over the i^{th} observed trajectory t^i. This distribution is written as

$$p(t^i) = \sum_{j=1}^{M} p(b_j)p(t^i|b_j). \tag{4.18}$$

where b_j is motion pattern j and $p(b_j)$ is its prior probability. Since we are interested in vehicles traveling along a known road network we can assume the number of motion patterns, M, is known *a priori*.

Motion Model

The *motion model* is defined as the mixture of weighted motion patterns (Eq. 4.18). Each motion pattern is weighted by its probability and is modeled by a pair of Gaussian processes mapping (x, y) locations to distributions over trajectory derivatives $\frac{\Delta x}{\Delta t}$ and $\frac{\Delta y}{\Delta t}$. The motion model has been previously presented in [29, 78, 110] and reviewed in Chapter 3.

4.2.3 RR-GP Trajectory Prediction Algorithm

Section 4.2.2 outlined the approach of using GP mixtures to model mobility patterns. In practice, GPs suffer from two interconnected shortcomings: their high computational cost and their inability to embed static feasibility or vehicle dynamics

[2]Note that throughout the section a t with a superscript, such as t^i, refers to a trajectory and a t without a superscript is a time value.

constraints. Very dense training data may elevate this feasibility problem by capturing, in great detail, the environment configuration and physical limitations of the vehicle. Unfortunately, the computation time for predicting future trajectories using the resulting GPs would suffer significantly, rendering the motion model unusable for real-timeapplication.

To handle both of these problems simultaneously, we developed RR-GP, a trajectory prediction solution using Bayesian nonparametric reachability trees [29]. RR-GP augments RRT-Reach [32], a reachability based method which creates dense, feasible trajectories from sparse samples using the GP mixture model (See Section 3.4).

High Level Architecture

Figure 4-10 shows the high level architecture of the RR-GP solution approach. RR-GP is based on two main components: 1) an intent predictor based on the GP mixture model (Section 4.2.2) and 2) a trajectory generator based on the RRT-Reach algorithm. The Intention Predictor uses the history of sensor position measurements $(x_{0:t}, y_{0:t})$, along with a set of typical GP motion patterns b_j, $j \in \{1, \ldots, M\}$, to produce an intent distribution which is given to the trajectory generation component. Using the intent information, a dynamic model of the target vehicle, a map of the environment, and a sparse distribution of the future positions of the target vehicle, the trajectory generator uses closed-loop RRT (CL-RRT) [31] to grow a separate tree of smooth trajectories for each motion pattern, that embeds dynamical feasibility and collision avoidance to them. The resulting trees produce an improved probabilistic prediction of the future trajectories of the target vehicle. Note that to limit the scope of the problem to uncertainty in predictability, the sensors measurements are assumed to be noise-free despite our motion pattern representation being robust to noisy measurements.

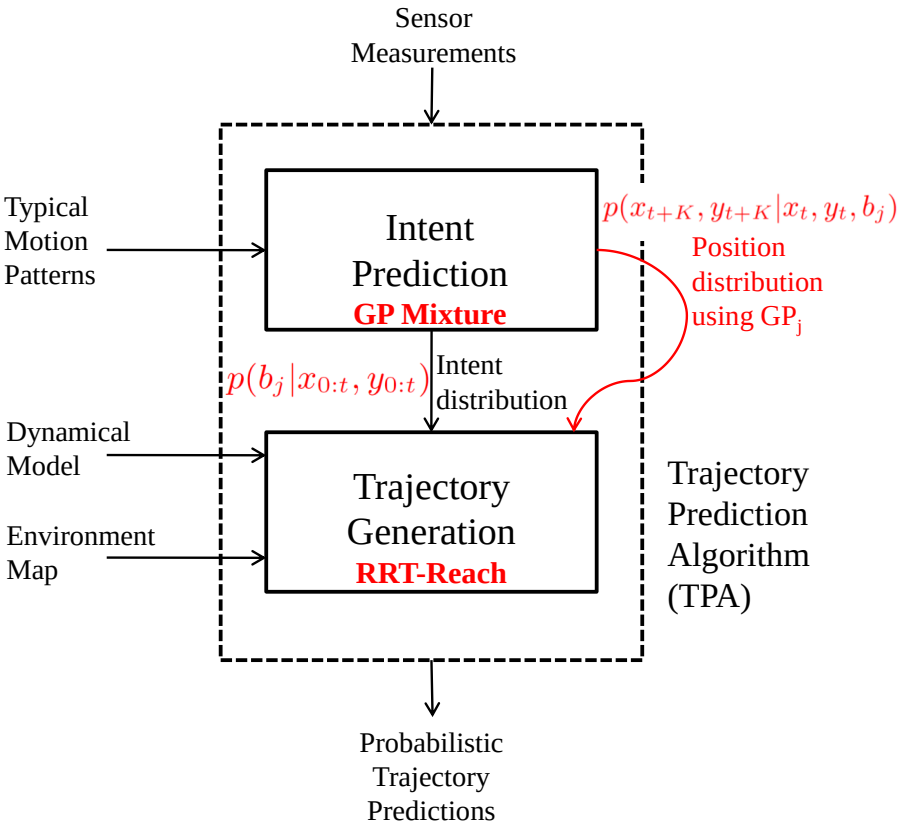


Figure 4-10: RR-GP High Level Architecture.

4.2.4 CC-RRT with Integrated RR-GP

This section integrates the model for uncertain, dynamic obstacles established in previous sections into a probabilistic planning framework, CC-RRT, to identify trajectories which can probabilistically avoid those obstacles. First, the CC-RRT formulation is reviewed under the assumption that each obstacle's uncertainty is modeled by a single Gaussian; the simple extension to the multi-Gaussian formulation yielded by RR-GP follows.

Review of Online CC-RRT [34]

In this section, the uncertainty of each obstacle is assumed to be represented by a single Gaussian:

$$c_{tj} \sim \mathcal{N}(\hat{c}_{jt}, P_{c_{jt}}) \; \forall \; j \in \mathbb{Z}_{1,B}, \; \forall \; t. \tag{4.19}$$

Given a sequence of inputs $u_0, \ldots, u_{N-1}$, under the assumptions of linear dynamics and Gaussian uncertainty, the distribution of the state x_t (represented as the random variable X_t) can be shown to be Gaussian [104]:

$$P(X_t|u_0, \ldots, u_{N-1}) \sim \mathcal{N}(\hat{x}_t, P_{x_t}) \ \forall\, t \in \mathbb{Z}_{0,N},$$

where N is some time step horizon. The mean $\hat{x}_t$ and covariance P_{x_t} can be updated implicitly using the relations

$$\hat{x}_{t+1} = A\hat{x}_t + Bu_t \ \forall\, t \in \mathbb{Z}_{0,N-1}, \tag{4.20}$$

$$P_{x_{t+1}} = AP_{x_t}A^T + P_w \ \forall\, t \in \mathbb{Z}_{0,N-1}. \tag{4.21}$$

To ensure that the probability of collision with any obstacle on a given time step does not exceed $\Delta \equiv 1 - p_{\text{safe}}$, it is sufficient to show that the probability of collision with each of the B obstacles at that time step does not exceed Δ/B. [104] The jth obstacle is represented through the conjunction of linear inequalities

$$\bigwedge_{i=1}^{n_j} a_{ij}^T x_t < a_{ij}^T c_{ijt} \ \forall\, t \in \mathbb{Z}_{0,t_f}, \tag{4.22}$$

where n_j is the number of constraints defining the jth obstacle, and c_{ijt} is a point nominally (i.e. $c_{jt} = \hat{c}_{jt}$) on the ith constraint at time step t; note that a_{ij} is not dependent on t, since the obstacle shape and orientation are fixed. To avoid all obstacles, the system must satisfy B disjunctions of constraints at each time step,

$$\bigvee_{i=1}^{n_j} a_{ij}^T x_t \geq a_{ij}^T c_{ijt} \ \forall\, j \in \mathbb{Z}_{1,B}, \ \forall\, t \in \mathbb{Z}_{0,N}. \tag{4.23}$$

For each obstacle – consider the jth one below – it is sufficient to not satisfy any one constraint in the conjunction (4.22). Thus, the probability of collision is *lower-bounded* by the probability of satisfying any single constraint:

$$P(\text{collision}) \leq P(a_{ij}^T X_t < a_{ij}^T c_{ijt}) \ \forall\, i \in \mathbb{Z}_{1,n_j}. \tag{4.24}$$

To prove the probability of collision with the jth obstacle does not exceed Δ/B, it is thus sufficient to show that

$$\bigvee_{i=1}^{n_j} P(a_{ij}^T X_t < a_{ij}^T C_{ijt}) \leq \Delta/B, \tag{4.25}$$

where $C_{ijt} = c_{ijt} + (c_{jt} - \hat{c}_{jt})$ is a random variable.

Next, a change of variables is made to render the problem tractable for path planning algorithms. For the ith constraint of the jth obstacle at time step t apply the change of variable

$$V = a_{ij}^T X_t - a_{ij}^T C_{ijt}; \tag{4.26}$$

it can be shown [34] that the mean and covariance for V are

$$\hat{v} = a_{ij}^T \hat{x}_t - a_{ij}^T c_{ijt}, \tag{4.27}$$

$$P_v = \sqrt{a_{ij}^T (P_{x_t} + P_{c_{jt}}) a_{ij}}. \tag{4.28}$$

With this change of variables, the probabilistic constraint can be shown to be equivalent to a deterministic constraint, [104]

$$\begin{aligned} P(V < 0) &\leq \Delta/B \\ \Leftrightarrow \ \hat{v} &\geq \gamma \equiv \sqrt{2} P_v \text{erf}^{-1}\left(1 - 2\frac{\Delta}{B}\right), \end{aligned}$$

where erf$(\cdot)$ denotes the standard error function. Using this, the constraints (4.23) are probabilistically satisfied for the true state x_t if the conditional mean $\hat{x}_t$ satisfies

$$\bigvee_{i=1}^{n_j} a_{ij}^T \hat{x}_t \geq b_{ij} + \bar{b}_{ijt} \ \ \forall\ j \in \mathbb{Z}_{1,B},\ \forall\ t \in \mathbb{Z}_{0,N}, \tag{4.29}$$

where $\bar{b}_{ijt} = \gamma$ represents the amount of *deterministic* constraint tightening necessary to ensure *probabilistic* constraint satisfaction.

Whereas the traditional RRT algorithm incrementally grows a tree of states which

are known to be feasible [97], the chance constrained RRT (CC-RRT) algorithm grows a tree of state distributions which are known to satisfy an upper bound on probability of collision. Furthermore, the CC-RRT algorithm can leverage a key property of the RRT algorithm – trajectory-wise constraint checking – by explicitly computing a bound on the probability of collision at each node, rather than simply satisfying tightened constraints for a fixed bound. In particular, the Online CC-RRT approach [34] leverages the relationship in (4.29) to compute the exact probability of satisfying each individual constraint for a given distribution $\mathcal{N}(\hat{x}, P_x)$ – an operation which is possible due to iterative constraint checking in the RRT algorithm. The key to the Online CC-RRT approach is the relationship

$$P(V < 0) = \frac{1}{2}\left(1 - \text{erf}\left[\frac{\hat{v}}{\sqrt{2}P_v}\right]\right), \tag{4.30}$$

which has been shown [34] to be derived from (4.29). Again consider the ith constraint of the jth obstacle at time step t, using the change of variables (4.26). Let $\Delta_{ijt}(\hat{x}, P_x)$ denote the probability that this constraint is satisfied for a Gaussian distribution with mean $\hat{x}$ and covariance P_x; using (4.30),

$$\Delta_{ijt}(\hat{x}, P_x) = \frac{1}{2}\left(1 - \text{erf}\left[\frac{a_{ij}^T\hat{x}_t - a_{ij}^T c_{ijt}}{\sqrt{2a_{ij}^T(P_{x_t} + P_{c_j})a_{ij}}}\right]\right) \tag{4.31}$$

Now define

$$\Delta_t(\hat{x}_t, P_{x_t}) \equiv \sum_{j=1}^{B} \min_{i=1,\ldots,n_j} \Delta_{ijt}(\hat{x}_t, P_{x_t}). \tag{4.32}$$

This term provides an upper bound on the probability of a collision with any obstacle

at time step t [34]:

$$\begin{aligned} P(\text{collision}) &\leq \sum_{j=1}^{B} P(\text{collision with obstacle } j) \\ &\leq \sum_{j=1}^{B} \min_{i=1,\ldots,n_j} P(a_{ij}^T X_t < a_{ij}^T C_{ijt}) \\ &= \sum_{j=1}^{B} \min_{i=1,\ldots,n_j} \Delta_{ijt}(\hat{x}_t, P_{x_t}) = \Delta_t(\hat{x}_t, P_{x_t}) \end{aligned} \tag{4.33}$$

Thus, for a node/timestep with state distribution $\mathcal{N}(\hat{x}_t, P_{x_t})$ to be probabilistically feasible, it is sufficient to check that $\Delta_t(\hat{x}_t, P_{x_t}) \leq 1 - p_{\text{safe}}$.

The CC-RRT algorithm consists of two main routines, a tree expansion step which incrementally adds probabilistically feasible nodes to the tree, and an execution loop which periodically chooses the cost-minimizing path in the tree; see [34] for more details.

RR-GP Integration

Suppose the jth obstacle is one of the dynamic obstacles modelled using RR-GP (Section 3.4); that dynamic obstacle may follow one of $k = 1, \ldots, b$ possible behaviors. At each timestep t, and for each behavior k, the RR-GP algorithm provides a likelihood δ^k and Gaussian distribution $\mathcal{N}(\hat{c}_{jt}^k, P_{c_{jt}}^k)$ for the uncertainty distribution of the obstacle if following that behavior. Thus, the overall state distribution for this obstacle at timestep t is given by

$$c_{tj} \sim \sum_{k=1}^{b} \delta^k \mathcal{N}(\hat{c}_{jt}^k, P_{c_{jt}}^k). \tag{4.34}$$

At each timestep, the probability of collision with dynamic obstacle j can be written as a weighted sum of the probabilities of collision for the dynamic obstacle j

under each behavior, using δ^k as the weights:

$$P(\text{collision with obstacle } j)$$
$$= \sum_{k=1}^{b} \delta^k P(\text{collision with obstacle } j, \text{behavior } k).$$

Comparing with (4.33), a dynamic obstacle with uncertain behaviors can actually be modelled as b separate obstacles, each with its own Gaussian uncertainty, with the modification that each such term is weighted by δ^k. The existing CC-RRT framework can be used by treating each behavior's state distribution as a separate obstacle, so long as the resulting risk is scaled by δ^k. Thus the existing guarantees of probabilistic feasibility [34] still hold under this modification.

Finally, we note that the guarantees presented in this section only hold under the assumptions of linear dynamics and Gaussian uncertainty. Modifications are possible to alleviate these assumptions [109], though this may yield an increase in computational load (particularly if the uncertainty is non-Gaussian).

4.2.5 Simulation Results

This section presents simulation results which demonstrate the effectiveness of the RR-GP algorithm in predicting the future behavior of an unknown, "hostile" vehicle, allowing the CC-RRT planner to design paths which can safely avoid it. As the probabilistic safety bound p_{safe} is increased at each timestep, the algorithm selects more conservative paths, which are more likely to reach the goal without colliding with any obstacles, but may require additional path length/time to do so. By placing the bound sufficiently high, however, the planner can design paths which consistently guide the host vehicle out of unsafe situations with high likelihood.

This section is organized as follows. First, the software infrastructure used to generate these results is explained. Three sets of simulation results then follow. The most thorough results consider a vehicle in an intersection scenario, where the agent must cross an intersection while avoiding another vehicle which is approaching the

intersection at a right angle without stopping. In this case, the host vehicle is modeled as a double integrator, such that the theoretical CC-RRT results still hold. In the second scenario, the host vehicle must navigate across a complex obstacle field; in this case, the hostile vehicle may follow as many as six different behaviors. Two versions of this scenario are presented – in the first version, a double integrator must travel to a sequence of fifty waypoints while avoiding a continuously-moving dynamic obstacle, while in the second version, a nonlinear Dubins vehicle must exchange positions with the dynamic obsatcle.

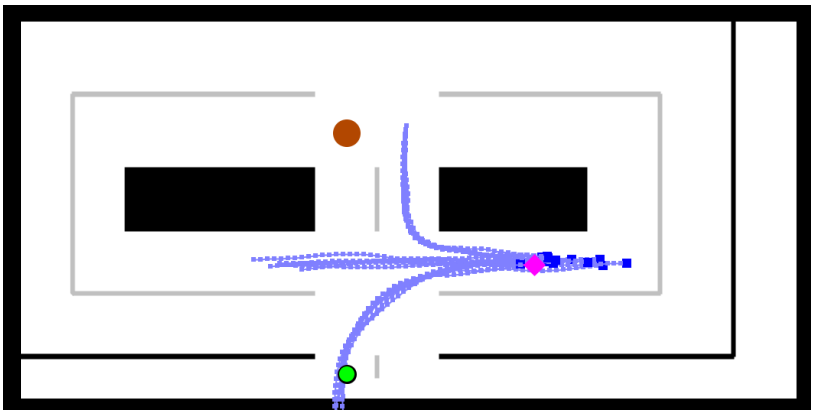
Infrastructure

The path planning algorithms in this work have been implemented using a multi-threaded, real-time Java application, executed on a 2.53GHz quad-core laptop with 3.48GB of RAM. The "hostile" vehicle's trajectories, both for GP training and simulated motion, were pre-generated by having a human operator manually drive the vehicle in simulation with a steering wheel, in a manner consistent with each behavior. The hostile vehicle dynamics are based on the iRobot Create skid-steered platform; a software wrapper imposes rate limits in acceleration and wheel speed differences, such that the vehicle emulates traditional automative dynamics at a maximum speed of 0.4 m/s. During each trial, one of these paths is randomly selected as the trajectory for the hostile vehicle.

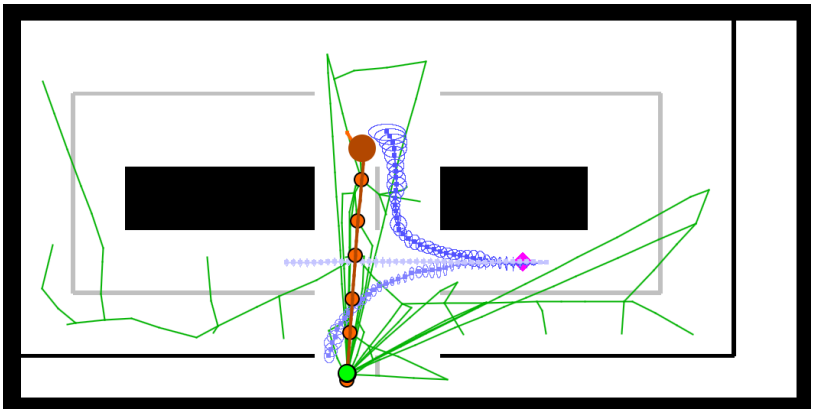
Intersection Scenario

Consider a ground vehicle operating in a constrained, two-dimensional environment (Figure 4-11(a)). This environment is a representative road network designed to emulate a real-world driving environment, and is based on a layout previously set up in hardware ([30]) in the RAVEN testbed ([89]). The road network is 11.2×5.5 m^2 in size, and is capable of accommodating multiple intersection types.

Consider a ground vehicle operating in a simple road network (Figure 4-11). In this scenario, which assumes left-hand traffic, the objective of the host vehicle is to go straight through the intersection at bottom-center of Figure 4-11, reaching a goal



(a) Environment used in the intersection scenario. The objective of the host vehicle (orange circle) is to reach the goal position (green circle) while avoiding all static obstacles (black) and the dynamic hostile vehicle (magenta diamond). The blue curves indicate the possible trajectories followed by the hostile vehicle. All objects are shown at true size; the grey lines are lane markings, which do not serve as constraints.



(b) Representative screenshot of planner in operation. The CC-RRT tree is in green; the host vehicle's path history and current path are in orange. The blue paths indicate the paths predicted by the RR-GP algorithm for each possible behavior, including $2-\sigma$ uncertainty ellipses; more likely paths are indicated with a brighter shade of blue.

Figure 4-11: Environment and display output used for the intersection scenario.

location 6 meters ahead on the opposite side. However, to get there, the host vehicle must successfully avoid a hostile vehicle, which incorrectly believes that right-hand traffic rules are in effect and thus is traveling in the wrong lane. There are three possible behaviors for the hostile vehicle as it enters the intersection: (a) left turn, (b) right turn, and (c) straight.

The vehicle is modeled as a double integrator:

$$\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ v^x_{t+1} \\ v^y_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \\ v^x_t \\ v^y_t \end{bmatrix} + \begin{bmatrix} \frac{dt^2}{2} & 0 \\ 0 & \frac{dt^2}{2} \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \left(\begin{bmatrix} u^x_t \\ u^y_t \end{bmatrix} + \begin{bmatrix} w^x_t \\ w^y_t \end{bmatrix} \right),$$

where $dt = 0.1$s, subject to avoidance constraints $\mathcal{X}$ (including velocity bounds) and input constraints

$$\mathcal{U} = \{(u^x, u^y) \mid |u^x| \leq 4, |u^y| \leq 4\}.$$

This choice of dynamics was made to ensure that the dynamics are still linear, such that the theoretical guarantees ([34]) are available; the subsequent example considers nonlinear dynamics. To emphasize the impact of the hostile vehicle's uncertainty, the host vehicle's own dynamics are assumed deterministic: $w^x_t \equiv w^y_t \equiv 0$. The vehicle is simulated and executed in closed-loop, using the controller

$$\begin{aligned} u^x_t &= -1.5(x_t - r^x_t) - 3(v^x_t - r^{v_x}_t), \\ u^y_t &= -1.5(y_t - r^y_t) - 3(v^y_t - r^{v_y}_t), \end{aligned}$$

where (r^x_t, r^y_t) is the reference position and $(r^{v_x}_t, r^{v_y}_t)$ is the reference velocity; the reference r_t is moved continuously between waypoints at a fixed speed of 0.35 m/s.

The hostile vehicle dynamics are based on the iRobot Create platform [87], a 2-wheeled, skid-steered vehicle with near-instant acceleration. When pre-generating trajectories (Section 4.2.5), a software wrapper imposes rate limits in acceleration

and wheel speed differences, such that the vehicle emulates traditional automative dynamics at a maximum speed of 0.4 m/s.

In the CC-RRT algorithm, the tree capacity is limited to 1000 nodes, with a replan time interval of 0.5s. In Algorithms 1-2, $T_h = 8$s and $\Delta t = 1$s, though the vehicle dynamics are simulated at 10 Hz. The limits of successful and infeasible connections per Δt (lines 9 and 11, Algorithm 1) are 20 and 100, respectively. The RR-GP algorithms are called once every 1.0 seconds, each time giving Algorithm 2 a total of 0.3 seconds to grow its trees. If the time limit is reached, the algorithm is terminated without reaching the time horizon T_h.

A total of 400 trials were performed, consisting of 50 trials each for eight different algorithms:

- Naive RRT: nominal RRT (no chance constraints) in which hostile vehicle is ignored; this sets a baseline for the minimum expected likelihood of safety
- Nominal RRT: nominal RRT in which hostile vehicle is treated as a static obstacle at its most recent location
- Velocity-Avoidance RRT: nominal RRT in which the future position of the hostile vehicle is predicted by propagating its current position based on its current velocity and heading
- CC-RRT (5 cases): CC-RRT Algorithm with $p_{\text{safe}} = 0.5$, 0.8, 0.9, 0.99, or 0.999

Each trial differs only in the path followed by the hostile vehicle and the random sampling used in the RR-GP and CC-RRT algorithms; the sequence of hostile vehicle paths is consistent across all sets of 50 trials. Four quantities were measured and averaged across these trials: the percentage of trials in which the vehicle safely reaches the goal; the average duration of such paths; the average time to generate an RRT/CC-RRT tree node; and the average time per execution of Algorithm 2.

Table 4.1 presents the averaged results over the 50 trials for each case. Note that in all five cases using CC-RRT, the host vehicle safely navigates the intersection with

Table 4.1: Simulation Results, Intersection Scenario

Algorithm	p_{safe}	**% to Goal**[a]	**Path Duration, s**[b]	**Time per Node, ms**[c]	**Time per RR-GP, s**[d]
Naive RRT	–	38%	10.01 (0.11%)	0.611	–
Nominal RRT	–	46%	10.90 (8.96%)	0.662	–
Velocity-Avoidance RRT	–	74%	11.14 (11.4%)	0.948	–
CC-RRT	0.5	92%	11.52 (15.2%)	1.610	0.598
CC-RRT	0.8	88%	11.65 (16.5%)	1.620	0.598
CC-RRT	0.9	92%	11.69 (16.9%)	1.620	0.590
CC-RRT	0.99	96%	12.51 (25.1%)	1.537	0.592
CC-RRT	0.999	100%	12.84 (28.4%)	1.492	0.587

[a] Number of trials where system executed a path to goal without colliding with any obstacles. (Recall that p_{safe} refers to feasibility for a single *time step*, whereas this entry corresponds to feasibility across the entire *path*.)
[b] Percentage is average increase in path duration relative to minimal-time (obstacle-free) path, 10.0s. Only paths which reach goal are included.
[c] Cumulative time spent growing the CC-RRT tree, divided by number of nodes generated.
[d] Time spent in Algorithm 2.

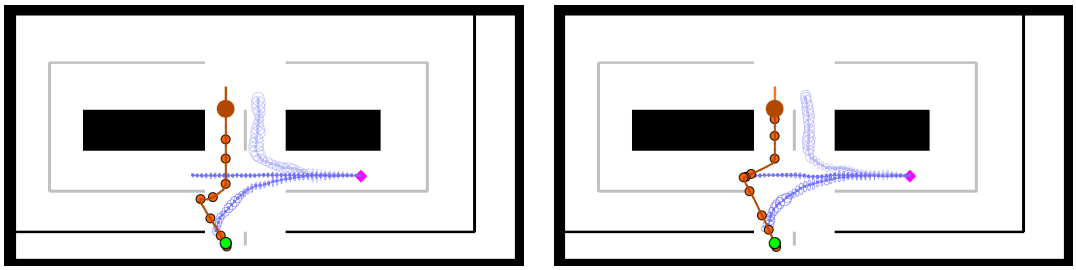
a much higher likelihood than any of the cases not using chance consrtaints. Furthermore, the CC-RRT results demonstrate the clear trade-off between overall path safety (in terms of percentage of trials which reach the goal) and average path duration when using CC-RRT. As p_{safe} is increased from 0.5 to 0.999, the percentage of safe trajectories generally increases (with the one exception of $p_{\text{safe}} = 0.5$), culminating with the host vehicle using CC-RRT with $p_{\text{safe}} = 0.999$ reaching the goal safely in all fifty trials.[3] On the other hand, as p_{safe} is increased and the planner becomes more conservative, the average degree of suboptimality of the safe trajectories also increases.

Figure 4-12 sheds some light on how different values of p_{safe} affect the types of paths chosen by the planner. In this particular trial, trial # 25, the hostile vehicle ultimately makes a left turn through the intersection, and would collide with the host vehicle if it did not deviate from an initial shortest-path trajectory. The RR-GP algorithm is initially undecided whether the hostile vehicle is going straight or turning left (as indicated by the shading on the predicted trajectories in Figures 4-

[3]Note that since p_{safe} is a bound on feasibility at each *timestep*, rather than over an entire *path*, it does *not* act as a bound on the percentage of paths which safely reach the goal - though a positive correlation between the two is expected.

