Safe and Interactive Autonomy: Control, Learning, and Verification

by

Dorsa Sadigh

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering - Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Summer 2017

Safe and Interactive Autonomy: Control, Learning, and Verification

Abstract

Safe and Interactive Autonomy: Control, Learning, and Verification

by

Dorsa Sadigh

Doctor of Philosophy in Engineering - Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Sanjit A. Seshia, Co-chair

Professor S. Shankar Sastry, Co-chair

The goal of my research is to enable safe and reliable integration of human-robot systems in our society by providing a unified framework for modeling and design of these systems. Today's society is rapidly advancing towards autonomous systems that interact and collaborate with humans, e.g., semiautonomous vehicles interacting with drivers and pedestrians, medical robots used in collaboration with doctors, or service robots interacting with their users in smart homes. The safety-critical nature of these systems require us to provide provably correct guarantees about their performance. In this dissertation, we develop a formalism for the design of algorithms and mathematical models that enable correct-by-construction control and verification of human-robot systems.

We focus on two natural instances of this agenda. In the first part, we study interaction-aware control, where we use algorithmic HRI to be mindful of the effects of autonomous systems on humans' actions, and further leverage these effects for better safety, efficiency, coordination, and estimation. We further use active learning techniques to update and better learn human models, and study the accuracy and robustness of these models. In the second part, we address the problem of providing correctness guarantees, while taking into account the uncertainty arising from the environment or human models. Through this effort, we introduce Probabilistic Signal Temporal Logic (PrSTL), an expressive specification language that allows representing Bayesian graphical models as part of its predicates. Further, we provide a solution for synthesizing controllers that satisfy temporal logic specifications in probabilistic and reactive settings, and discuss a diagnosis and repair algorithm for systematic transfer of control to the human in unrealizable settings. While the algorithms and techniques introduced can be applied to many human-robot systems, in this dissertation, we will mainly focus on the implications of my work for semiautonomous driving.

To Maman, Baba, Gelareh, Amir, Vivian,
and Nima.

Contents

Acknowledgments

I would like to first and foremost thank my advisors Sanjit Seshia and Shankar Sastry. Sanjit's enthusiasm and perceptiveness was what drew me to research in the first place. His patience and ability to see the positive in every bit of academia has motivated me throughout the past 7 years, and I would not have been where I am today without his help and support. I have also been fortunate to have Shankar's invaluable guidance throughout my Ph.D. Every word in a conversation with Shankar is a lifelong advice, and I am grateful that he took a chance on me after my elevator pitch in the elevator of Sutardja-Dai Hall in 2012.

Many of my recent work has been in close collaboration with Anca Dragan. Over the short time that Anca has been at Berkeley, she has given me a unique viewpoint on research and life. Anca has taught me how to be caring and compassionate at the same time as being candid and constructive. I now know there is so much love in it when she says "That was horrible!"

I would also like to thank the faculty who helped shape my research path during my time at Berkeley. Babak Ayazifar for introducing me to the world of research. Edward Lee, my undergraduate research advisor, for letting me hang out with his group and learn about model-based design. Ruzena Bajcsy for teaching me how to keep myself grounded and be my own critic. Claire Tomlin and Alberto Sangiovanni-Vincentelli for their support, advice, and constructive discussions. Jeannette Wing, Eric Horvitz, and Ashish Kapoor for fruitful conversations that shaped my research during the awesome summer of 2014 in Redmond.

I would also like to thank all my co-authors and collaborators, specially people who helped with the work of this dissertation: Shromona Ghosh and Pierluigi Nuzzo for the awesome nights in DOP center, Nick Landolfi, Wenchao Li, Alexandre Donze, Vasumathi Raman, and Richard Murray whose ideas have helped with various parts of this dissertation.

Special thanks to all my mentors for letting me follow them around for a good chunk of my graduate school: Sam Burden for his wisdom, advice, and being there for all the younger folks in the group, Joh Kotker for slowly guiding me towards formal methods, Lillian Ratliff, Dan Calderon, Roy Dong, Aaron Bestick, Katherine Driggs-Campbell, and Sam Coogan for their continuing support, all the lunches, conferences, tea times, and hikes.

I also want to thank the people of TRUST, and Learn and Verify group for the fruitful conversations over the years: Rohit Sinha, Eric Kim, Ankush Desai, Daniel Fremont, Ben Caulfield, Marcell Vazquez-Chanlatte, Jaime Fisac, Dexter Scobee, Oladapo Afolabi, Eric Mazumdar, Tyler Westenbroek, Zach Wasson, Wei-Yang Tan, Markus Rabe, Tommaso Dreossi, Robert Matthew, Victor Shia, Indranil Saha, Ruediger Ehlers, Yi-Chin Wu, Susmit Jha, Bryan Brady, Dan Holcomb, Nikhil Naikal, Nishant Totla, and Garvit Juniwal.

I wouldn't be able to go through the interview season this past year without the help and unsparing advice of Ramtin Pedarsani, Yasser Shoukry, and Justine Sherry.

Many thanks to special people of TRUST and DOP center, who made everything run smoothly: Jessica Gamble, Carolyn Winter, Mary Stewart, Larry Rohrbough, Christopher Brooks, Annie Ren, Aimee Tabor, Shirley Salanio, and Shiela Humphreys.

I have had some of the most wonderful years of my life at Berkeley, and I owe that to my awesome and supportive friends: Zhaleh Amini, Nazanin Pajoom, Negar Mehr, Ramtin Pedarsani, Payam Delgosha, Sana Vaziri, Hedyeh Ahmadi, Sina Akhbari, Faraz Tavakoli, and Yasaman Bahri.

I wanted to thank my family. My parents for all their sacrifices, love, and support. My sister, Gelareh, for listening to my ‘ghors’ almost everyday since the day I can remember, and for constantly pushing me to go beyond my comfort zone. Special thanks to Amir Razavi, Vivian Razavi, Babak Sadigh, and Minoo Bahadori for being an important part of my life.

The final thanks goes to Nima Anari for making my life more enjoyable and exciting than it has ever been. His ‘kari nadare ke!’ has made me believe in myself; his genius has led me to look at the world differently, and his peacefulness is my endless comfort.

Chapter 1

Introduction

Today our society is rapidly advancing towards autonomous systems that interact and collaborate with humans, e.g., semiautonomous vehicles interacting with drivers and pedestrians, medical robots used in collaboration with doctors, or service robots interacting with their users in smart homes. Humans play a central role in such autonomous systems. They are either part of the environment that interacts with autonomy, part of the control algorithm that provides the policy for the overall system, or have a supervisory role. For instance, in autonomous driving, the autonomous vehicle inevitably interacts with humans both inside and outside of the vehicle. The car needs to interact with the driver inside of the vehicle, who might at times take over and steer the vehicle in a shared autonomy setting. We see this in lane keeping systems, where the controller decides to bring a distracted driver back into the lane. We also see it in autonomous highway driving, when the vehicle asks the person to take over control as the highway ends. Similarly, autonomous cars will not drive in an isolated space. They will need to interact with everything else that lives in the environment. This includes the pedestrians or human-driven vehicles around them. For example, the human driver in the next lane of an autonomous car, might decide to slow down or speed up, when she sees an autonomous vehicle. Her actions are simply based on her model of the autonomous car's driving and her interaction with the autonomous car. On the other hand, the actions of the autonomous car is based on its model and interaction with the human. The autonomous car might decide that the human is distracted, so its safest strategy is to not change lanes. The autonomous car might also decide to nudge in a bit into the next lane to initiate an interaction, which can potentially affect the human driver (if attentive) to slow down, making some room for the autonomous car to merge in the front. In all these settings, it is crucial to understand the interaction between the human and the rest of the system, which further requires representative models of the humans' behavior. We refer to these systems as human-robot systems, or interchangeabily as human-cyber-physical systems (h-CPS). These are cyber-physical systems (CPS) since they unite the physical processes with the cyber-world, and they are h-CPS as they have a human as part of their plant, control loop, or environment interacting with them.

As these autonomous systems enter the humans' world, their continuing interaction results in many safety concerns. These systems don't live in a vacuum, and their actions have direct consequence on the environment the humans live in. In addition, these actions highly depend on *learned* models of the environment or the humans they interact with. So robots can easily unhinge humans' safety by relying on inaccurate models that are learned from data. The safety-critical nature of these human-robot systems demands providing provably correct guarantees about their actions, models, control, and performance. This brings us to a set of fundamental problems we would like to study for human-robot systems. How do we model humans? How do we model the interaction between the human and the robot? How do we leverage the growing amount of data in the design of human-CPS in a principled manner? What mathematical models are suitable for formal analysis and design of human-robot systems? How do we address safety in reactive, high-dimensional, probabilistic environments? How do we recover in the case of an unsafe event?

1.1 Thesis Approach

One of the key aspects for achieving safe controllers for human-robot systems is the design of the interaction between the human and autonomy. This is usually overlooked by assuming humans act as external disturbances just like moving obstacles. *Humans are not simply a disturbance that needs to be avoided; they are intelligent agents with approximately rational strategies.* To model and control any human-robot systems, we are required to develop verifiable models of humans, understand the interaction between them and the other agents, and leverage this interaction in construction of safe and efficient controllers. For the design of safe controllers, we further need to formally express the desirable properties the human-robot system should satisfy, and only then we can construct a strategy for the robot that would satisfy the formalism.

In order to address safety and interaction, we divide this dissertation into two parts first focusing on *interaction-aware control*, and then discussing *safe control*.

Our approach in interaction-aware control is to model the interaction between the human and the robot as an underactuated dynamical system, where the actions of the robot can influence the actions of the human. We assume the human and the robot are reward maximizing agents. We further study the human's reward function by:

(i) actively updating the reward function in online manner to address the deviation from different people's behaviors;

(ii) actively synthesizing comparison queries to learn human's preference reward functions, and

(iii) verifying the safety of the overall system by disturbance analysis of the learned reward function.

This disturbance analysis is a step towards addressing the safety of the overall human-robot system while using learned models, which brings us to the discussion of safe control.

We take a formal methods approach to address the safety question, where we formalize the desired specifications as temporal logic properties. We then design controllers that would either satisfy the specification, or in the case that such controllers do not exist, they would systematically diagnose the failure, transfer control to the human (or some other supervisor), and even provide suggested repairs for the specifications. We study different types of specifications to address continuity and stochasticity present in human-robot systems.

Here, we bridge ideas from formal methods, control theory, and human-robot interaction to understand and design controllers that are interactive, can influence people, and can guarantee to satisfy high level properties such as safety.

1.2 Contributions

This thesis makes the following contributions.

The goal of my thesis is to develop a safe and interactive control framework for human-robot systems.

Planning for Robots to Coordinate with Other People:

We provide a formalism for interaction between a human and a robot as a partially observable two-player game. The human and the robot can both act to change the state of the world, and they have partial information because they don't know each others' reward functions. We model the human as an agent who is approximately optimizing her reward function learned through demonstrations [129, 3, 188, 107], and the robot as a rational agent optimizing its own reward function.

This formulation has two issues: intractability, especially in continuous state and action spaces, and failing to capture human behavior, because humans tend to not follow Nash equilibria in day to day tasks [76]. We introduce a simplification of this formulation to an underactuated system. We assume that the robot decides on a trajectory $\mathbf{u}_{\mathcal{R}}$, and the human computes a best response to $\mathbf{u}_{\mathcal{R}}$ (as opposed to trying to influence $\mathbf{u}_{\mathcal{R}}$ as would happen in a game).

In addition, we derive an approximate solution for our system based on Model Predictive Control and a quasi-newton optimization. At every step, the robot replans a trajectory $\mathbf{u}_{\mathcal{R}}$ by reasoning about the optimization that the human would do based on a candidate $\mathbf{u}_{\mathcal{R}}$. We use implicit differentiation to obtain a gradient of the human's trajectory with respect to the robot's. This enables the robot to compute a plan in close to real-time. We evaluate our algorithm through a user study in an autonomous driving

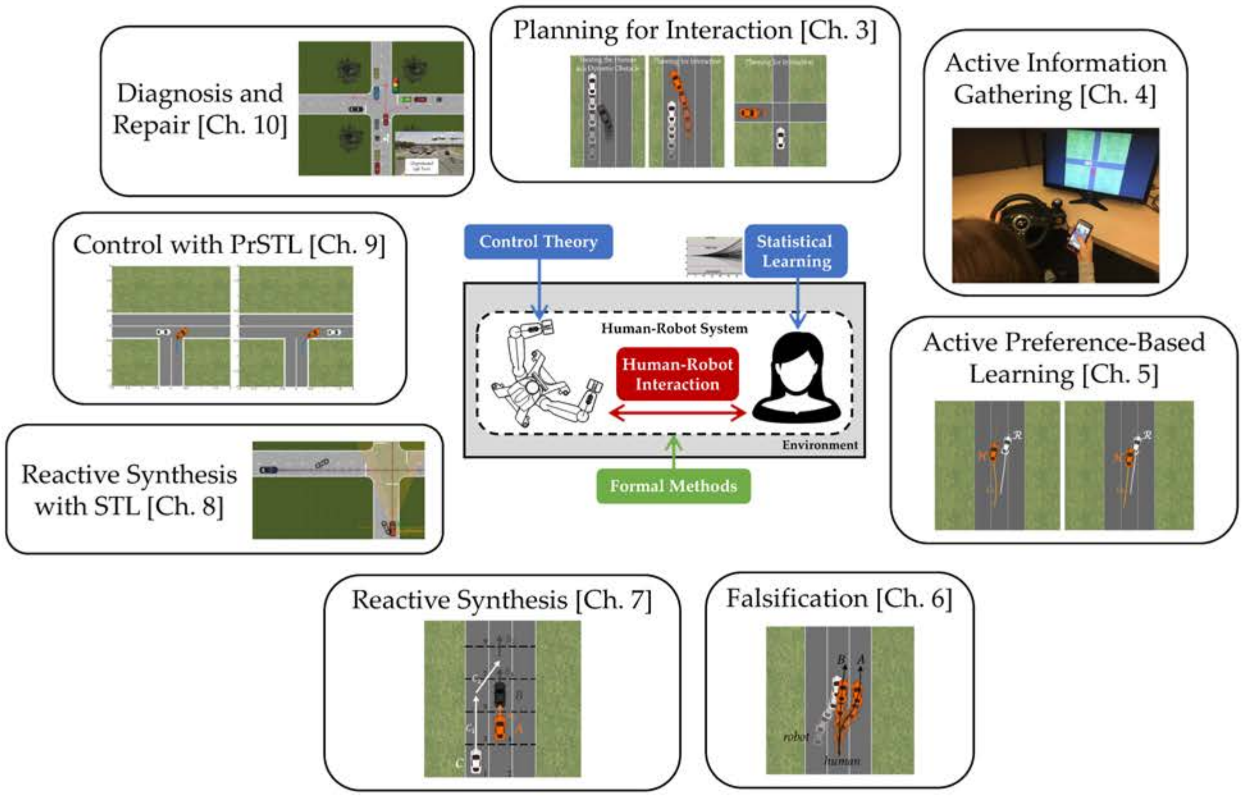

Planning for Interaction [Ch. 3]
Diagnosis and Repair [Ch. 10]
Active Information Gathering [Ch. 4]
Control with PrSTL [Ch. 9]
Control Theory
Statistical Learning
Human-Robot System
Human-Robot Interaction
Environment
Formal Methods
Active Preference-Based Learning [Ch. 5]
Reactive Synthesis with STL [Ch. 8]
Reactive Synthesis [Ch. 7]
Falsification [Ch. 6]
robot
human

Figure 1.1: Thesis overview: We develop a framework for safe and interactive autonomy for human-robot systems. In *interaction-aware control*, we first model the interaction between humans and robots using computational models of human behaviors, we then actively update the human models by online information gathering. We also efficiently learn human's preferences by actively synthesizing comparison queries, and then analyze the accuracy of our human models for the purposes of verification. In *safe control*, we take a reactive synthesis approach, where we find autonomous controllers that would satisfy temporal logic specifications, and systematically transfer control to the human if a safe controller does not exist. We further design a new formalism to express stochastic properties and find controllers that would satisfy such stochastic temporal logic specifications.

scenario, which suggests that robots are capable of affecting human's actions and driving them to a desired state (Chapter 3) [159, 160].

Active Information Gathering over Human's Internal State:

Our human-robot interaction model depends on accurate estimations of human reward function. This can be done by estimating the human reward offline from training data, but ultimately every driver is different, and even the same driver is sometimes more or less aggressive, more or less attentive, and so on. We thus explore estimating the human reward function online. This turns the problem into a partially observable Markov decision process (POMDP), with the human reward parameters as the hidden state. Prior work that incorporates some notion of human state into planning has thus far separated estimation and planning, always using the current estimate of the human state to determine what the robot should do [81, 58, 18]. Although efficient, these approximations sacrifice an important aspect of POMDPs: the ability to *actively gather information*.

We take advantage of the underactuated system to gather information about the human reward parameters. Rather than relying on passive observations, the robot actually accounts for the fact that humans will react to their actions: it uses this knowledge to select actions that will trigger human reactions which in turn will clarify the internal state. We apply our algorithm to estimating a human driver's style during the interaction with an autonomous vehicle, and our results in simulation and a user study suggest that our algorithm is capable of leveraging robot's actions for estimation that leads to significantly higher accuracy in identifying the correct human internal state (Chapter 4) [158, 160].

Active Preference-Based Learning of Reward Functions:

Reward functions play a central role in specifying how dynamical systems should act. For many systems, human's have a difficult time providing demonstrations of what they want. We propose a *preference-based* approach to learning desired reward functions in a dynamical system. Instead of asking for demonstrations, or for the value of the reward function for a sample trajectory (e.g., "rate the safety of this driving maneuver from 1 to 10"), we ask people for their relative preference between two sample trajectories (e.g., "is ξ_1 more safe or less safe than ξ_2?").

We provide an algorithm for *actively synthesizing* such comparison queries from scratch. Our algorithm uses continuous optimization in the query space to maximize the expected volume removed from the hypothesis space. The human's response assigns weights to the hypothesis space in the form of a *log-concave distribution*, which provides an approximation of the objective. We provide a bound on the number of iterations required to converge. In addition, we show that our algorithm converges faster than non-active and non-synthesis techniques in learning the reward function in an autonomous driving setting. We illustrate the performance of our algorithm in terms of accuracy of the reward function learned through an in-lab usability study (Chapter 5) [156].

Falsification for Human-Robot Systems:

Safe and interactive human-robot systems strongly depend on reliable models of human behavior. It is crucial to be able to formally analyze such human models (e.g. learned reward functions) to address the safety and robustness of the overall system. We provide a new approach for rigorous analysis of human-robot systems that are based on *learned* models of human behavior. We formalize this problem as a constrained optimization, where we examine the existence of a falsifying controller for the human that lies within a bounded region of the learned model and could possibly lead to unsafe outcomes. For instance, in a driving scenario, we find a sequence of possible human actions that potentially leads to a collision between the vehicles. We reduce this optimization to a more efficiently-solvable problem with linear constraints. In addition, we provide an efficient (almost linear time) optimization-driven approach to estimate the error bound between the true human reward function and the controller's estimate. We evaluate our technique for an autonomous driving example, where we find such falsifying actions within the learned safety bound (Chapter 6) [155].

Reactive Synthesis for Human-Robot Systems:

We formalize the problem of *safe control* for human-robot systems as a reactive synthesis problem [139]. We consider a shared control setting, where either the human or the robot can operate the system, and we provide controllers that are guaranteed to satisfy a set of high level specifications. Our algorithm satisfies four criteria for the overall system: i) *monitoring:* the robot determines if any human intervention is required based on monitoring the environment. ii) *minimal intervention:* the robot asks for human intervention only if it is necessary. iii) *prescient:* the robot determines if a specification is going to be violated ahead of time to give enough takeover time to the human based on the human's reaction time. iv) *conditional correctness:* the robot satisfies the high level specification until when the human needs to intervene.

We use a discrete state and action model of the robot to be able to guarantee satisfaction of high level Linear Temporal Logic (LTL) specifications. We leverage counterstrategy graphs in our algorithm to formalize the intervention problem. Based on the four desired criteria, our algorithm finds a s-t minimum cut of the counterstrategy graph to determine when the human needs to takeover. We only monitor the human's reaction time as part of the human model; however, our algorithm is able to automatically find the environment assumptions that need to be monitored and systematically transfers control to the human if it can't guarantee satisfaction of the high level specifications. We showcase our algorithm for a system motivated by driver-assistance systems (Chapter 7) [110].

Reactive Synthesis from Signal Temporal Logic

Reactive synthesis from LTL is a powerful technique that enables us to provide correctness guarantees for autonomous systems. However, using LTL requires a discrete state and action dynamical system. Such discretization is not very realistic for many robotics and control applications as they are usually inherently continuous. So instead, we formalize the problem of reactive synthesis from Signal Temporal Logic (STL), a specification language that is defined over continuous time and real-valued signals. Synthesizing reactive controllers under STL allows us to provide correctness guarantees even when we are dealing with continuous state and action dynamical systems.

We provide a new algorithm using ideas from *counterexample-guided inductive synthesis* (CEGIS) [166]. We solve a series of counterexample-guided optimization problems that result in finding a controller that satisfies the given STL specifications in a receding horizon fashion. We demonstrate the effectiveness of our algorithm in a driving example with a simple adversarial model of other human-driven vehicles. Similar to [146], we rely on transforming STL specifications to mixed integer linear program (MILP) encodings. Our method is a fundamentally novel approach to *reactive* synthesis for hybrid systems, different from most current methods, which often rely on model transformations, e.g., abstraction and discretization (Chapter 8) [149].

Safe Control under Uncertainty

As powerful as STL specifications are, they do not have the capability of encoding stochastic properties that can arise in human-robot systems. The desired properties in many robotics applications are based on the output of estimation and learning algorithms. For instance, safe control of a flying quadrotor depends on how certain the estimation algorithm is about the location of the other agents in the environment including the human agents. This is based on the current sensor data and specific classification methods being used.

We formally define a specification language to address the safe control of robots under such uncertain environment or human models. Probabilistic Signal Temporal Logic (PrSTL) is our specification language that enables expressing probabilistic properties, which can embed Bayesian graphical models. This probabilistic logical specification language enables reasoning about safe control strategies by embedding various predictions and their associated uncertainty. Furthermore, we solve a receding horizon control problem to satisfy PrSTL specifications using mixed integer semidefinite programs. We showcase our algorithm for examples in autonomous driving and control of quadrotors in uncertain environments (Chapter 9) [154].

Diagnosis and Repair for Synthesis from Signal Temporal Logic

When synthesizing safe controllers that satisfy high level specifications such as LTL, STL, or PrSTL, we usually transform the problem to a game (in the case of LTL), or an optimization (in the case of STL and PrSTL), and use existing techniques to solve for a safe control strategy. However, there could be situations where there does not exist a controller that satisfies all the given specifications. In this context, an unrealizable STL specification leads to an infeasible optimization problem. We leverage the ability of existing mixed integer linear programming (MILP) solvers to localize the cause of infeasibility to so-called *Irreducibly Inconsistent Systems* (IIS). We propose an algorithm that uses the IIS to localize the cause of unrealizability to the relevant parts of the STL specification. Additionally, we give a method for generating a *minimal set of repairs* to the STL specification such that, after applying those repairs, the resulting specification is realizable. The set of repairs is drawn from a suitably defined space that ensures that we rule out vacuous and other unreasonable adjustments to the specification. Our algorithms are sound and complete, i. e., they provide a correct diagnosis, and always terminate with a reasonable specification that is realizable using the chosen synthesis method, when such a repair exists in the space of possible repairs.

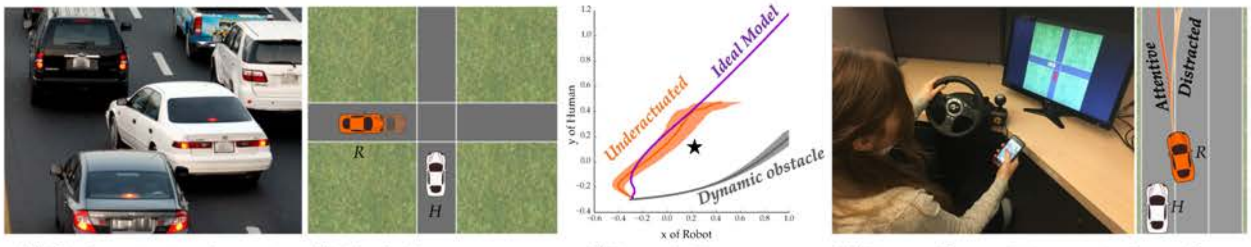
1.3 Overview

Interaction-Aware Control

Effects on Human Actions. Safe and interactive human-robot systems requires modeling the human, and taking into account the interaction that weaves the agents together. For instance, currently autonomous cars tend to be overly *defensive* and obliviously *opaque*. When needing to merge into another lane, they will patiently wait for another driver to pass first. When stopped at an intersection and waiting for the driver on the right to go, they will sit there unable to wave them by. They are very capable when it comes to obstacle avoidance, lane keeping, localization, active steering and braking [170, 108, 56, 55, 54, 44, 106]. But when it comes to other human drivers, they tend to rely on simplistic models: for example, assuming that other drivers will be bounded disturbances [71, 149], they will keep moving at the same velocity [175, 118, 154], or they will approximately follow one of a set of known trajectories [174, 77].

These models predict the trajectory of other drivers as if those drivers act in isolation. They reduce human-robot interaction to obstacle avoidance: the autonomous car's task is to do its best to stay out of the other drivers' way. It will not nudge into the lane to test if the other driver yields, nor creep into the intersection to assert its turn for crossing.

Our insight is that the actions of an autonomous car affect the actions of other drivers, and leveraging these effects in planning improves efficiency and coordination.



(a) Vehicle merging in and expecting other cars to slow down. (b) Robot backing up to encourage human cross first. (c) User study. Human crosses first. (d) Distracted human driving on a simulator, when the robot does active info gathering.