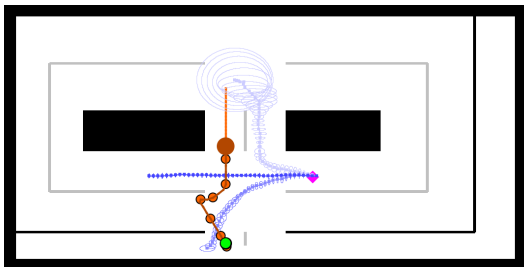
12(a) through 4-12(d)); by $t = 6$ seconds it has confirmed the vehicle is turning left (Figures 4-12(e) and 4-12(f)). When $p_{\text{safe}} = 0.8$, the planner selects a path with the minimum perturbation needed to avoid the hostile vehicle's most likely trajectories (Figure 4-12(a); note that $2-\sigma$ uncertainty ellipses are shown). As the hostile vehicle closes in on the intersection (Figures 4-12(c) and 4-12(e)), the host vehicle continues to hedge that it can cross the intersection safely and avoid the hostile vehicle's approach from either direction, and thus does not modify its plan. In contrast, when $p_{\text{safe}} = 0.999$, the planner selects a larger initial perturbation to maintain the host vehicle's distance from the hostile vehicle (Figure 4-12(b)). After several RR-GP updates, the host vehicle demonstrates a much more risk-averse behavior, by loitering outside the intersection (Figures 4-12(d) and 4-12(f)) for several seconds before making its approach. Ultimately, the host vehicle using $p_{\text{safe}} = 0.8$ reaches the goal (Figure 4-12(g)) before the host vehicle using $p_{\text{safe}} = 0.999$ (Figure 4-12(h)). In realistic driving scenarios, the most desirable behavior is likely somewhere between these two extremes.

Returning to Table 4.1 for Naive RRT, we see that by ignoring the hostile vehicle, the time-optimal path is almost always achieved, but that a collision takes place in a majority of trials, with collisions occurring in most instances of the hostile vehicle going straight or left. (Naive RRT can be thought of as the limiting case for CC-RRT as $p_{\text{safe}} \to 0$). In some instances, the Nominal RRT algorithm is able to maintain safety by selecting an alternative trajectory when the hostile vehicle's current position renders the host vehicle's current trajectory infeasible (Figure 4-13); however, the overall likelihood of safety is still quite low. In many cases, the hostile vehicle collides with the host vehicle from the side, such that a replan is not possible. Of the nominal RRT algorithms, the velocity-avoidance RRT algorithm performs the most competitively with CC-RRT, with 74% of trials yielding a safe trajectory. Since the hostile vehicle always starts by driving to the left, the host vehicle responds in nearly all trials by immediately perturbing its own path, based on the assumption that the hostile vehicle will go straight through the intersection (Figure 4-14(a)). This contributes to the larger average path duration obtained by velocity-avoidance

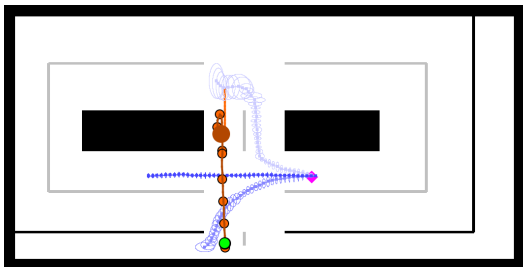


(a) $p_{\text{safe}} = 0.8$, $t = 2$ seconds

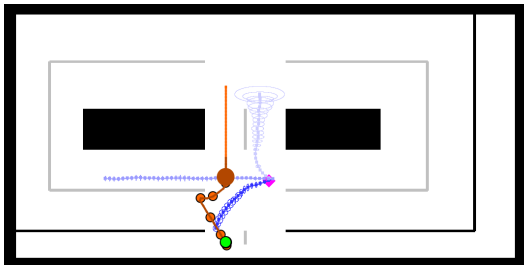
(b) $p_{\text{safe}} = 0.999$, $t = 2$ seconds



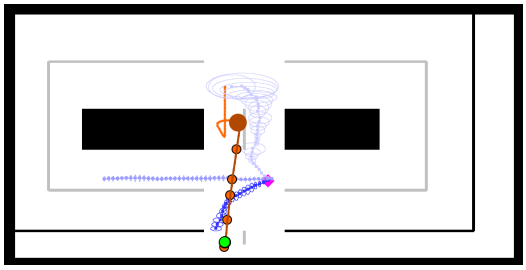
(c) $p_{\text{safe}} = 0.8$, $t = 4$ seconds



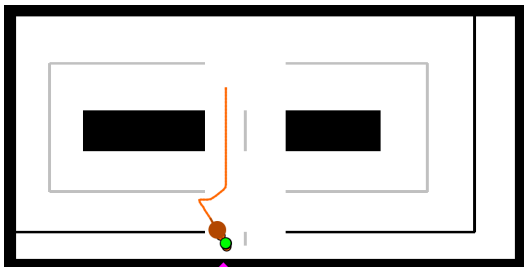
(d) $p_{\text{safe}} = 0.999$, $t = 4$ seconds



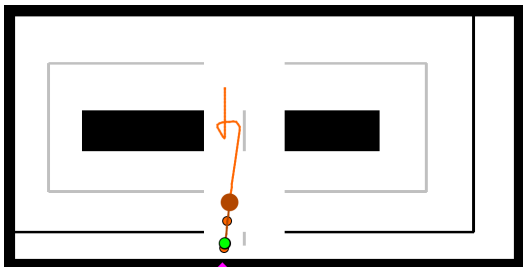
(e) $p_{\text{safe}} = 0.8$, $t = 6$ seconds



(f) $p_{\text{safe}} = 0.999$, $t = 6$ seconds



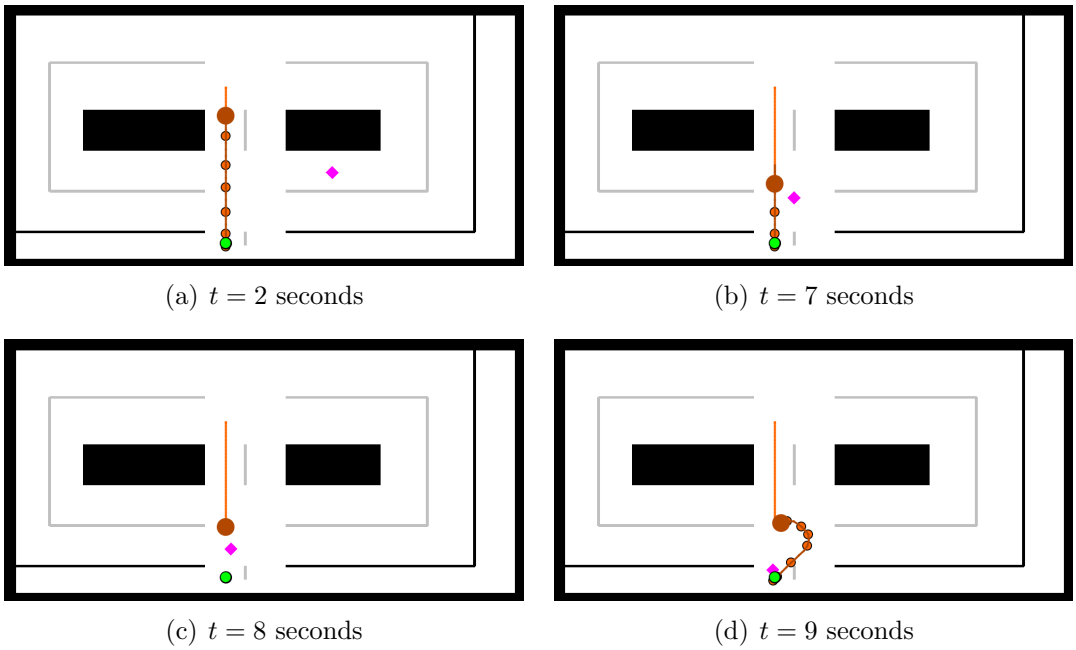
(g) $p_{\text{safe}} = 0.8$, $t = 11$ seconds



(h) $p_{\text{safe}} = 0.999$, $t = 11$ seconds

Figure 4-12: Representative screenshots of the RR-GP and CC-RRT algorithms during trial #25 of the intersection scenario, for two different values of p_{safe}. The host vehicle's path history and current path are in orange. The blue paths indicate the paths predicted by the RR-GP algorithm for each possible behavior, including $2-\sigma$ uncertainty ellipses; more likely paths are indicated with a brighter shade of blue.

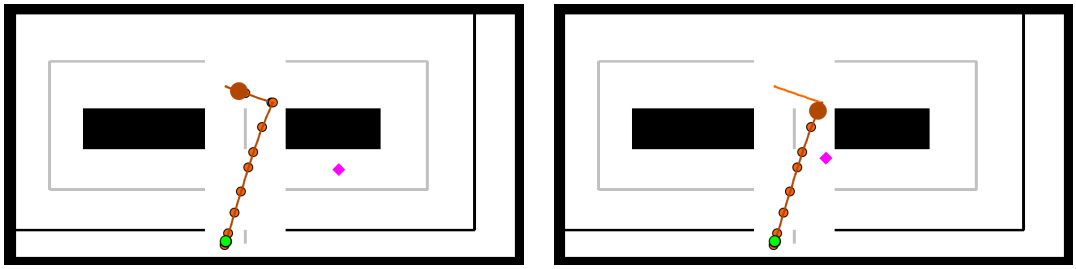
RRT compared to the other nominal algorithms in Table 4.1). However, in many instances, velocity-avoidance RRT is still unable to respond to rapid changes in the



(a) $t = 2$ seconds (b) $t = 7$ seconds

(c) $t = 8$ seconds (d) $t = 9$ seconds

Figure 4-13: Representative screenshots of the nominal RRT algorithm during trial #14 of the intersection scenario. The host vehicle plans a direct path to the goal (Figures 4-13(a) and 4-13(b)) until the hostile vehicle crosses the path, rendering it infeasible (Figure 4-13(c)); the host vehicle then replans around the hostile vehicle (Figure 4-13(d)).



(a) $t = 2$ seconds (b) $t = 6$ seconds

Figure 4-14: Representative screenshots of the velocity-avoidance RRT algorithm during trial #5 of the intersection scenario. In this trial, the hostile vehicle turns right and ultimately collides with the host vehicle.

hostile vehicle's heading, such as when the hostile vehicle turns right (Figure 4-14(b)).

Finally, we note that the average time to either generate an RRT node or call RRGP is largely independent of p_{safe} for CC-RRT. There is a modest increase in average time per node when moving from naive or nominal RRT to velocity-avoidance RRT (scales by a factor of 1.5) or CC-RRT (scales by a factor of 1.5), though previ-

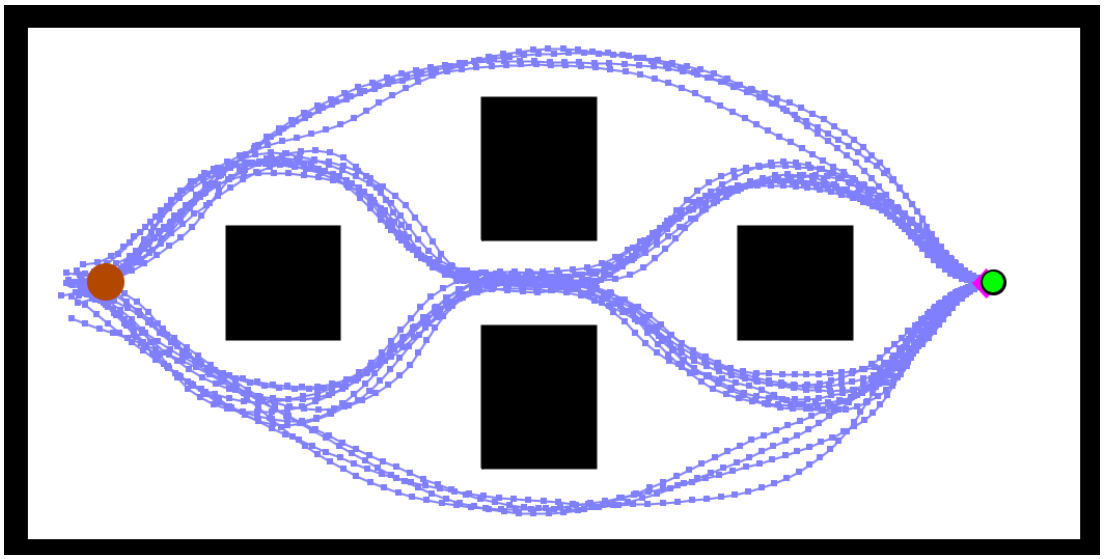


Figure 4-15: Environment used in the complex scenario, including possible behaviors for the hostile vehicle (at goal on right); see Figure 4-11 for legend.

ous work has demonstrated that this increases scales well with environment complexity ([34]).

Complex Scenario

This next case increases the complexity of the scenario, with more obstacles and more possible behaviors for the target vehicle (Figure 4-15. This environment is the same size as the previous one, with rearranged obstacles; as the target vehicle moves from one side of the environment to the other, it may display as many as six possible behaviors, corresponding to whether each of the four obstacles is passed by the target vehicle on its left or right. Furthermore, by design the target vehicle has a higher maximum speed than the host vehicle, meaning that the host vehicle is at risk of being overtaken from behind if its path is not planned carefully. Two versions of this scenario are presented. In the first version, a double integrator must travel to a sequence of fifty waypoints while avoiding a continuously-moving dynamic obstacle; all CC-RRT theory still holds in this case. In the second version, a nonlinear Dubins vehicle must exchange positions with the dynamic obsatcle; representative screenshots of these trials are presented.

Linear Dynamics Example: Consider the use of the same double integrator used in the results in Section 4.2.5; the same dynamics and controller are used, but in this case the reference movement speed is increased from 0.35 m/s to 0.6 m/s. Due to the increased number of behaviors and more complex environment, the Gaussian process formulation is more challenging for this scenario than in Section 4.2.5; as a result, several of the parameters in Algorithms 1-2 have been modified to improve performance. Here, the limits of successful and infeasible connections per Δt (lines 9 and 11, Algorithm 1) are 10 and 100, respectively. The RRGP algorithms are called once every 1.25 seconds, each time giving Algorithm 2 up to a full second to grow trees for each behavior, with the time horizon T_h increased from 8 seconds to 12 seconds. Both vehicles have a radius of 0.2m and start at zero velocity.

The objective of this scenario is to demonstrate the ability of the CC-RRT algorithm, using RR-GP dynamic obstacle data, to exhibit safe driving behavior for long-duration missions. The same eight algorithms used in Section 4.2.5 are again used here; however, rather than performing 50 trials, each algorithm is used to guide the host vehicle through a continuous sequence of 50 waypoints. The host vehicle starts on the left side of the room, and each subsequent waypoint is among a set of four, located near each of the room's 4 corners. The host vehicle is given the next waypoint as soon as the current waypoint is reached; consecutive waypoints are required to be on opposite sides of the room, with respect to the room's long axis. While the host vehicle completes this sequence, the hostile vehicle moves continuously back and forth between the left and right ends of the room, each time selecting one of the six possible behaviors and one of the four pre-generated trajectories for that behavior (Figure 4-15).[4] Note that both the sequence of waypoints and hostile vehicle behaviors are consistent across all algorithms.

If the host vehicle collides with each the hostile vehicle or an element of the environment, the mission continues; however, the hostile vehicle is penalized for this collision by being re-set back to its last reached waypoint. This allows the trial for each

[4]When the hostile vehicle moves from left to right, the trajectories shown in Figure 4-15 are reflected across the room's short axis.

Table 4.2: Simulation Results, Complex Scenario

Algorithm	p_{safe}	**# Collisions**[a]	**Mission Duration, s**[b]	**Time per Node, ms**	**Time per RR-GP, s**
Naive RRT	–	13	737.1	0.731	–
Nominal RRT	–	20	819.5	0.812	–
Velocity-Avoidance RRT	–	4	766.3	1.421	–
CC-RRT	0.5	6	821.2	5.112	0.665
CC-RRT	0.8	8	818.9	5.596	0.654
CC-RRT	0.9	2	811.7	4.343	0.635
CC-RRT	0.99	4	820.5	5.142	0.643
CC-RRT	0.999	6	834.1	4.693	0.639

[a] Number of collisions which took place over a single trial. Collisions which take place within 0.5 seconds of each other are not counted as separate collisions.
[b] Total time required for host vehicle to reach 50 waypoints; note that the vehicle is reset to its last reached waypoint each time a collision takes place.

algorithm to be performed as a single, continuous simulation, while also providing a figure of merit which factors in both path duration/length and risk of collision.

Four quantities were measured and averaged for each algorithm: the total time required to reach all 50 waypoints, including collision time penalties; the number of collisions which take place; the average time to generate an RRT/CC-RRT tree node; and the average time per execution of Algorithm 2.

Table 4.2 presents the results for each algorithm. A key trend in these results is that as p_{safe} increases, both the number of collisions and mission duration tend to decrease, reach minimum values at $p_{\text{safe}} = 0.9$, then actually increase beyond that value. For lower values of p_{safe}, the host vehicle is willing to execute trajectories with a significant amount of risk, by bringing the vehicle closer to the dynamic obsatcle, and thus collisions are more probable. Even though the resulting trajectories selected by the planner will be shorter than for larger values of p_{safe}, the time penalties for collisions actually result in an overall longer mission duration. On the other hand, for higher values of p_{safe}, the planner is more likely to mark a path as probabilistically infeasible for small amounts of risk, causing the vehicle to come to a stop at the conclusion of its probabilistically feasible path. This is typically a desirable behavior; however, the hard example (Figure 4-15) features many narrow corridors and a dynamic obstacle whose top speed exceeds the planner. In this case, coming to a

stop may actually increase the risk of collision, particularly if the host vehicle stops in the path of the hostile vehicle (whose trajectory ingnores the host vehicle's current position).

As a result, the most desirable behavior is achieved using CC-RRT with $p_{\text{safe}} = 0.9$, with only 2 collisions taking place over a sequence of 50 waypoints. Two aspects of the complex scenario tend to increase the likelihood of collision across all algorithms, such that 100% safety becomes unreasonable for this scenario. First, when the hostile vehicle changes direction, the RR-GP prediction environment changes rapidly (compare Figures 4-16(e) and 4-16(f) with Figures 4-16(g) and 4-16(h)), and the host vehicle may not be able to react quickly enough if nearby. Second, the host vehicle may find itself in a corridor being pursued by the hostile vehicle; since the hostile vehicle has a larger maximum speed, a collision may become inevitable.

Figure 4-16 demonstrates how CC-RRT can exhibit complex, robust avoidance behavior for large values of p_{safe} in order to remain risk-averse. In each trial, the agent's first task (shown in the figure) is to move from the left side of the room to the waypoint at bottom-right; the shortest path is to move along the bottom of the room. The hostile vehicle's trajectory takes it to the left of the first and last obstacles, and down the central corridor. When the first RR-GP update is performed, there is little data available to infer which way the hostile vehicle is going, and thus all six behaviors are equally likely. For $p_{\text{safe}} = 0.8$, the planner selects a complete trajectory which reaches the goal (Figure 4-16(a)). Even though this would lead to a head-on collision for one of the behaviors, the likelihood of that behavior being active is roughly 1 in 6, an acceptable risk for $p_{\text{safe}} = 0.8$ ($< 5/6$). On the other hand, when $p_{\text{safe}} = 0.999$, the planner is not willing to select a trajectory with crosses the hostile vehicle's path for *any* possible behavior. Instead, it selects a partial path behind one of a central obstacles, the location which brings it closest to the goal without being in any of the hostile vehicle's possible paths (Figure 4-16(b)).

As the mission progresses, the hostile vehicle's path is revealed to go through the central corridor. For $p_{\text{safe}} = 0.8$, this was not the behavior that risked a head-on collision, so it continues on its initial trajectory (Figures 4-16(c) and 4-16(e)), reaching

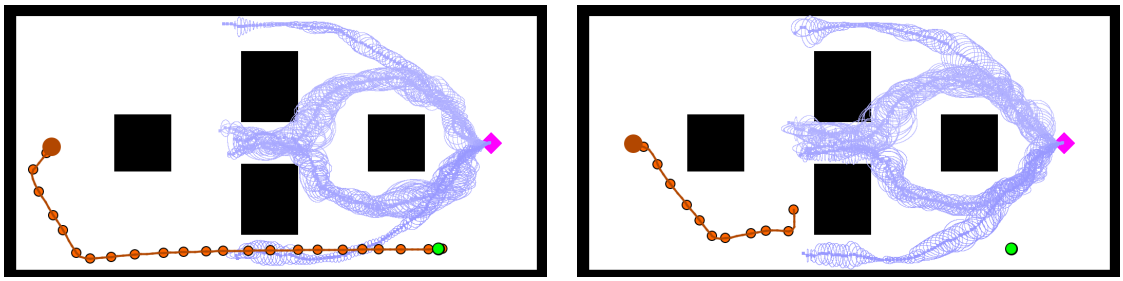
the goal state quickly (Figure 4-16(g)). On the other hand, when $p_{\text{safe}} = 0.999$, the agent holds its position behind the obstacle until the hostile vehicle has passed through the central corridor (Figure 4-16(d)); once the hostile vehicle has passed by, the identifies a new trajectory which reaches the goal (Figure 4-16(f)), though it arrives at the goal significantly later than if a lower value of p_{safe} were used (Figure 4-16(h)).

Observing the nominal RRT results in Table 4.2, it is clear that neither Naive RRT nor Nominal RRT can provide a sufficient level of safety for the host vehicle, with a double-digit number of collisions occurring in each case. On the other hand, Velocity-Avoidance RRT is quite competitive with CC-RRT. Though it does not achieve the minimum number of collisions obtained by CC-RRT for $p_{\text{safe}} = 0.9$, velocity-avoidance RRT still only has 4 collisions, as well as a mission duration significantly shorter than any of the CC-RRT trials. Since the trajectories executed by the hostile vehicle are relatively straight, with few sharp turns (especially compared to Section 4.2.5), the forward propgation done in this case actually tends to be a good prediction, allowing the host vehicle to make a rapid response when needed.

Concerning runtimes, it is observed that the average runtime needed to generate a tree node using CC-RRT is larger than the average runtime for nominal RRT, by a factor which is larger than the one observed in Section 4.2.5. Nonetheless, the runtime per node averages only 5ms, meaning many hundreds of nodes can be generated every second. Coupled with the fact that the RR-GP update averages a fraction of a second (about 0.65s), the CC-RRT algorithm with RR-GP is clearly amenable to real-time implementation for this more complex example.

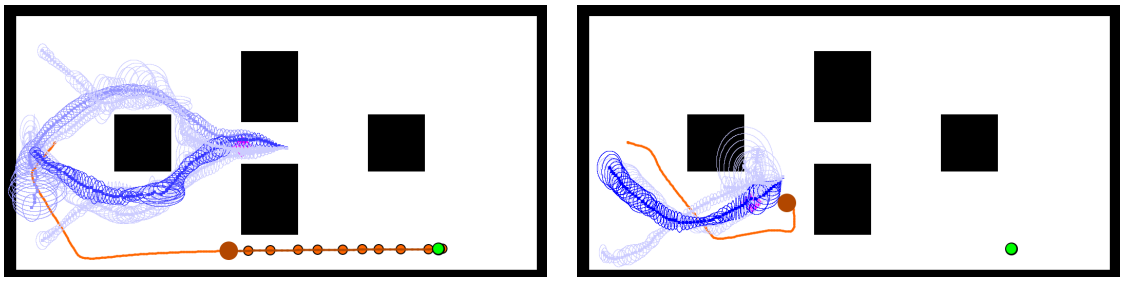
A video showing an autonomous vehicle navigating the complex scenario using CC-RRT with Integrated RR-GP (for $p_{\text{safe}} = 0.999$) is available at `http://acl.mit.edu/rrgp.mp4`.

Nonlinear Dynamics Example: For this scenario, the host vehicle is modeled as nonlinear car dynamics with a fixed speed $v = 0.4$ m/s, half that of the target



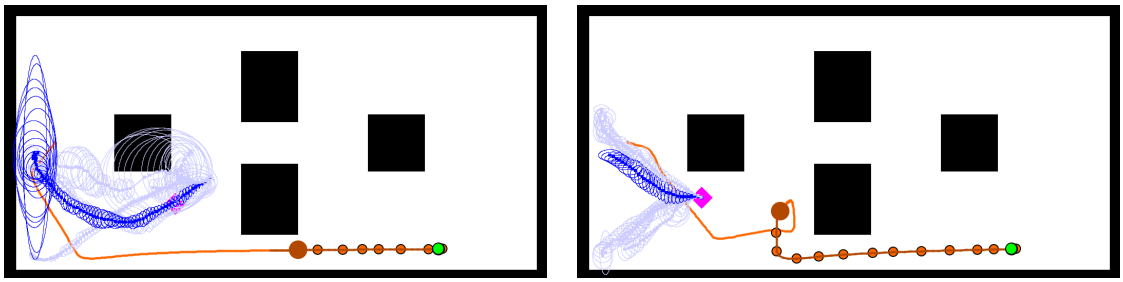
(a) $p_{\text{safe}} = 0.8$, $t = 1$ seconds

(b) $p_{\text{safe}} = 0.999$, $t = 1$ seconds



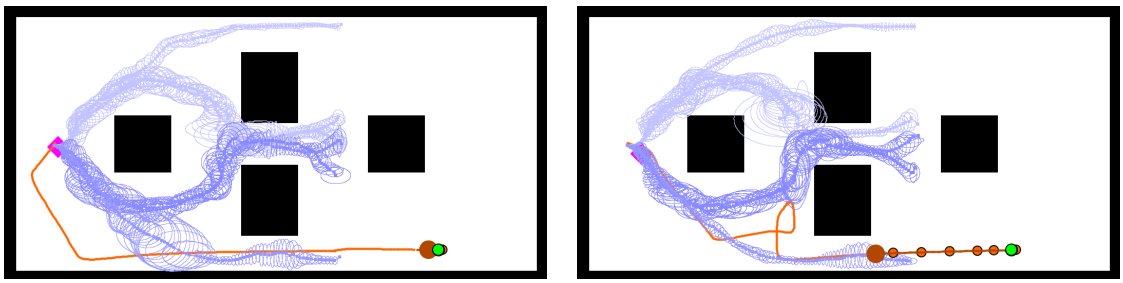
(c) $p_{\text{safe}} = 0.8$, $t = 8$ seconds

(d) $p_{\text{safe}} = 0.999$, $t = 8$ seconds



(e) $p_{\text{safe}} = 0.8$, $t = 9$ seconds

(f) $p_{\text{safe}} = 0.999$, $t = 9$ seconds



(g) $p_{\text{safe}} = 0.8$, $t = 15$ seconds

(h) $p_{\text{safe}} = 0.999$, $t = 15$ seconds

Figure 4-16: Representative screenshots of the RR-GP and CC-RRT algorithms as the host vehicle approaches the first waypoint in the complex scenario, for two different values of p_{safe}.

vehicle:

$$\begin{aligned} x_{t+1} &= x_t + (dt)v\cos\theta_t + w_t^x, \\ y_{t+1} &= y_t + (dt)v\sin\theta_t + w_t^y, \\ \theta_{t+1} &= \theta_t + (dt)\frac{v}{L_w}\tan\delta_t + w_t^\theta, \end{aligned}$$

where $dt = 0.1$s, (x, y) is the vehicle position, θ is the heading, $L_w = 0.2$ m, and $\delta_t \in [-\pi/4, +\pi/4]$ is the steering angle input. The host vehicle and target vehicle are to swap positions; the host vehicle must move from left to right, while avoiding a collision with a faster-moving dynamic obstacle moving in the opposite direction.

The vehicle is controlled in closed-loop using the nonlinear steering controller

$$\delta_t = -K_y \tan^{-1}(e_y) - K_\theta e_\theta,$$

where $K_y = K_\theta = 1$, e_θ is the difference in headings between the vehicle and waypoint, and e_y is the lateral distance between the vehicle and waypoint (as projected onto the waypoint heading). In this scenario, the vehicle must contend with both uncertainty in the dynamic obstacle and process noise, where the covariance on the disturbance $w_t = (w_t^x, w_t^y, w_t^\theta)$ is

$$P_w = \begin{bmatrix} 5\times10^{-5} & 0 & 0 \\ 0 & 5\times10^{-5} & 0 \\ 0 & 0 & 2\times10^{-4} \end{bmatrix},$$

with zero localization error ($P_{x_0} = 0$). In this version, the RR-GP algorithms are called once every 1.25 seconds, each time giving Algorithm 2 a total of 1 second to grow trees for each behavior, with T_h again set to 12 seconds.

Figure 4-17 demonstrates the operation of the developed algorithms for a typical trial using the nonlinear dynamics, with $p_{\text{safe}} = 0.99$. In these figures, the current RRT tree at each timestep is shown, including $2-\sigma$ uncertainty ellipses. Note that

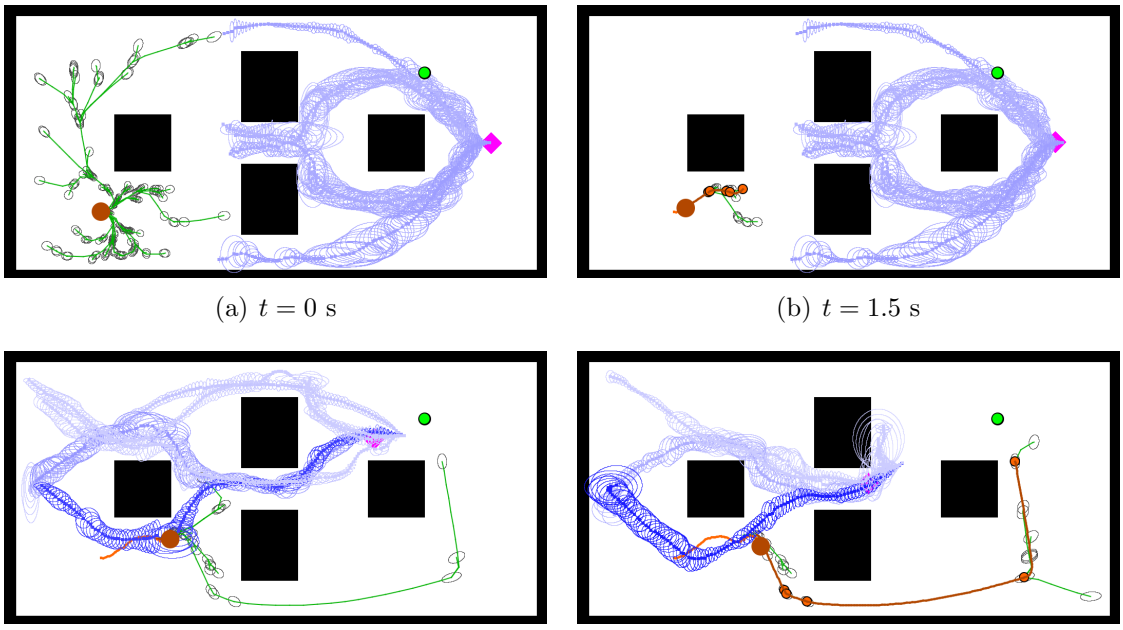
the uncertainty ellipses are generally aligned with the direction of motion, consistent with the existence of heading angle process noise [109].

When the simulation is initialized, all possible behaviors are perceived as equally likely (Figure 4-17(a)). The host vehicle initially selects a path guiding it forward a small amount toward the goal, awaiting a path to be identified in the tree which is both probabilistically feasible and brings it closer to the goal (Figure 4-17(b)). Three seconds later, two distinct features are clear in the RRT tree (Figure 4-17(c). A portion of the tree near the host vehicle would bring it directly toward the goal, but does not reach the goal – the path is terminated prematurely due to the path becoming probabilistically infeasible, when the target vehicle reaches the central corridor. On the other hand, a longer path is found through the lower corridor which reaches the goal, since RR-GP has indicated the likelihood of any behavior passing through that corridor is virtually zero. The host vehicle selects this path, and is able to safely proceed toward the goal without interacting with the target vehicle again (Figure 4-17(d)). The host vehicle safely reaches the goal despite uncertainty both in its own motion and in the motion of a dynamic obstacle.

Perhaps most importantly, RR-GP is able to update its six behavior trajectory predictions in an average time of 0.80 s. This demonstrates the ability of RR-GP to use Gaussian processes to generate a complex set of generalizable trajectory predictions while operating in real-time.

4.2.6 Conclusion

This section has developed a real-time path planning framework which allows autonomous agents to safely navigate environments while avoiding dynamic obstacles with uncertain motion patterns. The developed solution combines chance-constrained rapidly-exploring random trees [34], with RR-GP, Bayesian nonparametric reachability trees [29], resulting in a probabilistic path planning formulation that guarantees probabilistic feasibility and is suitable for real-time implementation. Simulation results have demonstrated the effectiveness of the integrated approach in achieving high levels of vehicle safety for a variety of dynamics, environments, and behaviors, with



(a) $t = 0$ s (b) $t = 1.5$ s

(c) $t = 4.5$ s (d) $t = 6$ s

Figure 4-17: Demonstration of the CC-RRT algorithm with RR-GP applied to a representative trial in the complex, nonlinear scenario. Uncertainty ellipses for the host vehicle's process noise are indicated by $2 - \sigma$ uncertainty ellipses on the RRT tree.

modest increases in path duration and computational load.

Chapter 5

Threat Assessment Algorithms and Application to Road Intersections

While the previous chapter developed algorithms to improve the safety of autonomous vehicle navigation, this chapter focuses on solution approaches that assist human drivers in urban environments. More specifically, it considers the decision-making problem for a human-driven vehicle crossing a road intersection in the presence of other, potentially errant, drivers [30]. Our approach relies on a novel threat assessment module, which combines an intention predictor based on support vector machines (Chapter 2) with an efficient threat assessor using rapidly-exploring random trees. The threat assessment module was introduced in Chapter 4, and is extended in this chapter to provide a threat warning to assist human drivers at road intersections. The module warns the host driver with the computed threat level and the corresponding best escape maneuver through the intersection. Through experimental results with small autonomous and human-driven vehicles in the Aerospace Controls Laboratory RAVEN testbed, we demonstrate that this threat assessment module can be successfully used in real-time to minimize the risk of collision in urban-like environments. This chapter also includes another game-theoretic formulation that extends the developed reachability algorithm to a dual exploration-pursuit mode to consider the worst-case scenario of errant drivers capable of causing intentional collisions [35].

5.1 Introduction

The field of road safety and safe driving has witnessed rapid advances due to improvements in sensing and computation technologies. Active safety features like anti-lock braking systems and adaptive cruise control have been widely deployed in automobiles to reduce road accidents [3]. However, the US Department of Transportation (DOT) still classifies road safety as "a serious and national public health issue." In 2008, road accidents in the US caused 37,261 fatalities and about 2.35 million injuries. A particularly challenging driving task is negotiating a traffic intersection safely; an estimated 45 percent of injury crashes and 22 percent of roadway fatalities in the US are intersection-related [23]. A main contributing factor in these accidents is the driver's inability to correctly assess and/or observe the danger involved in such situations [24].

This data suggests that driver assistance or warning systems may have an appropriate role in reducing the number of accidents, improving the safety and efficiency of human-driven ground transportation systems. Driver assistance systems can be classified into two categories: infrastructure-based systems, such as intelligent vehicle highway systems (IVHS), and vehicle-based systems, such as car-to-car (C2C) communications [111]. Vehicle-based systems are expected to become available more quickly on commercial vehicles, compared to their infrastructure-based counterparts [112]. Such systems typically augment the driver's situational awareness and can also act as collision mitigation systems. This research is focused on threat assessment algorithms that can be applied to vehicle-based systems.

Research on intersection decision support systems has become quite active in both academia and the automotive industry. In the US, the federal DOT, in conjunction with the California, Minnesota, and Virginia DOTs and several US research universities, is sponsoring the Intersection Decision Support (IDS) project [24]. In Europe, the InterSafe project was created by the European Commission to increase safety at intersections. The partners in the InterSafe project include European vehicle manufacturers and research institutes [25]. Both projects try to explore the requirements,

tradeoffs, and technologies required to create an intersection collision avoidance system, and demonstrate its applicability on selected dangerous scenarios [24, 25].

Several measures have been proposed to characterize the threat level of dynamic road situations. These approaches typically measure collision risk by time-to-collision (TTC) and its variants [14], such as headway time [113] or required deceleration [114]. However, these measures are tailored to frontal collision warning systems, where the unpredictable dangerous driver is leading the host driver, and cannot typically be applied to intersection scenarios where the dangerous driver may approach from a variety of angles. Model Predictive Control (MPC) techniques have been also used to compute the threat level [115, 116]. However, their formulation limits them from considering other vehicles with uncertain behaviors. Other approaches have considered safety assessment for more general road scenarios using Monte Carlo techniques [117–119], but their computational burden prevents them from real-time implementation. Recent work has proposed new techniques for assessing, in real-time, the threat at intersections. But they are typically limited to straight crossing path scenarios [120], or assume deterministic future trajectories for the other vehicles and simplistic driver models [121, 122].

This section proposes a new approach for threat assessment at intersections that addresses several of the limitations of the current approaches. More specifically, it considers the problem of assisting human drivers with negotiating busy intersections in the presence of possibly errant drivers with uncertain intentions. It proposes a novel design for a threat assessment module, which combines a learning-based intention predictor with an efficient sampling-based threat assessor to compute the threats of errant drivers in real-time. This threat data is used to evaluate the safety of several possible escape paths, which may be proposed to the human driver if evasive maneuvers are warranted. The approach is demonstrated through experimental results demonstrating its effectiveness on different intersection scenarios in the MIT RAVEN testbed [89].

5.2 Problem Statement

We now define the road intersection threat assessment problem that is analyzed in this section.

Definition 1 (Intersection Threat Assessment Problem) *Consider a host vehicle $\mathcal{HV}$ approaching an intersection involving one or more other possibly errant vehicles $\mathcal{OV}s$; compute the escape maneuver u^* that minimizes the threat level $\mathcal{T}$ that the $\mathcal{HV}$ incurs over a fixed time horizon T_h.*

In this work, we assume the region of interest is localized to a finite volume around the host vehicle, called the *active region* and represented by a detection radius. This assumption bounds the number of vehicles affecting decision making, and partially represents the limitations of the sensors and wireless communications systems likely to be installed on the vehicles (Section 5.3.2).

We consider $\mathcal{HV}$ to be safe if it avoids colliding with all other vehicles, denoted here as $\mathcal{OV}_i$, $i = 1 \dots N$. This constraint can typically be written in the form

$$\min_{t_0 \le t \le t_f} \|p_s(t) - p_{r_i}(t)\| > \epsilon \quad \forall i = 1 \dots N, \tag{5.1}$$

where $s(t)$ and $r_i(t)$ are the states of $\mathcal{HV}$ and $\mathcal{OV}_i$ at time t, respectively, $p_s(t)$ is the position vector corresponding to $s(t)$, $p_{r_i}(t)$ is the position vector corresponding to $r_i(t)$, ϵ is the minimum allowed distance between the $\mathcal{HV}$ and the $\mathcal{OV}s$, and $t_f = t_0 + T_h$. We say that $\mathcal{HV}$ is in collision with $\mathcal{OV}_i$ if the tuple $(p_s(t), p_{r_i}(t))$ enters the closed "collision" set Ω_i, whose boundary is given by the scalar equation

$$\|p_s(t) - p_{r_i}(t)\| = \epsilon. \tag{5.2}$$

In typical (i.e. uncertain and/or non-cooperative) driving scenarios, the future trajectories $r_i(t)$, $t \in [t_0, t_f]$ of the $\mathcal{OV}s$ are not known bythe $\mathcal{HV}$. Moreover, some realizations of $r_i(t)$ may cause (5.1) to become infeasible for every admissible escape maneuver. Let $u : [t_0, t_f] \mapsto U$ denote the set of all admissible control sequences. To maximize the safety of $\mathcal{HV}$, we compute the admissible escape maneuver u^* that

minimizes the threat level $\mathcal{T}$, defined below, for all possible realizations of $r_i(t)$, $i = 1 \dots N$.

Definition 2 (Threat Level) *Define the threat level $\mathcal{T}_i(u)$ corresponding to vehicle $\mathcal{OV}_i$ and time parametrized escape maneuver $u(t)$ as inversely proportional to t_{c_i}, the earliest possible time of collision:*

$$\begin{aligned} \mathcal{T}_i(u) &= \frac{1}{t_{c_i}(u)}, \\ t_{c_i}(u) &= \inf\{t \mid (p_s(t), p_{r_i}(t)) \in \Omega_i\}. \end{aligned} \tag{5.3}$$

The threat level $\mathcal{T}$ corresponding to escape maneuver $u(t)$ is then defined as

$$\mathcal{T}(u) = \max_{i \in 1 \dots N} \mathcal{T}_i(u). \tag{5.4}$$

Definition 3 (Best Escape Maneuver) *The best escape maneuver u^* is defined as*

$$u^* = \arg\min_{u \in U} \mathcal{T}(u). \tag{5.5}$$

Note that in (5.3), $p_s(t)$ is generated by the escape path $u(t)$.

5.3 Solution Approach

The proposed threat assessment provides assistance to the host vehicle $\mathcal{HV}$ in navigating dynamic environments populated by multiple other vehicles with uncertain intentions. The approach is demonstrated in intersection scenarios, but can be generalized to arbitrary road geometries. The assessment is computed in a threat assessment module (TAM) that is designed as a black box (Figure 5-1), allowing it to be easily incorporated in a driver assistance system. The TAM presented in this work is an adaptation of the architecture developed in Ref. [32] and presented in Section 4.1 to generate safe trajectories for autonomous vehicles. high-level architecture of the TAM is

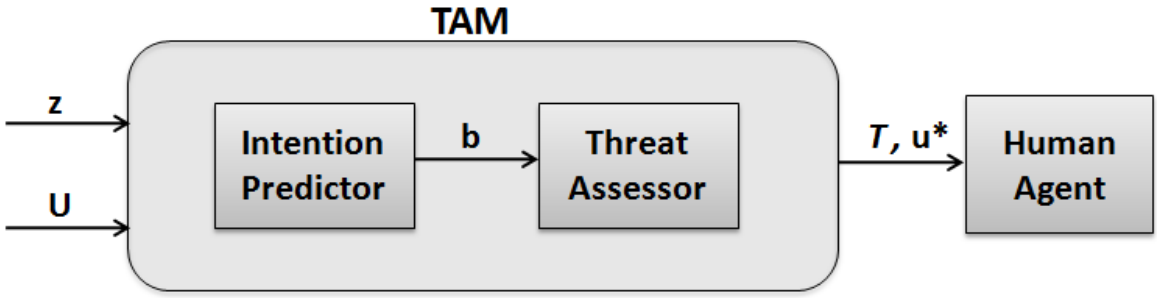


Figure 5-1: High-level architecture of the Threat Assessment Module (TAM). The inputs of the TAM are the measurement history z of the $\mathcal{OV}s$ and the set of control escape sequences U generated for the $\mathcal{HV}$. The IP computes the intention vector b for each $\mathcal{OV}$, used by the TA to identify the threat level $\mathcal{T}$and best escape maneuver u^*.

5.3.1 High-level Architecture

The TAM consists of two main components, an intention predictor (IP) and a threat assessor (TA). The IP incorporates high-level reasoning in modeling the intentions of the $\mathcal{OV}s$, and uses observations to make a prediction of future behaviors. Such a predictor might, for example, classify the threat posed by other vehicles based on whether or not their modeled intentions conform to "typical" driving behavior [96, 123]. The TA converts this prediction into a set of potential paths that the $\mathcal{OV}$ is likely to follow. The threat level can then be evaluated for each candidate $\mathcal{HV}$ escape maneuver, in order to identify the best escape maneuver.

Several distinct representations have been proposed to model intentions for humans and autonomous systems; here we adopt the Ecological Recogniser architecture introduced by Ref. [95] and shown in Figure 5-2. Using the language of the figure, the $\mathcal{HV}$ is the recognising agent, while each $\mathcal{OV}$ is an intending agent. It is implicitly assumed that the list of intentions fully partitions the space of all possible behaviors for the intending agents. We will also follow the abstract definition that these intentions are directly responsible for the actions or activities executed by the intending agents [95]. Intentions may be desired plans (e.g., following a straight path) or other high level intention states (e.g., not following the rules of the road).

Let the M-vector b_i denote the intention vector of $\mathcal{OV}_i$; the j^{th} entry b_i^j corresponds

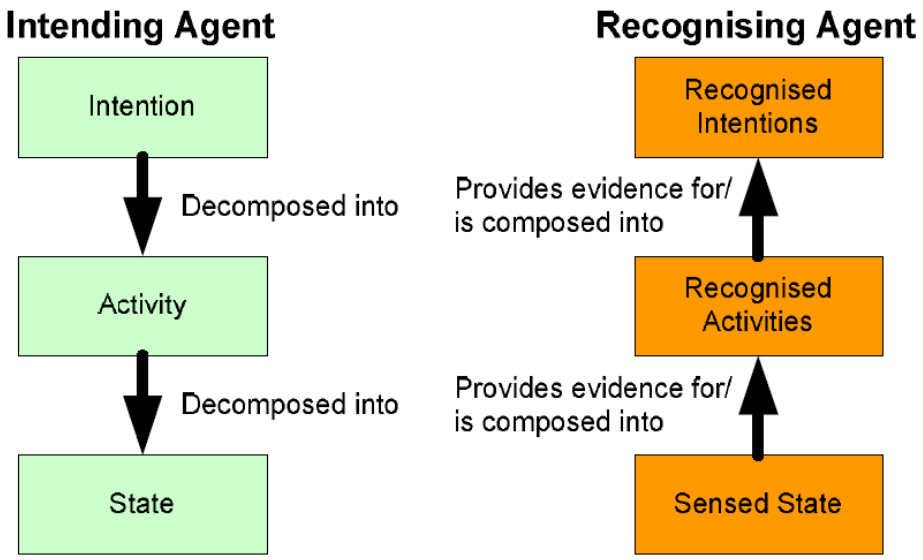


Figure 5-2: The intentional behavior of the Intending Agent and the intention recognition of the Recognizing Agent. Adapted from Ref. [95].

to the belief $\mathcal{OV}_i$ is operating under the jth intention. The definition of the threat level (5.4) is modified to embed the expectation of each vehicle's threat over all possible intentions,

$$\mathcal{T}(u(.)) = \max_{i \in 1...N} \left(\sum_{j \in 1...M} b_i^j \mathcal{T}_{ij}(u(.)) \right) \tag{5.6}$$

where $\mathcal{T}_{ij}$ is the threat value of the trajectories followed by $\mathcal{OV}_i$ as a realization of its j^{th} intention.

5.3.2 Agent Model

Each agent is modeled using the standard bicycle model

$$\begin{aligned} \dot{x} &= v\cos(\theta), \quad \dot{y} = v\sin(\theta), \\ \dot{\theta} &= \frac{v}{L}\tan(\delta), \quad \dot{v} = a, \end{aligned} \tag{5.7}$$

where (x, y) is the rear axle position, v is the forward speed, θ is the heading, L is the wheelbase, a is the forward acceleration, and δ is the steering angle (positive counter-clockwise). The state of the vehicle is $s = (x, y, \theta, v) \in \mathcal{S}$, while the input is $u = (\delta, a) \in \mathcal{U}$, including the constraints $a_{\min} \leq a \leq a_{\max}$ and $|\delta| \leq \delta_{\max}$.

In the work that follows, we make several assumptions to constrain the focus on uncertainty in the prediction of obstacles or other agents. We assume that the current states and models of the $\mathcal{HV}$ and the $\mathcal{OV}s$ are perfectly known by the $\mathcal{HV}$ within the active region, and that all sensor measurements are noise-free. While this assumption is somewhat unrealistic, the addition of estimation filters on noisy measurements should have a minimal effect on the TAM architecture. The focus of this chapter is on the planning subproblem, rather than sensing. Additionally, the TAM is provided with a complete map of the environment *a priori*, excluding any dynamic obstacles or agents. Finally, the number of escape control functions of the $\mathcal{HV}$ is assumed to be finite. Since the $\mathcal{HV}$ typically follows the rules of the road, it is reasonable to predefine some typical escape maneuvers that a "good" host driver might follow, e.g., maximum braking or an increase in throttle.

5.4 Developed Algorithms

This section describes the implementation of the different components of the TAM, tailored specifically for the road intersection problem. The IP component is based on the Ecological Recogniser architecture presented in Ref. [95]. The TA component combines knowledge of the intentions produced by the IP and domain-specific information of the environment to quickly propagate reachable paths for the moving vehicles using a reachability-based algorithm. Then, for each admissible escape maneuver, it evaluates the threat generated by these paths. It finally returns the maneuver with the lowest threat level over all vehicles. The result is an intention-reasoning TAM that runs efficiently and is suitable for real-time implementation.

5.4.1 Intention Predictor (IP)

The IP translates observations that $\mathcal{HV}$ makes of the other $\mathcal{OV}s$ into a prediction of their intentions. Instead of using low-level reasoning to predict future *trajectories*, the IP uses higher-level logic to provide the TAM with predicted future *intentions*, which is key to more timely and accurate prediction of collisions. Each intention can then

be translated to a richer set of future paths that exploit domain-specific knowledge using the TA (Section 5.4.2).

The chosen Ecological Recogniser architecture [95] is applicable to our problem because it is designed for an intelligent agent trying to recognize the intention of other agents with access to their state data but not actions. Additionally, this architecture can handle intentions which are not easily described by a simple set of rules, as can be the case in this problem. The Ecological Recogniser architecture has two key components, a pattern matcher and a reasoning module. The pattern matcher is a classifier that is trained offline to recognize the different intentions by observing agent trajectories, and operates online by continuously giving an estimate of the intentions over time. The reasoning module filters these estimates of intentions, along with knowledge of previous encounters with the agent, to give a final estimate of the intention vector b.

Since the focus is on estimating the intentions of vehicles near intersections, we use a specific implementation of the Ecological Recogniser that is based on an approach previously developed and demonstrated in simulation [96] for classifying other agent intentions using support vector machines (SVM) and Bayesian filtering. Even before considering autonomous vehicles, the human driver intention classification problem is very complex because of the various nuances of human behaviors. SVM is a suitable method for the IP because it has been shown to be a robust and efficient approach for classification problems [54], such as lane-change detection [51]. The IP design consists of an SVM combined with a Bayesian filter that uses the SVM outputs over a specific time period to compute the intention vector b. SVM-BF also includes a threshold detector, so the final b vector is a unit basis vector specifying the most likely intention according to the threshold value. The SVM-BF was trained using human driving data collected in RAVEN [89]. For more details about the SVM-BF approach, please refer to [96]. Based on experimenting with different kernel functions and several combinations of features, the best results were obtained using the Gaussian radial basis function and combining the following three features:

- the relative distance Δx between the $\mathcal{OV}_i$ and the entrance of the intersection

it is approaching;

- the speed of the $\mathcal{OV}_i$;
- the longitudinal acceleration of the $\mathcal{OV}_i$.

The SVM-BF algorithm is only activated when the distance between the $\mathcal{OV}_i$ and the $\mathcal{HV}$ is within some danger radius, and both vehicles are approaching the same intersection, which is representative of a limited computation budget and restricted knowledge of the world.

Finally, note that the SVM-BF is presented in the context of intention prediction rather than behavior classification (as in Chapter 2) because there is an implicit assumption in this chapter that the choice of following the rules or not is intentional rather than behavioral. In other words, the $\mathcal{OV}s$ consciously and intentionally decide whether to be errant or not. They are not unconsciously following a behavior that could be non-compliant.

5.4.2 Threat Assessor (TA)

The TA (Algorithm 8) computes a threat value of a candidate escape maneuver for the $\mathcal{HV}$ given a knowledge of the current states of the $\mathcal{OV}s$. It combines a fast sampling-based reachability method with intention prediction information provided by the IP (Section 5.4.1) to efficiently estimate the threat level. A similar version of the TA algorithm that assumes worst-case behavior for the other drivers was developed in Ref. [35]. The threat assessor has a finite time horizon, limiting the $\mathcal{HV}$ lookahead horizon of the possible future paths of the other $\mathcal{OV}s$, in order to focus calculations on imminent threats. This choice is domain-specific, but it should be long enough to allow the $\mathcal{HV}$ sufficient time to react in a dynamic environment. To obtain the best escape maneuver u^*, the TAM calls Algorithm 8 for each each maneuver u in the set of escape maneuvers, and returns the maneuver with the minimum threat level.

Algorithm 8 first calls the Compute-Intention-Reachability subroutine (Algorithm 9) on line 3 to create the reachability set for each $\mathcal{OV}_i$. The resulting sets are biased

Algorithm 8 Intention-based-Threat-Assessment (u)

1: $\mathcal{T}_i \leftarrow 0 \quad \forall i$
2: **for** each agent $\mathcal{OV}_i$ **do**
3: $\quad$ *Reach-Set* $\leftarrow$ Compute-Intention-Reachability $(\boldsymbol{b}_i)$
4: $\quad$ **for** each intention e_j with probability b_i^j **do**
5: $\quad\quad$ $\mathcal{T}_i \leftarrow \mathcal{T}_i + b_i^j \cdot$ Eval-Threat(*Reach-Set*,$\mathcal{OV}_i, e_j, u$)
6: $\quad$ **end for**
7: **end for**
8: **return** $\max_{i=1\ldots N} \mathcal{T}_i$

based on the perceived intentions of the $\mathcal{OV}s$. For each intention e_j of each $\mathcal{OV}_i$, Algorithm 10 is called to compute the threat incurred by the $\mathcal{HV}$ escape maneuver u in the region reached by the paths that correspond to the intention e_j. Note that the earliest time to collision is converted into a threat value using (5.3) (line 3 of Algorithm 10). This value is weighted by the probability b_i^j provided by the IP. Finally, the threat returned is computed as the maximum value of all threats created by each $\mathcal{OV}_i$ (line 8 of Algorithm 8). Note that, in practice, the computed reachability sets for each $\mathcal{OV}_i$ are saved between successive calls to Algorithm 1.