Figure 1.2: Modeling the *interaction* between human and autonomy, and leveraging *effects* on human actions for better safety and performance (a,b,c). Active information gathering over human's internal state (d).

In this thesis, we develop an optimization-based method for planning an autonomous vehicle's behavior in a manner that is cognizant of the effects it will have on human driver actions. This optimization leads to plans like the ones in Figure 1.2. For example, when a car nudges into my lane, I slow down to make room for it (Figure 1.2(a)). We formulate this problem as an *incomplete information two-player game*, where the two agents (human and robot) don't have access to each other's reward functions, and we approximate this game as an underactuated system [159, 160]: the robot's actions change not only robot state, but also influence human actions and thus human state. We model other drivers as acting approximately optimally according to some reward function that depends on state, human actions, as well as robot actions. We explore learning this human reward function offline. Our algorithm enables autonomous cars to be more efficient, less defensive, and better at coordinating with people through motion (e.g., inching backwards at an intersection to signal the human driven vehicles to cross first, which normally needs to be hand-coded, but has now instead *emerged out of the optimization* as in Figure 1.2(b),(c)). Our user studies on a car simulator (Figure 1.2 (d)) suggest that the robot has the effects it anticipates on human drivers, which leads to efficient *coordination* [159, 160].

Online Active Information Gathering over Human Internal State. It is clear that not all humans behave the same way. For example, in the case of driving, some drivers are aggressive or timid, some are attentive or distracted, and an accurate estimate of this knowledge is crucial in autonomous car's decision making. This estimate can be passively updated based on observations of driver actions. However, often we need to *get lucky* to observe informative behavior from drivers. For example, it is very difficult to tell a priori if a particular driver is going to let us merge into her lane without first trying to slowly nudge into that lane and observe her reactions. Thus, autonomous cars need a more intelligent approach to update this estimate.

Humans respond to robot actions; therefore, the robot has the opportunity to actively gather information about humans' internal state, e.g., whether they are distracted.

We formalize this problem as a continuous state and action Partially Observable Markov

Decision Process (POMDP), where the human's internal state denotes the unobserved variable. We augment the robot's reward function with an exploration term (i.e. entropy over the belief of the human's internal state), which results in a strategy for the robot that actively takes actions to improve its estimate. Exploration actions emerge out of optimizing for information gain, such as the robot *nudging into* the other lane and responding *safely* (going back to its lane or completing the merge) if the human driver is distracted or attentive respectively (Figure 1.2(d)) [158, 160].

Active Preference Based Learning of Reward Functions. Most of our work in interaction-aware control depends on acquiring representative models of human behaviors such as their reward functions. Efficiently learning reward functions that encode humans' preferences for how a dynamical system should act results in various challenges. It is quite difficult for people to demonstrate trajectories for robots with more than a few degrees of freedom or even to provide labels for precisely how much reward an action or trajectory should get, like a *robot motion* or a *driving maneuver*. Moreover, the learned reward function strongly depends on what environments and trajectories were experienced during the training phase. Our approach to efficiently learn human's preference reward functions is by *actively synthesizing* comparison queries that human's can respond to, and use ideas from volume-maximization and adaptive submodular optimization to actively synthesize such queries, which allow us to quickly converge to human's reward function [156].

Measuring Human Variations. Since many human-robot systems are emerging into our every day lives, we are required to understand and measure their performance and safe interaction in this environment. For instance, our framework depends on the quality of the learned human models, e.g., we would like to analyze how good of a reward function we have learned. Under the assumption that humans are *approximately rational*, we can either use the principle of maximum entropy to learn such reward functions, as we do in [159, 158, 160], or apply our active preference-based learning technique to quickly converge to human's preference reward functions [156].

These data-driven human models such as human reward functions are usually constructed based on large datasets that give access to a single model of rationality. However, humans vary in how they handle different situations, and we cannot fit a single model to all humans. In safety-critical scenarios, we need to be able to quantize and measure *how humans can differ from the fitted model.* As a step towards verified human modeling, we construct an efficient (almost linear, i.e., $\mathcal{O}(n \log n)$, where n is the number of queries) algorithm to quantize and learn a distribution over the variations from the fitted model by querying individual humans on actively generated scenarios.

Overall, this thesis takes a step towards robots that account for their effects on human actions in situations that are not entirely cooperative, and leverage these effects to coordinate with people. Natural coordination and interaction strategies are not hand-coded, but emerge out of planning in our model. Further, we study various models of our reward function through comparison based learning or falsification in our human-robot systems. The material in the first part of this thesis is based on joint work with Anca D.

Dragan, S. Shankar Sastry, and Sanjit A. Seshia [159, 158, 160, 156, 155].

Safe Control

Reactive and Stochastic Controller Synthesis from Temporal Logics. The problem of correct-by-construction controller synthesis has been addressed in the area of formal methods. We formalize a set of desired high level specification, then the *realizability* problem is to find an autonomous strategy for the robot so no matter what the environment or the other agents, e.g., humans do, the strategy for the robot is guaranteed to satisfy the specification. Linear Temporal Logic (LTL) is a popular formalism for stating these specifications. However, there is a significant gap between the expressivity of specifications in LTL and the desired requirements in robotics applications including human-robot systems.

Most of these applications deal with inherently continuous systems, and the discrete nature of LTL is incapable of representing properties over continuous time and space trajectories. Signal Temporal Logic (STL) is a specification language that addresses some of these shortcomings by providing a language over real-valued and continuous-time signals. However, synthesizing reactive controllers that satisfy STL specifications is a challenging problem. Previous work by Raman et al. has studied synthesizing *non-reactive* controllers under STL specifications by translating the specifications to mixed-integer linear program (MILP) constraints [146]. As part of this thesis in collaboration with Vasumathi Raman, Alexandre Donze, Richard Murray, and Sanjit A. Seshia [149], we study the problem of *reactive* synthesis under STL specifications. We use a similar approach to encode the specifications as MILPs, and leverage a series of counterexample guided optimizations to find a controller that would satisfy the desired property under reactive specifications [149].

Even though STL addresses continuous time and real-valued specifications, it still lacks the ability to express the stochasticity arising from the environment or the human models. It would be unrealistic to assume, we deterministically know where every agent is located. Our sensors are noisy, hence our estimation algorithms at best can probabilistically locate every agent in the environment. To address these limitations, we introduce a new formalism, *Probabilistic Signal Temporal Logic (PrSTL)*, which is an expressive language that closes this gap by including machine learning techniques, namely Bayesian classifiers as part of its predicates. This is joint work done at Microsoft Research, Redmond with Ashish Kapoor [154]. We further formalize a controller synthesis problem under satisfaction of *PrSTL* specifications, and solve a model predictive control problem, where the synthesized strategy would satisfy the desired *PrSTL* properties. We reduce this optimization to a mixed-integer second-order cone program (MISOCP) [154].

Explaining Failures: Accountable Human Intervention. Synthesizing safe controllers under stochastic or adversarial environments does not always result in *feasible* solutions. Sometimes the *extremely safe* strategy for the robot does not exist.

For example, imagine making an unprotected left turn at an intersection. When the light turns green, one enters the intersection, and yield by waiting for the queue of oncoming vehicles to pass through as shown in Figure 8.1. Humans can comfortably make these left turns even in complex scenarios, e.g., when there is a large number of oncoming vehicles. They either decide to go straight and make a left turn at the next intersection (*change the specification*), or cut in front of the oncoming traffic after making eye-contact with some oncoming vehicle *(violate the specification)* [1]. However, there does not necessarily exist an autonomous controller that is capable of making an unprotected left turn while satisfying all the specifications.

> *Our goal is to detect these infeasibilities, explain them, and systematically transfer control back to the human.*

Of course, other requirements regarding the human operator such as her *reaction time* or *minimal intervention* need to be considered for safe an reliable human intervention [110].

We address this problem both in the setting of reactive synthesis from LTL and STL. When the specification is written in LTL, together with Wenchao Li, Sanjit A. Seshia, and S. Shankar Sastry, we use an automata-based approach by extracting counterstrategy graphs. Counterstragegy graphs encode all the possible transitions that the reactive agent (environment or other human agents) can make in order to forcibly drive the robot to a failing state. We explain the failure reasons by computing a minimum-cut of this graph. When the specification is written in STL, together with Shromona Ghosh, Pierluigi Nuzzo, Vasumathi Raman, Alexandre Donze, Alberto Sangiovanni-Vincentelli, S. Shankar Sastry, and Sanjit A. Seshia, we use an optimization-based approach to *diagnosis* and *repair* the failure. We translate the problem to a mixed-integer linear program, and use slack variables to detect the specifications that cause the infeasibilities, and provide a minimal repair [65]. Our algorithm shows its ***accountability*** in failure scenarios by presenting the repaired formulas during systematic human intervention.

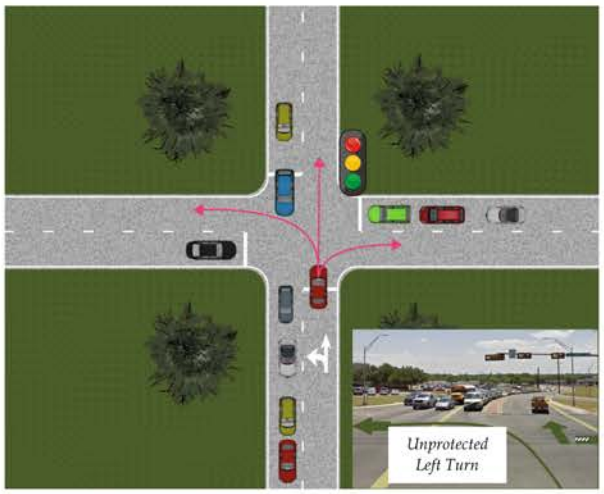


Figure 1.3: Unprotected Left Turn. There does not exist a *feasible* or *realizable* strategy for the red car to safely make a left turn since the oncoming queue vehicles can act adversarially and never end during the duration of green light.

[1]Pittsburgh Left.

Overall this thesis takes a step towards the design of safe and interactive autonomous controllers for human-robot systems. It leverages computational models of human behaviors to design an interaction model between the human and robot in order to influence the human for better safety, efficiency, coordination, and estimation. We also take an effort in better designing humans' reward functions through active preference based learning as well as finding a sequence of falsifying actions for the human within an error bound of the reward function. Under such uncertain human models, we address the safe control problem by defining relevant specifications and synthesizing a controller that would satisfy such specifications using either automata-based or optimization-based techniques. In addition, we diagnose and repair the specifications in case of infeasibilities, which enables a systematic transfer of control to the human.

Chapter 2

Preliminaries

In this chapter, we discuss some of the preliminaries required for the design of algorithms for safe and interactive autonomy. We start by discussing the formalism and the dynamical system that describes human-robot systems. Further, we explain the use of model predictive control in such systems. We then describe our choice of human models using ideas from learning from demonstrations. In addition, we discuss various specification languages such as temporal logics, as well as the details of the simulators and simulation models, we used in our experiments.

2.1 Formalism for Human-Robot Systems

We focus on a human-robot system consisting of an autonomous agent (robot) interacting in an environment with human agents, e.g. an autonomous vehicle interacting with human-driven vehicles on a shared road as shown in Figure 2.1. Here, the orange car (robot) is interacting with the white car (human). Our goal is for the autonomous agent to plan its actions in a manner that is cognizant of their effects and interactions with the human. We restrict ourselves to the two agent case, we have an autonomous agent $\mathcal{R}$ sharing an environment (such as a road in the driving case) with a human agent $\mathcal{H}$.

We model the problem as a fully observable dynamical system, but one in which the robot actions have consequences beyond their immediate effects on the robot itself: they will also affect human actions which in turn will affect state.

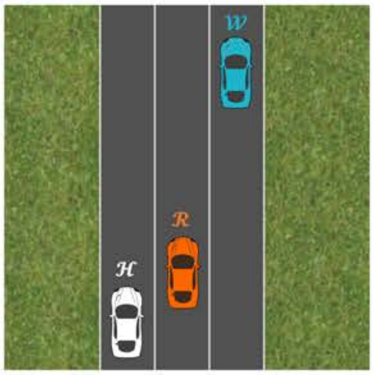


Figure 2.1: A human-robot system. Here $\mathcal{R}$ is the robot car, $\mathcal{H}$ is the human car, and $\mathcal{W}$ is another vehicle on the road that we treat as disturbance.

This modeling choice addresses how the actions of the human and robot together can drive the dynamics of the system to desirable states. This would require modeling actual actions of humans in a computational setting rather than a high level behavioral model of cognition. The model of the human should explain how the actions of the human influence or gets influenced in this dynamical system.

A state $x \in X$ in our system is continuous, and includes the state of the human and robot. The robot can apply continuous controls $u_{\mathcal{R}}$, which affect state immediately through a dynamics model $f_{\mathcal{R}}$:

$$x' = f_{\mathcal{R}}(x, u_{\mathcal{R}}) \tag{2.1}$$

However, the next state the system reaches also depends on the control the human chooses, $u_{\mathcal{H}}$. This control affects the intermediate state through a dynamics model $f_{\mathcal{H}}$:

$$x'' = f_{\mathcal{H}}(x', u_{\mathcal{H}}) \tag{2.2}$$

The overall dynamics of the system combines the two. We note that the ordering of the actions of human or robot does not matter, and we assume these control inputs are taken simultaneously:

$$x^{t+1} = f_{\mathcal{H}}\big(f_{\mathcal{R}}(x^t, u^t_{\mathcal{R}}), u^t_{\mathcal{H}}\big) \tag{2.3}$$

We let f denote the dynamics of such discrete-time system:

$$x^{t+1} = f(x^t, u^t_{\mathcal{R}}, u^t_{\mathcal{H}}) \tag{2.4}$$

Further, we assume the robot has a particular reward function at every time step. The robot's reward function depends on the current state, the robot's action, as well as the action that the human takes at that step in response, $r_{\mathcal{R}}(x^t, u^t_{\mathcal{R}}, u^t_{\mathcal{H}})$. At every time step, the robot's actions can be explained by maximizing this reward function. We assume, this reward function is a weighted combination of features that the robot cares about, e.g., in the driving example such features include collision avoidance, staying on the road, or distance to the final goal.

In Figure 2.1, imagine the orange car's goal is to go to the left lane. So its actions would be based on optimizing a reward function that has a term regarding distance to the left lane, distance to the blue and white car for collision avoidance, the heading and speed of the orange car, and its distance to the lanes or road boundaries. Ideally, the orange car would optimize such function to take safe and interactive actions towards its goal. It is clear that such a reward function depends on the current state of the world x^t, as well as the actions of the other vehicles on the road $u^t_{\mathcal{H}}$ and its own actions $u^t_{\mathcal{R}}$.

The key aspect of this formulation is that the robot will have a model for what $u_{\mathcal{H}}$ will be, and use that in planning to optimize its reward.

Model Predictive Control (MPC):

The robot will use Model Predictive Control (MPC) [124] (also known as Receding Horizon Control (RHC)) at every iteration. MPC is a popular framework for the design of autonomous controllers since generating a closed-loop policy is intractable in a complex, nonlinear, and non-convex human-robot dynamical system. In addition, computing controllers for a finite horizon has the benefit of computational tractability as well as addressing the limited range of sensors.

It will compute a finite horizon sequence of actions to maximize its reward. We reduce the computation required by planning for a shorter horizon of N time steps. We execute the control only for the first time step, and then re-plan for the next N at the next time step [35].

Let $\mathbf{x} = (x^1, \ldots, x^N)^\top$ denote a sequence of states over a finite horizon, N, and let $\mathbf{u}_\mathcal{H} = (u^1_\mathcal{H}, \ldots, u^N_\mathcal{H})^\top$ and $\mathbf{u}_\mathcal{R} = (u^1_\mathcal{R}, \ldots, u^N_\mathcal{R})^\top$ denote a finite sequence of continuous control inputs for the human and robot, respectively. We define $R_\mathcal{R}$ as the robot's reward over the finite MPC time horizon:

$$R_\mathcal{R}(x^0, \mathbf{u}_\mathcal{R}, \mathbf{u}_\mathcal{H}) = \sum_{t=1}^{N} r_\mathcal{R}(x^t, u^t_\mathcal{R}, u^t_\mathcal{H}), \tag{2.5}$$

where x^0 denotes the present physical state at the current iteration, and each state thereafter is obtained from the previous state and the controls of the human and robot using the given dynamics model, f.

At each iteration, we desire to find the sequence $\mathbf{u}_\mathcal{R}$ which maximizes the reward of the robot, but this reward ostensibly depends on the actions of the human. The robot might attempt to influence the human's actions, and the human, rationally optimizing for her own objective, might likewise attempt to influence the actions of the robot.

$$\mathbf{u}^*_\mathcal{R} = \arg\max_{\mathbf{u}_\mathcal{R}} R_\mathcal{R}\left(x^0, \mathbf{u}_\mathcal{R}, \mathbf{u}^*_\mathcal{H}(x^0, \mathbf{u}_\mathcal{R})\right) \tag{2.6}$$

Here, $\mathbf{u}^*_\mathcal{H}(x^0, \mathbf{u}_\mathcal{R})$ is what the human would do over the next N steps if the robot were to execute $\mathbf{u}_\mathcal{R}$.

The robot does not actually know $\mathbf{u}^*_\mathcal{H}$, but in the future sections we propose a *model* for the human behavior that the robot can use to make this problem tractable.

Similarly the robot's actions and dynamics can be affected by other elements in the environment. These could be other existing agents, or simply disturbances present in the environment. For instance, in Figure 2.1, we can consider other vehicles on the road (e.g. blue car) that are not in close interaction with the robot as such disturbances. We can then extend our formulation, and consider a continuous-time system Σ of the form:

$$\dot{x} = f_c(x, u^t_\mathcal{R}, u^t_\mathcal{H}, w)$$

where $w \in W$ is the external input provided by the environment that can possibly be adversarial. We will refer to w as the *environment* input. Here, f_c is the continuous dynamics of the system.

Given a sampling time $\Delta t > 0$, we assume that Σ admits a discrete-time approximation Σ_d of the form:

$$x^{t+1} = f(x^t, u^t_\mathcal{R}, u^t_\mathcal{H}, w^t) \tag{2.7}$$

An infinite *run*

$$\xi^{\omega} = (x^0, u^0_{\mathcal{R}}, u^0_{\mathcal{H}}, w^0)(x^1, u^1_{\mathcal{R}}, u^1_{\mathcal{H}}, w^1)(x^2, u^2_{\mathcal{R}}, u^2_{\mathcal{H}}, w^2)...$$

of Σ_d is a sequence of state and actions starting from the initial state x^0 that follow the dynamical system f. Given $x^0 \in X$ and the finite sequence of actions $\mathbf{u}_{\mathcal{R}}$, $\mathbf{u}_{\mathcal{H}}$, and $\mathbf{w} = (w^1, \dots, w^N)^\top$, the finite run $\xi = (x_0, \mathbf{u}_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}, \mathbf{w})$ is a unique sequence generated following equation (2.7):

$$\xi = (x^0, \mathbf{u}_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}, \mathbf{w}) = (x^0, u^0_{\mathcal{R}}, u^0_{\mathcal{H}}, w^0)(x^1, u^1_{\mathcal{R}}, u^1_{\mathcal{H}}, w^1), \dots, (x^N, u^N_{\mathcal{R}}, u^N_{\mathcal{H}}, w^N) \tag{2.8}$$

In addition, we introduce a generic cost function $J(\xi)$ similar to the generic reward function introduced in (2.5) for the robot $R_{\mathcal{R}}(\xi)$ that maps (infinite and finite) runs to $\mathbb{R}$.

2.2 Inverse Reinforcement Learning

Modeling human behavior is a challenging task that has been addressed in various fields. Our goal is to use computational models of human behaviors to be able to inform the design of our control algorithms for human-robot systems. Apprenticeship learning is a possible technique for constructing such computational models of humans; the learner finds a reward function that explains observations from an expert providing demonstrations [129, 3]. Similar to the robot's reward function, we model $\mathcal{H}$ as an agent who noisily optimizes her own reward function [188, 107, 165, 98]. We parametrize the human reward function as a linear combination of a set of hand-coded features:

$$r_{\mathcal{H}}(x^t, u^t_{\mathcal{R}}, u^t_{\mathcal{H}}) = w \cdot \phi(x^t, u^t_{\mathcal{R}}, u^t_{\mathcal{H}}) \tag{2.9}$$

Here, $\phi(x^t, u^t_{\mathcal{R}}, u^t_{\mathcal{H}})$ is a vector of such features, and w is a vector of the weights corresponding to each feature. The features describe different aspects of the environment or the robot that the human should care about. We apply the principle of maximum entropy [188, 187] to define a probability distribution over human demonstrations $\mathbf{u}_{\mathcal{H}}$, with trajectories that have higher reward being more probable:

$$P(\mathbf{u}_{\mathcal{H}} | x^0, w) = \frac{\exp(R_{\mathcal{H}}(x^0, \mathbf{u}_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}))}{\int \exp(R_{\mathcal{H}}(x^0, \mathbf{u}_{\mathcal{R}}, \tilde{\mathbf{u}}_{\mathcal{H}}))d\tilde{\mathbf{u}}_{\mathcal{H}}} \tag{2.10}$$

We then do an optimization over the weights w in the reward function that make the human demonstrations the most likely:

$$\max_{w} P(\mathbf{u}_{\mathcal{H}} | x^0, w) \tag{2.11}$$

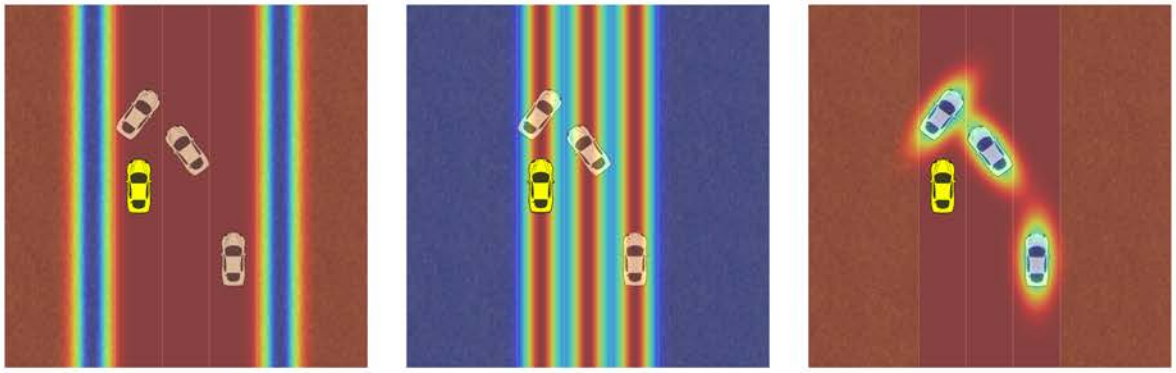

(a) Features for the road boundaries (b) Features for staying inside the lanes (c) Features for avoiding other vehicles

Figure 2.2: Features used in IRL for the human driven vehicle. In the heat map, the warmer colors correspond to higher reward. In (a), we show the features corresponding to staying within road boundaries, in (b), we show the features for staying within each lane, and in (c) we show non-spherical gaussian features corresponding to avoiding collisions.

We approximate the partition function in equation (2.10) following [107], by computing a second order Taylor approximation around the demonstration:

$$\begin{aligned} R_{\mathcal{H}}(x^0, \mathbf{u}_{\mathcal{R}}, \tilde{\mathbf{u}}_{\mathcal{H}}) \simeq R_{\mathcal{H}}(x^0, \mathbf{u}_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}) + (\tilde{\mathbf{u}}_{\mathcal{H}} - \mathbf{u}_{\mathcal{H}})^\top \frac{\partial R_{\mathcal{H}}}{\partial \mathbf{u}_{\mathcal{H}}} + \\ (\tilde{\mathbf{u}}_{\mathcal{H}} - \mathbf{u}_{\mathcal{H}})^\top \frac{\partial^2 R_{\mathcal{H}}}{\partial \mathbf{u}_{\mathcal{H}}^2} (\tilde{\mathbf{u}}_{\mathcal{H}} - \mathbf{u}_{\mathcal{H}}), \end{aligned} \tag{2.12}$$

which makes the integral in equation (2.10) a Gaussian integral, with a closed form solution [107].

Features

The heat map of features we have used for the autonomous driving example in this disseration are shown in Figure 2.2. The warmer colors correspond to higher rewards. In Figure 2.2(a), we show the features corresponding to staying within the boundaries of the roads. In Figure 2.2(b), we have features corresponding to staying within each lane, and in Figure 2.2(c), we have features corresponding to collision avoidance, which are non-spherical Gaussians, and their major axis is along the vehicle's heading.

More specifically for the autonomous driving example discussed in this dissertation, we choose 5 features. In general, we use Gaussian kernels for the features that encode distances. These features are based on:

- $\phi_1 \propto c_1 \cdot \exp(-c_2 \cdot d^2)$: distance to the boundaries of the road, where d is the distance between the vehicle and the road boundaries and c_1 and c_2 are appropriate scaling factors as shown in Figure 2.2(a).
- ϕ_2: distance to the middle of the lane, where the function is specified similar to ϕ_1 as shown in Figure 2.2(b).
- $\phi_3 = (v - v_{\max})^2$: higher speed for moving forward through, where v is the velocity of the vehicle, and $v_{\max}$ is the speed limit.
- $\phi_4 = \beta_{\mathcal{H}} \cdot \boldsymbol{n}$: heading; we would like the vehicle to have a heading along with the road using a feature, where $\beta_{\mathcal{H}}$ is the heading of $\mathcal{H}$, and $\boldsymbol{n}$ is a normal vector along the road.
- ϕ_5 corresponds to collision avoidance, and is a non-spherical Gaussian over the distance of $\mathcal{H}$ and $\mathcal{R}$, whose major axis is along the robot's heading as shown in Figure 2.2(c).

Demonstrations

We collected demonstrations of a single human driver in an environment with multiple autonomous cars, which followed precomputed routes.

Despite the simplicity of our features and robot actions during the demonstrations, the learned human model is enough for the planner to produce behavior that is human-interpretable, and that can affect human action in the desired way as discussed in the future chapters.

2.3 Synthesis from Temporal Logic

In the *Safe Control* part of this dissertation, we focus on the idea of correct-by-construction control, which enables the design of controllers that are guaranteed to satisfy various specifications such as safety of the human-robot system. Our work is based on the *reactive synthesis* approach introduced in the area of formal methods. The idea of temporal logic synthesis is to automatically construct an implementation that is guaranteed to satisfy a behavioral description of the system expressed in temporal logic [142]. In this section, we give an overview on synthesizing reactive modules from a specification given in Temporal Logic. This problem originates from Church's problem formulated in 1965 and can be viewed as a two-player game between the system and the environment.

Problem 1 (Alonzo Church's Synthesis Problem). *Given a requirement which a circuit is to satisfy, we may suppose the requirement expressed in some suitable logistic system which is an extension of restricted recursive arithmetic.*

The synthesis problem *is the to find recursion equivalences representing a circuit that satisfies the given requirement (or alternatively, to determine that there is no such circuit) [34].*

Linear Temporal Logic

Specifications are detailed descriptions of the desired properties of a system (e.g. autonomous agent, robot) along with its environment. We use Linear Temporal Logic (LTL) [141] to formally define such desired specifications. A LTL formula is built of *atomic propositions* $\omega \in \Pi$ that are over states of the system that evaluate to `True` or `False`, *propositional formulas* ϕ that are composed of atomic propositions and Boolean operators such as $\wedge$ (and), $\neg$ (negation), and *temporal operations* on ϕ. Some of the common temporal operators are defined as:

$\mathbf{G}\ \phi$	ϕ is true all future moments.
$\mathbf{F}\ \phi$	ϕ is true some future moments.
$\mathbf{X}\ \phi$	ϕ is true the next moment.
$\phi_1\ \mathbf{U}\ \phi_2$	ϕ_1 is true until ϕ_2 becomes true.

Using LTL, we can define interesting *liveness* and *safety* properties. For example, $\mathbf{GF}\ \phi$ defines a surveillance property specifying that ϕ needs to hold true infinitely often. On the other hand, $\mathbf{FG}\ \phi$ represents a stability specification by requiring ϕ to stay true after a particular point in the future. Similarly $\mathbf{G}(\phi \rightarrow \mathbf{F}\ \psi)$ represents a response operator meaning that at all times if ϕ becomes true then at some point in the future ψ must turn true as well.

The goal of reactive synthesis from LTL is to automatically construct a controller that is guaranteed to satisfy the given LTL specifications. The solution to this problem is computed by first constructing an automaton from the given specification, which then translates to a two-player game between the system components and the environment components. A deterministic Rabin automaton is a formalism that enables representing LTL specifications in the form of an automaton that can then be translated to the two-player game.

Definition 1. *A* deterministic Rabin automaton *is a tuple $\mathcal{R} = \langle Q, \Sigma, \delta, q^0, F\rangle$ where Q is the set of states; Σ is the input alphabet; $\delta : Q \times \Sigma \rightarrow Q$ is the transition function; q^0 is the initial state and F represents the acceptance condition: $F = \{(G_1, B_1), \ldots, (G_{n_F}, B_{n_F})\}$ where $G_i, B_i \subset Q$ for $i = 1, \ldots, n_F$.*

A *run* of a Rabin automaton is an infinite sequence $r = q^0, q^1 \ldots$ where $q^0 \in Q^0$ and for all $i > 0$, $q^{i+1} \in \delta(q^i, \sigma)$, for some input $\sigma \in \Sigma$. For every run r of the Rabin automaton, $\inf(r) \in Q$ is the set of states that are visited infinitely often in the sequence $r = q^0, q^1 \ldots$. A run $r = q^0, q^1 \ldots$ is *accepting* if there exists $i \in \{1, \ldots, n_F\}$ such that:

$$\inf(r) \cap G_i \neq \emptyset \quad \text{and} \quad \inf(r) \cap B_i = \emptyset \tag{2.13}$$

For any LTL formula ϕ over Π, a deterministic Rabin automaton (DRA) can be constructed with input alphabet $\Sigma = 2^{\Pi}$ that accepts all and only words over Π that satisfy ϕ [161]. We let $\mathcal{R}_\phi$ denote this DRA.

Intuitively, a run of the Rabin automaton is accepted if and only if it visits the accepting (good) states G_i infinitely often, and visits the non-accepting (bad) states B_i only finitely often. Acceptance of a run in DRA $\mathcal{R}_\phi$ is equivalent to satisfaction of the LTL formula ϕ by that particular run.

The solution to the *reactive synthesis* problem is then a winning strategy for the system that is extracted from this two-player game created by the DRA. Such a strategy would be winning for the system under *any* sequence of possible environment inputs.

There also exists situations, where a winning strategy does not exist for the environment, i.e., there exist an adversarial environment input sequence that can lead to the violation of the given LTL property. In such scenarios, we instead extract a winning strategy for the environment. Such a strategy is called a *counterstrategy* and provides a policy for the environment that summarizes all the possible adversarial moves the environment component can possibly take to drive the system to the violation of the specification.

Signal Temporal Logic

LTL provides a rich and expressive specification language that enables formally specifying high level properties of a reactive system. However, one major downside of using LTL is the need to discretize the state space, which is quite unrealistic for many robotics and control applications that are inherently continuous. Signal Temporal Logic (STL) is another specification language that can address some of these limitations. We consider STL formulas defined recursively according to the grammar

$$\varphi ::= \pi^\mu \mid \neg\pi^\mu \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \mathbf{G}_{[a,b]}\ \psi \mid \varphi\ \mathbf{U}_{[a,b]}\ \psi$$

where π^μ is an atomic predicate $\mathbb{R}^n \to \mathbb{B}$ whose truth value is determined by the sign of a function $\mu : \mathbb{R}^n \to \mathbb{R}$ and ψ is an STL formula.

An interesting property of STL is its ability to express specifications for continuous-time, real-valued signals. However, in the rest of this dissertation, we focus only on discrete-time, real-valued signals, which is already sufficient to avoid space discretization. STL also has the advantage of naturally admitting a *quantitative* semantics which, in addition to the binary answer to the question of satisfaction, provides a real number indicating the quality of the satisfaction or violation. Such quantitative semantics have been defined for timed logics e.g. Metric Temporal Logic (MTL) [52] and STL [45] to assess the *robustness* of the systems to parameter or timing variations.

The validity of a formula φ with respect to the discrete-time signal ξ at time t, noted

$(\xi, t) \models \varphi$ is defined inductively as follows:

$$\begin{array}{lcl}
(\xi,t) \models \pi^\mu & \Leftrightarrow & \mu(\xi^t) > 0 \\
(\xi,t) \models \neg\pi^\mu & \Leftrightarrow & \neg((\xi,t) \models \pi^\mu) \\
(\xi,t) \models \varphi \wedge \psi & \Leftrightarrow & (\xi,t) \models \varphi \wedge (\xi,t) \models \psi \\
(\xi,t) \models \varphi \vee \psi & \Leftrightarrow & (\xi,t) \models \varphi \vee (\xi,t) \models \psi \\
(\xi,t) \models \mathbf{G}_{[a,b]}\varphi & \Leftrightarrow & \forall t' \in [t+a, t+b], (\xi,t') \models \varphi \\
(\xi,t) \models \mathbf{F}_{[a,b]}\varphi & \Leftrightarrow & \exists t' \in [t+a, t+b], (\xi,t') \models \varphi \\
(\xi,t) \models \varphi \ \mathbf{U}_{[a,b]} \ \psi & \Leftrightarrow & \exists t' \in [t+a, t+b] \text{ s.t. } (\xi,t') \models \psi \\
 & & \wedge \forall t'' \in [t,t'], (\xi,t'') \models \varphi.
\end{array}$$

Here, ξ^t is the value of sequence ξ at time t. For instance if ξ is a sequence of state, action pairs as in equation (2.8), $\xi^t = (x^t, u^t_{\mathcal{R}}, u^t_{\mathcal{H}}, w^t)$. A signal ξ satisfies φ, denoted by $\xi \models \varphi$, if $(\xi, t^0) \models \varphi$. Informally, $\xi \models \mathbf{G}_{[a,b]}\varphi$ if φ holds at every time step between a and b, and $\xi \models \varphi \ \mathbf{U}_{[a,b]} \ \psi$ if φ holds at every time step before ψ holds, and ψ holds at some time step between a and b. Additionally, we define $\mathbf{F}_{[a,b]}\varphi = \top \ \mathbf{U}_{[a,b]} \ \varphi$, so that $\xi \models \mathbf{F}_{[a,b]}\varphi$ if φ holds at some time step between a and b.

A STL formula φ is *bounded-time* if it contains no unbounded operators; the *bound* of φ is the maximum over the sums of all nested upper bounds on the temporal operators, and provides a conservative maximum trajectory length required to decide its satisfiability. For example, for $\mathbf{G}_{[0,10]}\mathbf{F}_{[1,6]} \ \varphi$, a trajectory of length $N \geq 10 + 6 = 16$ is sufficient to determine whether the formula is satisfiable. This bound can be computed in time linear in the length of the formula.

Robust Satisfaction of STL formulas

Quantitative or robust semantics define a real-valued function ρ^φ of signal ξ and t such that $(\xi, t) \models \varphi \equiv \rho^\varphi(\xi, t) > 0$. In this work, we utilize a quantitative semantic for *space*-robustness, which is defined as follows:

$$\begin{array}{lcl}
\rho^{\pi^\mu}(\xi,t) & = & \mu(\xi^t) \\
\rho^{\neg\pi^\mu}(\xi,t) & = & -\mu(\xi^t) \\
\rho^{\varphi\wedge\psi}(\xi,t) & = & \min(\rho^\varphi(\xi,t), \rho^\psi(\xi,t)) \\
\rho^{\varphi\vee\psi}(\xi,t) & = & \max(\rho^\varphi(\xi,t), \rho^\psi(\xi,t)) \\
\rho^{\mathbf{G}_{[a,b]}\varphi}(\xi,t) & = & \min_{t'\in[t+a,t+b]} \rho^\varphi(\xi,t') \\
\rho^{\mathbf{F}_{[a,b]}\varphi}(\xi,t) & = & \max_{t'\in[t+a,t+b]} \rho^\varphi(\xi,t') \\
\rho^{\varphi \ \mathbf{U}_{[a,b]} \ \psi}(\xi,t) & = & \max_{t'\in[t+a,t+b]}(\min(\rho^\psi(\xi,t'), \min_{t''\in[t,t']} \rho^\varphi(\xi,t'')))
\end{array}$$

To simplify notation, we denote ρ^{π^μ} by ρ^μ for the remainder of this work. The robustness of satisfaction for an arbitrary STL formula is computed recursively from the above semantics in a straightforward manner, by propagating the values of the functions associated with each operand using min and max operators corresponding to the various STL operators. For example, for a signal $\mathbf{x} = x^0, x^1, x^2 \ldots$, the robust satisfaction of π^{μ_1} where $\mu_1(x) = x - 3 > 0$ at time 0 is $\rho^{\mu_1}(\mathbf{x}, 0) = x^0 - 3$. The robust satisfaction of $\mu_1 \wedge \mu_2$ is the minimum of ρ^{μ_1} and ρ^{μ_2}. Temporal operators are treated as conjunctions and disjunctions along the time axis: since we deal with discrete time, the robustness of satisfaction of $\varphi = \mathbf{G}_{[0,2]}\mu_1$ is $\rho^{\varphi}(\mathbf{x}, 0) = \min_{t \in [0,2]} \rho^{\mu_1}(\mathbf{x}, t) = \min\{x^0 - 3, x^1 - 3, \ldots, x^K - 3\}$ where $0 \leq K \leq 2 < K + 1$.

Note that for continuous time, the min and max operations would be replaced by inf and sup, respectively.

The robustness score $\rho^{\varphi}(\mathbf{x}, t)$ should be interpreted as *how much* model $\mathbf{x}$ satisfies φ. Its absolute value can be viewed as the distance of $\mathbf{x}$ from the set of trajectories satisfying or violating φ, in the space of projections with respect to the functions μ that define the predicates of φ. In addition, the robustness score over a trace or trajectory is analogous to having a reward function over that trajectory (equation (2.5)); both are a measure of quantitive satisfaction of the desired properties.

Remark 1. *We have introduced and defined a Boolean and a quantitative semantics for STL over discrete-time signals, which can be seen as roughly equivalent to Bounded Linear Temporal Logic (BLTL). There are several advantages of using STL over BLTL. First, STL allows us to explicitly use real time in our specifications instead of integer indices, which we find more elegant. Second, our goal is to use the resulting controller for the control of the continuous system Σ so the specifications should be independent from the sampling time Δt. Finally, note that the relationship between the continuous-time and discrete-time semantics of STL depending on discretization error and sampling time is beyond the scope of this work. The interested reader can refer to [51] for further discussion on this topic.*

2.4 Simulations

In the following chapters, we mainly focus on examples in autonomous driving and flying quadrotors. Here, we describe the simulation framework for our experiments.

Driving Simulator

We use a simple point-mass model of the car's dynamics. We define the physical state of the system $\mathbf{x} = [x \;\; y \;\; \psi \;\; v]^\top$, where x, y are the coordinates of the vehicle, ψ is the heading, and v is the speed. We let $\mathbf{u} = [u_1 \;\; u_2]^\top$ represent the control input, where u_1 is the steering input and u_2 is the acceleration. We denote the friction coefficient by μ. We can write the dynamics model:

$$[\dot{x} \;\; \dot{y} \;\; \dot{\psi} \;\; \dot{v}] = [v \cdot \cos(\psi) \quad v \cdot \sin(\psi) \quad v \cdot u_1 \quad u_2 - \mu \cdot v] \tag{2.14}$$

All the vehicle in our driving simulator follow the same dynamics model. The simulator provides a top-down view of the environment, and is connected to a steering

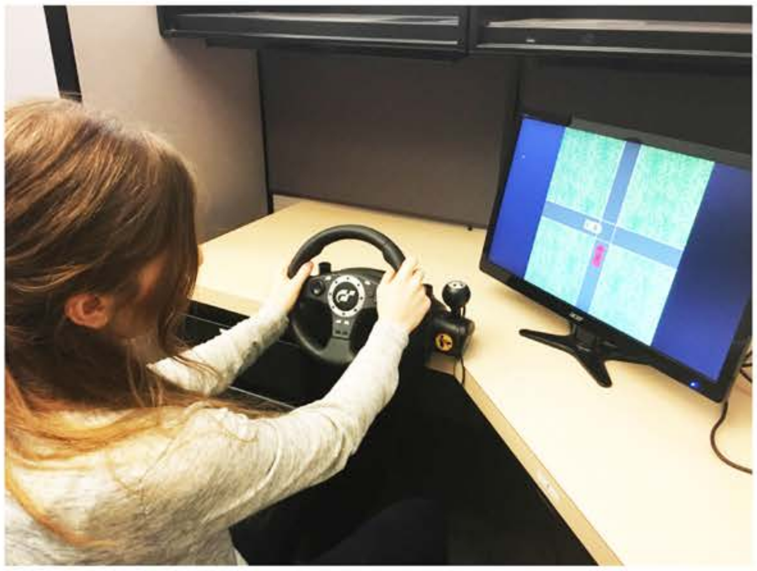


Figure 2.3: Driving Simulator. Top-down view driving simulator with steering wheel and braking pedal.

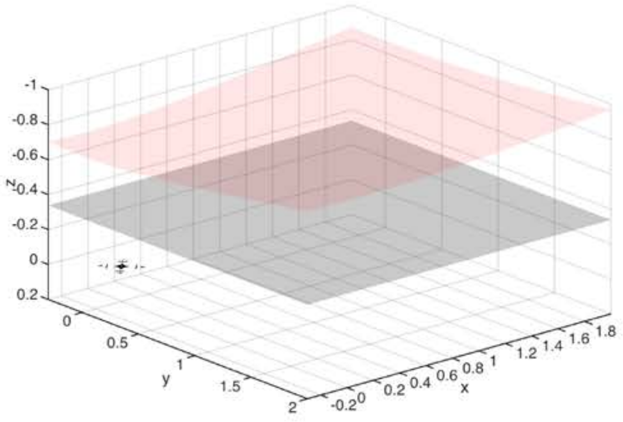


Figure 2.4: Flying Simulator. A quadrotor starting a trajectory from the origin. The gray surface represents the ceiling, and the pink surface is the quadrotors belief of the ceiling's location is based on the current sensor data.

wheel and a braking pedal as shown in Figure 2.3. The implementation of the driving simulator is available at: `https://github.com/dsadigh`.

Flying Simulator

We follow the derivation of the dynamics model of a quadrotor in [79]. We consider a 12 dimensional system, where the state consists of the position and velocity of the quadrotor x, y, z and $\dot{x}, \dot{y}, \dot{z}$, as well as the Euler angles ϕ, θ, ψ, i. e., roll, pitch, yaw, and the angular velocities p, q, r. Let $\mathbf{x}$ be:

$$\mathbf{x} = [x \ y \ z \ \dot{x} \ \dot{y} \ \dot{z} \ \phi \ \theta \ \psi \ p \ q \ r]^\top. \tag{2.15}$$

The system has a 4 dimensional control input $\mathbf{u} = \begin{bmatrix} u_1 & u_2 & u_3 & u_4 \end{bmatrix}^\top$, where u_1, u_2 and u_3 are the control inputs about each axis for roll, pitch and yaw respectively. u_4 represents the thrust input to the quadrotor in the vertical direction (z-axis). The nonlinear dynamics of the system is:

$$\begin{aligned}
f_1(x,y,z) &= \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^\top \\
f_2(\dot{x},\dot{y},\dot{z}) &= \begin{bmatrix} 0 & 0 & g \end{bmatrix}^\top - R_1(\dot{x},\dot{y},\dot{z}) \begin{bmatrix} 0 & 0 & 0 & u_4 \end{bmatrix}^\top / m \\
f_3(\phi,\theta,\psi) &= R_2(\dot{x},\dot{y},\dot{z}) \begin{bmatrix} \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^\top \\
f_4(p,q,r) &= I^{-1} \begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix}^\top - R_3(p,q,r) I \begin{bmatrix} p & q & r \end{bmatrix}^\top,
\end{aligned}$$

where R_1 and R_2 are rotation matrices, R_3 is a skew-symmetric matrix, and I is the inertial matrix of the rigid body. Here, g and m denote gravity and mass of the quadrotor, and for all our studies the mass and inertia matrix used are based on small sized quadrotors. Thus, the dynamics equation is $f(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} f_1 & f_2 & f_3 & f_4 \end{bmatrix}^\top$. Figure 2.4 shows the simulation environment. The implementation is available at: `https://github.com/dsadigh`.

Part I

Interaction-Aware Control

Chapter 3

Leveraging Effects on Human Actions

Traditionally, autonomous cars make predictions about other drivers' future trajectories, and plan to stay out of their way. This tends to result in defensive and opaque behaviors. Our key insight is that an autonomous car's actions will actually affect what other cars will do in response, whether these other cars are aware of it or not. Our thesis is that we can leverage these responses to plan more efficient and communicative behaviors. We model the interaction between an autonomous car and a human driver as a dynamical system, in which the robot's actions have immediate consequences on the state of the car, but also on human actions. We model these consequences by approximating the human as an optimal planner, with a reward function that we acquire through Inverse Reinforcement Learning. When the robot plans with this reward function in this dynamical system, it comes up with actions that purposefully change human state: it merges in front of a human to get them to slow down or to reach its own goal faster; it blocks two lanes to get them to switch to a third lane; or it backs up slightly at an intersection to get them to proceed first. Such behaviors arise from the optimization, without relying on hand-coded signaling strategies and without ever explicitly modeling communication. Our user study results suggest that the robot is indeed capable of eliciting desired changes in human state by planning using this dynamical system.

3.1 Human-Robot Interaction as a Two-Player Game

Our goal is to design controllers that autonomously generate behavior for better interaction and coordination with the humans in the environment. We set up a formalism that goes beyond modeling robots as agents acting in the physical world among moving obstacles as we see in multi-agent systems. Interaction with people is quite different from multi-agent planning; humans are not just moving obstacles that need to be avoided. In reality, the robot's actions has direct control over what the human is trying to do and how she performs it. Further, the human does not know what the robot is trying to do or even how the robot is going to do it.

We can think of the formalism introduced in Chapter 2.1 as a two-player game setting to formulate the interaction between the robot and the human [46]. However, this formulation can lead to various issues such as computational complexity in planning and inaccurate human behavior models. We further propose approximations that help resolve the computational complexities. Specifically, we simplify the planning problem to planning in an *underactuated* system. The robot can directly control its own actions, but also has a model of how it can influence human's actions through its own actions.

Much of robotics research focuses on how to enable a robot to achieve physical tasks, often times in the face of perception and movement error – of partially observable worlds and nondeterministic dynamics [143, 82, 137]. Part of what makes human-robot interaction difficult is that even if we assume the physical world to be fully observable and deterministic, we are still left with a complex problem, which is modeling and understanding the interaction between the human and the robot. Our proposed two-player game model would include a human agent who is approximately rational, i.e., she can take actions to maximize her own expected utility. In addition, the robot is a rational agent optimizing its reward function. We also assume the agents do not necessarily know each others' reward functions, which leaves us with an *incomplete information two-player game* [14].

Partially Observable Two-Player Game. We model this incomplete information two-player game in a partially observable setting similar to Partially Observable Markov Decision Processes (POMDP): as discussed in Chapter 2, there are two "players", the robot $\mathcal{R}$ and the human $\mathcal{H}$; at every step t, they can apply control inputs $u^t_{\mathcal{R}} \in \mathcal{U}_{\mathcal{R}}$ and $u^t_{\mathcal{H}} \in \mathcal{U}_{\mathcal{H}}$; they each have a reward function, $r_{\mathcal{R}}$ and $r_{\mathcal{H}}$; and there is a state space $\mathcal{S}$ with states s consisting of both the physical state x, as well as *reward parameters* θ_R and θ_H.

Here, we include the reward parameters in the state: $\mathcal{R}$ does not observe $\theta_{\mathcal{H}}$, and $\mathcal{H}$ does not observe $\theta_{\mathcal{R}}$, but both agents can technically evaluate each reward at any state, action pair $(s^t, u^t_{\mathcal{R}}, u^t_{\mathcal{H}})$ just because s contains the needed reward parameter information: $s = (x, \theta_{\mathcal{R}}, \theta_{\mathcal{H}})$ – if an agent knew the state, it could evaluate the other agent's reward [160, 46].

Further, we assume full observability over the physical state x. Our system follows a deterministic dynamics which is reasonable for relatively short interactions.

Robots do not know exactly what humans want, humans do not know exactly what robots have been programmed to optimize for, and their rewards might have common terms but will not be identical. This happens when an autonomous car interacts with other drivers or with pedestrians, and it even happens in seemingly collaborative scenarios like rehabilitation, in which very long horizon rewards might be aligned but not short-term interaction ones.

Limitations of the Game Formulation. The incomplete-information two-player game formulation is a natural way to characterize interaction from the perspective of MDP-like models, but is limited in two fundamental ways: 1) its computational complexity is prohibitive even in discrete state and action spaces [27, 75] and no methods are known to handle continuous spaces, and 2) it is not a good model for how people actually work

– people do not solve games in everyday tasks when they are not playing chess [76]. Furthermore, solutions here are tuples of policies that are in a Nash equilibrium, and it is not clear what equilibrium to select.

3.2 Approximate Solution as an Underactuated System

To alleviate the limitations from above, we introduce an approximate close to real-time solution, with a model of human behavior that does not assume that people compute equilibria of the game.

Assumptions to Simplify the Game

Our approximation makes several simplifying assumptions that turn the game into an offline learning phase in which the robot learns the human's reward function, followed by an online planning phase in which the robot is solving an underactuated control problem:

Separation of Estimation & Control
We separate the process of computing actions for the robot into two stages. First, the robot estimates the human reward function parameters $\theta_{\mathcal{H}}$ offline. Second, the robot exploits this estimate as a fixed approximation to the human's true reward parameters during planning. In the offline phase, we estimate $\theta_{\mathcal{H}}$ from user data via Inverse Reinforcement Learning [129, 3, 188, 107]. This method relies heavily on the approximation of all humans to a constant set of reward parameters, but we will relax this separation of estimation and control in Chapter 4.

Model Predictive Control (MPC)
Solving the incomplete information two-player game requires planning to the end of the full-time horizon. We reduce the computation required by planning for a shorter horizon of N time steps. We execute the control only for the first time step, and then re-plan for the next N at the next time step [35]. We have described the details of MPC in Chapter 2.1.

Despite our reduction to a finite time horizon, the game formulation still demands computing equilibria to the problem. Our core assumption, which we discuss next, is that this is not required for most interactions: that a simpler model of what the human does suffices.

Simplification of the Human Model
To avoid computing these equilibria, we propose to model the human as responding rationally to some fixed extrapolation of the robot's actions. At every time step, t, $\mathcal{H}$ computes a simple estimate of $\mathcal{R}$'s plan for the remaining horizon, $\tilde{\mathbf{u}}_{\mathcal{R}}^{t+1:N}$, based on the robot's previous actions $\mathbf{u}_{\mathcal{R}}^{\mathbf{0:t}}$. Then the human computes its plan $\mathbf{u}_{\mathcal{H}}$ as a *best response* [62] to this estimate. With this simplification, we reduce the general game to a Stackelberg competition: the human computes its best outcome while holding the robots plan fixed.