Algorithm 9, also referred to as the RRT-Reach algorithm, was presented in Section 4.1 in the context for threat-aware motion planning and is reproduced in this section for clarity. Algorithm 9 extends the rapidly-exploring random tree (RRT) [84, 97] algorithm, which grows a tree by randomly sampling points toward which dynamically feasible trajectories are simulated. In particular, it uses the closed-loop RRT (CL-RRT) algorithm of Ref. [31], which samples inputs to a controller rather than the vehicle itself. The algorithm thus maintains the exploration bias of traditional RRT algorithms, while allowing for generation of smooth trajectories more efficiently. A key advantage of the RRT algorithm is the scalability of the planning tree to use whatever computational resources are available.

Unlike traditional RRT approaches, the RRT-Reach tree is not used to identify some path that reaches a goal location. Rather, the entire tree is analyzed to find the maximum threat along each of the trajectories. The choice of samples is designed to be biased towards regions corresponding to the learned intentions of the $\mathcal{OV}s$. It is done through sampling some percent of the time in regions corresponding to the

Algorithm 9 Compute-Intention-Reachability (b)

```
Measure current vehicle state and environment
repeat
    Take sample for input to controller using bias from intention probability vector b
    repeat
        Update time range in time heuristics
        Find list of nearest neighbors using time range
        Sort list using distance heuristics
        for each sorted node do
            Call propagation function
            if propagated portion is collision free then
                Add sample to Tree break
            end if
        end for
    until timestamp reaches time horizon and no collision free portion was found
until time limit for growing tree is reached
return Tree
```

Algorithm 10 Eval-Threat (*Reach-Set*, $\mathcal{OV}$, e, u)

```
for each path path_k in Reach-Set that ends in a region of intention e do
    t_k ← compute earliest time of collision of path_k with HV escape maneuver u within
    time horizon T_h
    T_k ← 1/t_k
end for
return max_k T_k
```

component e_j of intention e. The remaining percent of the time is spent sampling uniformly in the environment. Note that Algorithm 9 was presented in Section 4.1 in the context for threat-aware motion planning, and is reproduced in this section for clarity.

Algorithm 9 includes several additional extensions to the RRT approach introduced in Ref. [31], based on time parameterization of the RRT tree. The extensions tailor the RRT algorithm to the efficient computation of intented paths by the $\mathcal{OV}s$. Refer to Section 4.1 for more details.

5.5 Experimental Results

This section presents experimental results which validate the effectiveness of the TAM in assisting human drivers approaching intersections in the presence of possibly er-

rant drivers, subject to real-world uncertainty. These results verify that the TA can accurately identify drivers who do not observe a stop sign and enter the intersection out of turn, constituting a violation of the rules of the road. The results also show that the IP is capable of providing the human driver with the correct course of action, based on the intention and likely paths of the errant driver.

Including a human driver in the experiment helps to validate the TAM, since it will ultimately be used to assist human drivers. For example, the response to TAM alerts can be measured to determine the appropriate notification time. The autonomous driver can also be used to emulate a variety of human driving behaviors acquired from actual urban traffic data within the testbed.

The hardware results are presented after an overview of the experimental infrastructure, including testbed, hardware, and software. A video showing several different experimental scenarios, including the results below, is available at http://acl.mit.edu/ITSC10TAM.mov and http://hdl.handle.net/1721.1/46720 [124].

5.5.1 Testbed

Hardware demonstrations were performed in the Real-time indoor Autonomous Vehicle test ENvironment (RAVEN), a testbed designed for the rapid prototyping of decision-making and control algorithms for unmanned aerial and ground vehicles [89]. Vehicle state data is collected using motion-capture cameras [125], which detect each vehicle via a unique pattern of attached reflective dots. Many different configurations can be tested with minimal setup effort, yielding high-fidelity state data for potentially dangerous driving scenarios without risk of injury.

A representative road network was constructed within the RAVEN testbed, including multiple intersection types and road signs, to emulate a real-world driving environment for testing the TAM (Figure 5-3). The road network is 11.2 m $\times$ 5.5 m in size, and is capable of accommodating multiple intersection types and as many as 10 vehicles running simultaneously.

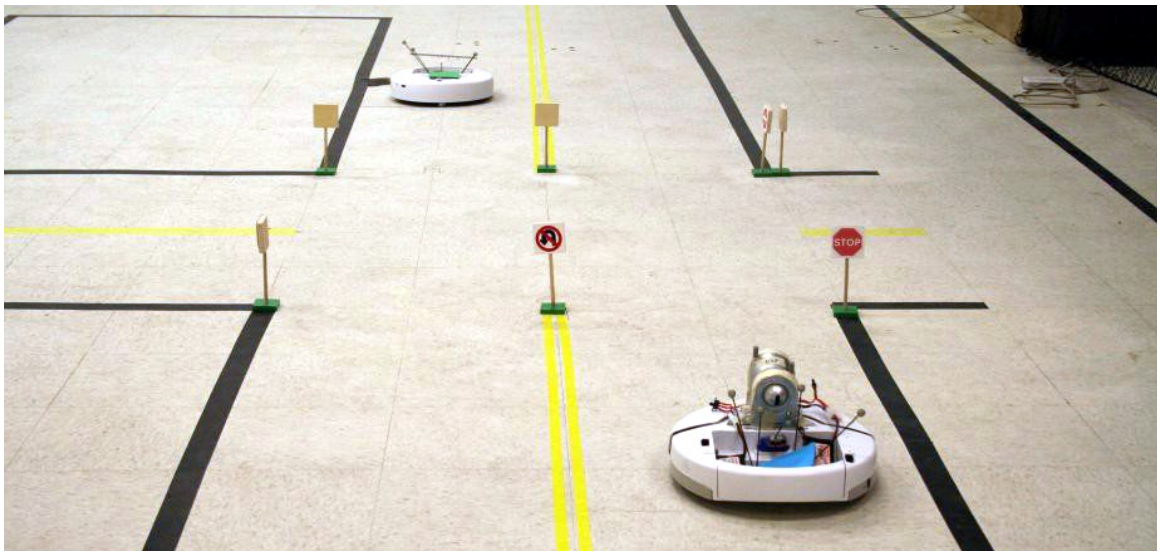


Figure 5-3: The road network constructed within the RAVEN testbed to test the TAM algorithm. Here both a human-driven vehicle (front-right) and autonomous vehicle (back-left) are approaching a four-way, stop-sign intersection. The rear-mounted camera, one of two, is clearly visible on the human-driven vehicle.

5.5.2 Hardware Infrastructure

All vehicles in this experiment use the iRobot Create platform [87]. The vehicle is a 2-wheeled, skid-steered vehicle with a maximum speed of 0.5 m/s and near-instantaneous acceleration. A software wrapper imposes rate limits in acceleration and wheel speed differences, such that the vehicle emulates traditional automative dynamics (5.7).

Each experiment consists of one human-driven vehicle and one autonomous vehicle. The human-driven vehicle is controlled through a wireless steering wheel [100], including acceleration and brake pedals. The mapping of the steering and pedal inputs to the vehicle motion has also been tuned to emulate traditional control of an automobile, including turning, acceleration, and braking behaviors. The human driver may steer their vehicle through direct visual inspection of the testbed, or via a "virtual dashboard" interface which simulates the first-person driving perspective (Figure 5-4). Two cameras mounted onboard the human-driven vehicle provide real-time visual feedback in both the forward and rear directions (Figure 5-3), while a world map "GPS" guides the driver through the desired sequence of navigation way-

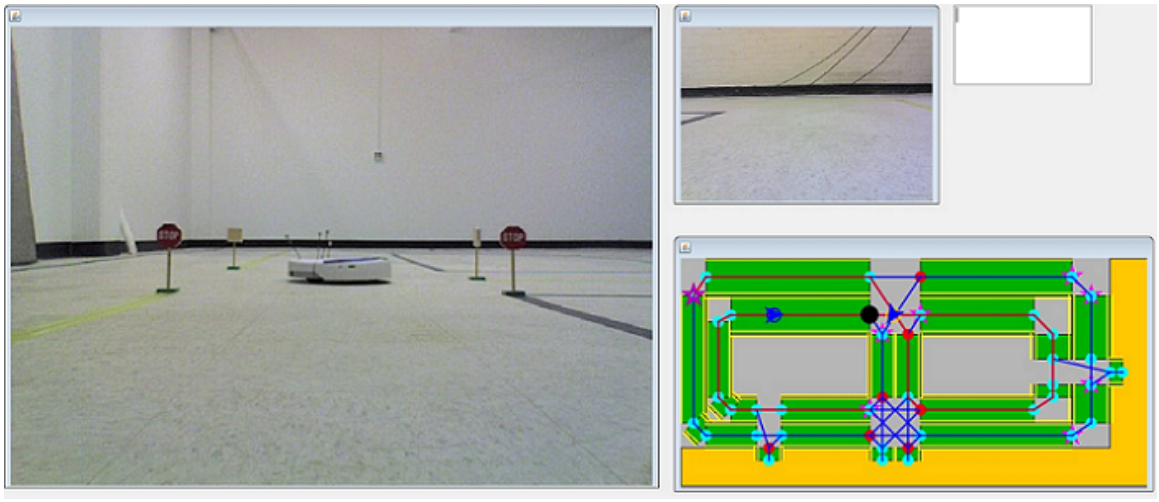


Figure 5-4: Human interface for the human-driven vehicle, including real-time video feeds of the forward (left) and rear (top-center) directions of the vehicle, a dialog box (top-right), and a world map "GPS" guiding the driver to their next waypoint.

points. The autonomous vehicle is controlled through an off-board planning software (Section 5.5.3).

5.5.3 Software Implementation

The TAM is implemented across three software modules for the host vehicle. The *Intention Predictor* classifies each driver according to the observed behaviors. The *Threat Assessor* determines the likely paths of those vehicles based on the perceived behaviors. Finally, if a vehicle poses a threat, the *Escape Path* module calculates suitable evasive trajectories for the human driver and offers the best course of action given the errant driver's paths. A vehicle is perceived to be a threat if three conditions are satisfied: (1) the IP module has classified the vehicle's behavior as errant; (2) the vehicle is sufficiently close to the host vehicle (within the black circle in Figure 5-5); and (3) both vehicles are approaching the same intersection.

The Intention Predictor module accesses observations of the states of each vehicle on the road network and provides these inputs to the classifier. Depending on the high-level behaviors, the classifier determines the most probable intent of each vehicle in the environment. In this work, classification is performed on an autonomous vehicle; the same classification process is performed on a human-driven vehicle in

Ref. [32].

The Threat Assessor module (Algorithm 8) predicts paths for each vehicle, based on the perceived intentions, using the Compute-Intention-Reachability module (Algorithm 9). This latter module is embedded in the external planner as a separate thread which runs asynchronously for each vehicle $\mathcal{OV}_i$ in the environment. Over fixed time intervals (1 s), the algorithm grows a new reachability tree based on the current $\mathcal{OV}$ position in the testbed. At the end of each time interval, the reachability tree is relayed to the host vehicle thread, which then calls the Escape Path algorithm (Section 5.4.2) if the threat conditions are met.

The Escape Path module (Algorithm 10) generates evasive maneuvers by simulating multiple possible actions of the human driver for sufficiently long time intervals (here 6 s), and determining whether and/or when the resulting trajectories enter the time-parameterized reachable set of the errant vehicle. In the current implementation, three possible actions are evaluated for the human driver – maintaining current speed, maximum acceleration (up to 0.4 m/s), or maximum braking. Escape paths are measured against the intention-reachability paths to determine the time to a potential collision, with the minimum-threat escape maneuver identified via (5.5). If the best escape maneuver requires acceleration or braking, the human driver is alerted to do so; no alert is provided if the driven can safely maintain their current speed over the prediction time horizon.

The autonomous vehicle is controlled via an offboard, multi-threaded Java implementation of the standard CL-RRT algorithm [31]. This is the same framework as used for autonomous vehicles in Ref. [32], but without the threat-aware components; only the human-driven vehicle is assessing the threat level of other vehicles. Instead, the autonomous agent simply treats other vehicles as static obstacles to avoid. The vehicle is controlled via pure pursuit steering control [99] and proportional-integral speed control. When approaching each intersection, a weighted coin flip within the autonomous vehicle logic determines whether or not the vehicle will properly decelerate and come to a stop.

A top-level navigator module is used in both vehicles' software packages to provide

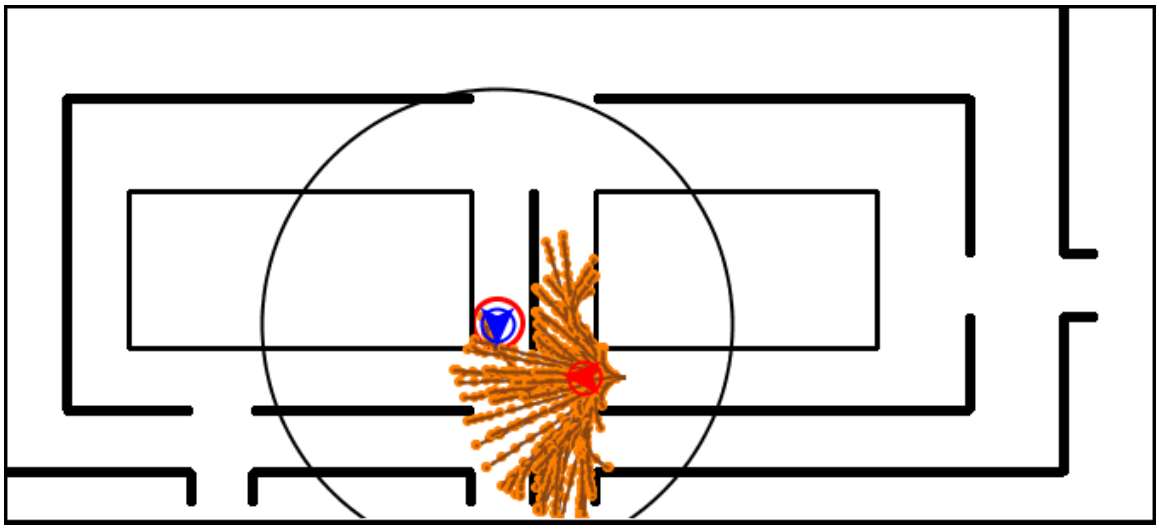


Figure 5-5: Display representation of the RAVEN road network, including the host vehicle (blue chevron; detection radius in black), autonomous vehicle (red chevron), and reachability tree (brown edges, orange nodes). In this figure, the human driver has been alerted to stop to avoid an imminent collision. See Figure 5-6 for a full legend.

a sequence of geographic waypoints. The navigator module leverages a sparse representation of the RAVEN road network, constructed using the Route Network Data File (RNDF) specification from the 2007 DGC [101]. This representation includes the location, size, and connectivity of lane segments and "zones" (e.g., parking lots); intersections are implicitly defined by the exit-entrance connections between lane/zone endpoints. An A^* implementation is used to select the shortest-distance waypoint path in the graph between the current and desired waypoints, respecting lane directionality constraints [14]. As the vehicle (whether human or autonomous) approaches each intermediate or desired waypoint, the navigator selects the next waypoint on the list as the new target. An arbitrary list of desired waypoints is provided for each vehicle; the ultimate objective is to observe the vehicles' interactions at intersections. The diagram at bottom-right in Figure 5-4 shows the RNDF for the RAVEN road network, constructed from a simple text file. Figure 5-5 shows the display representation of the world used in the results below.

Given the geographic waypoint sequence, each escape path is constructed through forward simulation of the vehicle model connecting these waypoints, in a manner

similar to the RRT algorithm.

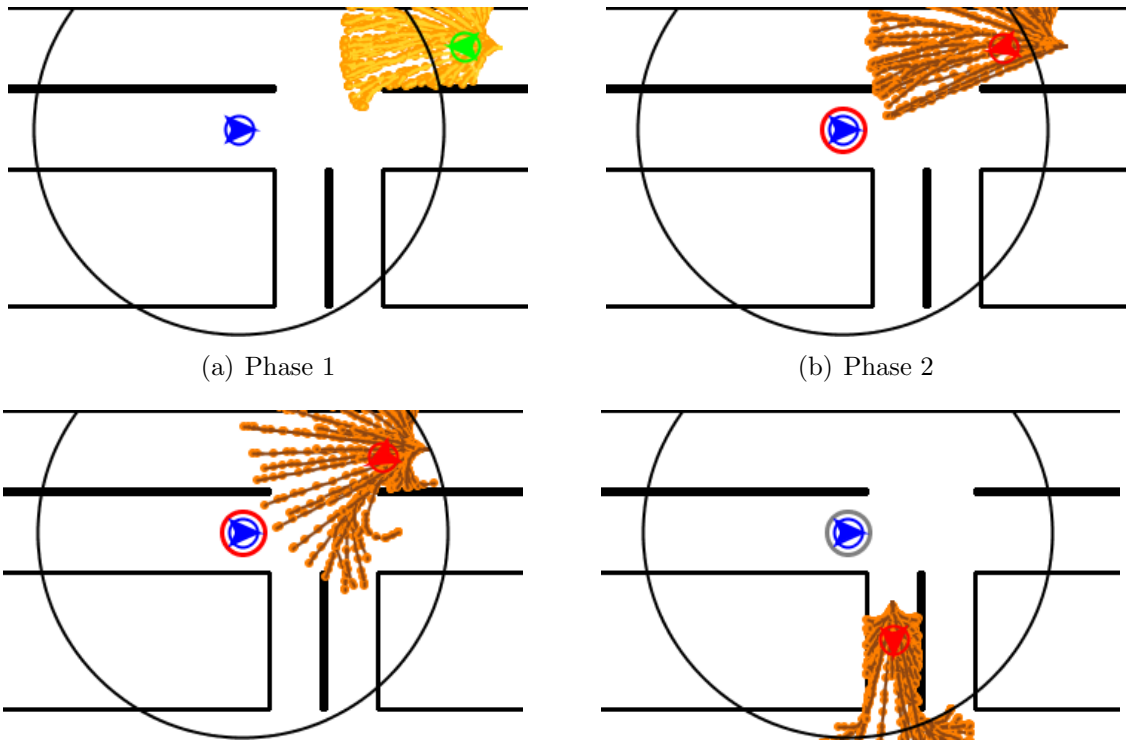
5.5.4 Results

The following results demonstrate the TAM providing a human driver with feedback during a simultaneous intersection approach with another driver. In this hardware demonstration, both the human-driven vehicle and autonomous vehicle are navigating through the RAVEN road network simultaneously, following a random sequence of waypoints. While the human driver is to respect all rules of the road, the autonomous vehicle may choose to not stop at an intersection and instead enter it out of turn (Section 5.5.3). In this case, the TAM should identify the autonomous driver as errant based on observed behavior, and use the reachability tree to alert the human driver if a corrective action is necessary.

Over the course of this experiment set, 20 intersection encounters occurred. Of these encounters, only two resulted in a collision; in both cases the human-driven vehicle did not react on time to the warning of the TAM algorithm, making the collision inevitable with the errant autonomous vehicle. An improved tactile or auditory warning and a more involved model of the driver reaction time should improve the response of the drivers, thus reducing the number of accidents.

Figure 5-6 shows four successive timesteps, or phases, of a representative scenario where the human driver interacts with an autonomous driver at one of the RAVEN intersections. Figure 5-7 shows the threat levels incurred by executing either the "same speed," "accelerate," or "stop" escape maneuvers at each. In this scenario, both the human and autonomous vehicles are attempting to enter the southbound lane of the intersection: the human driver is approaching from the left and making a right turn, while the autonomous vehicle is approaching from the right and making a left turn.

The human driver reaches the intersection first, and has right-of-way to proceed into the intersection (Figure 5-6(a)). The autonomous driver behavior is classified by the IP as normal, and the TAM does not post any warnings because the autonomous vehicle is outside the host vehicle's detection radius. After stopping for



(a) Phase 1 (b) Phase 2

(c) Phase 3 (d) Phase 4

Figure 5-6: Zoomed-in display view of a representative interaction between the human-driven host vehicle (blue chevron) and autonomous vehicle at an intersection; each figure shows a successive snapshot of this interaction. The autonomous vehicle is classified in real-time as safe (green chevron) or errant (red chevron). If the autonomous vehicle is perceived to be a threat, the reachability tree darkens, and the outer ring around the host vehicle indicates the alert type (grey = maintain speed, red = brake, green = accelerate). Escape paths are not shown.

several seconds, the human driver accelerates into the intersection, anticipating that the autonomous driver will decelerate. However, the IP observes the autonomous driver is actually accelerating into the intersection, and classifies the driver as errant (Figure 5-6(b)). The TA module generates the autonomous driver's reachability tree, and the escape path module determines the threat of either proceeding at the same speed, accelerating, or stopping. Since stopping minimizes the threat level (Figure 5-7), the TAM alerts the human driver to stop. (Even though the reachability tree does not reach the human-driven vehicle, the escape paths—which are not shown—do intersect with the set.) As the autonomous driver continues to move, the human driver stops within a few seconds, in response to the TAM's recommendation (Fig-

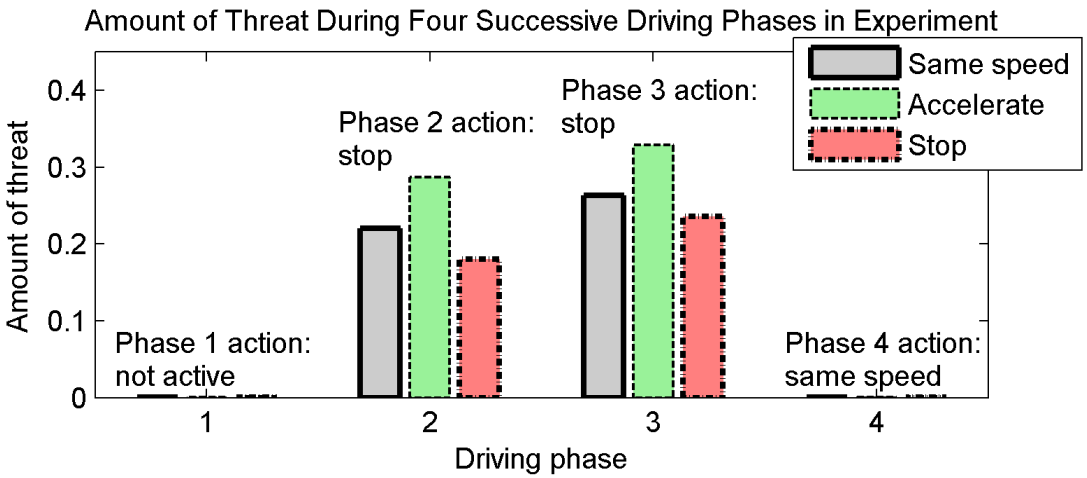


Figure 5-7: Threat levels for the possible escape paths during each phase of Figure 5-6.

ure 5-6(c)). Once the errant driver clears the intersection, the TAM determines that all actions are safe, causing the alert to switch off (Figure 5-6(d)).

Figures 5-8 and 5-9 present snapshots of a hardware demonstration of a particular scenario involving a human driven-vehicle and a possible errant autonomous driver. It is known as the Left Turn Across Path / Opposite Direction (LTAP/OD). It is one of the most dangerous and fatal intersection encounter [23, 102]. The human-driven vehicle is approaching from the right, and arrives first to the intersection. Before proceeding into the intersection, it detects (through the IP) that the autonomous-driven vehicle approaching from the left is not decelerating (to stop at the intersection) and violating the stop sign. The TAM warns the driver by suggesting (through the TA) a stopping maneuver to minimize the threat level $\mathcal{T}$ (see the red outer ring around the human-driven vehicle in Figure 5-8(a)). The human driver responds by decelerating until coming to a full stop. The TAM then removes the warning as the threat of the current maneuver (i.e., being stopped) is lower than the other escape maneuvers (see the grey outer ring in Figure 5-8(b)). However, the human driver decides to accelerate again, and the TAM recognizes the threat of the autonomous vehicle which is still in the intersection. The TAM recommends again the stopping maneuver (Figure 5-9(a)). The human driver comes to a stop again, and the the threat caused by the autonomous vehicle vanishes (Figure 5-9(b)). This scenario shows how the TAM detects in real-time the threat of an errant vehicle. It also demonstrates the

TAM escape maneuvers suggestion for the human-driven vehicle dynamically changes depending on the errant vehicle and the response of the human driver.

As mentioned above, a video showing several different experimental scenarios along with detailed captions is available at http://acl.mit.edu/ITSC10TAM.mov and http://hdl.handle.net/1721.1/46720 [124]. Finally, this work has been featured on a TV documentary that highlights the latest high-tech research and technologies. The documentary can be watched online at http://www.youtube.com/watch?v=w_VWsaPEalA.

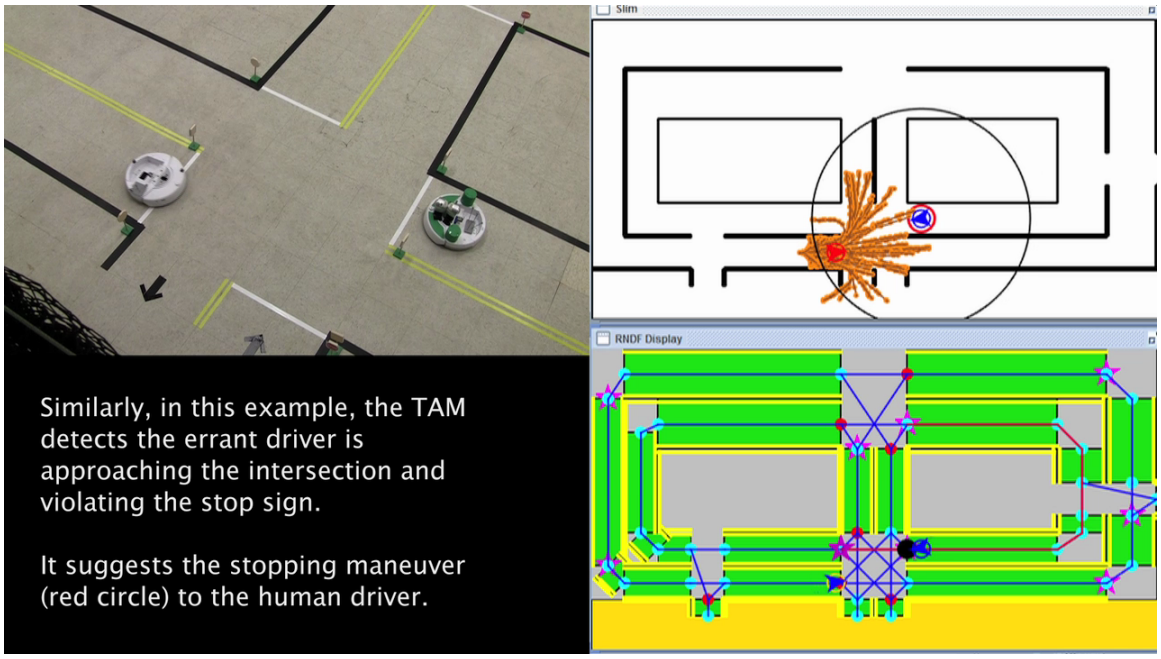
5.6 Pursuit-Evasion Formulation

This section introduces a formulation that extends the RRT-Reach algorithm to include a game-theoretic component. This formulation considers the worst-case scenario of errant drivers capable of causing intentional collisions [35].