Let $R_{\mathcal{H}}$ be the human reward over the time horizon:

$$R_{\mathcal{H}}(x^0, \mathbf{u}_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}) = \sum_{t=1}^{N} r_{\mathcal{H}}(x^t, u^t_{\mathcal{R}}, u^t_{\mathcal{H}}), \tag{3.1}$$

then we can compute the control inputs of the human from the remainder of the horizon by:

$$u^t_{\mathcal{H}}(x^0, \mathbf{u}^{0:t}_{\mathcal{R}}, \tilde{\mathbf{u}}^{t+1:N}_{\mathcal{R}}) = \arg\max_{\mathbf{u}^{t+1:T}_{\mathcal{H}}} R_{\mathcal{H}}(x^t, \tilde{\mathbf{u}}^{t+1:N}_{\mathcal{R}}, u^{t+1:N}_{\mathcal{H}}). \tag{3.2}$$

This human model would certainly not work well in adversarial scenarios, but our hypothesis, supported by our results, is that it is useful enough in day-to-day tasks to enable robots to be more effective and more fluent interaction partners.

In our work, we propose to make the human's estimate $\tilde{\mathbf{u}}_{\mathcal{R}}$ equal to the actual robot control sequence $\mathbf{u}_{\mathcal{R}}$. Our assumption that the time horizon is short enough that the human can effectively extrapolate the robot's course of action motivates this decision. With this presumption, the human's plan becomes a function of the initial state and robot's true plan:

$$\mathbf{u}^*_{\mathcal{H}}(x^0, \mathbf{u}_{\mathcal{R}}) = \arg\max_{\mathbf{u}_{\mathcal{H}}} R_{\mathcal{H}}(x^t, \mathbf{u}_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}). \tag{3.3}$$

This is now an underactuated system: the robot has direct control over (can actuate) $\mathbf{u}_{\mathcal{R}}$ and indirect control over (cannot actuate but does affect) $\mathbf{u}_{\mathcal{H}}$. However, the dynamics model in our setup is more sophisticated than in typical underactuated systems because it models the response of the humans to the robot's actions. Evaluating the dynamics model requires solving for the optimal human response, $\mathbf{u}^*_{\mathcal{H}}$.

The system is also a special case of an MDP, with the state as in the two-player game, the actions being the actions of the robot, and the world dynamics being dictated by the human's response and the resulting change on the world from both human and robot actions.

The robot can now plan in this system to determine which $\mathbf{u}_{\mathcal{R}}$ would lead to the best outcome for the itself:

$$\mathbf{u}^*_{\mathcal{R}} = \arg\max_{\mathbf{u}_{\mathcal{R}}} R_{\mathcal{R}}\left(x^0, \mathbf{u}_{\mathcal{R}}, \mathbf{u}^*_{\mathcal{H}}(x^0, \mathbf{u}_{\mathcal{R}})\right). \tag{3.4}$$

Planning with Quasi-Newton Optimization

Despite the reduction to a single agent complete information underactuated system, the dynamics remain too complex to solve in real-time. We lack an analytical form for $\mathbf{u}^*_{\mathcal{H}}(x^0, \mathbf{u}_{\mathcal{R}})$ which forces us to solve equation (3.3) each time we evaluate the dynamics.

Assuming a known human reward function $r_{\mathcal{H}}$ (which we can obtain through Inverse Reinforcement Learning (IRL), see Chapter 2.2), we can solve equation (3.4) locally, using gradient-based methods. Our main contribution is agnostic to the particular optimization method, but we use L-BFGS [11], a quasi-Newton method that stores an approximate inverse Hessian implicitly resulting in fast convergence.

To perform the local optimization, we need the gradient of equation (3.1) with respect to $\mathbf{u}_{\mathcal{R}}$. This gradient using chain rule would be the following:

$$\frac{\partial R_{\mathcal{R}}}{\partial \mathbf{u}_{\mathcal{R}}} = \frac{\partial R_{\mathcal{R}}}{\partial \mathbf{u}_{\mathcal{H}}} \frac{\partial \mathbf{u}^*_{\mathcal{H}}}{\partial \mathbf{u}_{\mathcal{R}}} + \frac{\partial R_{\mathcal{R}}}{\partial \mathbf{u}_{\mathcal{R}}} \tag{3.5}$$

We can compute both $\frac{\partial R_{\mathcal{R}}}{\partial \mathbf{u}_{\mathcal{H}}}$ and $\frac{\partial R_{\mathcal{R}}}{\partial \mathbf{u}_{\mathcal{R}}}$ symbolically through back-propagation because we have a representation of $R_{\mathcal{R}}$ in terms of $\mathbf{u}_{\mathcal{H}}$ and $\mathbf{u}_{\mathcal{R}}$.

What remains, $\frac{\partial \mathbf{u}^*_{\mathcal{H}}}{\partial \mathbf{u}_{\mathcal{R}}}$, is difficult to compute because $\mathbf{u}^*_{\mathcal{H}}$ is technically the outcome of a global optimization. To compute $\frac{\partial \mathbf{u}^*_{\mathcal{H}}}{\partial \mathbf{u}_{\mathcal{R}}}$, we use the method of implicit differentiation. Since $R_{\mathcal{H}}$ is a smooth function whose minimum can be attained, we conclude that for the unconstrained optimization in equation (3.3), the gradient of $R_{\mathcal{H}}$ with respect to $\mathbf{u}_{\mathcal{H}}$ evaluates to 0 at its optimum $\mathbf{u}^*_{\mathcal{H}}$:

$$\frac{\partial R_{\mathcal{H}}}{\partial \mathbf{u}_{\mathcal{H}}} \left(x^0, \mathbf{u}_{\mathcal{R}}, \mathbf{u}^*_{\mathcal{H}}(x^0, \mathbf{u}_{\mathcal{R}}) \right) = 0 \tag{3.6}$$

Now, we differentiate the expression in equation (3.6) with respect to $\mathbf{u}_{\mathcal{R}}$:

$$\frac{\partial^2 R_{\mathcal{H}}}{\partial \mathbf{u}^2_{\mathcal{H}}} \frac{\partial \mathbf{u}^*_{\mathcal{H}}}{\partial \mathbf{u}_{\mathcal{R}}} + \frac{\partial^2 R_{\mathcal{H}}}{\partial \mathbf{u}_{\mathcal{H}} \partial \mathbf{u}_{\mathcal{R}}} \frac{\partial \mathbf{u}_{\mathcal{R}}}{\partial \mathbf{u}_{\mathcal{R}}} = 0 \tag{3.7}$$

Finally, we solve for a symbolic expression of $\frac{\partial \mathbf{u}^*_{\mathcal{H}}}{\partial \mathbf{u}_{\mathcal{R}}}$:

$$\frac{\partial \mathbf{u}^*_{\mathcal{H}}}{\partial \mathbf{u}_{\mathcal{R}}} = \left[\frac{\partial^2 R_{\mathcal{H}}}{\partial \mathbf{u}^2_{\mathcal{H}}} \right]^{-1} \left[-\frac{\partial^2 R_{\mathcal{H}}}{\partial \mathbf{u}_{\mathcal{H}} \partial \mathbf{u}_{\mathcal{R}}} \right] \tag{3.8}$$

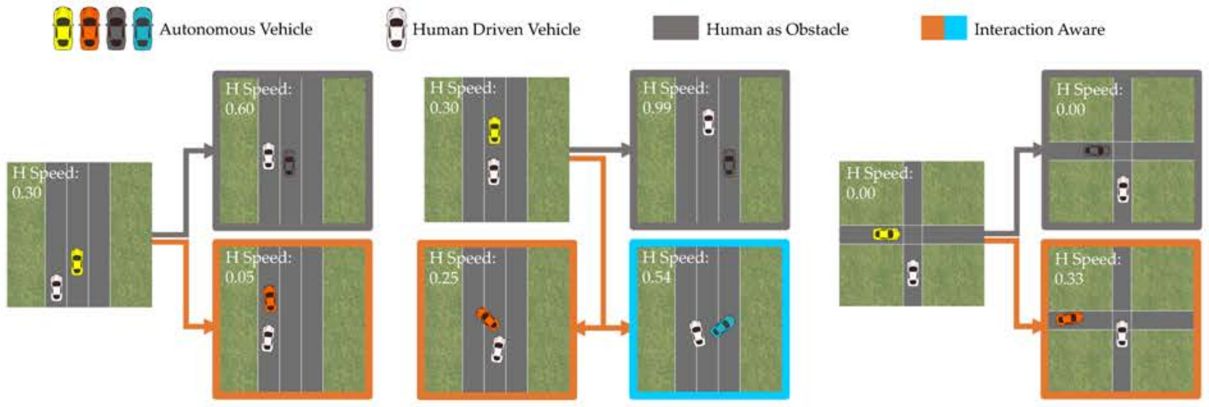
and insert it into equation (3.5), providing an expression for the gradient $\frac{\partial R_{\mathcal{R}}}{\partial \mathbf{u}_{\mathcal{R}}}$.

Offline Estimation of Human Reward Parameters

Thus far, we have assumed access to $r_{\mathcal{H}}(x^t, u^t_{\mathcal{R}}, u^t_{\mathcal{H}})$. In our implementation, we learn this reward function from human data. We collect demonstrations of a driver in a simulation environment, and use Inverse Reinforcement Learning [129, 3, 188, 107, 165, 98] to recover a reward function that explains the demonstrations.

To handle continuous state and actions space, and cope with noisy demonstrations that are perhaps only locally optimal, we use continuous inverse optimal control with locally optimal examples (Chapter 2.2).

Despite the simplicity of our features and robot actions during the demonstrations, the learned human model proved sufficient for the planner to produce human-interpretable behavior (case studies in Section 3.3), and actions which affected human action in the desired way (user study in Section 3.4).



(a) Scenario 1: Make human slow down (b) Scenario 2: Make human go left/right (c) Scenario 3: Make human cross first

Figure 3.1: Driving scenarios. In (a), the car plans to merge in front of the human in order to make them slow down. In (b), the car plans to direct the human to another lane, and uses its heading to choose which lane the human will go to. In (c), the car plans to back up slightly in order to make the human proceed first at the intersection. None of these plans use any hand coded strategies. They *emerge* out of optimizing with a learned model of how humans react to robot actions. In the training data for this model, the learned was *never exposed to situations* where another car stopped at an orientation as in (b), or backed up as in (c). However, by capturing human behavior in the form of a reward, the model is able to generalize to these situations, enabling the planner to find creative ways of achieving the desired effects.

Implementation Details

In our implementation, we used the software package Theano [26, 19] to symbolically compute all Jacobians and Hessians. Theano optimizes the computation graph into efficient C code, which is crucial for real-time applications.

This implementation enables us to solve each step of the optimization in approximately 0.3 seconds for horizon length $N = 5$ on a 2.3 GHz Intel Core i7 processor with 16 GB RAM. Future work will focus on achieving better computation time and a longer planning horizon.

3.3 Case Studies with Offline Estimation

We noted earlier that the state of the art autonomous driving plans conservatively because of its simple assumptions regarding the environment and vehicles on the road. In our experiments, we demonstrate that an autonomous vehicle can purposefully affect human drivers, and can use this ability to gather information about the human's driving style and goals.

In this section, we introduce 3 driving scenarios, and show the result of our planner assuming a simulated human driver, highlighting the behavior that emerges from different robot reward functions. In the next section, we test the planner with real users and measure the effects of the robot's plan. Figure 3.1 illustrates our three scenarios, and contains images from the actual user study data.

Conditions for Analysis Across Scenarios

In all three scenarios, we start from an initial position of the vehicles on the road, as shown in Figure 3.1. In the ***control*** condition, we give the car a reward function similar to the learned $R_{\mathcal{H}}$, i.e., a linear combination of the features discussed in Chapter 2.2. Therefore, this reward function is to avoid collisions and have high velocity. We refer to this as R_{control}. In the ***experimental*** condition, we augment this reward function with a term corresponding to a desired human action (e.g. low speed, lateral position, etc.). We refer to this as $R_{\text{control}} + R_{\text{affect}}$. Section 3.3 contrast the two plans for each of our three scenarios, and then show what happens when instead of explicitly giving the robot a reward function designed to trigger certain effects on the human, we simply task the robot with reaching a destination as quickly as possible.

Scenario 1: Make Human Slow Down

In this highway driving setting, we demonstrate that an autonomous vehicle can plan to cause a human driver to slow down. The vehicles start at the initial conditions depicted on left in Figure 3.1 (a), in separate lanes. In the experimental condition, we augment the robot's reward with the negative of the square of the human velocity, which encourages the robot to slow the human down.

Figure 3.1 (a) contrasts our two conditions. In the control condition, the human moves forward uninterrupted. *In the experimental condition, however, the robot plans to move in front of the person, anticipating that this will cause the human to brake.*

Scenario 2: Make Human Go Left/Right

In this scenario, we demonstrate that an autonomous vehicle can plan to affect the human's lateral location, making the human switch lanes. The vehicles start at the initial conditions depicted on left in Figure 3.1 (b), in the same lane, with the robot ahead of the human. In the experimental condition, we augment the robot's reward with the lateral position of the human, in two ways, to encourage the robot to make the human go either left (orange border image) or right (blue border image). The two reward additions are shown in Figure 3.2(a) and (b).

Figure 3.1 (b) contrasts our two conditions. In the control condition, the human moves forward, and might decide to change lanes. *In the experimental condition, however, the robot*

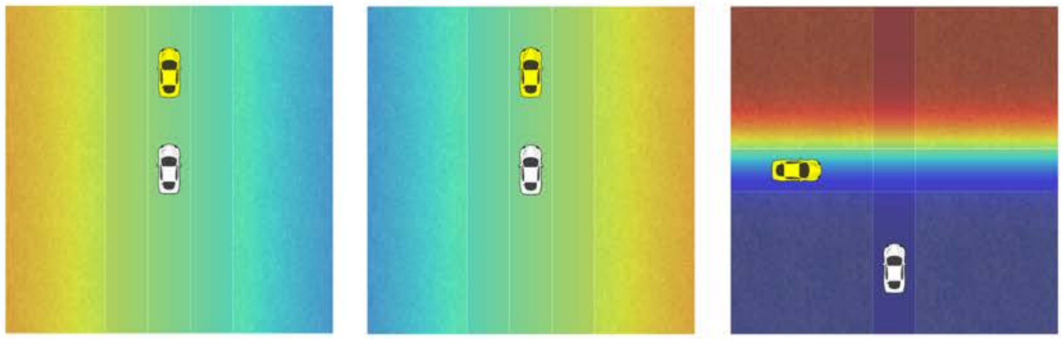

(a) Scenario 2 Reward: force human left (b) Scenario 2 Reward: force human right (c) Scenario 3 Reward: force human to cross first

Figure 3.2: Heat map of the reward functions in scenarios 2 and 3. The warmer colors show higher reward values. In (a), (b), the reward function of the autonomous vehicle is plotted, which is a function of the human driven vehicle's position. In order to affect the driver to go left, the reward is higher on the left side of the road in (a), and to affect the human to go right in (b), the rewards are higher on the right side of the road. In (c), the reward of the autonomous vehicle is plotted for scenario 3 with respect to the position of the human driven car. Higher rewards correspond to making the human cross the intersection.

plans to intentionally occupy two lanes (using either a positive or negative heading), anticipating this will make the human avoid collision by switching into the unoccupied lane.

Scenario 3: Make Human Go First

In this scenario, we demonstrate that an autonomous vehicle can plan to cause the human to proceed first at an intersection. The vehicles start at the initial conditions depicted on the left in Figure 3.1 (c), with both human and robot stopped at the 4-way intersection. In the experimental condition, we augment the robot's reward with a feature based on the y position of the human car $y_{\mathcal{H}}$ relative to the middle of the intersection y_0. In particular, we used the hyperbolic tangent of the difference, $\tanh(y_{\mathcal{H}} - y_0)$. The reward addition is shown in Figure 3.2 (c).

Figure 3.1 (c) contrasts our two conditions. In the control condition, the robot proceeds in front of the human. *In the experimental condition, however, the robot plans to intentionally reverse slightly, anticipating that this will induce the human cross first.* We might interpret such a trajectory as communicative behavior, but communication was never explicitly encouraged in the reward function. Instead, *the goal of affecting human actions led to this behavior.*

Reversing at an intersection is perhaps the most surprising result of the three scenarios, because it is not an action human drivers take. In spite of this novelty, our user study

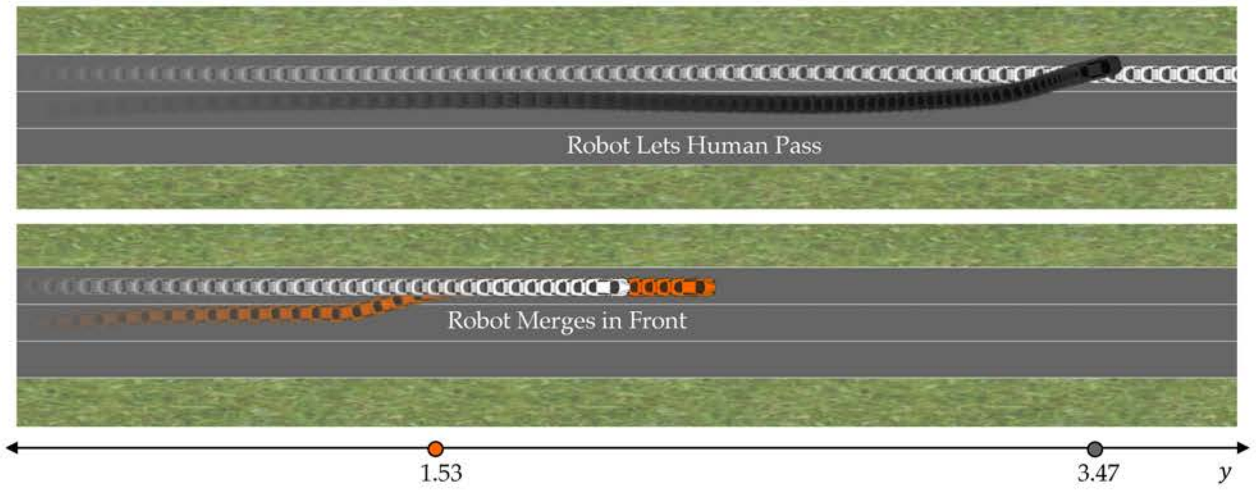


Figure 3.3: A time lapse where the autonomous vehicle's goal is to reach a final point in the left lane. In the top scenario, the autonomous vehicle has a simple model of the human driver that does not account for the influence of its actions on the human actions, so it acts more defensively, waiting for the human to pass first. In the bottom, the autonomous vehicle uses the learned model of the human driver, so it acts less defensively and reaches its goal faster.

suggests that human drivers respond in the expected way: they proceed through the intersection. Further, pedestrians sometimes exhibit behavior like the robot's, stepping back from an intersection in order to let a car pass first.

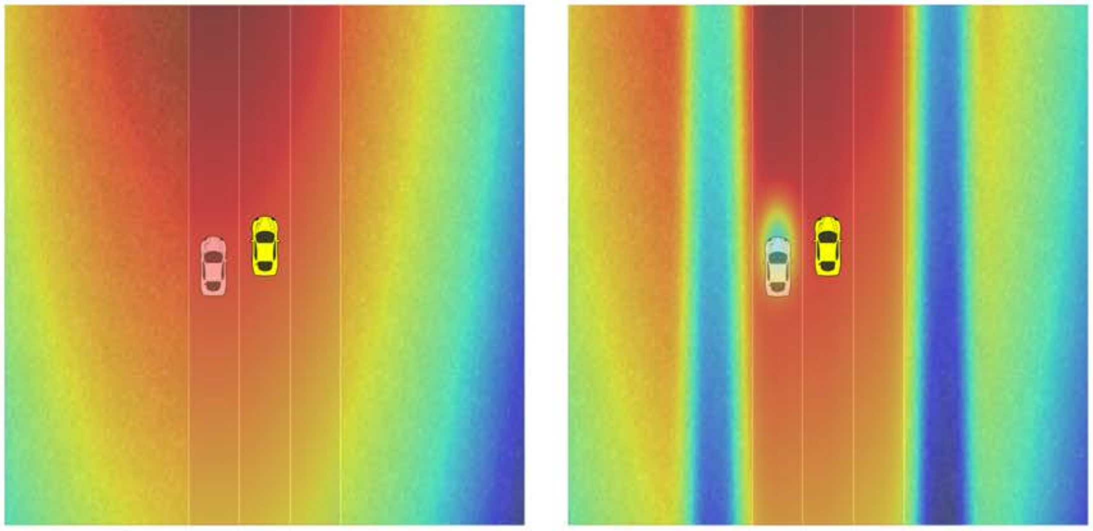
Behaviors Also Emerge from Efficiency

Thus far, we have explicitly encoded a desired effect on human actions, and optimized it as a component of the robot's reward. We have also found, however, that behaviors like those we have seen so far can emerge out of the need for efficiency.

Figure 3.3 (bottom) shows the generated plan for when the robot is given the goal to reach a point in the left lane as quickly as possible (reward shown in Figure 3.4). By modeling the effects its actions have on the human actions, the robot plans to merge in front of the person, expecting that they will slow down.

In contrast, the top of the figure shows the generated plan for when the robot uses a simple (constant velocity) model of the person. In this case, the robot assumes that merging in front of the person can lead to a collision, and defensively waits for the person to pass, merging behind them.

We hear about this behavior often in autonomous cars today: they are defensive. Enabling them to plan in a manner that is cognizant that they can affect other driver actions can make them more efficient at achieving their goals.



(a) Single goal feature (b) Superposition of all features

Figure 3.4: Heat map of reward function for reaching a final goal at the top left of the road. As shown in the figure, the goal position is darker showing more reward for reaching that point.

The Robot Behavior Adapts to the Situation

Throughout the case studies, we see examples of coordination behavior that emerges out of planning in our system: going in front of someone knowing they will brake, slowing and nudging into another lane to incentivize a lane change, or backing up to incentivize that the human proceeds first through an intersection. Such behaviors could possibly be hand-engineered for those particular situations rather than autonomously plan. However, we advocate that the need to plan comes from versatility: from the fact that the planner can adapt the exact strategy to each situation.

Figure 3.5 shows a spectrum of behaviors for the robot depending on where it starts relative to the human: from merging behind the person, to not merging, to merging in front.

3.4 User Study with Offline Estimation

The previous section showed the robot's plans when interacting with a simulated user that perfectly fits the robot's model of the human. Next, we present the results of a user study that evaluates whether the robot can successfully have the desired effects on real users.

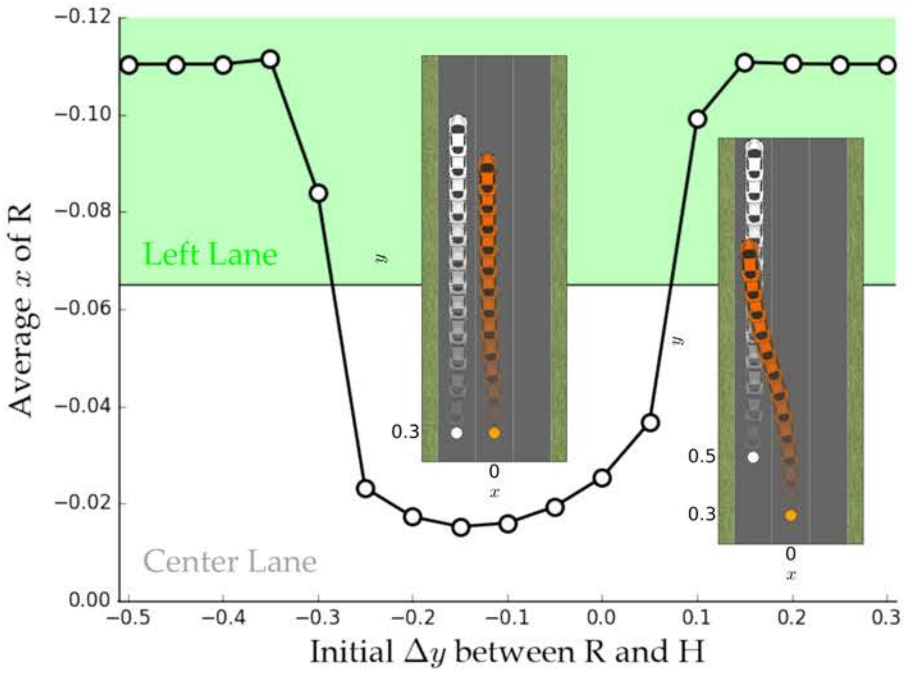


Figure 3.5: The robot adapts its merging behavior depending on the relative position of the person: it does not always cut the person off: sometimes it merges behind the person, and if it starts too close (depending on how the reward function is set up) it will not merge at all.

Experimental Design

We use the same 3 scenarios as in the previous section.

Manipulated Factors. We manipulate a single factor: the ***reward*** that the robot is optimizing, as described in Section 3.3. This leads to two conditions: the ***experimental*** condition where the robot is encouraged to have a particular effect on human state though the reward $R_{\text{control}} + R_{\text{affect}}$, and the ***control*** condition where that aspect is left out of the reward function and the robot is optimizing only R_{control} (three conditions for Scenario 2, where we have two experimental conditions, one for the left case and one for the right case).

Dependent Measures. For each scenario, we measure the value along the user trajectory of the feature added to the reward function for that scenario, R_{affect}. Specifically, we measure the human's negative squared velocity in Scenario 1, the human's x axis location relative to center in Scenario 2, and whether the human went first or not through the intersection in Scenario 3 (i.e. a filtering of the feature that normalizes for difference in

timing among users and measures the desired objective directly).
Hypothesis. We hypothesize that our method enables the robot to achieve the effects it desires not only in simulation, but also when interacting with real users:

> *The reward function that the robot is optimizing has a significant effect on the measured reward during interaction. Specifically, R_{affect} is higher, as planned, when the robot is optimizing for it.*

Subject Allocation. We recruited 10 participants (2 female, 8 male). All the participants owned drivers license with at least 2 years of driving experience. We ran our experiments using a 2D driving simulator, we have developed with the driver input provided through driving simulator steering wheel and pedals (see Chapter 2.4).

Analysis

Scenario 1: A repeated measures ANOVA showed the square speed to be significantly lower in the experimental condition than in the control condition ($F(1, 160) = 228.54$, $p < 0.0001$). This supports our hypothesis: the human moved slower when the robot planned to have this effect on the human.

We plot the speed and latitude profile of the human driven vehicle over time for all trajectories in Figure 3.6. Figure 3.6(a) shows the speed profile of the control condition trajectories in gray, and of the experimental condition trajectories in orange. Figure 3.6(b) shows the mean and standard error for each condition. In the control condition, human squared speed keeps increasing. In the experimental condition however, by merging in front of the human, the robot is triggering the human to brake and reduce speed, as planned. The purple trajectory represents a simulated user that perfectly matches the robot's model, showing the ideal case for the robot. The real interaction moves significantly in the desired direction, but does not perfectly match the ideal model, since real users do not act exactly as the model would predict.

The figure also plots the y position of the vehicles along time, showing that the human has not travelled as far forward in the experimental condition.
Scenario 2: A repeated measures ANOVA showed a significant effect for the reward factor ($F(2, 227) = 55.58$, $p < 0.0001$). A post-hoc analysis with Tukey HSD showed that both experimental conditions were significantly different from the control condition, with the user car going more to the left than in the control condition when R_{affect} rewards left user positions ($p < 0.0001$), and more to the right in the other case ($p < 0.001$). This supports our hypothesis.

We plot all the trajectories collected from the users in Figure 3.7. Figure 3.7(a) shows the control condition trajectories in gray, while the experimental conditions trajectories are shown in orange (for left) and blue (for right). By occupying two lanes, the robot triggers an avoid behavior from the users in the third lane. Here again, purple curves show a simulated user, i.e. the ideal case for the robot.

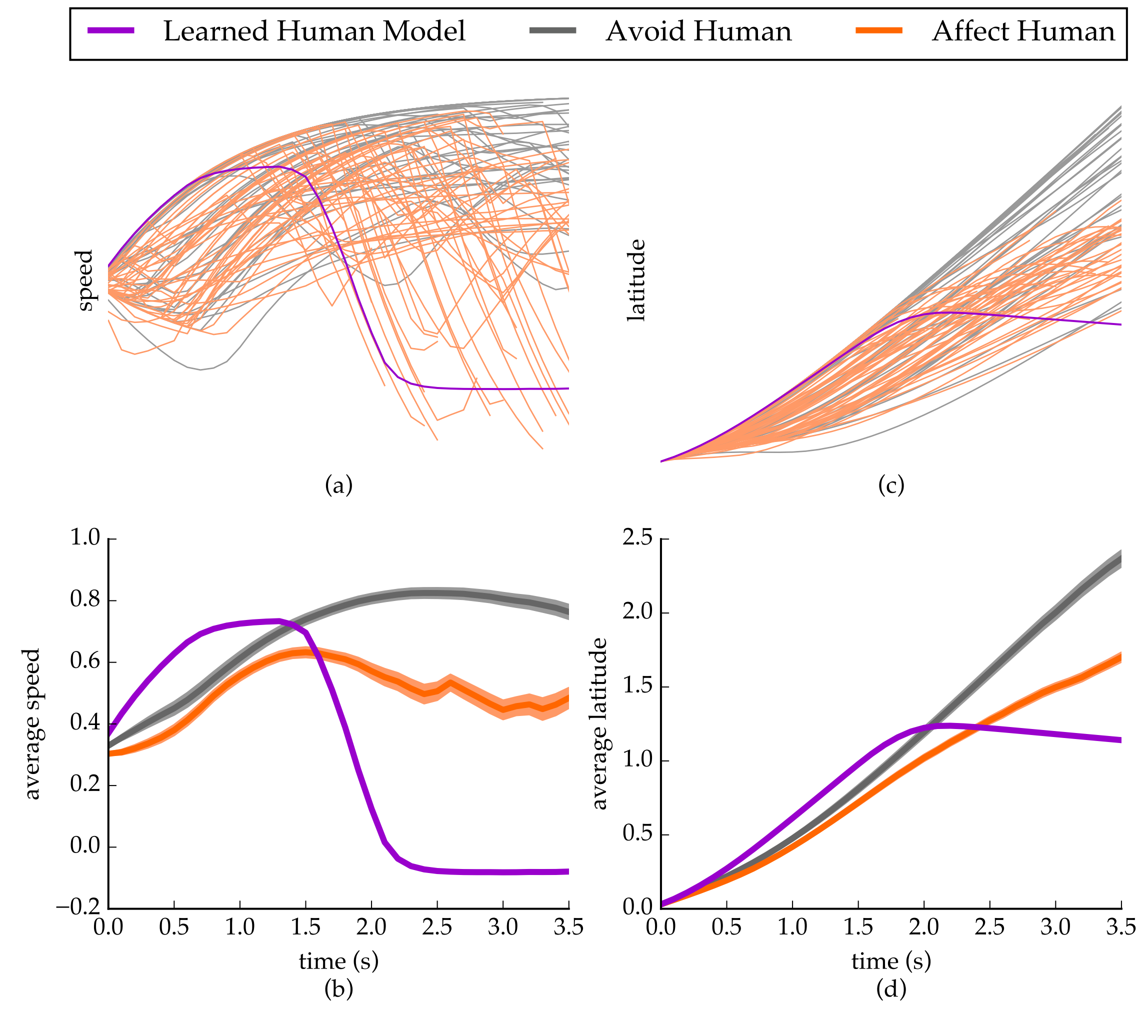


Figure 3.6: Speed profile and latitude of the human driven vehicle for Scenario 1. The first column shows the speed of all trajectories with its mean and standard errors in the bottom graph. The second column shows the latitude of the vehicle over time; similarly, with the mean and standard errors. The gray trajectories correspond to the control condition, and the orange trajectories correspond to the experimental condition: the robot decides to merge in front of the users and succeeds at slowing them down. The purple plot corresponds to a simulated user that perfectly matches the model that the robot is using.

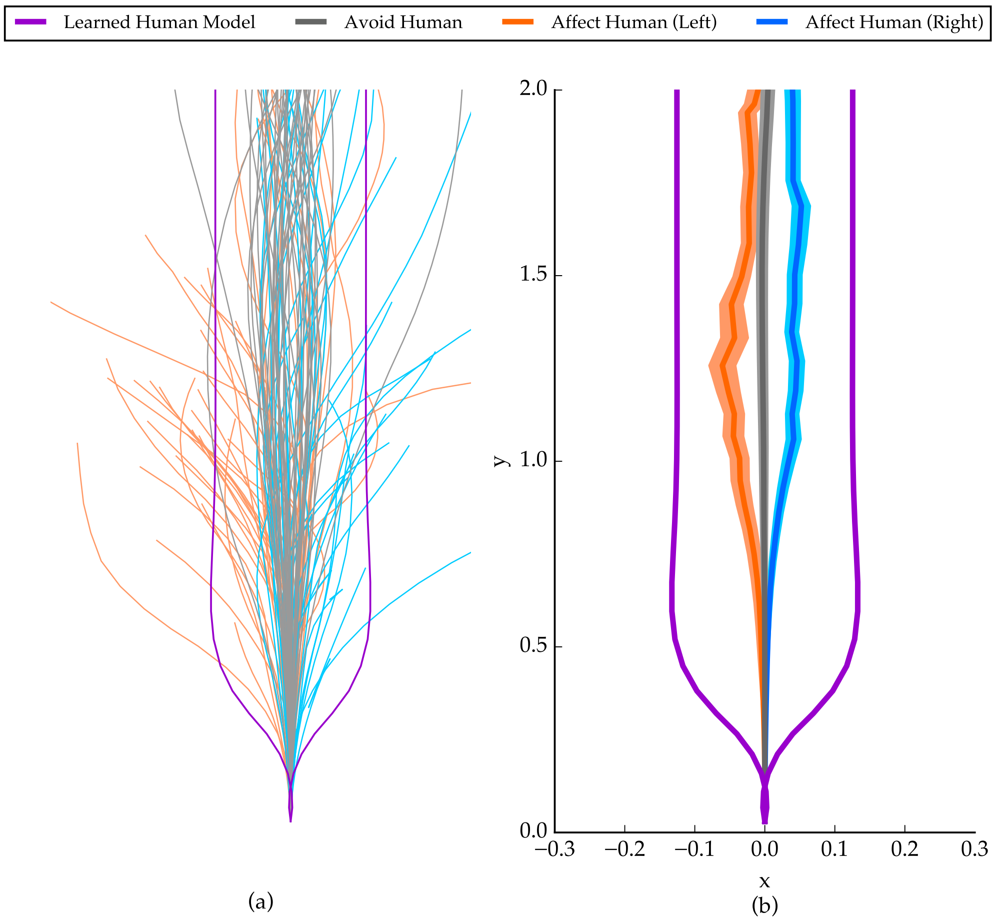


Figure 3.7: Trajectories of the human driven vehicle for Scenario 2. The first column (a) shows all the trajectories, and the second column (b) shows the mean and standard error. Orange (blue) indicates conditions where the reward encouraged the robot to affect the user to go left (right).

Scenario 3: An ordinal logistic regression with user as a random factor showed that significantly more users went first in the intersection in the experimental condition than in the baseline ($\chi^2(1, 129) = 106.41$, $p < .0001$). This supports our hypothesis.

Figure 3.8 plots the y position of the human driven vehicle with respect to the x position of the autonomous vehicle. For trajectories that have a higher y position for the human vehicle than the x position for the robot, the human car has crossed the intersection before the autonomous vehicle. The lines corresponding to these trajectories travel above the origin, which is shown with a blue square in this figure. The mean of the orange lines travel above the origin, which means that the autonomous vehicle has successfully affected the humans to cross first. The gray lines travel below the origin, i.e. the human crossed second.

Overall, our results suggest that the robot was able to affect the human state in the

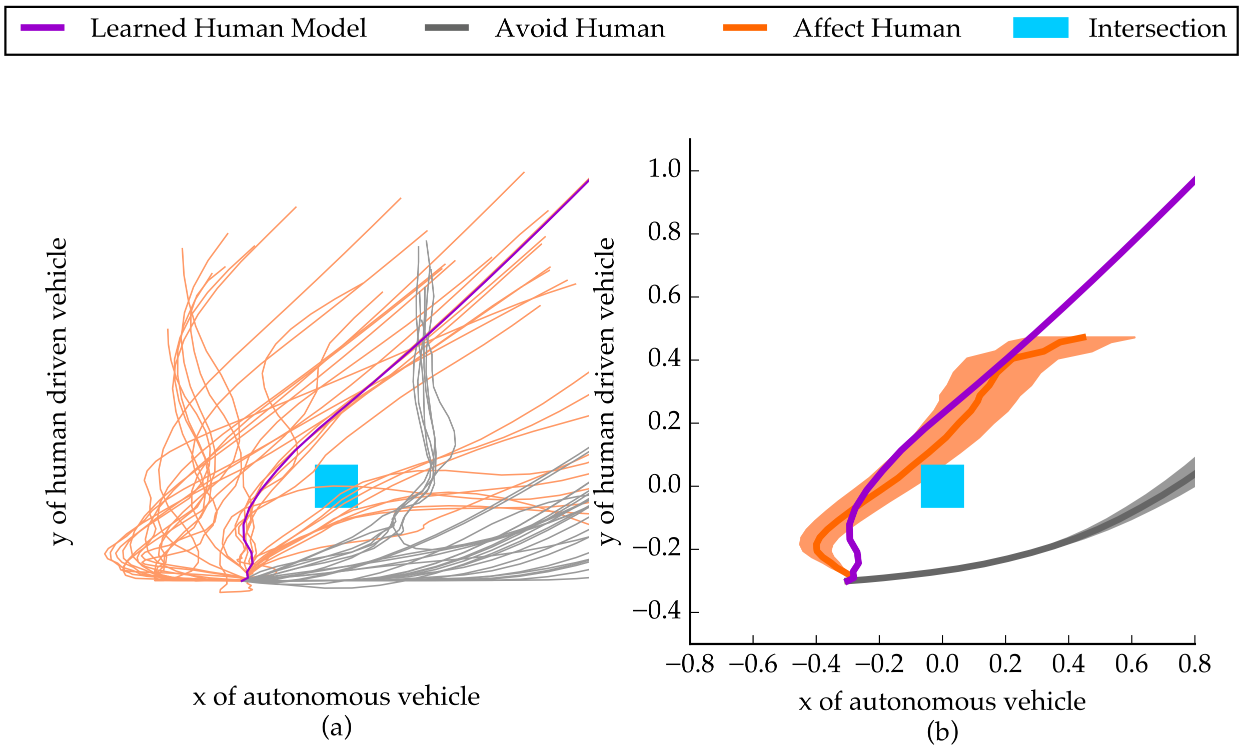


Figure 3.8: Plot of $y_{\mathcal{H}}$ with respect to $x_{\mathcal{R}}$. The orange curves correspond to when the autonomous vehicle affects the human to cross the intersection first. The gray curves correspond to the nominal setting.

desired way, even though it does not have a perfect model of the human.

3.5 Chapter Summary

In this chapter, we formalized the interaction between an autonomous (robot) vehicle and a human driver as a dynamical system, in which the actions of the robot affect those of the human and vice-versa. We introduced an approximate solution that enables the robot to optimize its own reward within this system. The resulting plans can purposefully modify human behavior, and can achieve the robot's goal more efficiently. Our user study suggests that this is not only true in simulation, but also true when tested with real users.

Like any research, our work is limited in many ways. All this work happened in a simple driving simulator. To put this on the road, we will need more emphasis on safety, as well as a longer planning horizon. The former involves the use of formal methods and safe control as well as better models of users: not all drivers act the same. Using a probabilistic dynamics model as opposed to planning with the most probable human actions, as well as estimating driving style, will be important next steps that we discuss in the future chapters.

An even bigger limitation is that we currently focus on a single human driver. Looking to the interaction among multiple vehicles is not just a computational challenge, but also a modeling one – it is not immediately clear how to formulate the problem when multiple human-driven vehicles are interacting and reacting to each other. We address some of these limitations such as use of formal methods for safe control and estimating driving style in the future chapters.

Despite these limitations, we are encouraged to see autonomous cars generate human-interpretable behaviors though optimization, without relying on hand-coded heuristics. We also look forward to applications of these ideas beyond autonomous driving, to mobile robots, UAVs, and in general to human-robot interactive scenarios where robot actions can influence human actions.

Chapter 4

Active Information Gathering over Human Internal State

Imagine driving on the highway. Another driver is in the lane next to you, and you need to switch lanes. Some drivers are aggressive and they will never brake to let you in. Others are more defensive and would gladly make space for you. You don't know what kind of driver this is, so you decide to gently nudge in towards the other lane to *test* their reaction. At an intersection, you might *nudge in* to test if the other driver is distracted and they might just let you go through (Figure 6.1 bottom left). Our goal in this chapter is to extend the optimization based approach of Chapter 3 to give robots the capability to differentiate between human agents and plan accordingly.

In general, human behavior is affected by internal states that a robot would not have direct access to: intentions, goals, preferences, objectives, driving style, etc. Work in robotics and perception has focused thus far on estimating these internal states by providing algorithms with observations of humans acting, be it intent prediction [189, 111, 47, 16, 17, 131], driver style prediction [102], affective state prediction [99], or activity recognition [172].

Human state estimation has also been studied in the context of human-robot interaction tasks. Here, the robot's reward function depends (directly or indirectly) on the human internal state, e.g., on whether the robot is able to adapt to the human's plan or preferences. Work in assistive teleoperation or in human assistance has cast this problem as a Partially Observable Markov Decision Process, in which the robot does observe the physical state of the world but not the human internal state — that it has to estimate from human actions. Because POMDP solvers are not computationally efficient, the solutions proposed thus far use the current estimate of the internal state to plan (either using the most likely estimate, or the entire current belief), and adjust this estimate at every step [81, 58, 18]. Although efficient, these approximations sacrifice an important aspect of POMDPs: the ability to *actively gather information*.

Our key insight is that robots can leverage their own actions to estimate the human

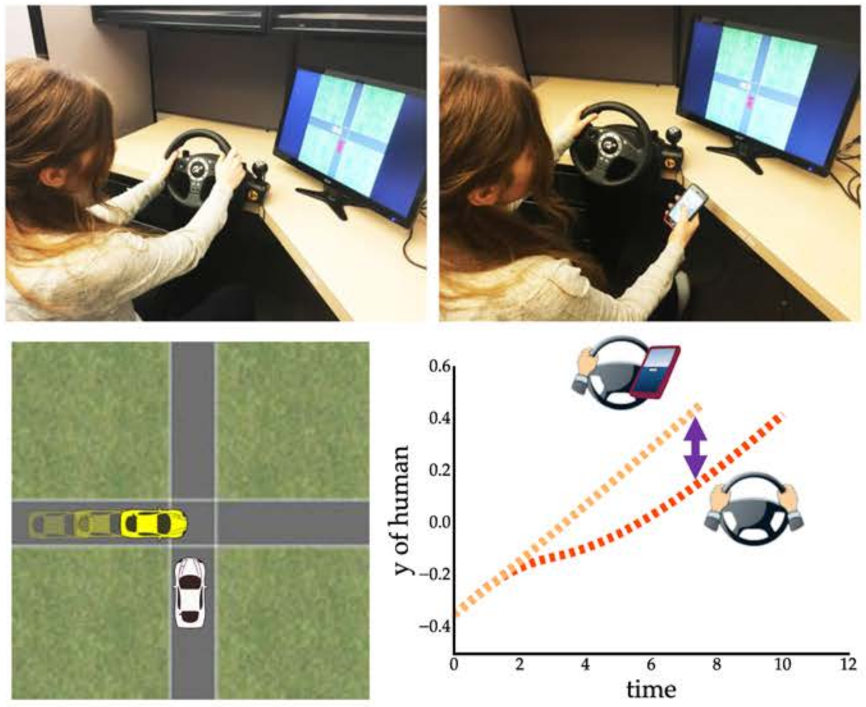


Figure 4.1: We enable robots to generate actions that actively probe humans in order to find out their internal state. We apply this to autonomous driving. In this example, the robot car (yellow) decides to inch forward in order to test whether the human driver (white) is attentive. The robot expects drastically different reactions to this action (bottom right shows attentive driver reaction in light orange, and distracted driver reaction in dark orange). We conduct a user study in which we let drivers pay attention or distract them with cellphones in order to put this state estimation algorithm to the test.

internal state.

Rather than relying on passive observations, robots can actually account for the fact that humans will react to their actions: they can use this knowledge to select actions that will trigger human reactions which in turn will clarify the internal state.

In this chapter, we make two contributions:

An Algorithm for Active Information Gathering over Human Internal State. We introduce an algorithm for planning robot actions that have high expected information gain. Our algorithm uses a reward-maximization model of how humans plan their actions in response to those of the robot's [159], and leverages the fact that different human internal states will lead to different human reactions to speed up estimation. Figure 6.1 shows an example of the anticipated difference in reaction between a distracted and an attentive driver.

Application to Driver Style Estimation. We apply our algorithm to estimating a human driver's style during the interaction of an autonomous vehicle with a human-driven vehicle. Results in simulation as well as from a user study suggest that our algorithm's ability to leverage robot actions for estimation leads to significantly higher accuracy in identifying the correct human internal state. The autonomous car plans actions like inching forward at an intersection (Figure 6.1), nudging into another car's lane, or braking slightly in front of a human-driven car, all to estimate whether the human driver is attentive.

4.1 Extension to Online Estimation of the Human Model

In Chapter 3, we described the incomplete information two-player game model of the interaction between the human and robot, and in our approximate solution, we treated the human's reward function as estimated once, offline. This has worked well in our user study on seeking specific coordination effects on the human, like slowing down or going first through the intersection. But in general, this is bound to run into problems, because not all people behave according to the same estimated $\theta_{\mathcal{H}}$.

Different drivers have different *driving styles*. Some are very defensive, more so than our learned model. Others are much more aggressive, and for instance would not actually brake when the car merges in front of them. Even for the same driver, their style might change over time, for instance when they get distracted on their phone.

In this chapter, we relax our assumption of an offline estimation of the human's reward parameters $\theta_{\mathcal{H}}$. Instead, we explore estimating this online. We introduce an algorithm which maintains a belief over a space of candidate reward functions, and enable the robot to perform inference over this space throughout the interaction. We maintain tractability by clustering possible $\theta_{\mathcal{H}}$s into a few options that the robot maintains a belief over.

A POMDP with Human Reward as the Hidden Variable

The human's actions are influenced by their internal reward parameters $\theta_{\mathcal{H}}$ that the robot does not directly observe. So far, we estimated $\theta_{\mathcal{H}}$ offline and solved an underactuated system, a special case of an MDP. Now, we want to be able to adapt our estimate of $\theta_{\mathcal{H}}$ online, during interaction. This turns the problem into a partially observable Markov decision process (POMDP) with $\theta_{\mathcal{H}}$ as the hidden state. By putting $\theta_{\mathcal{H}}$ in the state, we now have a known dynamics model like in the underactuated system before for the robot and the human state, and we assume $\theta_{\mathcal{H}}$ to remain fixed regardless of the robot's actions.

If we could solve the POMDP, the robot would estimate $\theta_{\mathcal{H}}$ from the human's actions, optimally trading off between exploiting it's current belief over $\theta_{\mathcal{H}}$ and actively taking information gathering actions intended to cause human reactions, which result in a better estimate of $\theta_{\mathcal{H}}$.

Because POMDPs cannot be solved tractably, several approximations have been proposed for similar problem formulations [81, 102, 58]. These approximations are *passively* estimating the human internal state, and exploiting the belief to plan robot actions.[1]

In this work, we take the opposite approach: we focus explicitly on active information gathering Our formulation enables the robot to choose to actively probe the human, and thereby improve its estimate of $\theta_{\mathcal{H}}$. We leverage this method in conjunction with exploitation methods, but the algorithm we present may also be used alone if human internal state (reward parameters) estimation is the robot's primary objective.

Simplification to Information Gathering

We denote a belief in the value of the hidden variable, θ, as a distribution $b(\theta)$, and update this distribution according to the likelihood of observing a particular human action, given the state of the world and the human internal state:

$$b^{t+1}(\theta) \propto b^t(\theta) \cdot P(u^t_{\mathcal{H}} \mid x^t, u_{\mathcal{R}}, \theta). \tag{4.1}$$

In order to update the belief b, we require an observation model. Similar to before, we assume that actions with lower reward are exponentially less likely, building on the principle of maximum entropy [188]:

$$P(u_{\mathcal{H}} \mid x, u_{\mathcal{R}}, \theta) \propto \exp\left(R^{\theta}_{\mathcal{H}}(x^0, \mathbf{u}_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}})\right). \tag{4.2}$$

To make explicit our emphasis on taking actions which effectively estimate θ, we redefine the robot's reward function to include an information gain term, i.e., the difference between entropies of the current and updated beliefs: $H(b^t) - H(b^{t+1})$. The entropy over the belief $H(b)$ evaluates to:

$$H(b) = -\frac{\sum_{\theta} b(\theta)\log(b(\theta))}{\sum_{\theta} b(\theta)}. \tag{4.3}$$

We now optimize our expected reward with respect to the hidden state θ, and this optimization explicitly entails reasoning about the effects that the robot actions will have on the observations, i.e., the actions that the human will take in response, and how useful these observations will be in shattering ambiguity about θ.

Explore-Exploit Trade-Off

In practice, we use information gathering in conjunction with exploitation. We do not solely optimize the information gain term $H(b^t) - H(b^{t+1})$, but optimize it in conjunction with the robot's actual reward function *assuming the current estimate of* θ:

[1]One exception is Nikolaidis et al. [132], who propose to solve the full POMDP, albeit for discrete and not continuous state and action spaces.

$$r_{\mathcal{R}}^{augmented}(x^t, \mathbf{u}_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}) = \lambda(H(b^t) - H(b^{t+1})) \\ + r_{\mathcal{R}}(x^t, \mathbf{u}_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}, b^t) \tag{4.4}$$

At the very least, we do this as a measure of safety, e.g., we want an autonomous car to keep avoiding collisions even when it is actively probing a human driver to test their reactions. We choose λ experimentally, though existing techniques that can better adapt λ over time [173].

Solution via Model Predictive Control

To find the control inputs for the robot we locally solve:

$$\mathbf{u}_{\mathcal{R}}^{*} = \arg\max_{\mathbf{u}_{\mathcal{R}}} \mathbb{E}_{\theta}\left[R_{\mathcal{R}}\left(x^0, \mathbf{u}_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}^{*,\theta}(x^0, \mathbf{u}_{\mathcal{R}})\right)\right] \tag{4.5}$$

over a finite horizon N, where $\mathbf{u}_{\mathcal{H}}^{*,\theta}(x^0, \mathbf{u}_{\mathcal{R}})$ corresponds to the actions the human *would* take from state x^0 if the robot executed actions $\mathbf{u}_{\mathcal{R}}$. This objective generalizes equation (3.4) with an expectation over the current belief over θ, b^0.

We still assume that the human maximizes their own reward function, $r_{\mathcal{H}}^{\theta}(x^t, u_{\mathcal{R}}^t, u_{\mathcal{H}}^t)$; we add the superscript θ to indicate the dependence on the hidden state. We can write the sum of human rewards over horizon N as:

$$R_{\mathcal{H}}^{\theta}(x^0, \mathbf{u}_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}) = \sum_{t=0}^{N-1} r_{\mathcal{H}}^{\theta}(x^t, u_{\mathcal{R}}^t, u_{\mathcal{H}}^t) \tag{4.6}$$

Computing this over the continuous space of possible reward parameters θ is intractable even with discretization. Instead, we learn clusters of θs offline via IRL, and online use estimation to figure out which cluster best matches the human.

Despite optimizing the trade-off in equation (4.4), we do not claim that our method as-is can better solve the general POMDP formulation: only that it can be used to get better estimates of human internal state. Different tradeoffs λ will result in different performance. Our results below emphasize the utility of gathering information, but also touch on the implications for active information gathering on $R_{\mathcal{R}}$.