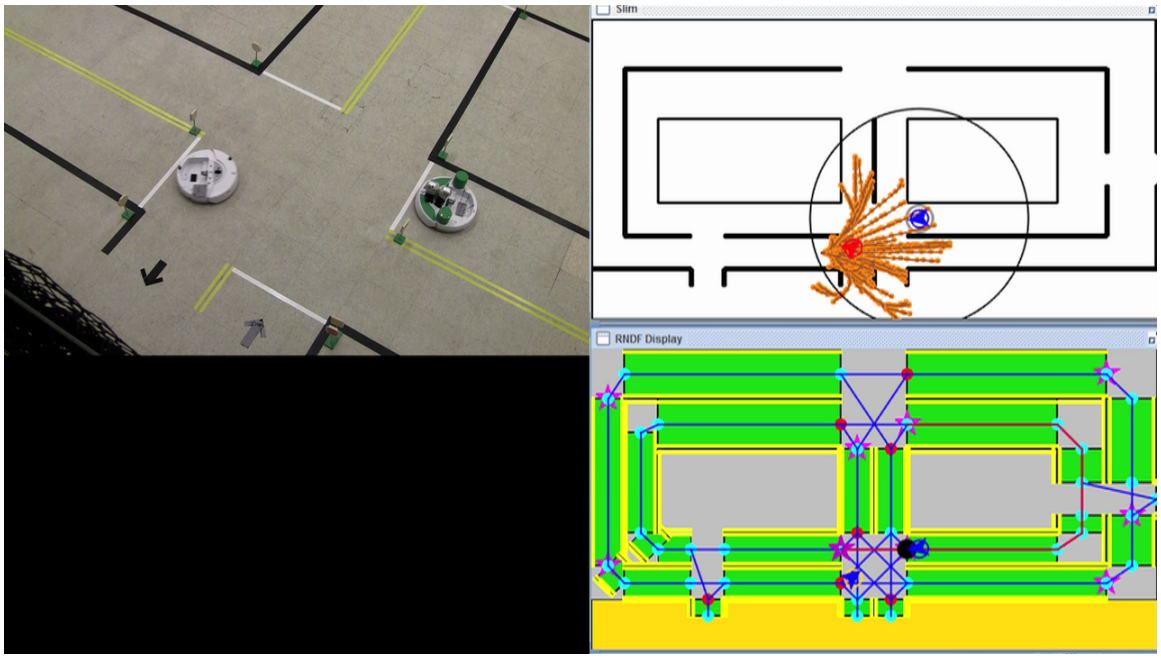
5.6.1 Problem Statement

To bound the number of vehicles affecting the decision making, the region of interest is localized to a finite volume around the intersection, called the *active region* (See Section 5.2). Every vehicle in the active region is assumed to be classified as either predictable or errant by a classification algorithm [96], part of the onboard threat assessment module. To simplify the problem, it is assumed that only two agents are involved: an errant vehicle, modeled as a hostile agent which may not be following the rules of the road; and a host vehicle, assisted by the threat assessment algorithm to minimize the threat of an intersection collision. Each vehicle is modeled using the standard nonlinear bicycle model defined in Section 5.3.2, where each admissible control function $u(t)$ is a piecewise constant function $u : [0, T_h] \mapsto U$, and T_h is a finite and fixed time horizon.

Game theoretic formulation of InP: The intersection threat assessment problem (hereafter referred to as InP) can be formulated as a perfect information zero-sum differential game $\mathcal{G}$ with state constraints and free final time [126]. The formulation

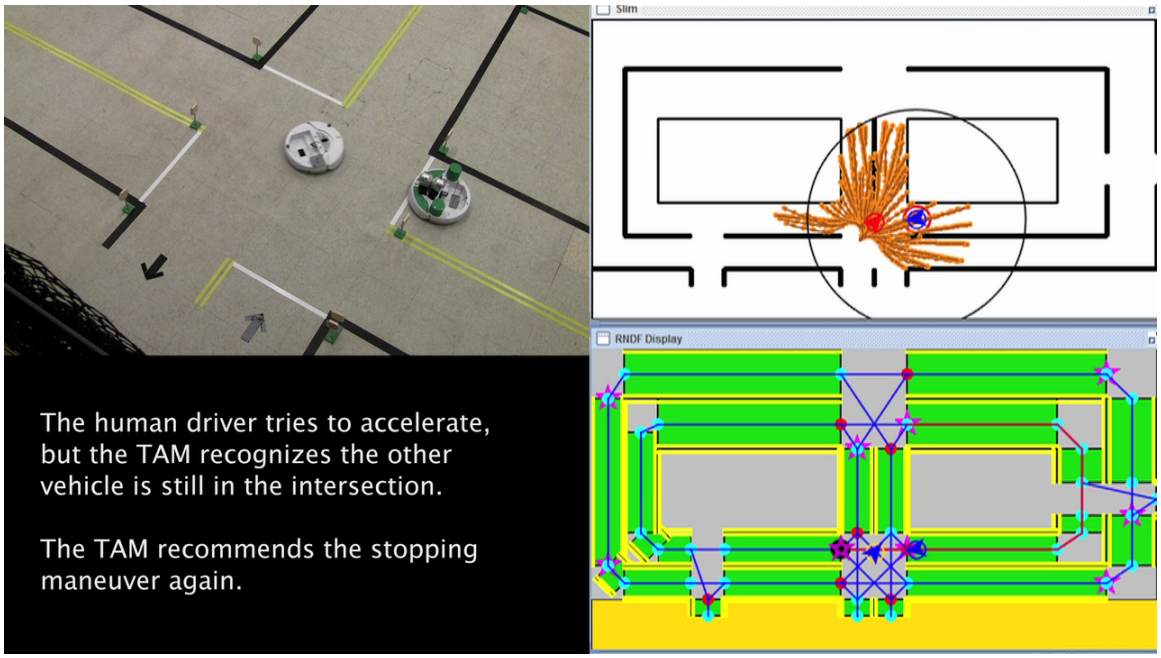


(a) Part 1

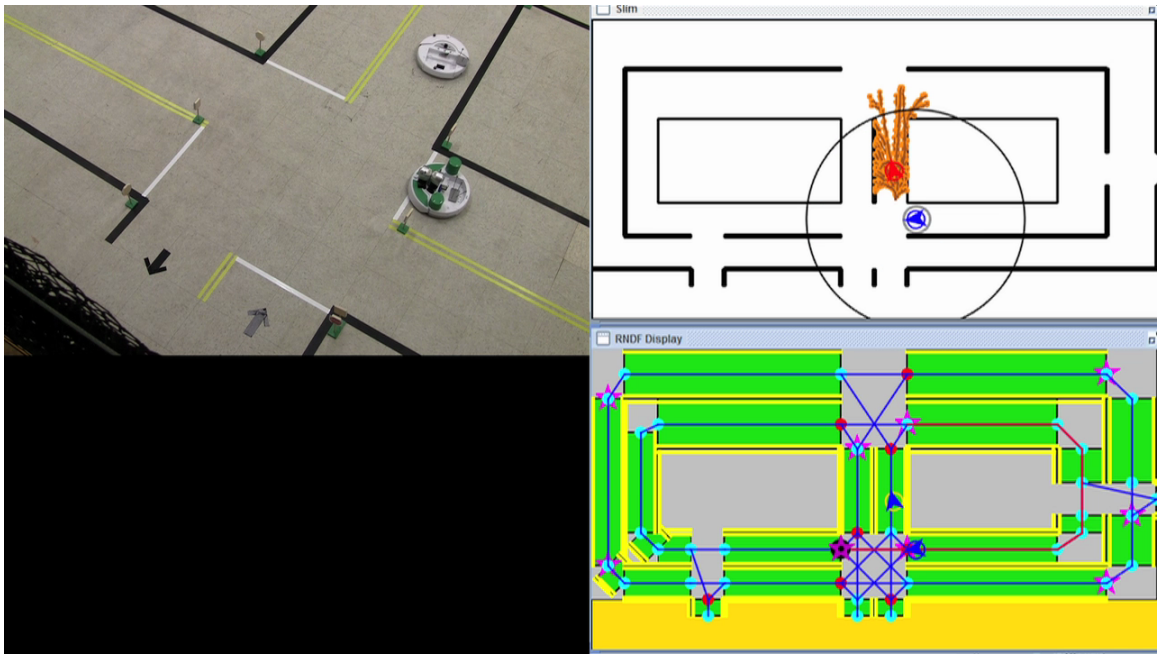


(b) Part 2

Figure 5-8: RAVEN road network view of an interaction between a a human-driven vehicle and a possibly errant autonomous vehicle at an intersection in the three vehicle experiment: Phases 1 and 2. Human-driven vehicle is warned of the threat of the autonomous-vehicle at an LTAP/OD encounter.



(a) Part 3



(b) Part 4

Figure 5-9: RAVEN road network view of an interaction between a a human-driven vehicle and a possibly errant autonomous vehicle at an intersection in the three vehicle experiment: Phases 3 and 4. Human-driven vehicle is warned of the threat of the autonomous-vehicle at an LTAP/OD encounter.

is similar to the one presented in Ref. [127]. Let subscripts 1 and 2 denote the errant vehicle and host vehicle, respectively. If we let $s_i(t)$ and $u_i(t)$ refer to the state and control of vehicle i at time t, where $i \in \{1, 2\}$, then (5.7) can be represented as

$$\begin{aligned} \dot{s}(t) &= \begin{bmatrix} \dot{s}_1(t) \\ \dot{s}_2(t) \end{bmatrix} = \begin{bmatrix} f(s_1(t), u_1(t)) \\ f(s_2(t), u_2(t)) \end{bmatrix}, \\ s(0) &= s_0, \quad t \in [0, \infty). \end{aligned} \tag{5.8}$$

The controls and states are assumed to satisfy the constraints

$$C_i(s_i(t), u_i(t)) \leq 0, \qquad i \in \{1, 2\}, \tag{5.9}$$

Equation (5.9) may include obstacle avoidance constraints, bounds on controls, and bounds on states.

The game terminates with a collision if the state of the game $s(t)$ enters the closed target set Ω, expressed as

$$\|(x_1(t_f), y_1(t_f)) - (x_2(t_f), y_2(t_f))\| \leq \epsilon, \tag{5.10}$$

where ϵ represents the collision distance between the two vehicles, and the free final time t_f of $\mathcal{G}$ represents the time to collision

$$t_f = \inf\{t | (s(t)) \in \Omega\}. \tag{5.11}$$

The value $t_f = +\infty$ indicates the game terminated without a collision. The objective of the host vehicle is to maximize the final time t_f, while the objective of the errant vehicle is to minimize it. Thus the payoff function of $\mathcal{G}$ is simply

$$J(s(t), u(t), t) = t_f. \tag{5.12}$$

Suppose that $\mathcal{G}$ admits a saddle point $(\gamma_1^*, \gamma_2^*) \in \Gamma_1 \times \Gamma_2$ in feedback strategies. If

both players execute their feedback saddle-point strategies, the value of the game is

$$V(s(t),t) = \min_{\gamma_1 \in \Gamma_1} \max_{\gamma_2 \in \Gamma_2} J(s(t),u(t),t) \tag{5.13}$$

$$= \max_{\gamma_2 \in \Gamma_2} \min_{\gamma_1 \in \Gamma_1} J(s(t),u(t),t). \tag{5.14}$$

Computing the feedback solutions requires solving the Hamilton-Jacobi-Isaacs equation [126]. Except for some simple cases, the solution quickly becomes intractable [128]. Instead, open-loop representations of the feedback saddle-point strategies are typically solved. The resulting solutions typically coincide with saddle-point solutions. Please see Ref. [127] and references therein for more details. The above formulation $\mathcal{G}$ can be solved using the technique of Ref. [129], which decomposes the problem into two ordinary optimal control subproblems to be solved iteratively using nonlinear programming. Examples including a car collision avoidance scenario were solved and shown to converge to the solutions obtained by indirect methods. However, it involves solving iteratively nonlinear programs which can be can be time-consuming, with the performance heavily on the quality of the initial guess. Thus, it not a reliable method to solve the InP in real-time.

This section proposes an approximate open-loop solution using sampling-based algorithms, which can efficiently identify feasible solutions for complex motion planning or reachability problems [84, 128, 130].

Sampling-based relaxation of InP game: To find an approximate solution of $\mathcal{G}$ using sampling-based algorithms, we introduce the following two assumptions:

- *The number of control functions (i.e., policies) of the host vehicle is finite.* Since the host vehicle typically follows the rules of the road, it is reasonable to predefine some typical escape paths of a "good" host driver facing a dangerous situation; and,

- *The game $\mathcal{G}$ has a fixed time horizon T_h* (typically on the order of few seconds). The value of T_h should be long enough to ensure that the host vehicle has enough time to execute its escape maneuver in the active region. It also limits the size

of the problem by neglecting capture solutions outside the active region.

Then the sampling-based relaxation of the InP game will approximate the lower value (5.14) of the open loop representation of the InP by solving

$$\tilde{V}(s(t),t) = \max_{u_2 \in \tilde{\mathcal{U}}_2} \min_{u_1 \in \tilde{\mathcal{U}}_1} J(s(t),u(t),t) \tag{5.15}$$

$$\approx \max_{u_2 \in \mathcal{U}_2} \min_{u_1 \in \mathcal{U}_1} J(s(t),u(t),t), \tag{5.16}$$

where $\mathcal{U}_i$, $i \in \{1,2\}$ represents the set of admissible control functions for each vehicle, and $\tilde{\mathcal{U}}_1$ and $\tilde{\mathcal{U}}_2$ are subsets of $\mathcal{U}_1$ and $\mathcal{U}_2$ respectively. Here $\tilde{\mathcal{U}}_1$ is computed using a sampling-based algorithm (Section 5.6.2) that is probabilistically complete, while $\tilde{\mathcal{U}}_2$ is a user-defined approximation of $\mathcal{U}_2$ that consists of a finite number of typical control functions of an evader at an intersection.

5.6.2 Overview of the Approach

This section details the threat assessment algorithm, which computes the approximate sampling-based solution to the InP game (Section 4.1.2) for the host vehicle to safely avoid an erratic driver. Central to this algorithm is the RRT-Attain subroutine (Section 5.6.2), which efficiently approximates the reachability set for the errant vehicle via time parametrization (Section 5.6.2). The tree uses biases for both exploration and pursuit of the host vehicle, representing the errant vehicle's "objective" to minimize the collision time t_f. The threat assessment algorithm then uses the RRT-Attain tree to evaluate the safety of each available escape path (Section 5.6.2).

RRT-Attain Algorithm

To approximate the solution to the InP game, the RRT-Attain algorithm adds a game-theoretical component to RRT-Reach (See Algorithm 9). The approach is called RRT-Attain because it uses RRT to create trajectories for the pursuer to attain the evader escape paths in minimum time. Both the pursuer (errant driver) and evader (host driver) are assumed to have full knowledge of each other's policies. The approach

utilizes this perfect information assumption through efficient, biased sampling in the RRT-algorithm.

RRT-Attain operates in two modes, exploration mode and pursuit mode. It chooses the exploration mode with probability P_{exp}, and the pursuit mode with probability P_{exp}. The more the environment is constrained, the higher the value P_{exp} should be set. Algorithm 11 describes the flow of the RRT-Attain algorithm.

In the exploration mode (Algorithm 12), the algorithm explores the state space by approximating the reachability set of the pursuer. As with the traditional RRT algorithm, it generates a sample (line 1), uses some heuristic to sort the nodes (line 5), attempts to propagate a trajectory from each node to the sample (line 7), and checks for constraint violation (line 8). Note that Algorithm 12 is similar to RRT-Reach (Algorithm 6). For more details about RRT-Reach, refer to Section 5.4.2.

In the pursuit mode (Algorithm 13), the algorithm uses the knowledge of the escape paths of the evader (host vehicle) to bias the sampling towards the position of the evader. This step starts by updating the position of the evader using the escape timestamp (line 3), which increases every time the pursuit step is called. It then uses this position as a sample for the pursuer tree to grow toward. In order to increase its efficiency in capturing the evader, it checks if the pursuer has a chance to arrive "on time" to the sample by computing the unconstrained minimum time from the root to the sample using Dubins distance (line 4). If on-time arrival is not possible, it throws the sample away, and moves to the next escape path sample. Otherwise, it performs a propagation step (lines 7–17) similar to the one of Algorithm 12, but with an additional condition in the neighbor selection process: neighbors must have a timestamp smaller than the current escape timestamp to be selected, as capture is otherwise impossible.

To incorporate these time-based decisions, a "timestamp" has been explicitly added to the state of the vehicle to track the time along each generated trajectory. While propagating (Algorithm 14), if the timestamp reaches T_h, the propagation is interrupted and the current portion of the trajectory is checked for feasibility. Also, when searching for nearest neighbours, the algorithm skips any node with a times-

Algorithm 11 RRT-Attain Algorithm

```
Measure current vehicle state and environment
repeat
    Sample p_mode uniformly from [0,1]
    if p_mode is less or equal to exploration bias P_exp then
        Perform an exploration step (Algorithm 12)
    else
        Perform a pursuit step (Algorithm 13)
    end if
until time limit for growing tree is reached
```

Algorithm 12 Exploration Step

```
Take sample for input to controller
repeat
    Update time range in time heuristics
    Find list of nearest neighbors using time range
    Sort list using distance heuristics
    for each sorted node do
        Call propagation function using simulation of controller
        if propagated portion is collision free then
            Add sample to tree break
        end if
    end for
until timestamp reaches time horizon and no collision free portion was found
```

tamp already equal to T_h. Unlike traditional RRT approaches, the RRT-Attain tree is not used to identify some path which reaches a goal location. Rather, the entire tree is analyzed to find the maximum threat along each of the trajectories (Algorithm 14). Thus in this case the "goal" is not some location, but in fact the vehicles being in a state of collision. Finally, note that RRT-Attain does not include a completeness guarantee on the reachability set; there may be some feasible trajectories which are not included when work on constructing the reachability set is completed. However, the problem of computing the full reachability set in real-time is computationally intensive when subject to complex dynamics and complex, dynamic environments. The RRT-Attain algorithm is designed to rapidly approximate the reachability set and improve the approximation with more available time, regardless of the current problem complexity.

Algorithm 13 Pursuit Step

```
1: Update escape timestamp
2: for each escape path do
3:    Set sample equal to escape path at current path time
4:    if minimum unconstrained time to reach sample from root is less than escape
      timestamp then
5:       continue {continue to next escape path}
6:    else
7:       repeat
8:          Update time range in time heuristics
9:          Find list of nearest neighbors using time range and escape timestamp
10:         Sort list using distance heuristics
11:         for each sorted node do
12:            Call propagation function using simulation of controller
13:            if propagated portion is collision free then
14:               Add sample to tree break
15:            end if
16:         end for
17:      until timestamp reaches time horizon and no collision free portion was found
18:   end if
19: end for
```

Time Parametrization

Simulation results (Section 5.6.3) have suggested that the use of time-parametrized heuristics in the nearest node selection process (line 5 of Algorithm 12; line 10 of Algorithm 13) can result in shorter paths and a more efficient approximation of the InP minimization component. The heuristic acts as a suboptimal strategy for identifying time-minimal paths in the tree to new samples, resulting in a more realistic reachability set for the errant vehicle.

The time heuristic consists of partitioning the nodes into incremental time ranges, where the time of each node is measured from the root, and using only one range at a time in the nearest node search. The time ranges are considered in order from shortest time from root to longest time from root (line 4 in Algorithm 12; line 9 in Algorithm 13). Using the list of nodes whose timestamp lies inside the current time range, the nearest node function then uses the Dubins distance metric to compute the k nearest neighbors (line 5 in Algorithm 12; line 10 in Algorithm 13). If none of

the k neighbors generate a feasible trajectory, and the time range has not yet reached the time horizon T_h, the algorithm moves to the next-furthest time range and repeats the cycle (line 12 in Algorithm 12; line 17 in Algorithm 13). Each nearest node search is limited to a subset of the tree nodes, significantly reducing the complexity of this calculation.benefit of the time heuristic.

Threat Assessment Algorithm

This section introduces the threat assessment (TA) algorithm used to solve the sampling-based relaxation of the InP (See Algorithm 5). The inputs to the TA algorithm are the reachability tree of the errant vehicle computed by RRT-Attain (Algorithm 11), and a list of escape paths that the host driver could follow. These escape paths can be learnt from statistical traffic data. For each escape path and for each timestamp, the algorithm checks the distance between the escape path and each node of the reachability tree *with the same timestamp*. If it finds that an escape path is within a distance smaller than some safety threshold distance of the other vehicle, it flags that escape path as unsafe, and stores the time of collision.

The TA algorithm generates a list of times of collision for each escape path, and return the path(s) with the highest time. By doing so, the TA algorithm performs the maximization operation in the InP value function Eq. 5.16. Note that since the algorithm stops checking times beyond the time horizon, we tacitly assume that the escape paths flagged "safe" have a time of collision equal to infinity. In practice, the threat assessment could have a higher logic that decides to choose among the safest paths that have a time of collision higher than some threshold. The following section will illustrate the TA algorithm.

5.6.3 Illustration

In this section, the TA algorithm is illustrated for a rural stop-controlled intersection scenario, known to have a high risk of collisions. Consider the problem of helping a driver on a major road (e.g. a highway) avoid a collision with an errant driver on

Algorithm 14 Threat Assessment Algorithm

```
Compute reachability tree of errant vehicle using RRT-Attain (Algorithm 11)
Obtain list of escape paths of host vehicle
repeat
  for each escape path E_j flagged as safe do
    for each node n_k in the reachability tree with time stamp equal to t_i do
      if ||n_k − E_j(t_i)|| ≤ safety threshold then
        Set collision time of E_j to t_i and flag E_j as unsafe
        break
      end if
    end for
  end for
  Increment time t_i by Δt_i
until time t_i equals T_h or no more safe escape paths
return escape path(s) with highest collision time
```

a minor road (e.g. a rural road). As the errant driver approaches the major road, he/she misses the STOP sign or (equivalently here) loses control of the vehicle. It thus does not decelerate as expected, creating an unpredictable and dangerous behavior for other drivers. The host vehicle is equipped with a classifier [96] which would quickly flag the other vehicle as dangerous, and then launches the TA algorithm to compute the risk of available escape maneuvers. This scenario deals with two main classes of intersection accidents, straight crossing paths (SCP) and right-turn into path (RTIP), which together account for more than 40% of light vehicle intersection accidents [23].

The initial position and heading of each vehicle are shown in Figure 5-10. The initial velocity is $v_1(0) = 1.5$m/s for the errant vehicle and $v_2(0) = 2.0$m/s for the host vehicle, whose velocity is assumed constant. The RRT-Attain algorithm uses a tree size of 2000 nodes with a time horizon of $T_h = 3$ s, and relies on a pure-pursuit controller [131] to control the steering motion of the errant vehicle in its propagation step. The exploration bias is set to $P_{\text{exp}} = 0.3$. To make the problem even more constrained, a small obstacle is added to the minor road around location (x,y) = (2.5, 0.5). It may represent some unexpected object that is obstructing part of the errant driver's road.

Figure 5-10 shows the output of the reachability tree generated by RRT-Attain

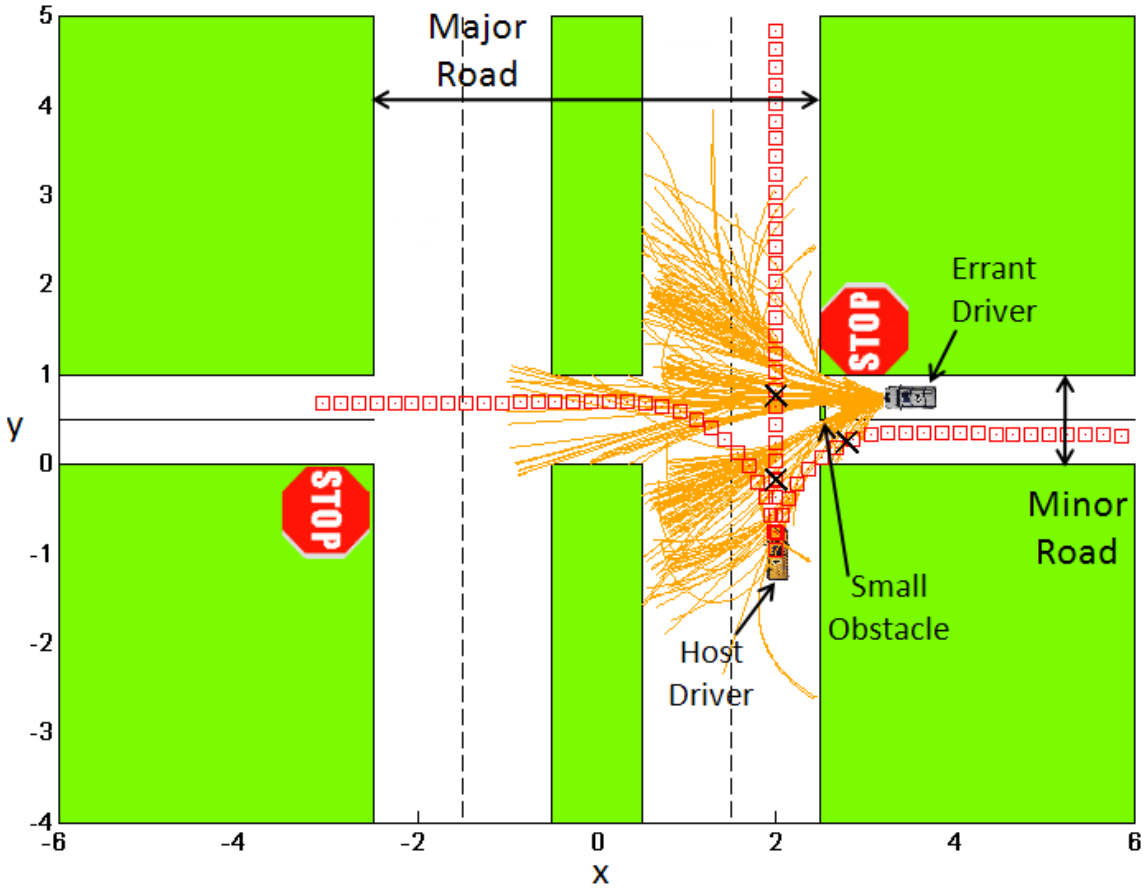


Figure 5-10: TA Algorithm applied to the stop-controlled intersection. The tree generated for the errant vehicle using RRT-Attain is shown, as well as four escape maneuvers for the host vehicle. For each escape maneuver, the location of earliest possible collision is marked $\times$.

along with the four possible escape maneuvers for the host vehicle: 1) decelerate with acceleration $a = -2.5$ m/s^2 until full stop, 2) keep moving straight with same initial velocity, 3) turn right with same initial velocity, and 4) turn left with same initial velocity. All four paths are dynamically feasible, and represent a subset of maneuvers that the host driver would typically follow to minimize the risk of collision at an intersection. Table 5.1 shows the results of the TA algorithm. Since Maneuver 4 has the highest time of collision, it is chosen as the recommended maneuver, and the value of the game $\mathcal{G}$ is set to infinity.