4.2 Case Studies with Online Estimation

In this section, we show simulation results that use the method from this chapter to estimate human driver type in the interaction between an autonomous vehicle and a human-driven vehicle. We consider three different autonomous driving scenarios. In these scenarios, the human is either distracted or attentive during different driving experiments.

The scenarios are shown in Figure 6.3, where the yellow car is the autonomous vehicle, and the white car is the human driven vehicle. Our goal is to plan to actively estimate the human's driving style in each one of these scenarios, by using the robot's actions.

Attentive vs. Distracted Human Driver Models

Our technique requires reward functions $r_{\mathcal{H}}^{\theta}$ that model the human behavior for a particular internal state θ. We obtain a generic driver model via continuous inverse optimal control with locally optimal examples [107] from demonstrated trajectories in a driving simulator in an environment with multiple autonomous cars, which followed precomputed routes, similar to the previous chapter and as described in Chapter 2.2.

We then adjust the learned weights to model attentive vs. distractive drivers. Specifically, we modify the weights of the collision avoidance features, so the distracted human model has less weight for these features. Therefore, the distracted driver is more likely to collide with the other cars while the attentive driver has high weights for the collision avoidance feature. In future work, we plan to investigate ways of automatically clustering learned $\theta_{\mathcal{H}}$s from data from different users, but we show promising results even with these simple options.

Manipulated Factors

We manipulate the *reward* function that the robot is optimizing. In the ***passive*** condition, the robot optimizes a simple reward function for collision avoidance based on the current belief estimate. It then updates this belief passively, by observing the outcomes of its actions at every time step. In the ***active*** condition, the robot trades off between this reward function and information gain in order to explore the human's driving style.

We also manipulate the *human internal reward parameters* to be ***attentive*** or ***distracted***. The human is simulated to follow the ideal model of reward maximization for our two rewards.

Scenarios and Qualitative Results

Scenario 1: Nudging In to Explore on a Highway. In this scenario, we show an autonomous vehicle actively exploring the human's driving style in a highway driving setting. We contrast the two conditions in Figure 6.3(a). In the passive condition, the autonomous car drives on its own lane without interfering with the human throughout the experiment, and updates its belief based on passive observations gathered from the human car. *However, in the active condition, the autonomous car actively probes the human by nudging into her lane in order to infer her driving style. An attentive human significantly slows down (timid driver) or speeds up (aggressive driver) to avoid the vehicle, while a distracted driver might not realize the autonomous actions and maintain their velocity, getting closer to the*

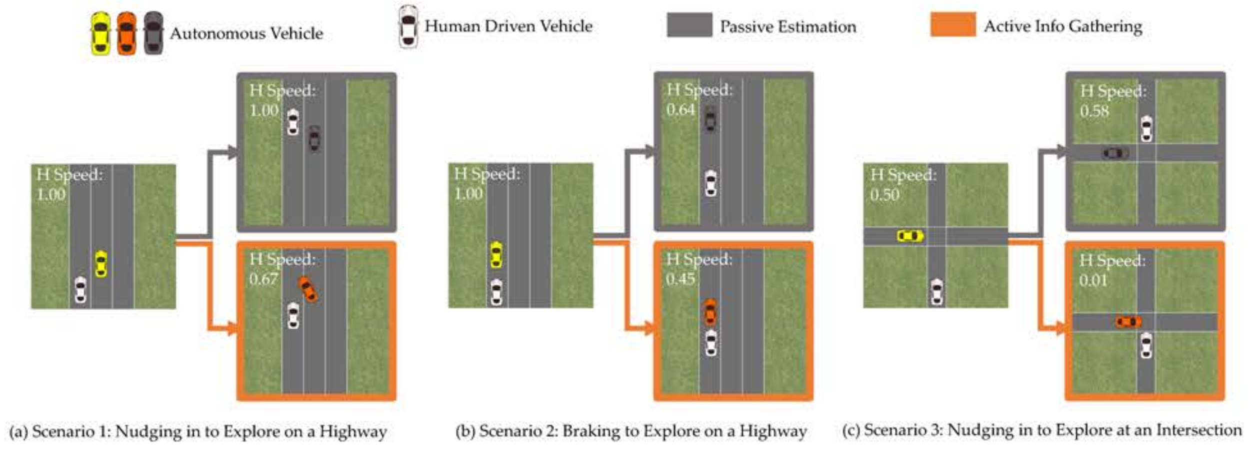


Figure 4.2: Our three scenarios, along with a comparison of robot plans for passive estimation (gray) vs active information gathering (orange). In the active condition, the robot is purposefully nudging in or braking to test human driver attentiveness. The color of the autonomous car in the initial state is yellow, but changes to either gray or orange in cases of passive and active information gathering respectively.

autonomous vehicle. It is this difference in reactions that enables the robot to better estimate θ.

Scenario 2: Braking to Explore on a Highway. In the second scenario, we show the driving style can be explored by the autonomous car probing the human driver behind it. The two vehicles start in the same lane as shown in Figure 6.3(b), where the autonomous car is in the front. In the passive condition, the autonomous car drives straight without exploring or enforcing any interactions with the human driven vehicle. *In the active condition, the robot slows down to actively probe the human and find out her driving style. An attentive human would slow down and avoid collisions while a distracted human will have a harder time to keep safe distance between the two cars.*

Scenario 3: Nudging In to Explore at an Intersection. In this scenario, we consider the two vehicles at an intersection, where the autonomous car actively tries to explore human's driving style by nudging into the intersection. The initial conditions of the vehicles are shown in Figure 6.3(c). In the passive condition, the autonomous car stays at its position without probing the human, and only optimizes for collision avoidance. This provides limited observations from the human car resulting in a low confidence belief distribution. *In the active condition, the autonomous car nudges into the intersection to probe the driving style of the human. An attentive human would slow down to stay safe at the intersection while a distracted human will not slow down.*

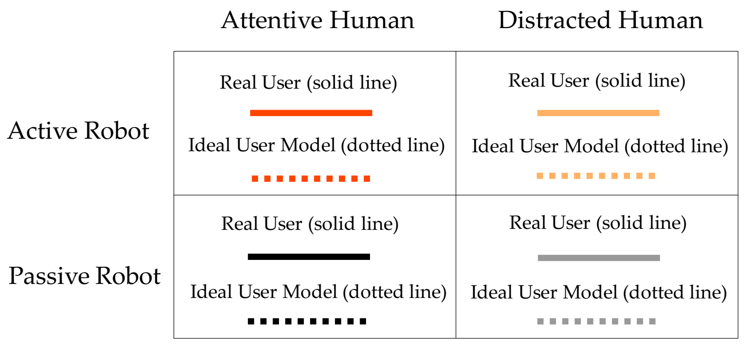


Figure 4.3: Legends indicating active/passive robots, attentive/distracted humans, and real user/ideal model used for all following figures.

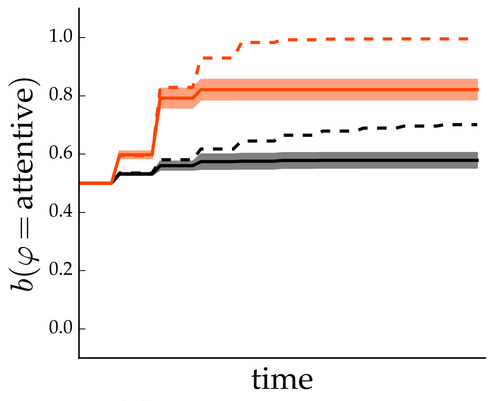
Quantitative Results

Throughout the remainder of this chapter, we use a common color scheme to plot results for our experimental conditions. We show this common scheme in Figure 4.3: darker colors (black and red) correspond to attentive humans, and lighter colors (gray and orange) correspond to distracted humans. Further, the shades of orange correspond to active information gathering, while the shades of gray indicate passive information gathering. We also use solid lines for real users, and dotted lines for scenarios with an ideal user model learned through inverse reinforcement learning.

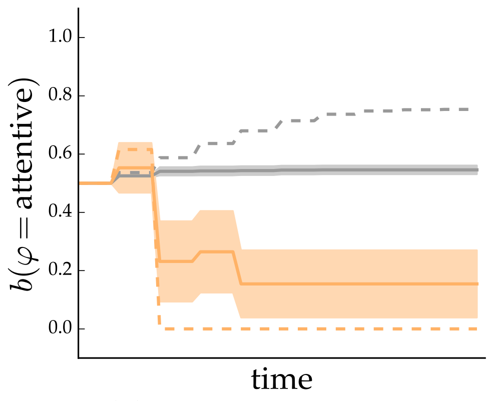
Figure 4.4 plots, using dotted lines, the beliefs over time for the attentive (left) and distracted (right) conditions, comparing in each the passive (dotted black and gray respectively) with the active method (dotted dark orange and light orange respectively). In every situation, the active method achieves a more accurate belief (higher values for attentive on the left, when the true θ is attentive, and lower values on the right, when the true θ is distracted). In fact, passive estimation sometimes incorrectly classifies drivers as attentive when they are distracted and vice-versa.

The same figure also shows (in solid lines) results from our user study of what happens when the robot no longer interacts with an ideal model. We discuss these in the next section.

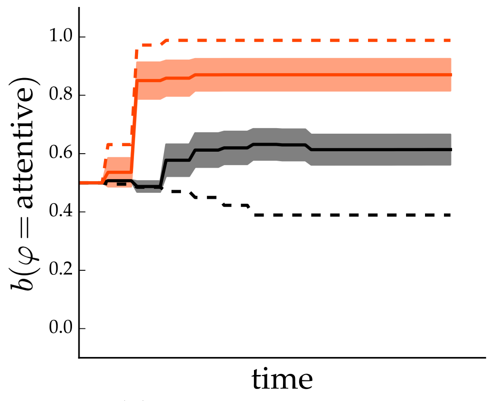
Figure 4.5 and Figure 5.6 plot the corresponding robot and human trajectories for each scenario. The important takeaway from these figures is that there tends to be a larger gap between attentive and distracted human trajectories in the active condition (orange shades) than in the passive condition (gray shades), especially in scenarios 2 and 3. It is this difference that helps the robot better estimate θ: *the robot in the active condition is purposefully choosing actions that will lead to large differences in human reactions*, in order to more easily determine the human driving style.



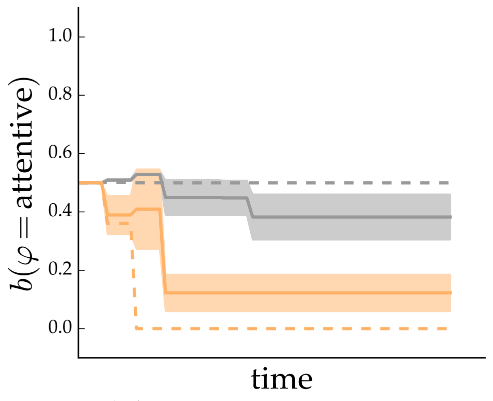
(a) Scenario 1. Attentive



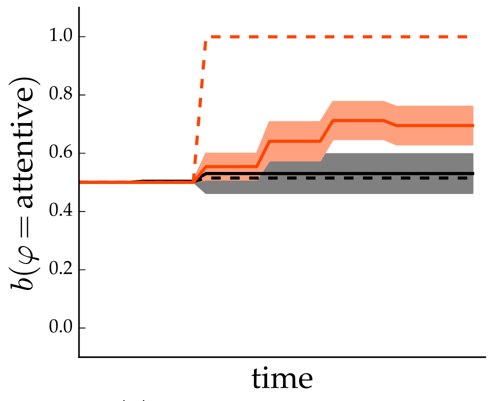
(b) Scenario 1. Distracted



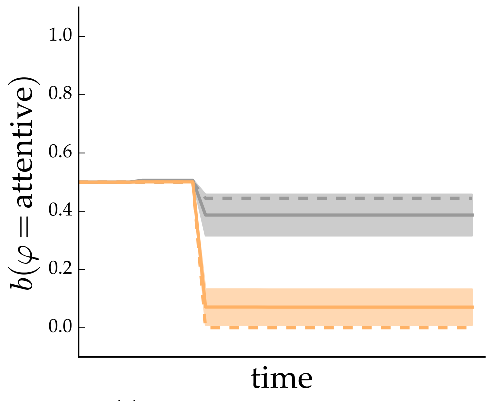
(c) Scenario 2. Attentive



(d) Scenario 2. Distracted



(e) Scenario 3. Attentive



(f) Scenario 3. Distracted

Figure 4.4: The probability that the robot assigns to attentive as a function of time, for the attentive (left) and distracted (right). Each plot compares the active algorithm to passive estimation, showing that active information gathering leads to more accurate state estimation, in simulation and with real users.

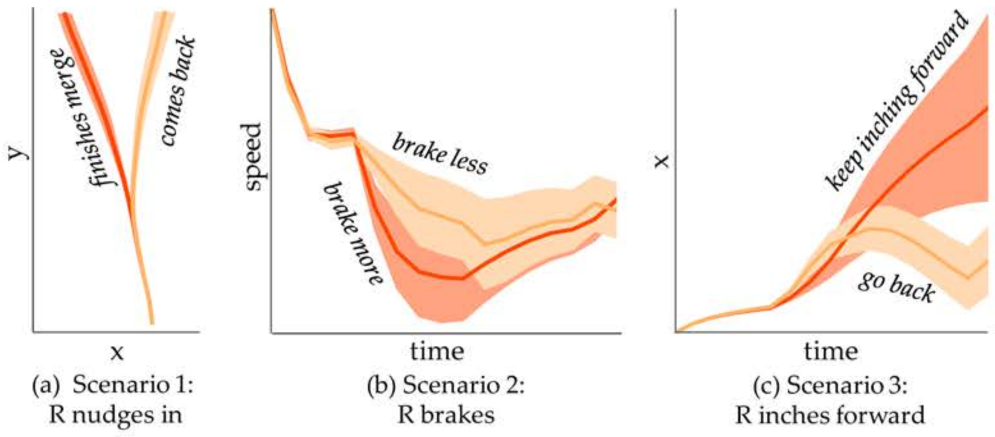


Figure 4.5: Robot trajectories for each scenario in the active information gathering condition. The robot acts differently when the human is attentive (dark orange) vs. when the human is distracted (light orange) due to the trade-off with safety.

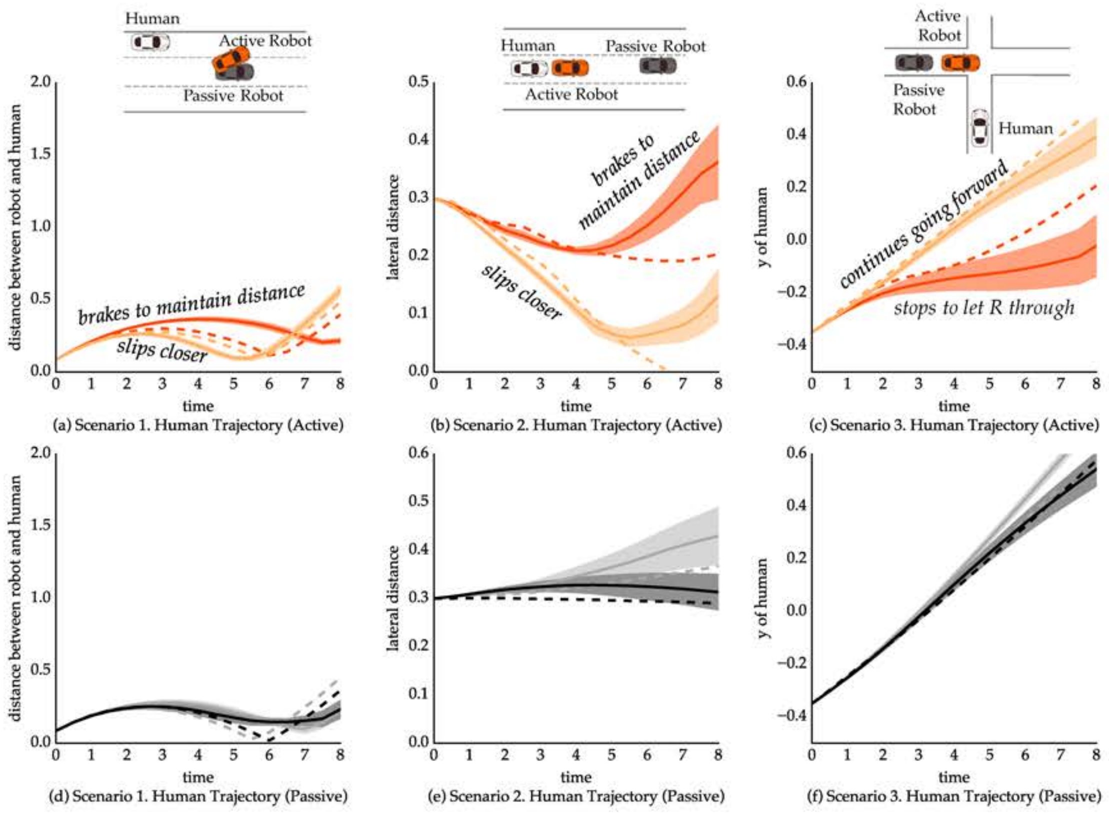


Figure 4.6: The user trajectories for each scenario. The gap between attentive and distracted drivers' actions is clear in the active information gathering case (first row).

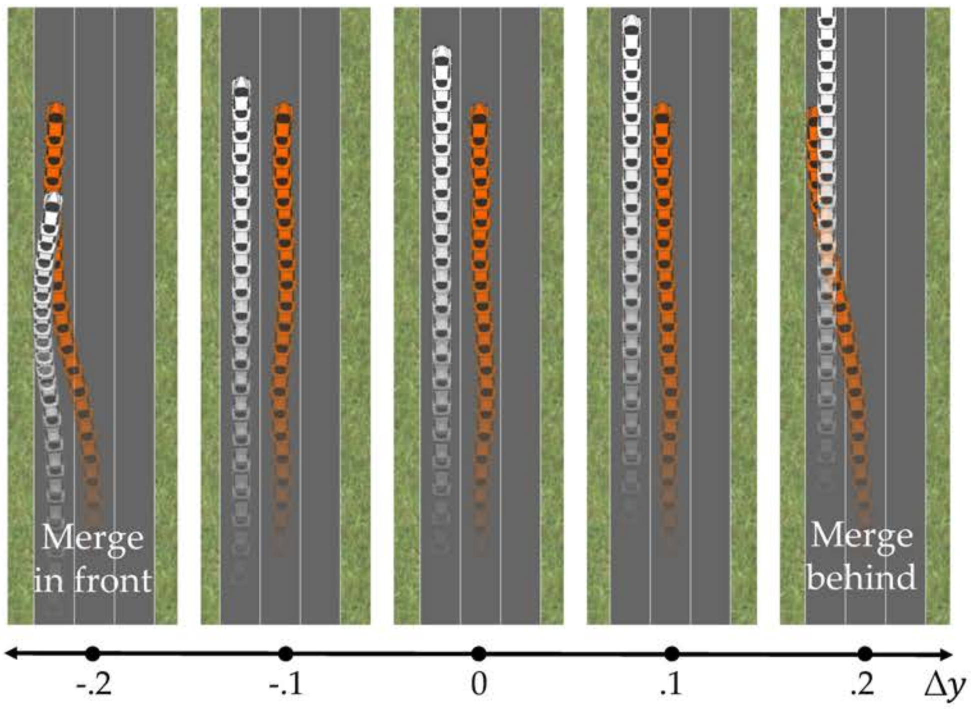


Figure 4.7: Effect of varying the initial condition (relative y position) in the active merge scenario. The robot adapts to merge when feasible and avoid otherwise. The human is attentive in all cases.

Robot Behavior Adapts to the Situation

As Figure 4.5 suggests, active info gathering results in interesting coordination behavior. In Scenario 1, the robot decides to nudge into the person's lane. But what follows next nicely reacts to the person's driving style. The robot proceeds with the merge if the person is attentive, but actually *goes back to its lane* if the person is distracted. Even more interesting is what happens in Scenario 3 at the 4-way-stop. The robot inches forward into the intersection, and proceeds if the person is attentive, but actually *goes back* to allow the person through if they are distracted! These all emerge as the optima in our system.

The behavior also naturally changes as the initial state of the system changes. Figure 4.7 shows different behaviors arising from an attentive driver model but different initial position of the human driver. This shows that even for the same driver model, the robot intelligently adapts its coordination behavior to the situation, sometimes deciding to merge but sometimes not.

This is particularly important, because it might be easy to handcode these coordination strategies for a particular situation. *Much like in the offline estimation case, the robot not only comes up with these strategies, but actually adapts them depending on the situation – the driver*

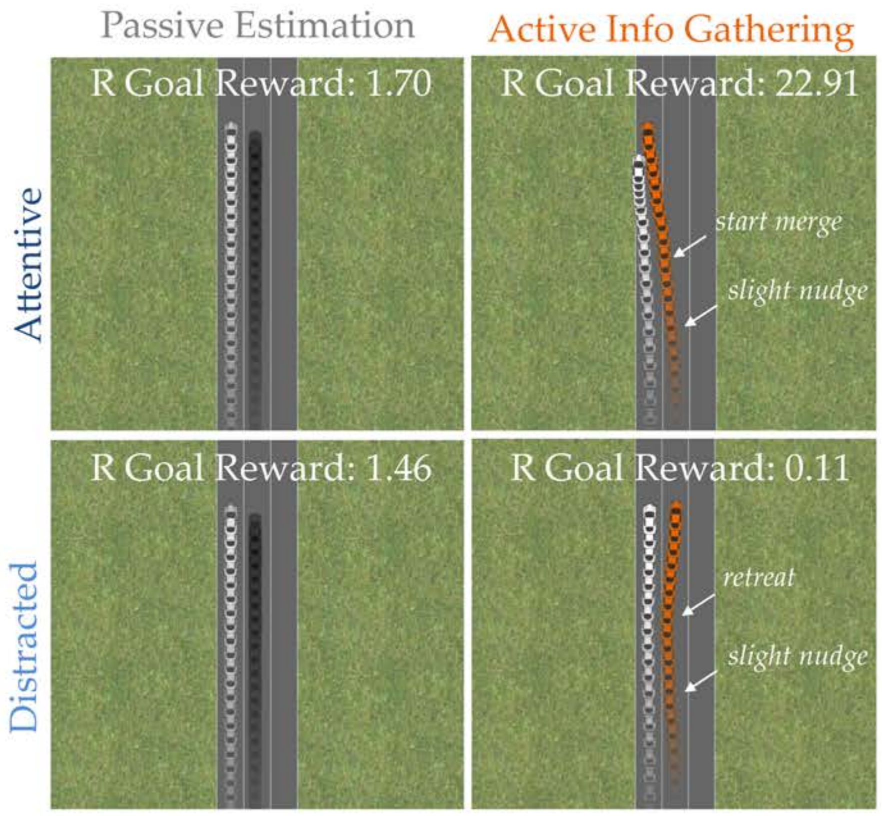


Figure 4.8: Active info gathering improves the robot's ability to efficiently achieve its goal in the case when the human is attentive: where a passive estimator never gets enough information to know that the person is paying attention, an active estimator nudges in, updates its belief, and proceeds with the merge. At the same time, active info gathering does not hurt too much when the person is distracted: the robot nudges in slightly (this does decrease its reward relative to the passive case, but not by much), updates its belief, and retreats to its lane.

style, the initial state, and so on.

Active Information Gathering Helps the Robot's Actual Reward

So far, we have looked at how active information gathering improves estimation of the driver model. This is useful in itself in situations where human internal state estimation is the end-goal. But it is also useful for enabling the robot to better achieve its goal.

Intuitively, knowing the human driving style more accurately should improve the robot's ability to collect reward. For instance, if the robot starts unsure of whether the human is paying attention or not, but collects enough evidence that she is, then the robot can safely merge in front of the person and be more efficient. Of course, this is not always

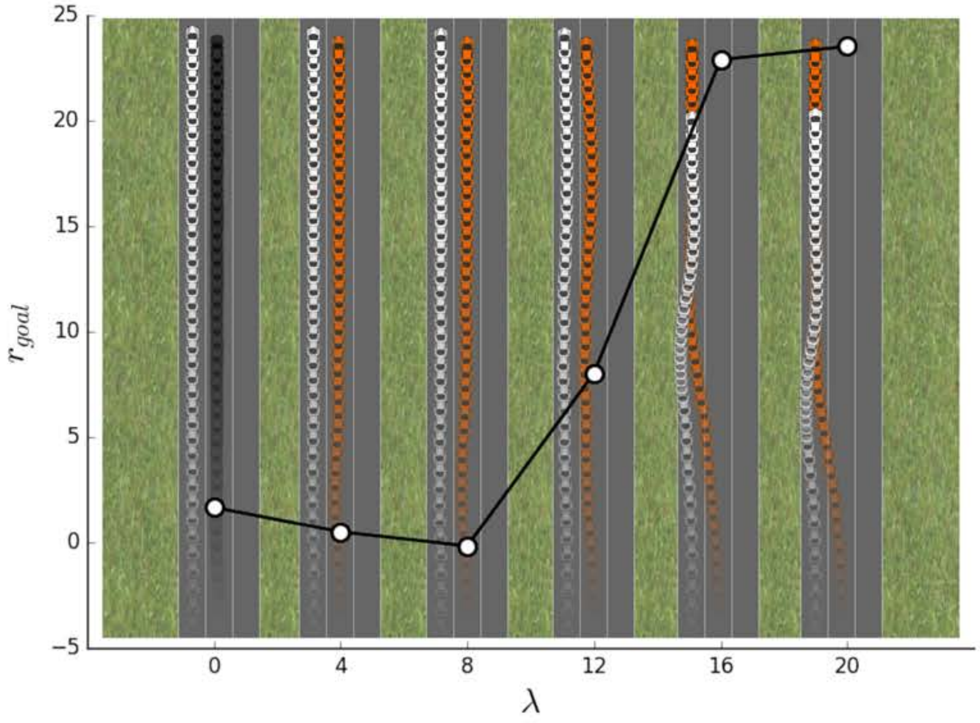


Figure 4.9: Active information gathering behavior when the robot's goal is to merge into the left lane for different values of λ, together with the reward the robot obtains. $\lambda = 0$ results in low reward because the robot does not figure out that the person is attentive and does not merge. A small λ hurts the reward because the information gathering costs but does not buy anything. For higher values, the robot gets enough information that it forces a merge in front of the human.

the case. If the person is distracted, then the information gathering actions could be a waste because the robot ends up not merging anyway.

Figure 4.8 shows what happens in the merging scenario: the robot gains more reward compared to passive estimation by doing information gathering with attentive drivers, because it figures out it is safe to merge in front of them; the robot looses some reward compared to passive estimation with distracted drivers, because it makes the effort to nudge in but has to retreat back to its lane anyway because it cannot merge.

Of course, all this depends on choosing λ, the tradeoff between exploitation and exploration (information gain). Figure 4.9 shows the effect of λ has on the robot's goal reward (not its information gain reward), which shows that not all λs are useful. In other situations, we would also expect to see a large decrease in reward from too much weight on information gain.

Beyond Driving Style: Active Intent Inference

Driving style is not the only type of human internal state that our method enables robots to estimate. If the human has a goal, e.g. of merging into the next lane or not, or of exiting the highway or not, the robot could estimate this as well using the same technique.

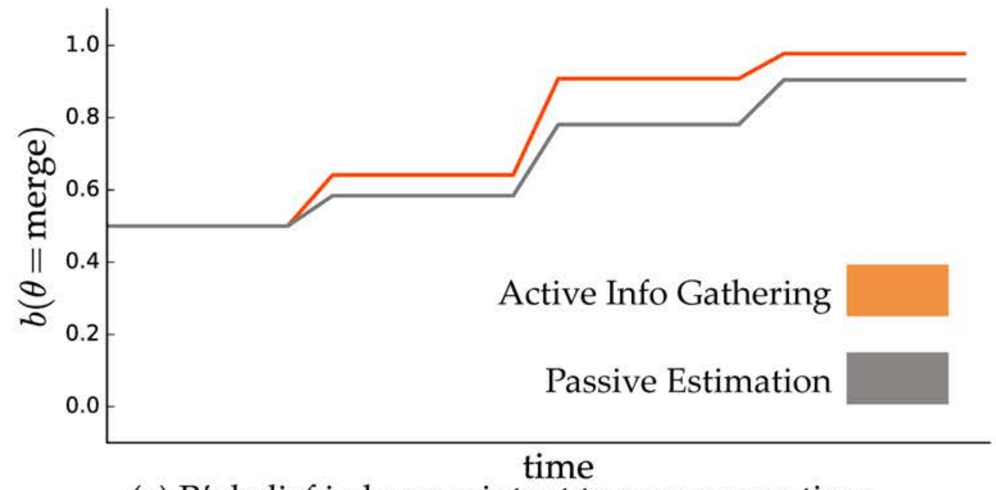
Each possible goal corresponds to a feature. When estimating which goal the human has, the robot is deciding among θs which place weight on only one of the possible goal features, and 0 on the others. Figure 4.10 shows the behavior that emerges from estimating whether the human wants to merge into the robot's lane. In the passive case, the human is side by side with the robot. Depending on the driving style, they might slow down slightly, accelerate slightly, or start nudging into the robot's lane, but since the observation model is noisy the robot does not get quite enough confidence in the human's intent early on. Depending on the robot's reward, it might take a long time before the person can merge. In the active case, the robot decides to probe the person by slowing down and shifting away from the person in order to make room. It then becomes optimal for the person wanting to merge to start shifting towards the robot's lane, giving the robot enough information now to update its belief. In our experiment, we see that this is enough for the person to be able to complete the merge faster, despite the robot not having any incentive to help the person in its reward.

4.3 User Study with Online Estimation

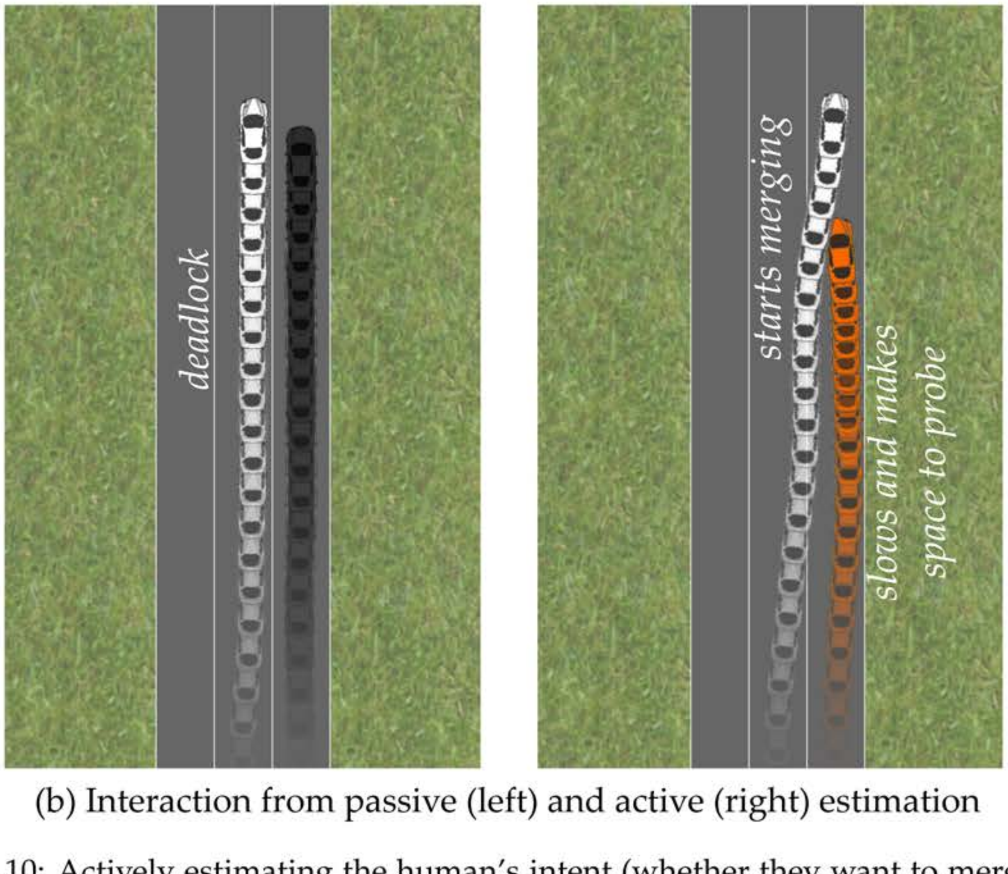
In the previous section, we explored planning for an autonomous vehicle that actively probes a human's driving style, by braking or nudging in and expecting to cause reactions from the human driver that would be different depending on their style. We showed that active exploration does significantly better at distinguishing between attentive and distracted drivers using simulated (ideal) models of drivers. Here, we show the results of a user study that evaluates this active exploration for attentive and distracted human drivers.

Experimental Design

We use the same three scenarios discussed in the previous section.
Manipulated Factors. We manipulated the same two factors as in our simulation experiments: the *reward* function that the robot is optimizing (whether it is optimizing its reward through passive state estimation, or whether it is trading off with active information gathering), and the *human internal state* (whether the user is attentive or distracted). We asked our users to pay attention to the road and avoid collisions for the attentive case, and asked our users to play a game on a mobile phone during the distracted driving experiments.
Dependent Measure. We measured the probability that the robot assigned along the way to the human internal state.



(a) R's belief in human intent to merge over time



(b) Interaction from passive (left) and active (right) estimation

Figure 4.10: Actively estimating the human's intent (whether they want to merge in the right lane or not). The robot slows down and shifts slightly away from the person, which would make someone who wants to merge proceed. This could be useful for robots trying to optimize for the good of all drivers (rather than for their selfish reward function).

Hypothesis. *The active condition will lead to more accurate human internal state estimation, regardless of the true human internal state.*
Subject Allocation. We recruited 8 participants (2 female, 6 male) in the age range of 21-26 years old. All participants owned a valid driver license and had at least 2 years of driving experience. We ran the experiments using a 2D driving simulator with the steering input and acceleration input provided through a steering wheel and a pedals as shown in Figure 6.1 and described in Chapter 2.4. We used a within-subject experiment design with counterbalanced ordering of the four conditions.

Analysis

We ran a factorial repeated-measures ANOVA on the probability assigned to "attentive", using reward (active vs passive) and human internal state (attentive vs distracted) as factors, and time and scenario as covariates. As a manipulation check, attentive drivers had significantly higher estimated probability of "attentive" associated than distracted drivers (.66 vs .34, $F = 3080.3$, $p < .0001$). More importantly, there was a signifiant interaction effect between the factors ($F = 1444.8$, $p < .000$). We ran a post-hoc analysis with Tukey HSD corrections for multiple comparisons, which showed all four conditions to be significantly different from each other, all contrasts with $p < .0001$. In particular, the active information gathering did end up with higher probability mass on "attentive" than the passive estimation for the attentive users, and lower probability mass for the distracted user. This supports our hypothesis that our method works, and active information gathering is better at identifying the correct state.

Figure 4.4 compares passive (grays and blacks) and active (light and dark oranges) across scenarios and for attentive (left) and distracted (right) users. It plots the probability of attentive over time, and the shaded regions correspond to standard error. From the first column, we can see that our algorithm in all cases detects human's attentiveness with much higher probably than the passive information gathering technique shown in black. From the second column, we see that our algorithm places significantly lower probability on attentiveness, which is correct because those users were distracted users. These are in line with the statistical analysis, with active information gathering doing a better job estimating the true human internal state.

Figure 4.5 plots the robot trajectories for the active information gathering setting. Similar to Figure 4.4, the solid lines are the mean of robot trajectories and the shaded regions show the standard error. We plot a representative dimension of the robot trajectory (like position or speed) for attentive (dark orange) or distracted (light orange) cases. The active robot probed the user, but ended up taking different actions when the user was attentive vs. distracted in order to maintain safety. For example, in Scenario 1, the trajectories show the robot is nudging into the human's lane, but the robot decides to move back to its own lane when the human drivers are distracted (light orange) in order to stay safe. In Scenario 2, the robot brakes in front of the human, but it brakes less when the human is distracted. Finally, in Scenario 3, the robot inches forward, but again it stops when if the human is distracted, and even backs up to make space for her.

Figure 5.6 plots the user trajectories for both active information gathering (first row) and passive information gathering (second row) conditions. We compare the reactions of distracted (light shades) and attentive (dark shades) users. There are large differences directly observable, with user reactions tending to indeed cluster according to their internal state. These differences are much smaller in the passive case (second row, where distracted is light gray and attentive is black). For example, in Scenario 1 and 2, the attentive users (dark orange) keep a larger distance to the car that nudges in front of them or brakes in front of them, while the distracted drivers (light orange) tend to keep a smaller distance. In Scenario 3, the attentive drivers tend to slow down and do not cross the intersection, when the robot actively inches forward. None of these behaviors can be detected clearly in the passive information gathering case (second row). This is the core advantage of active information gathering: the actions are purposefully selected by the robot such that users would behave drastically differently depending on their internal state, clarifying to the robot what this state actually is.

Overall, these results support our simulation findings, that our algorithm performs better at estimating the true human internal state by leveraging purposeful information gathering actions.

4.4 Chapter Summary

In this chapter, we took a step towards autonomously producing behavior for interaction and coordination between autonomous cars and human-driven vehicles. We introduced an online estimation algorithm in which the robot actively uses its actions to gather information about the human model so that it can better plan its own actions. Our analysis again shows coordination strategies arising out of planning in our formulation: the robot nudges into someones's lane to check if the human is paying attention, and only completes the merge if they are; the robot inches forward at an intersection, again to check if the human is paying attention, and proceeds if they are, but backs up to let them through if they are not; the robot slows down slightly and shifts in its lane away from the human driver to check if they want to merge into its lane or not.

Importantly, these behaviors change with the human driver style and with the initial conditions – the robot takes different actions in different situations, emphasizing the need to start generating such coordination behavior autonomously rather than relying on hand coded strategies. Even more importantly, the behaviors seem to work when the robot is planning and interacting with real users.

Limitations. While performing our experiments, we found the robot's nominal reward function (trading off between safety and reaching a goal) to be insufficient – in some cases it led to getting dangerously close to the human vehicle and even collisions, going off the road, oscillating in the lane due to minor asymmetries in the environment, etc.

Figure 4.11 shows an example of such behavior that comes from the 4-way-stop domain. For the most part, the car plans to back up to incentivize the human to go through first. But for some values of the human's initial velocity, we observed bad behavior, likely due to convergence to local maxima: the car did not figure out to slow down or back up, and instead if proceeded forward – then it tried to avoid collisions with the person and went off the road, and in the wrong direction (i.e. in the person's way).

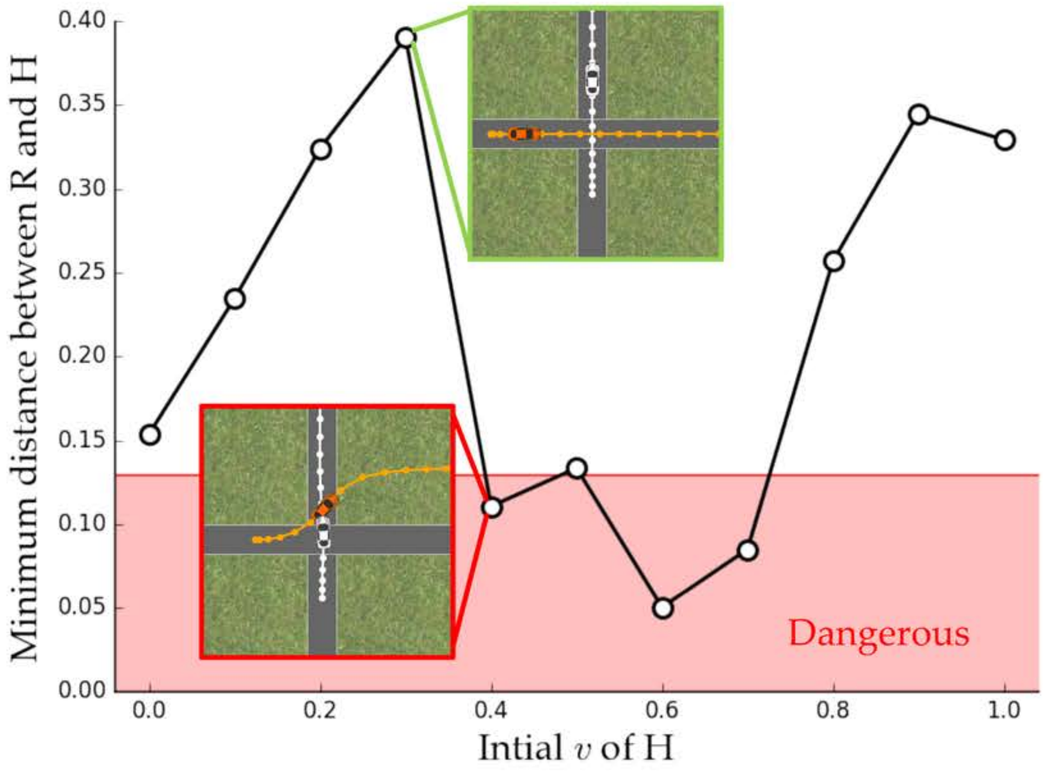


Figure 4.11: Example of bad local optima occurring for certain initial velocities of the human in the 4way intersection scenario.

It seems like while a reward function might be a good enough model for the human, it might be difficult to devise such a universal function for the robot, and the use of hard constraints to ensure safe control would be welcome. We address some of these limitations by introducing temporal logic constraints in the *Safe Control* part of this dissertation.

Another limitation is that we currently focus on a single human driver. Looking to the interaction among multiple vehicles is not just a computational challenge, but also a modeling one. We also have not tested the users' acceptance of information gathering actions. Although these actions are useful, people might not always react positively to being probed.

Conclusion. We are encouraged by the fact that robots can generate useful behavior for interaction autonomously, and plan to explore information-gathering actions on human state further, including beyond autonomous driving scenarios.

Chapter 5

Active Preference-Based Learning of Reward Functions

Reward functions play a central role in specifying how dynamical systems should act: how an end-user wants their assistive robot arm to move, or how they want their autonomous car to drive. For many systems, end-users have difficulty providing demonstrations of what they want. For instance, they cannot coordinate 7 degrees of freedom (DOFs) at a time [6], and they can only show the car how *they* drive, not how they want *the car* to drive [20]. In such cases, another option is for the system to regress a reward function from labeled state-action pairs, but assigning precise numeric reward values to observed robot actions is also difficult.

In this chapter, we propose a *preference-based* approach to learning desired reward functions in a dynamical system. Instead of asking for demonstrations, or for the value of the reward function for a sample trajectory (e.g., "rate the safety of this driving maneuver from 1 to 10"), we ask people for their relative preference between two sample trajectories (e.g., "is ξ_1 more safe or less safe than ξ_2?").

Active preference-based learning has been successfully used in many domains [4, 90, 33, 32], but what makes applying it to learning reward functions difficult is the complexity of the queries, as well as the continuous nature of the underlying hypothesis space of possible reward functions. We focus on dynamical systems with continuous or hybrid discrete-continuous state. In this setting, queries consist of two candidate continuous state and action space trajectories that satisfy the system's dynamics, in an environment or scenario that the learning algorithm also needs to decide on, consisting of an initial state and the trajectories of *other* agents in the scene. Consider again the example of autonomous driving. In this situation, a query would consist of two trajectories for the car from some starting state among other cars following their own trajectories.

Typically in preference-based learning, the queries are actively selected by searching some discrete or sampled set (e.g., [78, 80, 63, 180, 167, 7]). Our first hypothesis is that in our setting, the continuous and high-dimensional nature of the queries renders relying on a discrete set ineffective. Preference-based work for such spaces has thus far collected data passively [181, 151]. Our second hypothesis is that active generation of queries leads to better reward functions faster.

We contribute an algorithm for *actively synthesizing* queries from scratch. We do continuous optimization in query space to maximize the expected volume removed from the hypothesis space. We use the human's response to assign weights to the hypothesis space in the form of a *log-concave distribution*, which provides an approximation of the objective via a Metropolis algorithm that makes it differentiable w.r.t. the query parameters. We provide a bound on the number of iterations required to converge.

We compare our algorithm to non-active and non-synthesis approaches to test our hypotheses. We use an experimental setup motivated by autonomous driving, and show that our approach converges faster to a desired reward function. Finally, we illustrate the performance of our algorithm in terms of accuracy of the reward function learned through an in-lab usability study.

5.1 Preference-Based Learning Problem

Modeling Choices

Our goal is to model the behavior and preferences of a human $\mathcal{H}$ for how a dynamical system should act. Similar to the previous chapters (as discussed in Chapter 2), we model the overall system including $\mathcal{H}$, and all the other agents (robot $\mathcal{R}$) as a fully-observable dynamical system.

We define a trajectory $\xi \in \Xi$, where $\xi = (x^0, u_R^0, u_H^0), \dots, (x^N, u_R^N, u_H^N)$ is a finite horizon sequence of states and actions of all agents. Here, Ξ is a set of all feasible continuous trajectories. A feasible trajectory is one that satisfies the dynamics of $\mathcal{H}$ and $\mathcal{R}$. As in IRL (see Chapter 2.2), we parameterize the preference reward function as a linear combination of a set of features:

$$r_{\mathcal{H}}(x^t, u_{\mathcal{R}}^t, u_{\mathcal{H}}^t) = w^\top \phi(x^t, u_{\mathcal{R}}^t, u_{\mathcal{H}}^t), \tag{5.1}$$

with w being a vector of the weights for the feature function $\phi(x^t, u_{\mathcal{R}}^t, u_{\mathcal{H}}^t)$ evaluated at every state and action pair. We assume a d-dimensional feature function, so $\phi(x^t, u_{\mathcal{R}}^t, u_{\mathcal{H}}^t) \in \mathbb{R}^d$.

For simpler notation, we combine the $N+1$ elements of ϕ so $\Phi = \sum_{t=0}^{N} \phi(x^t, u_{\mathcal{R}}^t, u_{\mathcal{H}}^t)$. Therefore, $\Phi(\xi)$ evaluates over a trajectory ξ. We finally reformulate the reward function as the following inner product:

$$R_{\mathcal{H}}(\xi) = w \cdot \Phi(\xi). \tag{5.2}$$

Our goal is to learn $R_{\mathcal{H}}$.

Approach Overview

Inverse Reinforcement Learning (IRL) [188, 107, 159] enables us to learn $R_{\mathcal{H}}$ through demonstrated trajectories. However, IRL requires the human to show demonstrations of the optimal sequence of actions. Providing demonstrations can be challenging for many human-robot tasks. Furthermore, generating interesting training data that actively explores $R_{\mathcal{H}}$ through atypical, interesting environments (which is necessary in many cases for resolving ambiguities) works well [116, 41] but in practice can make (good) demonstrations infeasible: the algorithm cannot physically manufacture environments, and therefore relies on simulation, which makes demonstrations only possible through teleoperation.

For these reasons, we assume demonstrations are not available in this chapter. Instead, we propose to leverage preference-based learning, which queries $\mathcal{H}$ to provide comparisons between two candidate trajectories. We propose a new approach for active learning of a reward function for human preferences through comparisons.

We split trajectories ξ into two parts: a scenario and the trajectory of the agent $\mathcal{H}$ whose reward function we are learning. We formalize a scenario to be the initial state of the environment as well as the sequence of actions for the other agent(s) $\mathcal{R}$, $\tau = (x^0, u^0_{\mathcal{R}}, \ldots, u^N_{\mathcal{R}})$. Given an environment specified by scenario $\tau \in \mathcal{T}$, the human agent will provide a finite sequence of actions $\mathbf{u}_{\mathcal{H}} = u^0_H, \ldots, u^N_H$ in response to scenario τ. This response, along with the scenario τ defines a trajectory ξ showing the evolution of the two systems together in the environment.

We iteratively synthesize queries where we ask the human to compare between two trajectories ξ_A and ξ_B defined over the same fixed scenario τ as shown in Figure 5.1 (a). Their answer provides us information about w. In what follows, we discuss how to update our probability distribution over w given the answer to a query, and how to actively synthesize queries in order to efficiently converge to the right w.

5.2 Learning Reward Weights from Preferences of Synthesized Queries

In this section, we describe how we update a distribution over reward parameters (weights w in equation (5.2)) based on the answer of one query. We first assume that we are at iteration t of the algorithm and an already synthesized pair of trajectories ξ_A and ξ_B in a common scenario is given (we discuss in the next section how to generate such a query). We also assume $\mathcal{H}$ has provided her preference for this specific pair of trajectories at iteration t. Let her answer be I, with $I_t = +1$ if she prefers the former, and $I_t = -1$ if she prefers the latter. This answer gives us information about w: she is more likely to say $+1$ if ξ_A has higher reward than ξ_B, and vice-versa, but she might not be exact in determining I_t. We thus model the probability $p(I|w)$ as noisily capturing the preference w.r.t. $R_{\mathcal{H}}$:

$$p(I_t|w) = \begin{cases} \frac{\exp(R_{\mathcal{H}}(\xi_A))}{\exp(R_{\mathcal{H}}(\xi_A))+\exp(R_{\mathcal{H}}(\xi_B))} & I_t = +1 \\ \\ \frac{\exp(R_{\mathcal{H}}(\xi_B))}{\exp(R_{\mathcal{H}}(\xi_A))+\exp(R_{\mathcal{H}}(\xi_B))} & I_t = -1 \end{cases} \tag{5.3}$$

We start with a prior over the space of all w, i.e., w is uniformly distributed on the unit ball. Note that the scale of w does not change the preference I_t, so we can constrain $||w|| \leq 1$ to lie in this unit ball. After receiving the input of the human I_t, we propose using a Bayesian update to find the new distribution of w:

$$p(w|I_t) \propto p(w) \cdot p(I_t|w). \tag{5.4}$$

Let

$$\varphi = \Phi(\xi_A) - \Phi(\xi_B) \tag{5.5}$$

Then our update function that multiplies the current distribution at every step is:

$$f_\varphi(w) = p(I_t|w) = \frac{1}{1+\exp(-I_t w^\top \varphi)} \tag{5.6}$$

Updating the distribution of w allows us to reduce the probability of the undesired parts of the space of w, and maintain the current probability of the preferred regions.

5.3 Synthesizing Queries through Active Volume Removal

The previous section showed how to update the distribution over w after getting the response to a query. Here, we show how to synthesize the query in the first place: we want to find the next query such that it will help us remove as much volume (the integral of the unnormalized pdf over w) as possible from the space of possible rewards.

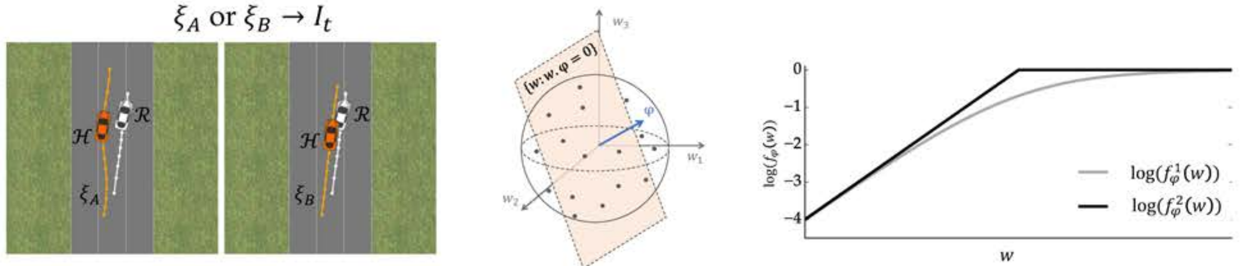
Formulating Query Selection as Constrained Optimization

We synthesize experiments by maximizing the volume removed under the feasibility constraints of φ:

$$\begin{aligned} \max_{\varphi} \quad & \min\{\mathbb{E}[1 - f_\varphi(w)], \mathbb{E}[1 - f_{-\varphi}(w)]\} \\ \text{subject to} \quad & \varphi \in \mathbb{F} \end{aligned} \tag{5.7}$$

The constraint in this optimization requires φ to be in the feasible set $\mathbb{F}$:

$$\begin{aligned} \mathbb{F} = \{\varphi : \varphi = \Phi(\xi_A) - \Phi(\xi_B), \xi_A, \xi_B \in \Xi, \\ \tau = (x^0, \mathbf{u}_{\mathcal{R}}^A) = (x^0, \mathbf{u}_{\mathcal{R}}^B)\} \end{aligned} \tag{5.8}$$



(a) Preference query. (b) Query response effect. (c) log-concave update of w.