Note that some higher-order logic could be added to the TA algorithm. For example, if any escape maneuver with time of collision exceeding 1 s were deemed

Table 5.1: Summary of TA simulation results

Escape	Description	Time of Collision
1	Decelerate until stop	1.14 s
2	Keep going straight	0.88 s
3	Turn right	0.78 s
4	Turn left	∞

acceptable, then both maneuver 1 and 4 could be considered safe, as well. The choice might also depend on other factors as the discomfort level of each maneuver. Finally, in practice, the host vehicle is only warned if the threat of its current trajectory is larger than some safety threshold, at which point it will be advised to follow the escape maneuver with the lowest threat i.e., highest time of collision.

5.7 Conclusion

This chapter introduced a new approach for human driver assistance planning at road intersections. It is based on combining an intention predictor (IP), threat assessor (TA), and escape paths to create a threat assessment module (TAM) which runs efficiently and is suitable for real-time implementation. The IP performs classification on vehicle state information to determine threat levels via high-level logic, including the Ecological Recogniser, support vector machines, and Bayesian filtering. The TA performs efficient sampling-based reachability computations, via RRT, to identify possible future paths of surrounding vehicles. Finally, the escape paths are evaluated to maximize time to collision and thus minimize the threat level. The TAM has been demonstrated and validated in the RAVEN testbed, using both human-driven and autonomous vehicles. This chapter also presented another game-theoretic formulation that extends RRT-Reach to a dual exploration-pursuit mode that considers the worst-case scenario of errant drivers that are capable of causing intentional collisions. This conservative formulation is tailored for scenarios where the errant driver is observed to be following atypical maneuvers; the worst case approach then provides the safest escape maneuvers for the host vehicle.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Behavior Classification at Intersections (Chapter 2)

Chapter 2 introduced two new approaches for classifying driver behaviors at road intersections. The first one, denoted SVM-BF, combines a Support Vector Machines classifier with a Bayesian filter, to discriminate between compliant drivers and violators based on vehicle speed, acceleration and distance to intersection. The second one, an HMM-based classifier, uses the EM algorithm to develop two distinct Hidden Markov Models for compliant and violating behaviors. To optimize safety while respecting driving acceptance levels, the algorithms were designed to maximize true positive rates while keeping false alarm rates below the 5% threshold. The two algorithms were successfully validated on more than 10,000 real-world intersection approaches. The results of several generalization tests showed consistent and significant improvements with developed algorithms compared to three common approaches in the active safety literature.

Trajectory Prediction (Chapter 3)

Chapter 3 presented a novel approach to improve the trajectory prediction of agents with uncertain intentions for collision avoidance (CA) and conflict detection (CD)

systems. It combined the RRT-Reach algorithm with a Gaussian Process (GP) mixture model. RRT-Reach predicts the trajectories using a sampling-based reachability algorithm that incorporates vehicle and environment constraints. The GP mixture, a nonparametric Bayesian model, captured the distribution over a wide range of possible target agent motion patterns. The resulting RR-GP algorithm, ensures that the predicted trajectories are both feasible by construction and probabilistically weighted to reflect the typical motion patterns of the agents. It improves over traditional GP approaches both in computation times and prediction accuracy. This was verified in both human-operated simulation and real-world traffic data. This accurate real-time prediction algorithm is an ideal candidate for future CA and CD systems.

Safe Motion Planning (Chapter 4)

Chapter 4 introduced two new path planning algorithms that build upon the tools that have been introduced in Chapters 2 and 3, with a focus on safe autonomous navigation in the presence of other vehicles with uncertain motion patterns. First, it presented a new threat assessment module (TAM) that combines the RRT-Reach algorithm with an SVM-based intention predictor, to develop a threat-aware path planner. Using this planner, an autonomous vehicle can successfully identify and avoid an errant driver in realistic intersection scenarios, as demonstrated through various hardware results. Second, another novel path planning technique was developed by integrating the RR-GP trajectory prediction algorithm with a state-of-the-art chance-constrained RRT planner. This framework provides several theoretical guarantees on the probabilistic satisfaction of collision avoidance constraints. Extensive simulation results demonstrated that the resulting approach can be used in real-time to efficiently and accurately execute safe paths.

Threat Assessment and Application to Road Intersections (Chapter 5)

Chapter 5 considered the decision-making problem for a human-driven vehicle crossing a road intersection in the presence of other, potentially errant, drivers. The

proposed approach uses the TAM framework that consists of intention predictor (IP) and threat assessor (TA) components. The IP performed classification on vehicle state information to determine threat levels via high-level logic that combines support vector machines and Bayesian filtering. The TA efficiently computes sampling-based reachability sets, via the RRT-Reach algorithm, to identify possible future paths of surrounding vehicles. Finally, the escape paths are evaluated to maximize time to collision and thus minimize the threat level. Experimental results with small autonomous and human-driven vehicles in the RAVEN testbed demonstrate that this approach can be successfully used in real-time to minimize the risk of collision in urban-like environments.

6.2 Future Work

Building on the contributions presented in this thesis, suggestions for future research directions are outlined below.

When dealing with environments where the typical patterns of the target vehicle are not well understood, the trajectory prediction algorithm should be extended with an incremental learning phase that is performed online as more trajectories of the vehicle are observed. A Dirichlet process (DP) prior [132] over the mixture of Gaussian Processes (GP) could be used to avoid assumptions about the rigid structure of the mobility patterns. Both the GPs and the DP ensure that the complexity of the resulting mobility pattern model is robustly adapted, based on the available data [79].

To implement the developed threat assessment algorithms on the roads, they have to be integrated with different vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication paradigms [133, 134]. Furthermore, communication issues such delays and packet loss should be formally accounted for. Future work is required to extend the algorithms to be robust to the stochastic nature of these communication systems.

Another line of research is the investigation of the human factors involved in

the threat assessment design of driver assistance systems. While several studies have compared the effectiveness of different warning mechanisms (e.g., tactile vs. auditory), they have mainly focused on driver responses to forward collision warnings [135–137]. Following the recent findings presented in Ref. [138], future work has to consider the driver response to warnings at different road intersection types and geometries. Careful design is required to display the various threat levels and suggested escape maneuvers of the developed threat assessment algorithms, in a way that maximizes the benefits to the driver.

Appendix A

Implementation Parameters for Behavior Classifiers

This appendix presents the best parameter combinations for the SVM-based and HMM-based classifiers introduced in Chapter 2.

More specifically, Figures A-1, A-2, A-3, A-4, A-5, A-6 show the best ten combinations for the classifiers for the basic generalization tests presented in Section 2.6. Similarly, Figures A-7, A-8, A-9, A-10, A-11, A-12 show the best ten combinations for the m-fold cross-validation tests.

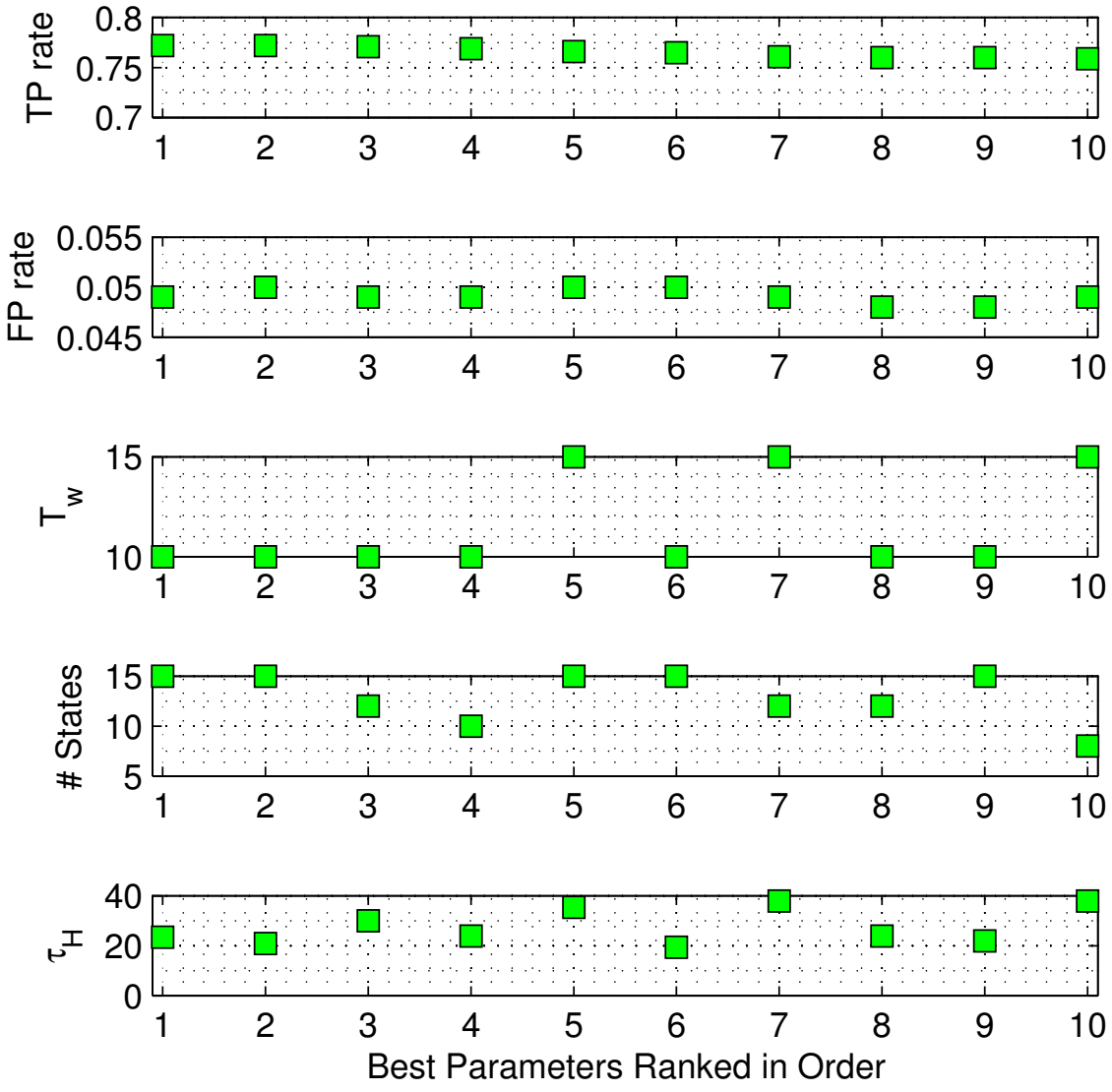


Figure A-1: Ten best parameter combinations for the HMM-based classifier with corresponding true positive rates for the basic generalization test with $p = 0.2$, $TTI_{\min}$ = 1.0 s and $d_{\min}$ = 6.25 m

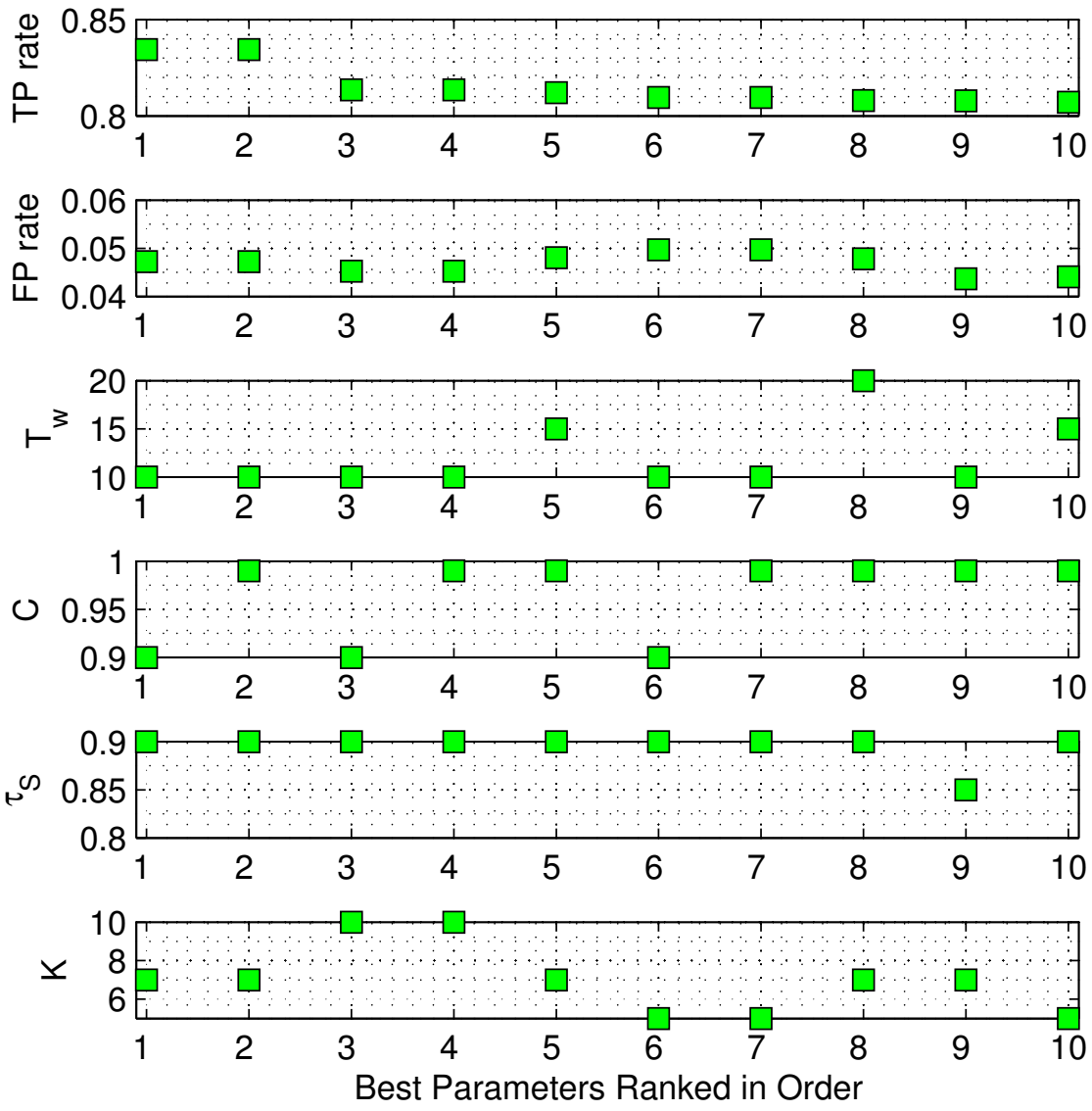


Figure A-2: Ten best parameter combinations for the SVM-based classifier with corresponding true positive rates for the basic generalization test with with $p = 0.2$, $TTI_{\min} = 1.0$ s and $d_{\min} = 6.25$ m

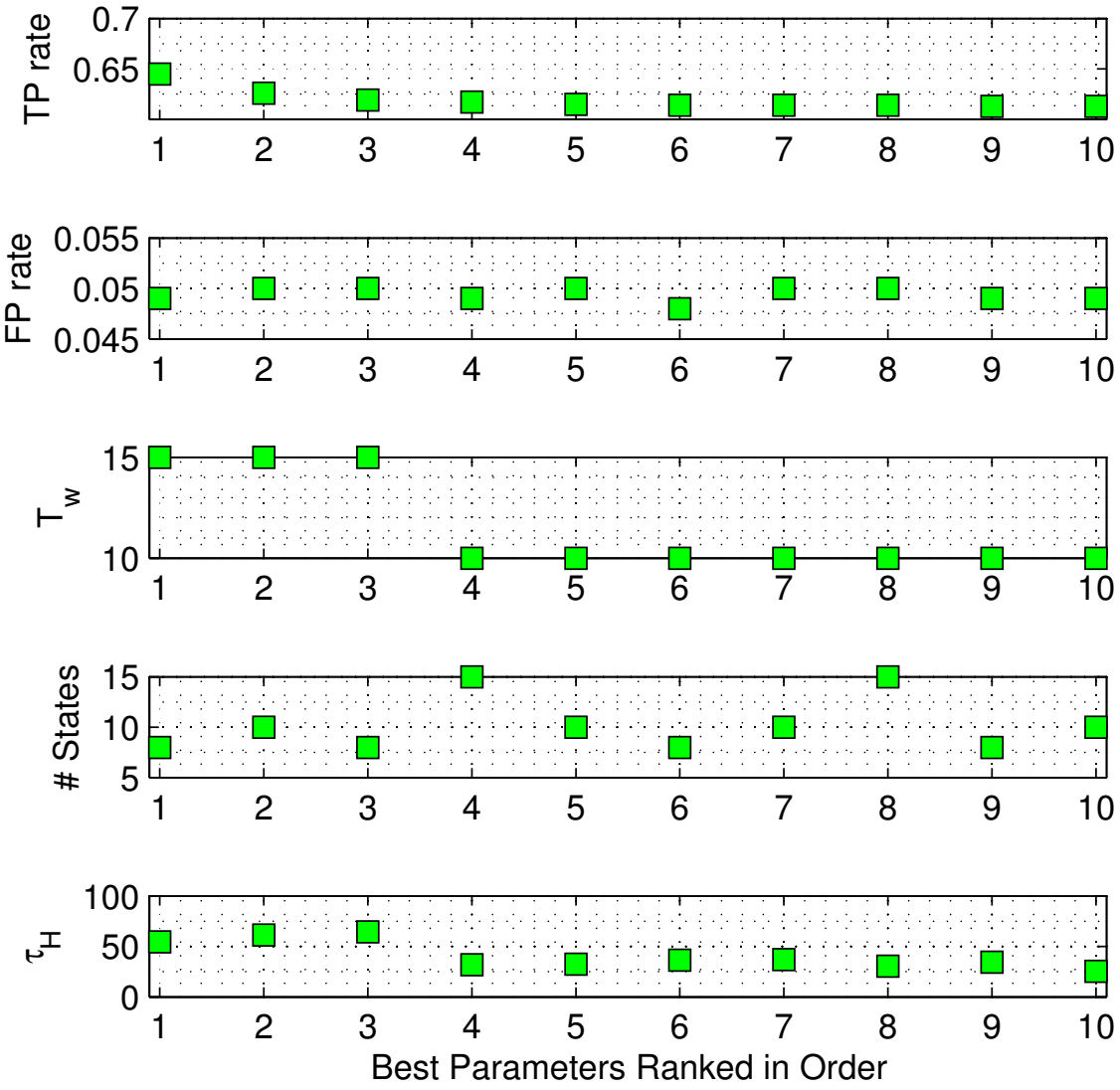


Figure A-3: Ten best parameter combinations for the HMM-based classifier with corresponding true positive rates for the basic generalization test with $p = 0.2$, $TTI_{\min}$ = 1.6 s and $d_{\min}$ = 10 m

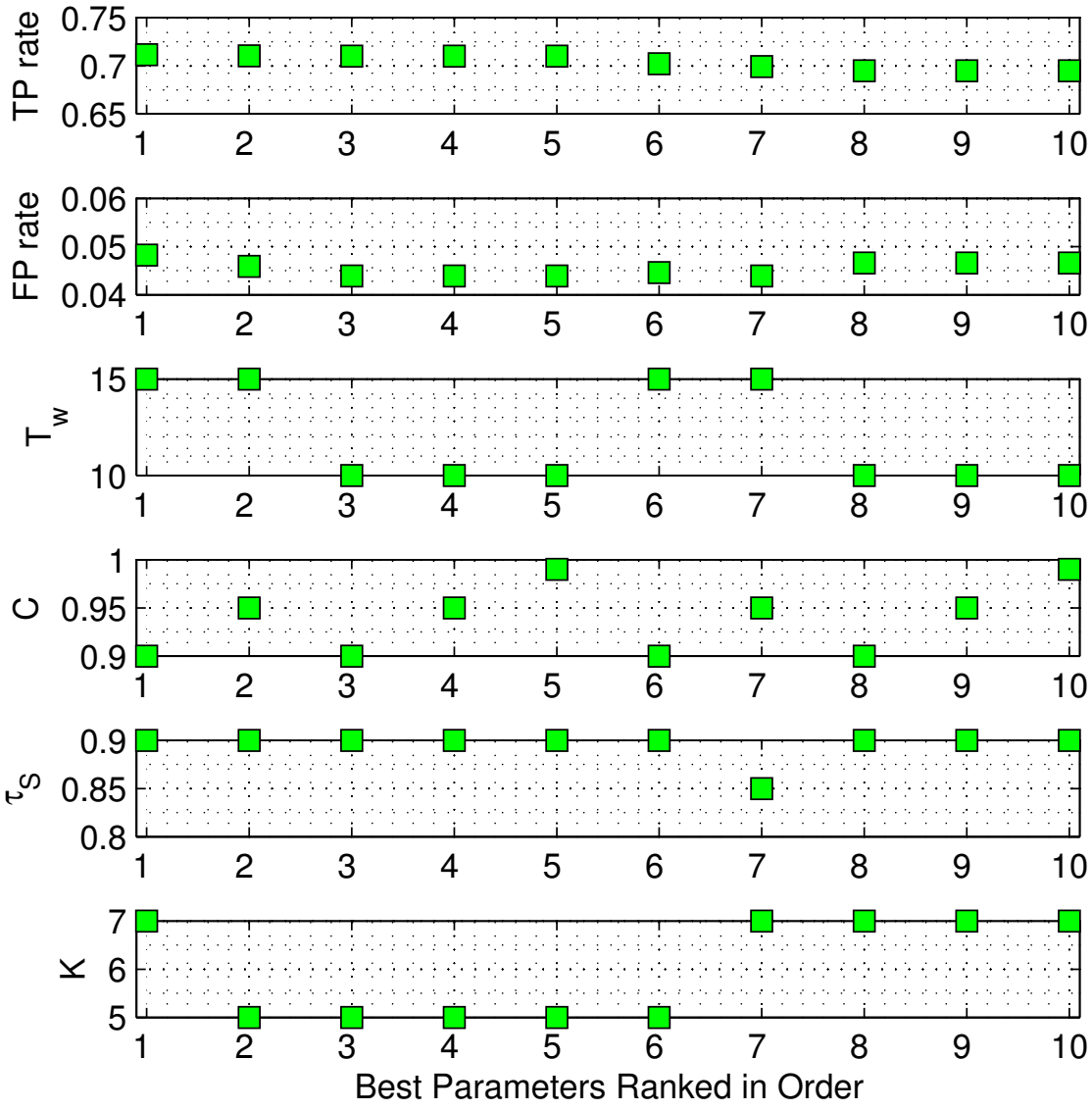


Figure A-4: Ten best parameter combinations for the SVM-based classifier with corresponding true positive rates for the basic generalization test with $p = 0.2$, $TTI_{\min}$ = 1.6 s and $d_{\min}$ = 10 m

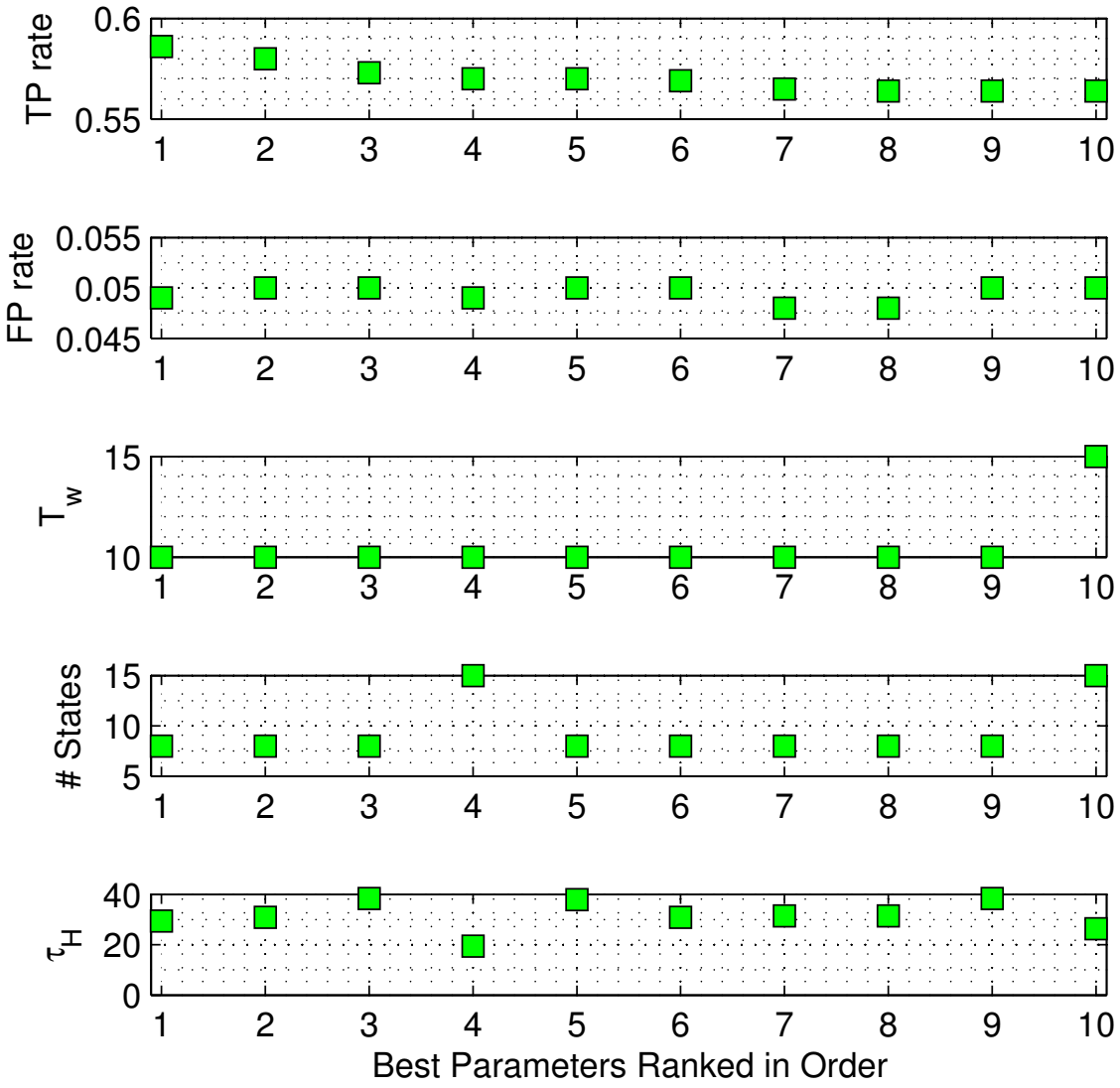


Figure A-5: Ten best parameter combinations for the HMM-based classifier with corresponding true positive rates for the basic generalization test with $p = 0.2$, $TTI_{\min}$ = 2.0 s and $d_{\min}$ = 12.5 m

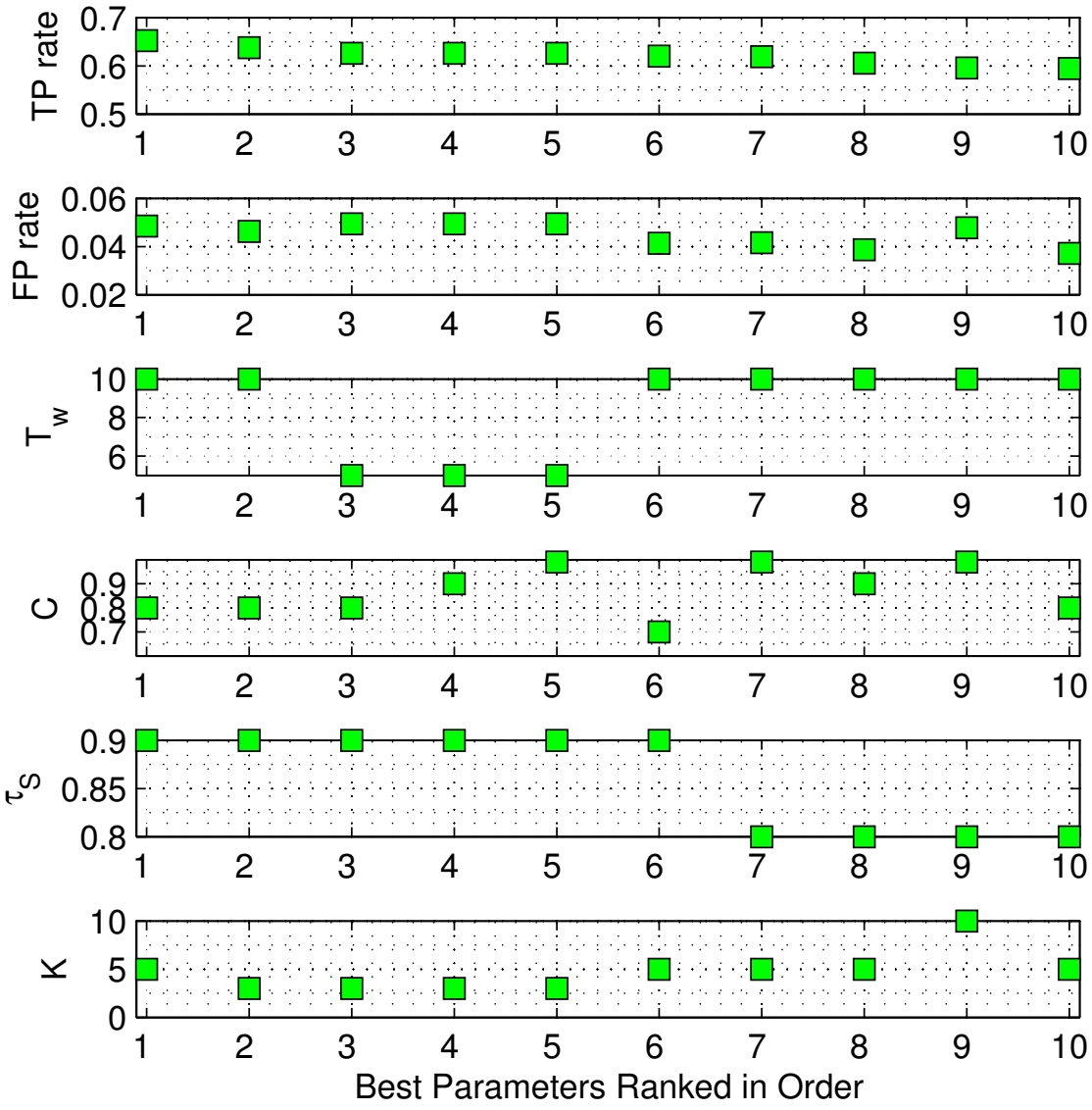


Figure A-6: Ten best parameter combinations for the SVM-based classifier with corresponding true positive rates for the basic generalization test with $p = 0.2$, $TTI_{\min}$ = 2.0 s and $d_{\min}$ = 12.5 m

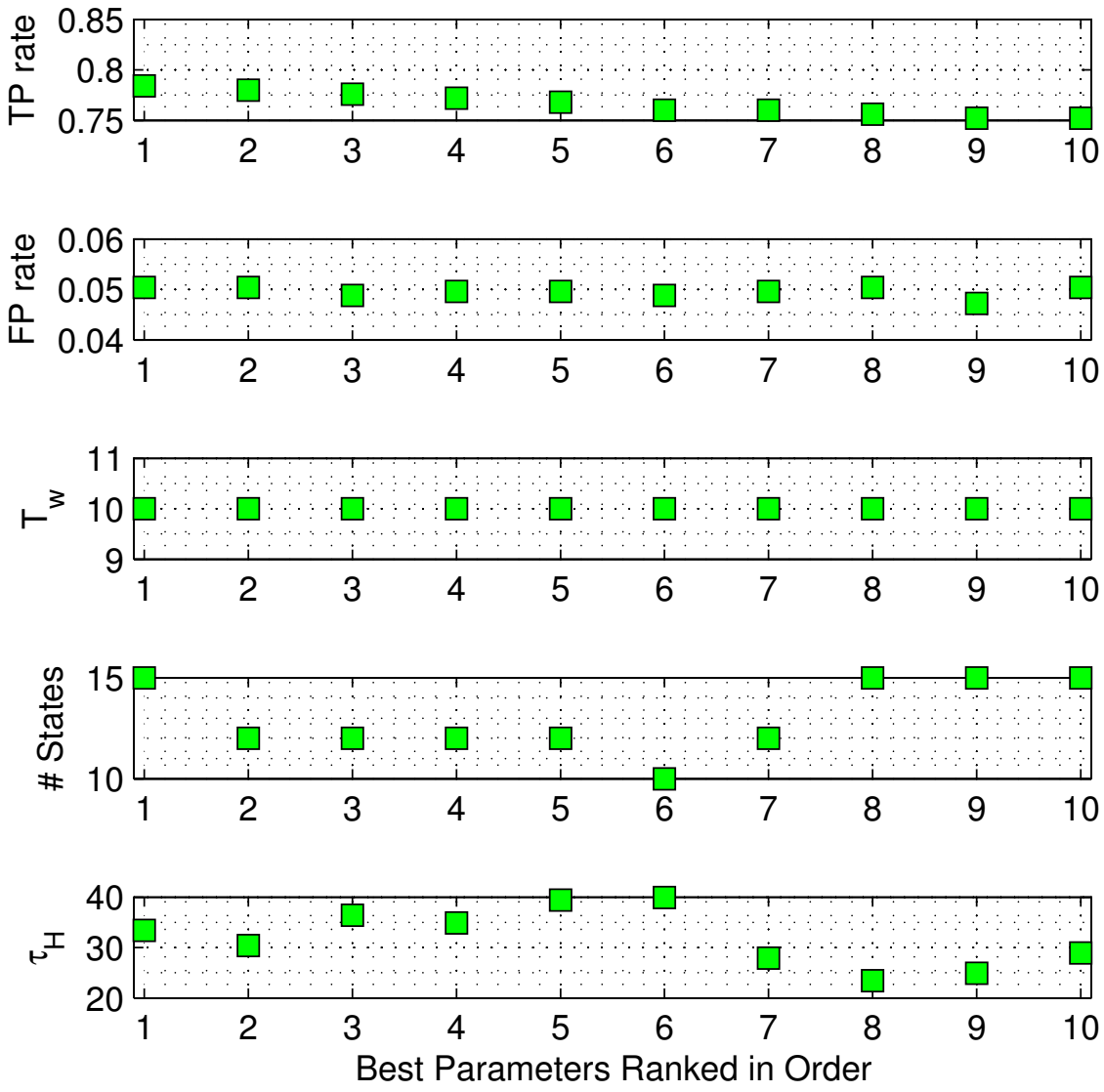


Figure A-7: Ten best parameter combinations for the HMM-based classifier with corresponding true positive rates for the m-fold cross-validation with $m = 4$, $TTI_{\min}$ = 1.0 s and $d_{\min}$ = 6.25 m

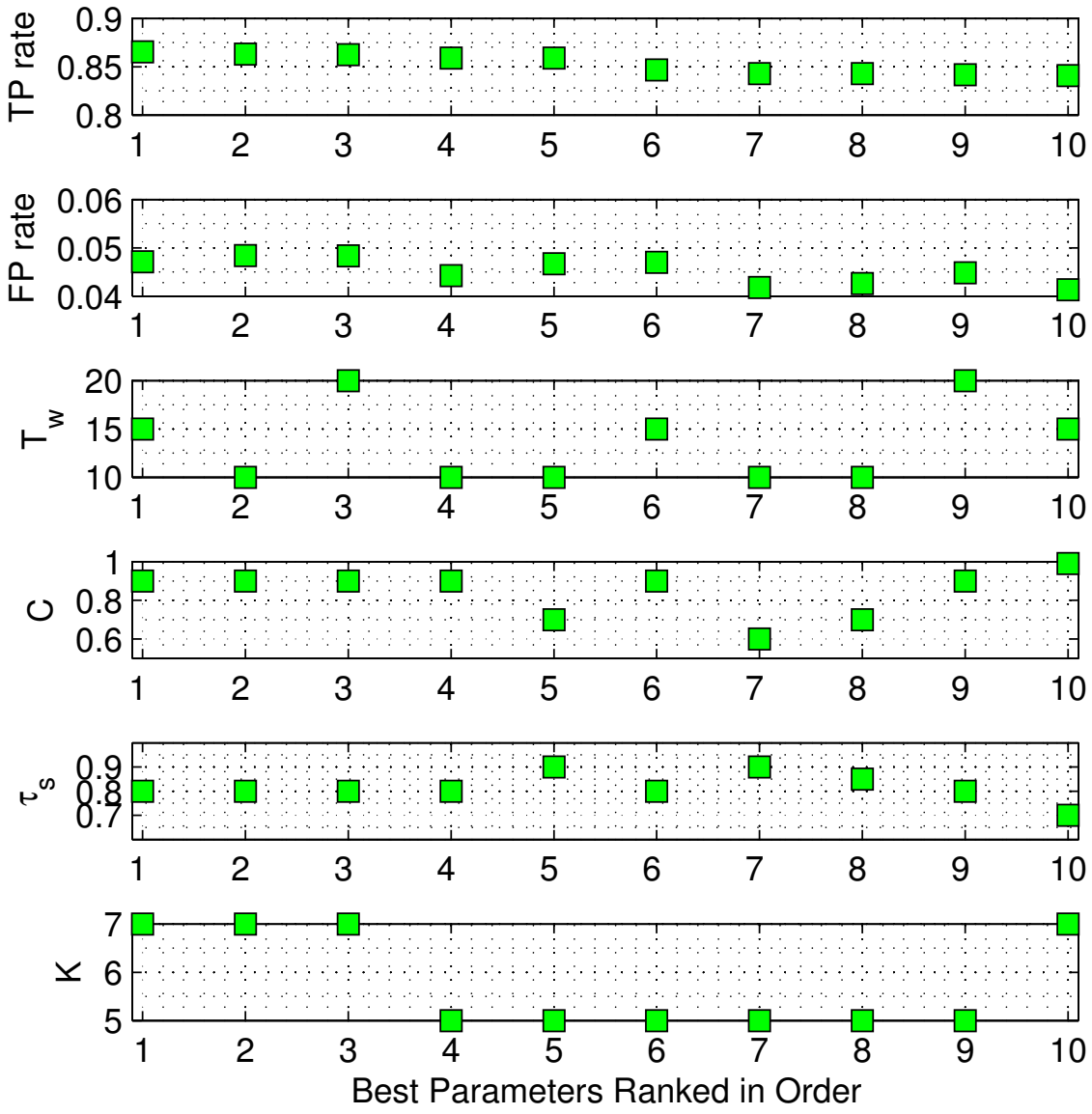


Figure A-8: Ten best parameter combinations for the SVM-based classifier with corresponding true positive rates for the m-fold cross-validation with $m = 4$, $TTI_{\min} = 1.0$ s and $d_{\min} = 6.25$ m

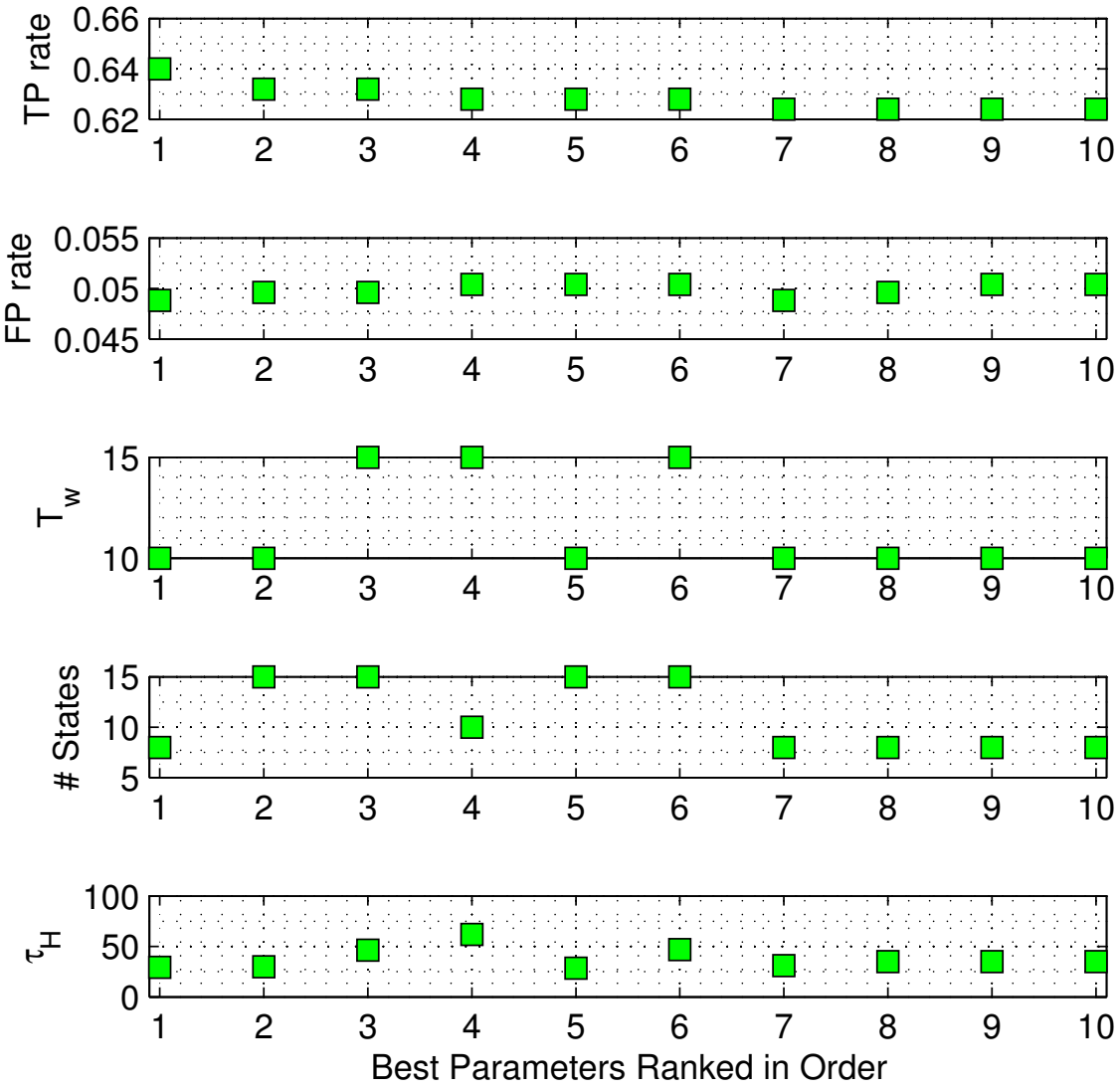


Figure A-9: Ten best parameter combinations for the HMM-based classifier with corresponding true positive rates for the m-fold cross-validation with $m = 4$, $TTI_{\min}$ = 1.6 s and $d_{\min}$ = 10 m

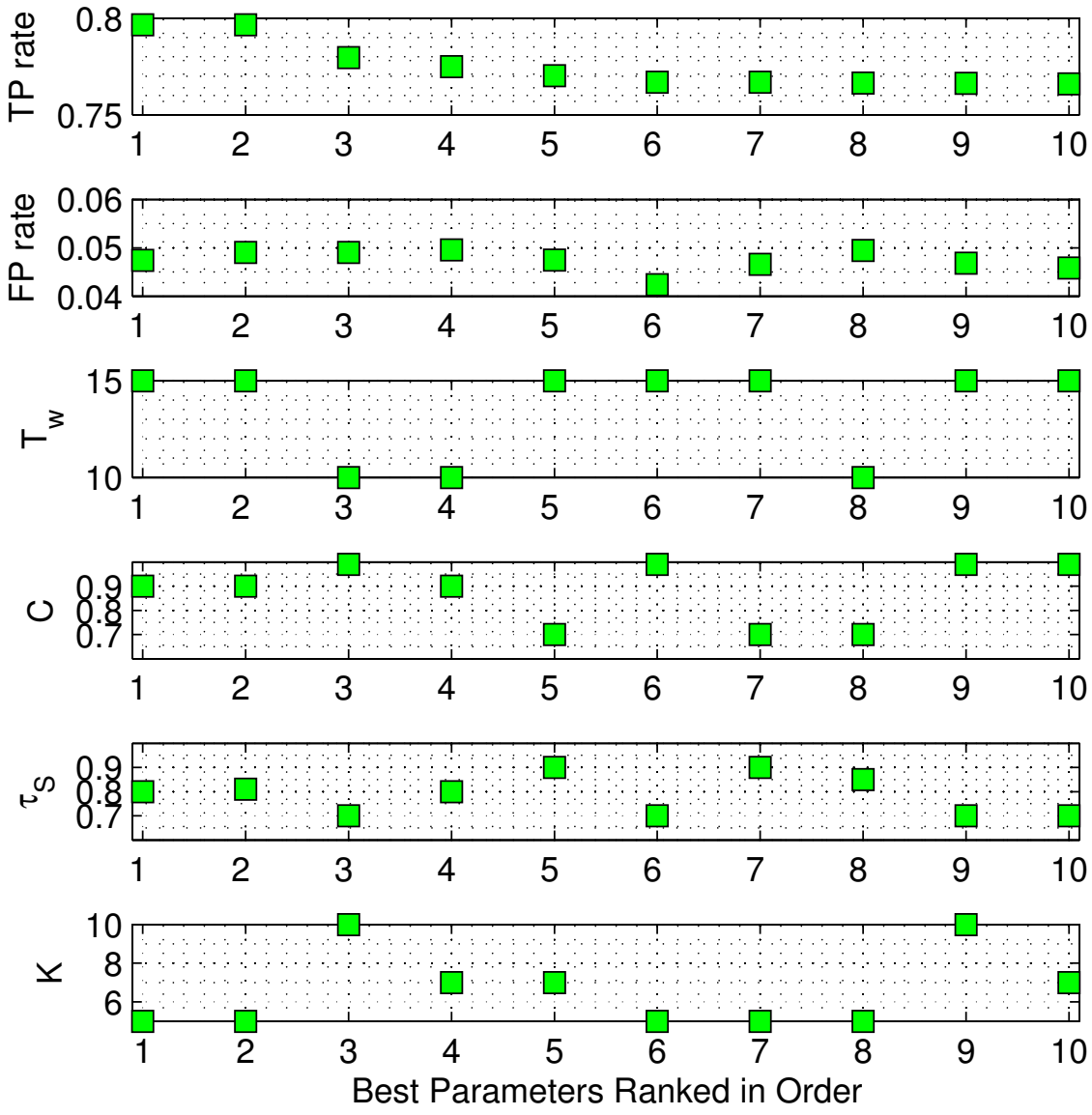


Figure A-10: Ten best parameter combinations for the SVM-based classifier with corresponding true positive rates for the m-fold cross-validation with $m = 4$, $TTI_{\min}$ = 1.6 s and $d_{\min}$ = 10 m

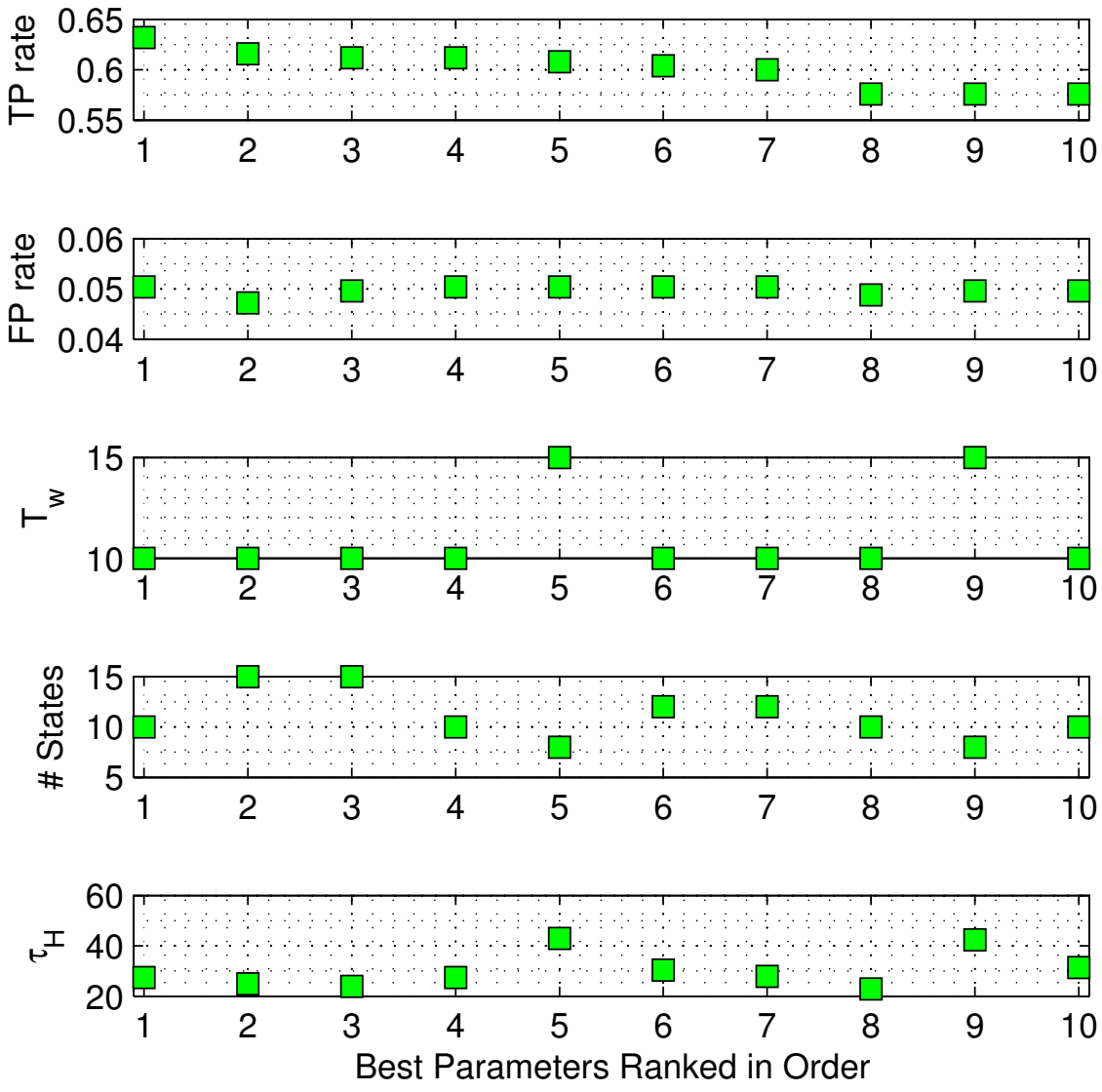


Figure A-11: Ten best parameter combinations for the HMM-based classifier with corresponding true positive rates for the m-fold cross-validation with $m = 4$, $TTI_{\min}$ = 2.0 s and $d_{\min}$ = 12.5 m

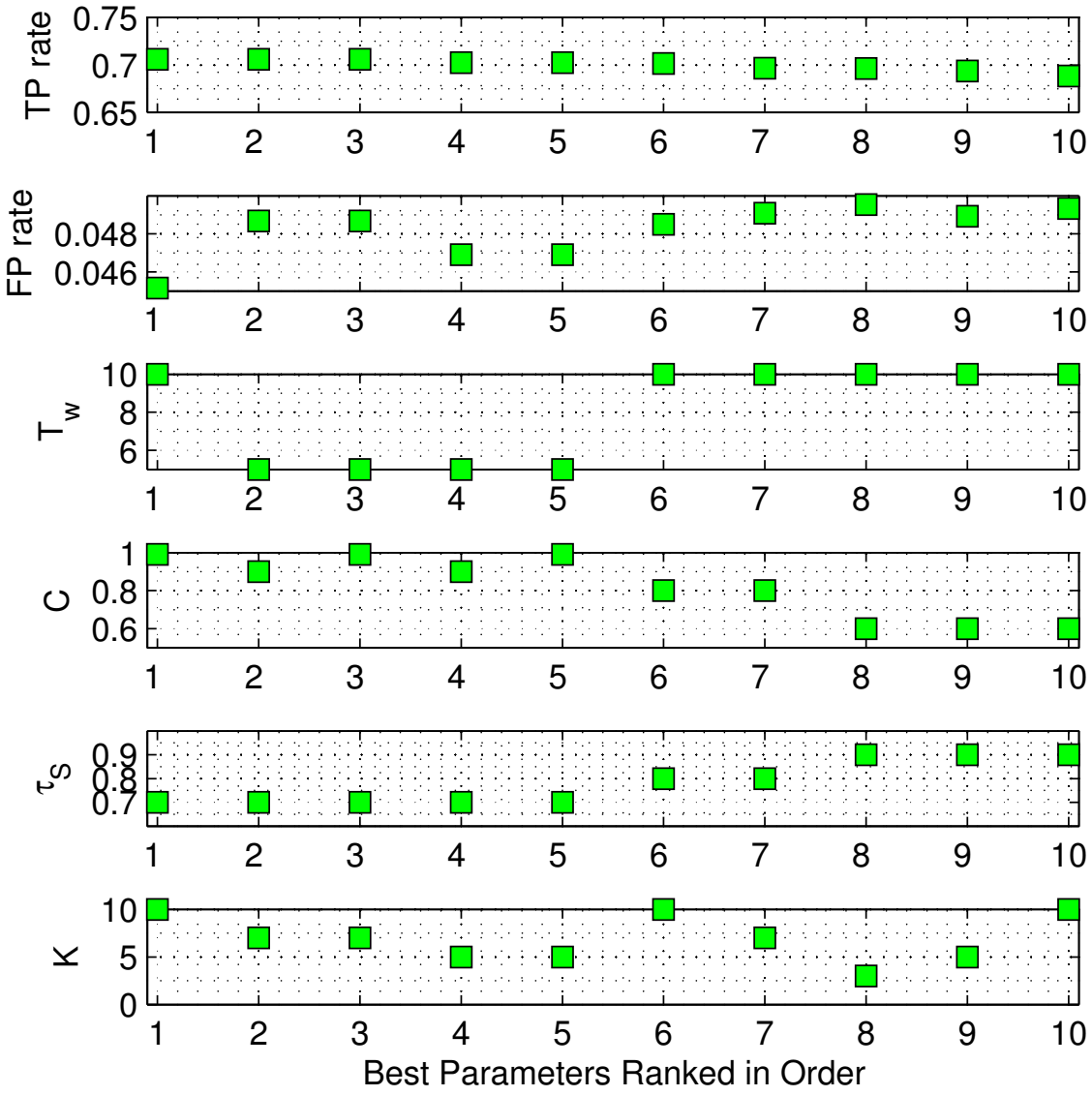


Figure A-12: Ten best parameter combinations for the SVM-based classifier with corresponding true positive rates for the m-fold cross-validation with $m = 4$, $TTI_{\min}$ = 2.0 s and $d_{\min}$ = 12.5 m

Bibliography

[1] T. Perry, "In search of the future of air traffic control," *Spectrum, IEEE*, vol. 34, no. 8, pp. 18–35, 1997.

[2] D. Sperling and D. Gordon, *Two billion cars: driving toward sustainability*. Oxford University Press, USA, 2009.

[3] W. D. Jones, "Keeping cars from crashing," *IEEE Spectrum*, vol. 38, no. 9, pp. 40–45, 2001.

[4] A. Eidehall, *Tracking and threat assessment for automotive collision avoidance*. PhD thesis, Department of Electrical Engineering, Link "opings universitet, 2007.

[5] Z. Sun and S. Chen, "Automotive active safety systems [introduction to the special section]," *Control Systems Magazine, IEEE*, vol. 30, no. 4, pp. 36–37, 2010.

[6] R. Petzold, "Proactive Approach to Safety Planning," *Federal Highway Administration Public Roads Magazine*, vol. 66, May/June 2003.

[7] "National Highway Traffic Safety Administration - 2008 Traffic Safety Annual Asessment - Highlights." http://www.nhtsa.dot.gov/. Online, Accessed 20-July-2011.

[8] T. Arino, K. Carpenter, S. Chabert, H. Hutchinson, T. Miquel, B. Raynaud, K. Rigotti, and E. Vallauri, "Studies on the Safety of ACAS II in Europe," tech. rep., 2002.

[9] E. Lester and R. Hansman, "Benefits and Incentives for ADS-B Equipage in the National Airspace System," Tech. Rep. ICAT-2007-2, Massachusetts Institute of Technology, 2007.

[10] J. Kuchar and L. Yang, "A review of conflict detection and resolution modeling methods," *IEEE Transactions on Intelligent Transportation Systems*, vol. 1, no. 4, pp. 179–189, 2002.

[11] M. Christodoulou and S. Kodaxakis, "Automatic commercial aircraft-collision avoidance in free flight: The three-dimensional problem," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 7, no. 2, pp. 242–249, 2006.

[12] "Real time Determination and Prediction of Aircraft Trajectories Using Limited Sensor Data." http://www.navysbir.com/n09_s/navst09-005.htm, 2009. Online, Accessed 20-July-2011.

[13] E. Frew and R. Sengupta, "Obstacle avoidance with sensor uncertainty for small unmanned aircraft," in *Conference on Decision and Control*, vol. 1, pp. 614–619, IEEE, 2004.

[14] J. Leonard, J. P. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, O. Koch, Y. Kuwata, D. Moore, E. Olson, S. Peters, J. Teo, R. Truax, M. Walter, D. Barrett, A. Epstein, K. Maheloni, K. Moyer, T. Jones, R. Buckley, M. Antone, R. Galejs, S. Krishnamurthy, and J. Williams, "A perception-driven autonomous urban vehicle," *Journal of Field Robotics*, vol. 25, no. 10, pp. 727–774, 2008.

[15] S. Thrun, "What we're driving at." http://googleblog.blogspot.com/2010/10/what-were-driving-at.html, October 2010. Google's official press release announcing the success of their driverless cars. Online, Accessed 20-July-2011.

[16] L. Fletcher, S. Teller, E. Olson, D. Moore, Y. Kuwata, J. How, J. Leonard, I. Miller, M. Campbell, D. Huttenlocher, A. Nathan, and F. R. Kline, "The MIT - Cornell Collision and Why it Happened," *Journal of Field Robotics*, vol. 25, pp. 775 – 807, October 2008.

[17] E. Valovage, "Enhanced ads-b research," *Aerospace and Electronic Systems Magazine, IEEE*, vol. 22, no. 5, pp. 35–38, 2007.

[18] "Vehicle infrastructure integration consortium (VIIC)." http://www.vehicle-infrastructure.org, 2010. Online, Accessed 20-July-2011.

[19] J. Yepes, I. Hwang, and M. Rotea, "New algorithms for aircraft intent inference and trajectory prediction," *Journal of guidance, control, and dynamics*, vol. 30, no. 2, pp. 370–382, 2007.

[20] J. McCall, D. Wipf, M. Trivedi, and B. Rao, "Lane change intent analysis using robust operators and sparse Bayesian learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 8, no. 3, pp. 431–440, 2007.

[21] D. Vasquez, T. Fraichard, O. Aycard, and C. Laugier, "Intentional motion online learning and prediction," *Machine Vision and Applications*, vol. 19, no. 5, pp. 411–425, 2008.

[22] S. M. Lavalle and R. Sharma, "On motion planning in changing, partially-predictable environments," *International Journal of Robotics Research*, vol. 16, pp. 775–805, 1997.

[23] National Highway Traffic Safety Administration, "Fatality analysis reporting system encyclopedia." http://www-fars.nhtsa.dot.gov/Crashes/CrashesLocation.aspx, 2010. Online, Accessed 20-July-2011.

[24] B. Bougler, D. Cody, and C. Nowakowski, "California Intersection Decision Support: A Driver-Centered Approach to Left-Turn Collision Avoidance System Design," tech. rep., UC Berkeley, 2008.

[25] K. Fuerstenberg and B. Rossler, "A new european approach for intersection safetythe ec-project intersafe," *Advanced Microsystems for Automotive Applications 2005*, pp. 493–504, 2005.

[26] H. Erzberger and R. Paielli, "Concept for next generation air traffic control system.," *Air Traffic Control Quarterly*, vol. 10, no. 4, pp. 355–378, 2002.

[27] G. S. Aoude, V. R. Desaraju, L. H. Stephens, and J. P. How, "Behavior Classification Algorithms at Intersections and Validation using Naturalistic Data," in *IEEE Intelligent Vehicles Symposium*, (Baden-Baden, Germany), June 2011.

[28] G. S. Aoude, V. R. Desaraju, L. H. Stephens, and J. P. How, "Driver behavior classification at intersections and validation on large naturalistic dataset," *IEEE Transactions on Intelligent Transportation Systems*, 2011 (submitted).

[29] G. S. Aoude, J. Joseph, N. Roy, and J. P. How, "Mobile Agent Trajectory Prediction using Bayesian Nonparametric Reachability Trees," in *AIAA Infotech@Aerospace*, (St. Louis, Missouri), March 2011.

[30] G. S. Aoude, B. D. Luders, D. S. Levine, K. K. H. Lee, and J. P. How, "Threat Assessment Design for Driver Assistance System at Intersections," in *IEEE Conference on Intelligent Transportation Systems*, (Madeira, Portugal), September 2010.

[31] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transactions on Control Systems Technology*, vol. 17, pp. 1105–1118, September 2009.

[32] G. S. Aoude, B. D. Luders, D. S. Levine, and J. P. How, "Threat-aware Path Planning in Uncertain Urban Environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Taipei, Taiwan), October 2010.

[33] G. Aoude, B. Luders, J. Joseph, N. Roy, and J. P. How, "Probabilistically Safe Motion Planning to Avoid Dynamic Obstacles with Uncertain Motion Patterns," *Autonomous Robots*, 2011 (submitted).

[34] B. Luders, M. Kothari, and J. P. How, "Chance constrained RRT for probabilistic robustness to environmental uncertainty," in *AIAA Guidance, Navigation, and Control Conference*, (Toronto, Canada), August 2010.

[35] G. S. Aoude, B. D. Luders, and J. P. How, "Sampling-Based Threat Assessment Algorithms for Intersection Collisions Involving Errant Drivers," in *IFAC Symposium on Intelligent Autonomous Vehicles*, (Lecce, Italy), September 2010.

[36] C. Y. Chan and B. Bougler, "Evaluation of cooperative roadside and vehicle-based data collection for assessing intersection conflicts," in *Intelligent Vehicles Symposium*, pp. 165–170, IEEE, 2005.

[37] M. Maile, F. A. Zaid, L. Caminiti, J. Lundberg, and P. Mudalige, "Cooperative intersection collision avoidance system limited to stop sign and traffic signal violations," 2008. Midterm Phase 1 Report.

[38] D. D. Salvucci, "Inferring driver intent: A case study in lane-change detection," in *in Proceedings of the Human Factors Ergonomics Society 48th Annual Meeting*, pp. 2228–2231, 2004.

[39] N. Oliver and A. P. Pentland, "Graphical models for driver behavior recognition in a smartcar," pp. 7–12, 2000.

[40] T. Gates, D. Noyce, L. Laracuente, and E. Nordheim, "Analysis of driver behavior in dilemma zones at signalized intersections," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 2030, no. 1, pp. 29–39, 2007.

[41] H. Rakha, I. El-Shawarby, and J. R. Sett, "Characterizing driver behavior on signalized intersection approaches at the onset of a yellow-phase trigger," *IEEE Transactions on Intelligent Transportation Systems*, vol. 8, no. 4, pp. 630–640, 2007.

[42] L. Zhang, K. Zhou, W. Zhang, and J. A. Misener, "Prediction of red light running based on statistics of discrete point sensors," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 2128, no. -1, pp. 132–142, 2009.

[43] J. Bonneson and H. Son, "Prediction of expected red-light-running frequency at urban intersections," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 1830, pp. 38–47, 2003.

[44] N. Elmitiny, X. Yan, E. Radwan, C. Russo, and D. Nashar, "Classification analysis of driver's stop/go decision and red-light running violation," *Accident Analysis & Prevention*, vol. 42, no. 1, pp. 101–111, 2010.

[45] V. Neale, M. Perez, Z. Doerzaph, S. Lee, S. Stone, and T. Dingus, "Intersection decision support: Evaluation of a violation warning system to mitigate straight crossing path crashes (report no. vtrc 06-cr10)," *Charlottesville, VA: Virginia Transportation Research Council*, 2006.

[46] Z. Doerzaph, V. Neale, and R. Kiefer, "Cooperative Intersection Collision Avoidance for Violations: Threat Assessment Algorithm Development and Evaluation Method," in *Transportation Research Board 89th Annual Meeting*, no. 10-2748, 2010.

[47] Z. Doerzaph, "Development of a threat assessment algorithm for intersection collision avoidance systems," 2007.

[48] D. McNicol, *A primer of signal detection theory*. Lawrence Erlbaum, 2004.

[49] "Car 2 Car Communication Consortium." http://www.car-to-car.org, 2011. Online, Accessed 20-July-2011.

[50] R. J. Kiefer, J. Salinger, and J. J. Ference, "Status of NHTSA's Rear-End Crash Prevention Research Program," 2005.

[51] H. M. Mandalia and D. D. Salvucci, "Using Support Vector Machines for Lane-Change Detection," *Human Factors and Ergonomics Society Annual Meeting Proceedings*, vol. 49, pp. 1965–1969, 2005.

[52] D. Wipf and B. Rao, "Driver Intent Inference Annual Report," tech. rep., University of California, San Diego, 2003.

[53] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. Wiley-Interscience, 2 ed., 2000.

[54] V. Vapnik, *The Nature of Statistical Learning Theory*. New York: Springer-Verlag, 1995.

[55] L. R. Rabiner., "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *IEEE Trans.*, vol. 77, no. 2, 1990.

[56] A. Van der Horst, *A time-based analysis of road user behaviour in normal and critical encounters*. PhD thesis, Delft University of Technology, 1990.

[57] H. Berndt, S. Wender, and K. Dietmayer, "Driver Braking Behavior during Intersection Approaches and Implications for Warning Strategies for Driver Assistant Systems," in *Intelligent Vehicles Symposium*, (Istanbul, Turkey), pp. 245–251, IEEE, 2007.

[58] "SVM Application List." http://www.clopinet.com/isabelle/Projects/SVM/applist.html, 2006. Online, Accessed 20-July-2011.

[59] Y. Liang, M. L. Reyes, and J. D. Lee, "Real-time detection of driver cognitive distraction using support vector machines," *IEEE Transactions on Intelligent Transportation Systems*, vol. 8, no. 2, pp. 340–350, 2007.

[60] C. Cortes and V. Vapnik, *Support-vector networks*, vol. 20. Springer, 1995.

[61] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, August 2006.

[62] D. Cutting, J. Kupiec, J. Pedersen, and P. Sibun, "A practical part-of-speech tagger," in *Proceedings of the third conference on Applied natural language processing*, pp. 133–140, 1992.

[63] J. Bilmes, "A gentle tutorial on the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models," no. ICSI-TR-97-021, 1997.

[64] Z. R. Doerzaph and V. Neale, "Data Acquisition Method for Developing Crash Avoidance Algorithms Through Innovative Roadside Data Collection," in *Transportation Research Board 89th Annual Meeting*, no. 10-2762, 2010.

[65] M. Maile and L. Delgrossi, "Cooperative Intersection Collision Avoidance System For Violations (CICAS-V) For Avoidance Of Violation-Based Intersection Crashes," *Enhanced Safety of Vehicles*, 2009.

[66] S. McLaughlin, J. Hankey, and T. Dingus, "A method for evaluating collision avoidance systems using naturalistic driving data," *Accident Analysis & Prevention*, vol. 40, no. 1, pp. 8–16, 2008.

[67] C. Chang and C. Lin, "Libsvm: a library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[68] M. Dunham and K. Murphy, "pmtk3 - probabilistic modeling toolkit for matlab/octave, version 3." http://code.google.com/p/pmtk3. Online, Accessed 20-July-2011.

[69] M. Salem and S. Stolfo, "Detecting masqueraders: a comparison of one-class bag-of-words user behavior modeling techniques," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, vol. 1, no. 1, pp. 3–13, 2010.

[70] M. Johnson, "Capacity and complexity of hmm duration modeling techniques," *Signal Processing Letters*, vol. 12, no. 5, pp. 407–410, 2005.

[71] T. Miloh and S. Sharma, *Maritime collision avoidance as a differential game*. Institut fur Schiffbau der Universitat Hamburg, 1976.

[72] R. Lachner, "Collision avoidance as a differential game: Real-time approximation of optimal strategies using higher derivatives of the value function," vol. 3, pp. 2308–2313, 2002.

[73] Q. Zhu, "Hidden Markov model for dynamic obstacle avoidance of mobile robot navigation," *Robotics and Automation, IEEE Transactions on*, vol. 7, no. 3, pp. 390–397, 2002.

[74] M. Bennewitz, W. Burgard, G. Cielniak, and S. Thrun, "Learning motion patterns of people for compliant robot motion," *The International Journal of Robotics Research*, vol. 24, no. 1, p. 31, 2005.

[75] H. Sorenson, *Kalman filtering: theory and application*. IEEE, 1985.

[76] E. Mazor, A. Averbuch, Y. Bar-Shalom, and J. Dayan, "Interacting multiple model methods in target tracking: a survey," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 34, no. 1, pp. 103–123, 2002.

[77] J. Joseph, F. Doshi-Velez, and N. Roy, "A bayesian nonparametric approach to modeling mobility patterns," in *AAAI*, 2010.

[78] J. Joseph, F. Doshi-Velez, A. S. Huang, and N. Roy, "A bayesian nonparametric approach to modeling motion patterns," tech. rep., 2011.

[79] J. Joseph, F. Doshi-Velez, A. S. Huang, and N. Roy, "A bayesian nonparametric approach to modeling motion patterns," *Autonomous Robots*, vol. Accepted, awaiting publication, 2011. Search and Pursuit/Evasion with Mobile Robots.

[80] M. Tay and C. Laugier, "Modelling smooth paths using gaussian processes," in *Field and Service Robotics*, pp. 381–390, Springer, 2008.

[81] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. The MIT Press, December 2005.

[82] C. Fulgenzi, C. Tay, A. Spalanzani, and C. Laugier, "Probabilistic navigation in dynamic environment using rapidly-exploring random trees and gaussian processes," in *Proceedings of the IEEE International Conference on Intelligent Robots and Systems*, pp. 1056–1062, 2008.

[83] S. Gan, K. Yang, and S. Sukkarieh, "3D Online Path Planning in a Continuous Gaussian Process Occupancy Map," in *Australian Conference on Field Robotics*, 2009.

[84] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[85] O. Amidi and C. Thorpe, "Integrated mobile robot control," *SPIE Mobile Robots V*, pp. 504–523, 1990.

[86] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *AIAA Journal of Guidance, Control, and Dynamics*, vol. 25, pp. 116–129, January-February 2002.

[87] iRobot, "iRobot: Education & research robots." Online http://store.irobot.com/shop/index.jsp?categoryId=3311368.

[88] P. Velleman and D. Hoaglin, *Applications, Basics, and Computing of Exploratory Data Analysis*. Duxbury Press, 1981.

[89] J. How, B. Bethke, A. Frank, D. Dale, and J. Vian, "Real-time indoor autonomous vehicle test environment," *IEEE Control Systems Magazine*, vol. 28, no. 2, pp. 51–64, 2008.

[90] J. Bresina, R. Dearden, N. Meuleau, S. Ramakrishnan, D. Smith, and R. Washington, "Planning under continuous time and resource uncertainty: A challenge for AI," in *AIPS Workshop on Planning for Temporal Domains*, pp. 91–97, Citeseer, 2002.

[91] A. Brooks, A. Makarenko, S. Williams, and H. Durrant-Whyte, "Parametric POMDPs for planning in continuous state spaces," *Robotics and Autonomous Systems*, vol. 54, no. 11, pp. 887–897, 2006.

[92] C. Schlenoff, R. Madhavan, and Z. Kootbally, "PRIDE: A Hierarchical, Integrated Prediction Framework for Autonomous on-Road Driving," pp. 2348–2353, 2006.

[93] M. Althoff, O. Stursberg, and M. Buss, "Model-based Probabilistic Collision Detection in Autonomous Driving," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 10, no. 2, pp. 299–310, 2009.

[94] R. Verma and D. D. Vecchio, "Safety in Semi-autonomous Multi-vehicle Systems: A Hybrid Control Approach," *IEEE Robotics and Automation Magazine*, 2011 (Accepted).

[95] C. Heinze, *Modelling Intention Recognition for Intelligent Agent Systems*. PhD thesis, Melbourne, Victoria: Defence Science and Technology Organisation, 2004.

[96] G. S. Aoude and J. P. How, "Using Support Vector Machines and Bayesian Filtering for Classifying Agent Intentions at Road Intersections," Tech. Rep. ACL09-02, Massachusetts Institute of Technology, September 2009.

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

[98] G. S. Aoude, B. D. Luders, D. S. Levine, and J. P. How, "Threat-aware Path Planning in Uncertain Urban Environments [Attached Video]," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Taipei, Taiwan), October 2010. http://hdl.handle.net/1721.1/64737.

[99] S. Park, J. Deyst, and J. P. How, "Performance and Lyapunov stability of a nonlinear path-following guidance method," *Journal of Guidance, Control, and Dynamics*, vol. 30, pp. 1718–1728, November-December 2007.

[100] Microsoft, "Description of the Xbox 360 Wireless Racing Wheel." http://support.microsoft.com/kb/927344?sd=xbox, 2011. Online, Accessed 20-July-2011.

[101] DARPA, "Urban challenge: Route network definition file (RNDF) and mission data file (MDF) formats," tech. rep., Defense Advanced Research Projects Agency, March 2007.

[102] W. Najm, J. Koopmann, and D. Smith, "Analysis of Crossing Path Crash Countermeasure Systems," in *Proceedings of the 17th International Technical Conference on the Enhanced Safety of Vehicles*, (Amsterdam, The Netherlands), 2001.

[103] B. Luders, G. Aoude, J. Joseph, N. Roy, and J. P. How, "Probabilistically safe avoidance of dynamic obstacles with uncertain motion patterns," tech. rep., Massachusetts Institute of Technology, 2011.

[104] L. Blackmore, H. Li, and B. Williams, "A probabilistic approach to optimal robust path planning with obstacles," in *Proceedings of the IEEE American Control Conference*, pp. 2831–2837, 2006.

[105] G. C. Calafiore and L. E. Ghaoui, "Linear Programming with Probability Constraints – Part 1," in *American Control Conference*, IEEE, 2007.

[106] L. Blackmore, "A probabilistic particle control approach to optimal, robust predictive control," in *Proceedings of the AIAA Guidance, Navigation and Control Conference*, 2006.

[107] L. Blackmore, "A probabilistic particle control approach to optimal robust predictive control," in *Proceedings of the IEEE American Control Conference*, 2007.

[108] L. Blackmore, M. Ono, A. Bektassov, and B. C. Williams, "A probabilistic particle-control approximation of chance-constrained stochastic predictive control," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 502–517, 2010.

[109] B. Luders and J. P. How, "Probabilistic Feasibility for Nonlinear Systems with Non-Gaussian Uncertainty using RRT," in *Proceedings of the AIAA Infotech@Aerospace Conference*, (St. Louis, MO), March 2011.

[110] J. Joseph, F. Doshi-Velez, and N. Roy, "A bayesian nonparametric approach to modeling mobility patterns," in *AAAI*, 2010.

[111] Y. Zhao and A. House, "Vehicle Location and Navigation Systems: Intelligent Transportation Systems," *Artech House*, pp. 221–224, 1997.

[112] J. Hillenbrand and K. Kroschel, "A Study on the Performance of Uncooperative Collision Mitigation Systems at Intersection-like Traffic Situations," in *Proceedings of the IEEE Conference on Cybernetics and Intelligent Systems*, pp. 1–6, 2006.

[113] A. Polychronopoulos, M. Tsogas, A. Amditis, U. Scheunert, L. Andreone, and F. Tango, "Dynamic situation and threat assessment for collision warning systems: the EUCLIDE approach," in *Proceedings of the IEEE Intelligent Vehicles Symposium*, pp. 636–641, 2004.

[114] R. Karlsson, J. Jansson, and F. Gustafsson, "Model-based statistical tracking and decision making for collision avoidance application," in *Proceedings of the IEEE American Control Conference*, vol. 4, 2004.

[115] S. Anderson, S. Peters, T. Pilutti, and K. Iagnemma, "An optimal-control-based framework for trajectory planning, threat assessment, and semi-autonomous control of passenger vehicles in hazard avoidance scenarios," *International Journal of Vehicle Autonomous Systems*, vol. 8, no. 2, pp. 190–216, 2010.

[116] P. Falcone, F. Borrelli, J. Asgari, H. Tseng, and D. Hrovat, "Predictive active steering control for autonomous vehicle systems," *IEEE Transactions on Control Systems Technology*, vol. 15, no. 3, pp. 566–580, 2007.

[117] A. Broadhurst, S. Baker, and T. Kanade, "Monte carlo road safety reasoning," in *Intelligent Vehicles Symposium*, pp. 319–324, IEEE, 2005.

[118] S. Danielsson, L. Petersson, and A. Eidehall, "Monte carlo based threat assessment: Analysis and improvements," in *Intelligent Vehicles Symposium*, pp. 233–238, IEEE, 2007.

[119] A. Eidehall and L. Petersson, "Statistical threat assessment for general road scenes using monte carlo sampling," *IEEE Transactions on Intelligent Transportation Systems*, vol. 9, no. 1, pp. 137–147, 2008.

[120] J. Hillenbrand, A. Spieker, and K. Kroschel, "A multilevel collision mitigation approachits situation assessment, decision making, and performance tradeoffs," *IEEE Transactions on Intelligent Transportation Systems*, vol. 7, no. 4, pp. 528–540, 2006.

[121] Y. Liu, O. Ozguner, and E. Ekici, "Performance evaluation of intersection warning system using a vehicle traffic and wireless simulator," in *Intelligent Vehicles Symposium*, pp. 171–176, IEEE, 2005.

[122] M. Brannstrom, E. Coelingh, and J. Sjoberg, "Model-based threat assessment for avoiding arbitrary vehicle collisions," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 3, pp. 658–669, 2010.

[123] T. Kumagai, Y. Sakaguchi, M. Okuwa, and M. Akamatsu, "Prediction of driving behavior through probabilistic inference," in *Proceedings of the Eighth International Conference on Engineering Applications of Neural Networks*, pp. 8–10, 2003.

[124] G. S. Aoude, B. D. Luders, D. S. Levine, K. K. H. Lee, and J. P. How, "Threat Assessment Design for Driver Assistance System at Intersections: Experiment Video," Tech. Rep. ACL11-03, Massachusetts Institute of Technology, April 2010. http://hdl.handle.net/1721.1/46720.

[125] Vicon, "Motion Capture Systems from Vicon." http://www.vicon.com, 2011. Online, Accessed 20-July-2011.

[126] R. Isaacs, *Differential games: a mathematical theory with applications to warfare and pursuit, control and optimization*. Courier Dover Publications, 1999.

[127] H. Ehtamo and T. Raivio, "On Applied Nonlinear and Bilevel Programming or Pursuit-Evasion Games," *Journal of Optimization Theory and Applications*, vol. 108, no. 1, pp. 65–96, 2001.

[128] V. Isler, D. Sun, and S. Sastry, "Roadmap based pursuit-evasion and collision avoidance," in *Proc. Robotics, Systems, & Science*, 2005.

[129] T. Raivio and H. Ehtamo, "On the numerical solution of a class of pursuit-evasion games," *Advances in dynamic games and applications*, p. 177, 2000.

[130] A. Bhatia and E. Frazzoli, "Incremental search methods for reachability analysis of continuous and hybrid systems," *Hybrid Systems: Computation and Control*, vol. 2993, pp. 142–156, 2004.

[131] Y. Kuwata, J. Teo, S. Karaman, G. Fiore, E. Frazzoli, and J. How, "Motion Planning in Complex Environments using Closed-loop Prediction," in *Proceedings of the AIAA Guidance, Navigation and Control Conference and Exhibit*, (Honolulu, Hawaii), August 2008.

[132] E. Fox, E. Sudderth, and A. Willsky, "Hierarchical Dirichlet processes for tracking maneuvering targets," in *10th International Conference on Information Fusion*, pp. 1–8, IEEE, 2007.

[133] S. Biswas, R. Tatchikou, and F. Dion, "Vehicle-to-vehicle wireless communication protocols for enhancing highway traffic safety," *Communications Magazine*, vol. 44, no. 1, pp. 74–82, 2006.

[134] J. Santa, A. Gómez-Skarmeta, and M. Sánchez-Artigas, "Architecture and evaluation of a unified v2v and v2i communication system based on cellular networks," *Computer Communications*, vol. 31, no. 12, pp. 2850–2861, 2008.

[135] R. Parasuraman, P. A. Hancock, and O. Olofinboba, "Alarm effectiveness in driver-centred collision-warning systems," *Ergonomics*, vol. 40, no. 3, pp. 390–399, 1997.

[136] C. Ho, N. Reed, and C. Spence, "Multisensory in-car warning signals for collision avoidance," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 49, no. 6, p. 1107, 2007.

[137] J. Scott and R. Gray, "A comparison of tactile, visual, and auditory warnings for rear-end collision prevention in simulated driving," *Human Factors: The Journal of the Human Factors and Ergonomics Society*, vol. 50, no. 2, p. 264, 2008.

[138] V. Inman and G. Davis, "Effects of in-vehicle and infrastructure-based collision warnings to nonviolating drivers at signalized intersections," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 2189, pp. 17–25, 2010.