Figure 5.1: In (a) two candidate trajectories are provided for comparison to the human oracle. ξ_A shows a smoother trajectory without any collisions. In (b) the unit ball is representing the space of w. We synthesize experiments that correspond to a separating hyperplane $\{w : w \cdot \varphi = 0\}$, and reweigh samples of w on each side of the hyperplane in order to update the distribution of w. In (c) we show the two choices of log-concave update functions. Here, $f^1_\varphi(w) = p(I_t|w)$ is the Bayesian update, and $f^2_\varphi(w) = \min(1, \exp(I_t w^\top \varphi))$ is our choice of simpler update function.

which is a set of the difference of features over feasible trajectories $\xi_A, \xi_B \in \Xi$ defined over the same scenario τ.

In this optimization, we maximize the minimum of the split between the two spaces preferred by either choices of the human agent. Each term in the minimum is the volume removed depending on the input of the human I_t, i.e., the preference. The expectation is taken w.r.t. to distribution over w.

Solution

We first reformulate our problem as an unconstrained optimization, where we enforce the feasibility of trajectories by directly optimizing over the query components, $x^0, \mathbf{u}_\mathcal{R}, \mathbf{u}^A_\mathcal{H}, \mathbf{u}^B_\mathcal{H}$, as opposed to optimizing in the desired feature difference φ:

$$\max_{x^0,\mathbf{u}_\mathcal{R},\mathbf{u}^A_\mathcal{H},\mathbf{u}^B_\mathcal{H}} \quad \min\{\mathbb{E}[1 - f_\varphi(w)], \mathbb{E}[1 - f_{-\varphi}(w)]\} \tag{5.9}$$

Here, φ remains a function of $x^0, \mathbf{u}_\mathcal{R}, \mathbf{u}^A_\mathcal{H}, \mathbf{u}^B_\mathcal{H}$. We will solve this by local optimization using a Quasi-Newton method (L-BFGS [11]). To do so, we need a differentiable objective.

The distribution $p(w)$ can become very complex and there is no simple way to compute the volume removed by a Bayesian update, let alone differentiate through it. We therefore resort to sampling, and optimize an approximation of the objective obtained via samples, where weights are sampled in proportion to their probability. Assume we sample $w_1, \ldots, w_M$ independently from the distribution $p(w)$. Then we can approximate

$p(w)$ by the empirical distribution composed of point masses at w_i's:

$$p(w) \sim \frac{1}{M} \sum_{i=1}^{M} \delta(w_i). \tag{5.10}$$

Then the volume removed by an update $f_\varphi(w)$ can be approximated by:

$$\mathbb{E}[1 - f_\varphi(w)] \simeq \frac{1}{M} \sum_{i=1}^{M} (1 - f_\varphi(w_i)). \tag{5.11}$$

Given such samples, the objective is now differentiable w.r.t. φ, which is differentiable w.r.t. the starting state and controls – the ingredients of the query which are the variables in equation (10.2). What remains is to get the actual samples. To do so, we take advantage of the fact that $p(w)$ is a log-concave function, and the update function $f_\varphi(w)$ defined here is also log-concave in w as shown in Figure 5.1 (c); therefore, the posterior distribution of w stays log-concave. Note we do not need to renormalize the distribution $p(w)$ after a Bayesian update, i.e. divide $p(w)$ by its integral. Instead, we use Metropolis Markov Chain methods to sample from $p(w)$ without normalization.

Log-concavity is useful because we can take advantage of efficient polynomial time algorithms for sampling from the current $p(w)$ [117][1]. In practice, we use an adaptive Metropolis algorithm, where we initialize with a warm start by computing the mode of the distribution, and perform a Markov walk [74].

We could find the mode of f_φ from (5.6), but it requires a convex optimization. We instead speed this up by choosing a similar log-concave function whose mode evaluates to zero always, and which reduces the probability of undesired w by a factor of $\exp(I_t w^\top \varphi)$:

$$f_\varphi(w) = \min(1, \exp(I_t w^\top \varphi)) \tag{5.12}$$

Figure 5.1 (c) shows this simpler choice of update function in black with respect to $p(I_t|w)$ in gray. The shape is similar, but enables us to start from zero, and a Markov walk will efficiently sample from the space of w.

In Figure 5.1 (b), we show a simple diagram demonstrating our approach. Here, w first uniformly lies on a unit d-dimensional ball. For simplicity, here we show a 3D ball. The result of a query at every step is a state and trajectories that result in a feature difference vector φ, normal to the hyperplane $\{w : w^\top \varphi = 0\}$, whose direction represents the preference of the human, and its value lies in $\mathbb{F}$ defined in equation (5.8). We reduce the volume of w by reducing the probability distribution of the samples of w on the rejected side of the hyperplane through the update function $f_\varphi(w)$ that takes into account noise in the comparison.

[1]Note that computing the actual volume removed can be converted to the task of integrating a log-concave function, for which efficient algorithms do exist. The purpose of using samples instead is to have an expression suitable for maximization of volume removed, i.e. an expression differentiable w.r.t. the query.

5.4 Algorithm and Performance Guarantees

Armed with a way of generating queries and a way of updating the reward distribution based on the human's answer to each query, we now present our method for actively learning a reward function in Algorithm 1. The inputs to the algorithm are a set of features ϕ, the desired horizon N, the dynamics of the system f, and the number of iterations *iter*. The goal is to converge to the true distribution of w, which is equivalent to finding the preference reward function $R_{\mathcal{H}}(\xi) = w \cdot \Phi(\xi)$.

We first initialize this distribution to be uniform over a unit ball in line 3. Then, for M iterations, the algorithm repeats these steps: volume estimation, synthesizing a feasible query, querying the human, and updating the distribution.

We sample the space of w in line 5. Using these samples, and the dynamics of the system, SynthExps solves the optimization in equation (10.2): it synthesizes a feasible pair of trajectory that maximizes the expected volume removed from the distribution of $p(w)$. The answer to the query is received in line 7. We compute $f_\varphi(w)$, and update the distribution in line 10.

Algorithm 1 Preference-Based Learning of Reward Functions

```
1:  **Input:** Features $\phi$, horizon $N$, dynamics $f$, $iter$
2:  **Output:** Distribution of $w$: $p(w)$
3:  Initialize $p(w) \sim \text{Uniform}(B)$, for a unit ball $B$
4:  **While** $t < iter$:
5:      $W \leftarrow M$ samples from AdaptiveMetropolis($p(w)$)
6:      $(x^0, \mathbf{u}_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}^A, \mathbf{u}_{\mathcal{H}}^B) \leftarrow \text{SynthExps}(W, f)$
7:      $I_t \leftarrow \text{QueryHuman}(x^0, \mathbf{u}_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}^A, \mathbf{u}_{\mathcal{H}}^B)$
8:      $\varphi = \Phi(x^0, \mathbf{u}_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}^A) - \Phi(x^0, \mathbf{u}_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}^B)$
9:      $f_\varphi(w) = \min(1, I_t \exp(w^\top \varphi))$
10:     $p(w) \leftarrow p(w) \cdot f_\varphi(w)$
11:     $t \leftarrow t + 1$
12: **End for**
```

Regarding the convergence of Algorithm 1, one cannot generally make any strict claims for several reasons: We replace the distribution $p(w)$ by an empirical distribution, which could introduce errors. The maximization in line 6 is via non-convex optimization which does not necessarily find the optimum, and even if it was guaranteed that the global optimum is found, it could potentially make very little progress in terms of volume removal, since the set $\mathbb{F}$ can be arbitrary.

Putting aside the issues of sampling and global optimization, we can compare what Algorithm 1 does to the best one could do with the set $\mathbb{F}$. Algorithm 1 can be thought of as a *greedy* volume removal.

Theorem 1. *Under the following assumptions:*

- *The update function is f_φ as defined in equation* (5.6),
- *The human inputs are noisy similar to equation* (5.3),
- *The errors introduced by sampling and non-convex optimization are ignored,*

Algorithm 1 removes at least $1-\epsilon$ times as much volume as removed by the best adaptive strategy after $\ln(\frac{1}{\epsilon})$ times as many iterations.

Proof. Removed volume can be seen to be an adaptive submodular function, defined in terms of the choices φ and the human input I_t, as defined in [69]. It is also adaptive monotone; thus, the results of [69] imply that greedy volume removal for l steps in expectation removes at least $(1-\exp(-l/k))\mathrm{OPT}_k$ where OPT_k is the best solution any adaptive strategy can achieve after k steps. Setting $l = k\ln(\frac{1}{\epsilon})$ gives us the desired result.

One caveat is that in equation (5.6) the human input I_t is treated as worst case, i.e., in the synthesis step, one maximizes the *minimum* removed volume (over the possible I_t). Namely maximizing the following quantity:

$$\min\{\mathbb{E}_w[1-\Pr[I_t=+1|w]]), \mathbb{E}_w[1-\Pr[I_t=-1|w]])\}. \tag{5.13}$$

Normally the greedy strategy in adaptive submodular maximization should treat I_t as probabilistic. In other words instead of the minimum one should typically maximize the following quantity:

$$\begin{aligned}\Pr[I_t=+1]\cdot\mathbb{E}_w[1-\Pr[I_t=+1|w]]+\\ \Pr[I_t=-1]\cdot\mathbb{E}_w[1-\Pr[I_t=-1|w]].\end{aligned} \tag{5.14}$$

However, note $\mathbb{E}_w[1-\Pr[I_t=+1|w]]$ is simply $\Pr[I_t=-1]$ and similarly $\mathbb{E}_w[1-\Pr[I_t=+1|w]]$ is $\Pr[I_t=+1]$. Therefore, Algorithm 1 is maximizing $\min(\Pr[I_t=-1], \Pr[I_t=+1])$, whereas greedy submodular maximization should be maximizing $2\Pr[I_t=-1]\Pr[I_t=+1]$. It is easy to see that these two maximizations are equivalent, since the sum $\Pr[I_t=-1]+\Pr[I_t=+1]=1$ is fixed. □

5.5 Simulation Experiment

In this section, we evaluate our theory using a semiautonomous driving example. Our goal is to learn people's preferred reward function for driving. In this context, our approach being preference-based is useful since it allows the users to only compare two candidate trajectories in various scenarios instead of requiring the user to demonstrate a full trajectory (of how they would like to drive, not how they actually drive). In addition, our approach being *active* enables choosing informative test cases that are otherwise difficult to encounter in driving scenarios. For example, we can address the preference of drivers in moral dilemma situations (deciding between two undesirable outcomes), which are very unlikely to arise in standard collected driving data. Finally, our approach

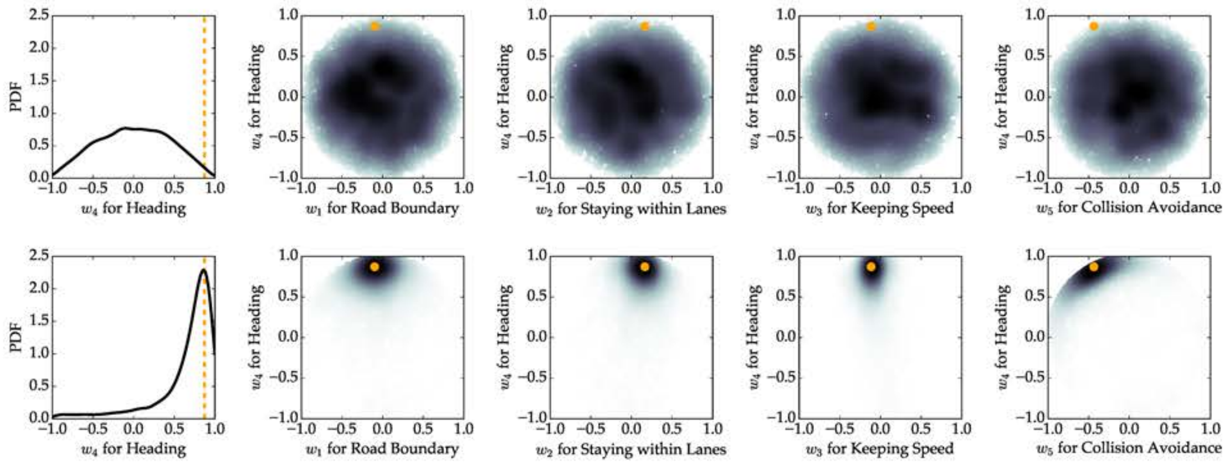


Figure 5.2: Distribution of w_4, the weight for the heading feature, relative to the other features. The top plots shows the starting distribution, and the bottom plot shows the distribution at convergence. The orange dot and dotted line show the ground truth for the weights.

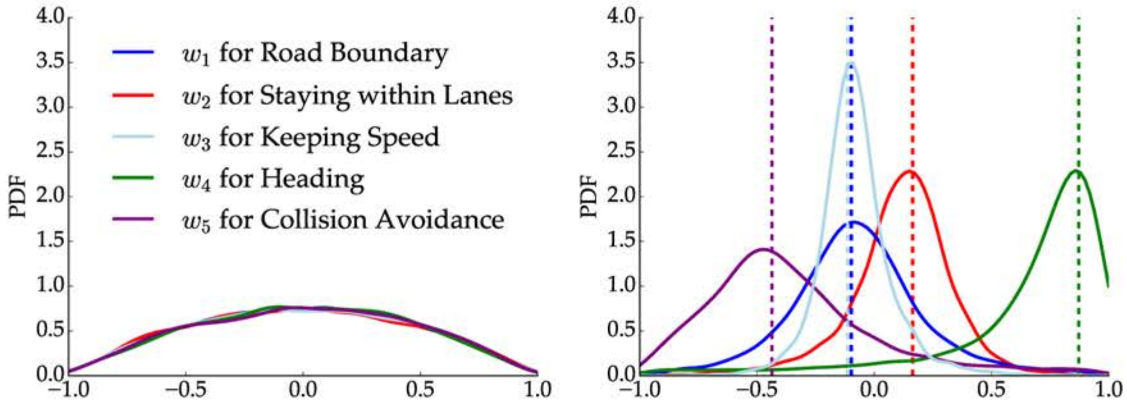


Figure 5.3: Distribution over all weights before/after convergence. The dotted lines show the ground truth of the weights.

synthesizes these test cases from scratch, which should help better exploit the continuous and high-dimensional space of queries. We put these advantages to the test in what follows.

Experimental Setup

We assume a human driven vehicle $\mathcal{H}$ living in an environment with another vehicle $\mathcal{R}$, and we synthesize trajectories containing two candidate sequence of actions for the human driven car, while for every comparison we fix the synthesized scenario (i.e., the initial state of the environment and the sequence of actions of $\mathcal{R}$). Figure 5.1 (a) shows an example of this comparison. The white vehicle is $\mathcal{R}$, and the orange vehicle corresponds to $\mathcal{H}$. The white and orange lines show the path taken by the human and robot, respectively, in the two cases over a horizon of $N = 5$.

Assuming the vehicles follow the same dynamics model as in Chapter 2.4, we learn the reward function of the human's preferences based on queries similar to Figure 5.1 (a). We define a set of features that allow representing this cost function. These are the same features introduced in Chapter 2.2, i.e., $\phi_1 \propto c_1 \cdot \exp(-c_2 \cdot d^2)$ corresponds to penalizing getting close to the boundaries of the road, where d is the distance between the vehicle and these boundaries, and c_1 and c_2 are appropriate scaling factors. We use a similar feature ϕ_2 for enforcing staying within a single lane by penalizing leaving the boundaries of the lane. We also encourage higher speed for moving forward through $\phi_3 = (v - v_{\max})^2$, where v is the velocity of the vehicle, and $v_{\max}$ is the speed limit. Also, we would like the vehicle to have a heading along with the road using a feature $\phi_4 = \beta_{\mathcal{H}} \cdot \boldsymbol{n}$, where $\beta_{\mathcal{H}}$ is the heading of $\mathcal{H}$, and $\boldsymbol{n}$ is a normal vector along the road. Our last feature ϕ_5 corresponds to collision avoidance, and is a non-spherical Gaussian over the distance of $\mathcal{H}$ and $\mathcal{R}$, whose major axis is along the robot's heading. Then, we aim to learn a distribution over the weights corresponding to these features $w = \begin{bmatrix} w_1 & w_2 & w_3 & w_4 & w_5 \end{bmatrix}$ so $R_{\mathcal{H}} = w \cdot \Phi(\xi)$ best represents the preference reward function.

Conditions

We compare our algorithm with two baselines: non-active and non-synthesis.

First, we compare it to a *non-active* version. Instead of actively generating queries that will remove the most volume, we uniformly sample a scenario. We do not use totally random trajectories for the cars as this would be a weak baseline, instead we sample two candidate weight vectors w^A and w^B from the current distribution on w. We then maximize the reward functions $w^A \cdot \Phi(\xi_A)$ and $w^B \cdot \Phi(\xi_B)$ to solve for optimal $\mathbf{u}_{\mathcal{H}}^A$ and $\mathbf{u}_{\mathcal{H}}^B$. That creates our query, comparing two reward functions, but no longer optimized to efficiently learn w. The trajectories generated through this other approach are then used to query the human, and update the distribution of $p(w)$ similar to Algorithm 1. Second, we compare it against a *non-synthesis* (discrete) version. Instead of solving a continuous optimization problem to generate a query, we do a discrete search over a sampled set of queries.

Metric

We evaluate our results using a hidden reward function $R_{\text{true}} = w_{\text{true}} \cdot \Phi(\xi)$. We query an ideal user who knows R_{true} and uses it to compare pairs of trajectories, and show the w computed by our algorithm efficiently converges to w_{true}.

At every step of the iteration, we compute the following measure of convergence:

$$m = \mathbb{E}[\frac{w \cdot w_{\text{true}}}{|w||w_{\text{true}}|}]. \tag{5.15}$$

Here, m computes the average heading of the current distribution of w with respect to w_{true} – how similar the learned reward is. Since the prior distribution of w is symmetric (uniformly distributed on a unit ball), this expectation starts at 0, and moves closer to 1 at every step of the iteration.

Hypothesis

H1. *The reward function learned through our algorithm is closer to the true reward compared to the non-active baseline.*
H2. *The reward function learned through our algorithm is closer to the true reward compared to the non-synthesis baseline.*

Results

We run a paired t-test for each hypothesis. Supporting H1, we find that our algorithm significantly outperforms the non-active version ($t(1999) = 122.379$, $p < .0001$), suggesting that it is important to be doing active learning in these continuous and high-dimensional query spaces. Supporting H2, we find that our algorithm significantly outperforms the non-synthesis version ($t(1999) = 35.39$, $p < .0001$), suggesting the importance of synthesizing queries instead of relying on a discrete set. Note these results hold even with conservative Bonferroni corrections for multiple comparisons.

Figure 5.3 shows the distribution of w for our algorithm corresponding to the five features after 200 iterations, showing convergence to close to the true weights (in dotted lines). The mode of the distribution of w_1 has a negative weight enforcing staying within the roads, and w_2 has a positive weight enforcing staying within your own lane. w_3 also shows a slight preference for keeping speed, and w_4 shows a significant preference for keeping heading. w_5 also shows the distribution for the weight of collision avoidance.

Further, we show the initial and final two dimensional projection of the density of w_4, the weight of the heading feature, with respect to all the other w's in Figure 5.2. This shift and convergence of the distribution is clear from the top plot to the bottom plot.

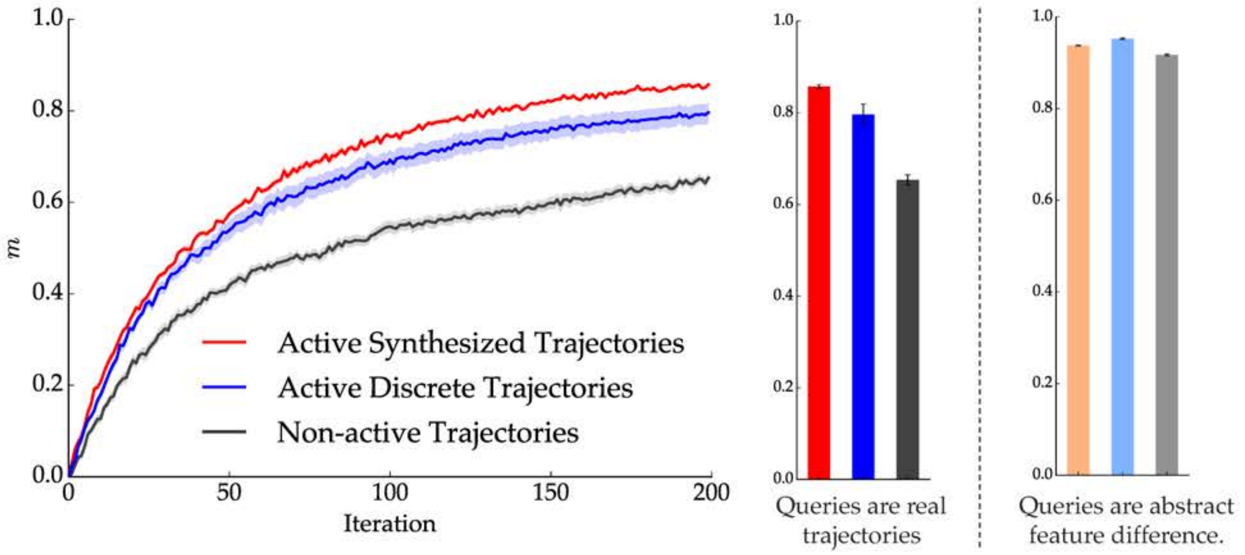


Figure 5.4: A comparison of our algorithm (red) with a non-active (black) and a non-synthesis (blue) version. Both deciding on queries actively and synthesizing them instead of relying on a predefined set improve performance by a statistically significant margin. On the right, we run an experiment to argue that these things are really important when synthesizing real queries for continuous high-dimensional and constrained systems: the lighter colors are for a version that only generates a query in feature difference space, without rendering it into actual (feasible) trajectories – the algorithm is allowed to make a fake query. There, being active and especially doing synthesis don't matter (synthesis even hurts, likely because it is a local search and a discrete set of queries can better cover the space).

The orange dot in all plots, as well as the orange dotted line show the ground truth of w_{true}. Figure 5.2 shows our algorithm clearly converges to close to the ground truth for all features.

We show the convergence of the distribution $p(w)$ in Figure 5.4. The dark red line shows the mean result of Algorithm 1. It outperforms, according to metric m, both the non-active condition (black), as well as the non-synthesis (discretized) condition (dark blue). The results are the average of 10 runs of the algorithms (since sampling is nondeterministc). The bar graph in the middle also shows the converged value of m after 200 queries.

In a thought experiment, we also investigated to what extent we see these effects because we are generating real trajectories for dynamical systems. To test this, we used the three algorithms to produce abstract queries that a simulation could answer, consisting of only a feature difference φ as opposed to two feasible real trajectories. The value of m after 200 iterations for these versions of the algorithms are shown in the bar graph with lighter colors. Because these approaches are unencumbered by the need for feasibility of the queries (or even producing actual concrete queries), they overall perform better than when the algorithms need to generate real queries. In this case, being active is still important, but the ability to perform synthesis is no longer useful compared to relying on a discrete set. Without needing to produce trajectories, a discrete set covers the much lower dimensional space of feature differences, and discrete search is not bottlenecked by local optima. This suggests that indeed, synthesis is important when we are dealing with learning reward functions for dynamical systems, requiring queries in a continuous, high-dimensional, and dynamically constrained space.

Overall, the results suggest that doing *active* query *synthesis* is important in reward learning applications of preference-based learning, with our results supporting both of our central hypotheses and demonstrating an improvement over prior methods which are either non-active or non-synthesis.

5.6 Usability Study

Our experiments so far supported the value of our contribution: active preference-based learning that synthesizes queries from scratch outperformed both non-active and non-synthesis learning. Next, we ran a usability study with human subjects. The purpose of this study is not to compare our algorithm once again with other methods. Instead, we wanted to test whether a person can interact with our algorithm to recover a reasonable reward function for dynamical systems with continuous state and action spaces.

Experimental Design

In the learning phase of the experiments, we jumpstart the weights w with a reference starting distribution, and only update that based on the data collected for each user. We ask for 10 query inputs to personalize the distribution of w for each user.

Unlike our simulation experiments, here we do not have access to the ground truth reward function. Still, we need to evaluate the learned reward function. We thus evaluate what the users think subjectively about the behavior produced on a 7 point Likert scale.

Manipulated Factors

In order to calibrate the scale, we need to compare the reward's rating with some baseline. We choose to *perturb* the learned reward for this calibration. Thus, we only manipulate one factor: *perturbation from the learned weights*. We ask users to compare trajectories obtained by optimizing for reward with three different values for the weights: i) $w^* = \mathbb{E}[w]$ providing the mode of the learned distribution from Algorithm 1, ii) w^2 a *large* perturbation of w^* which enables us to sanity check that the learned reward is better than a substantially different one, and iii) w^1 a *slight* perturbation of w^*, which is a harder baseline to beat and enables us to test that the learned reward is a local optimum.

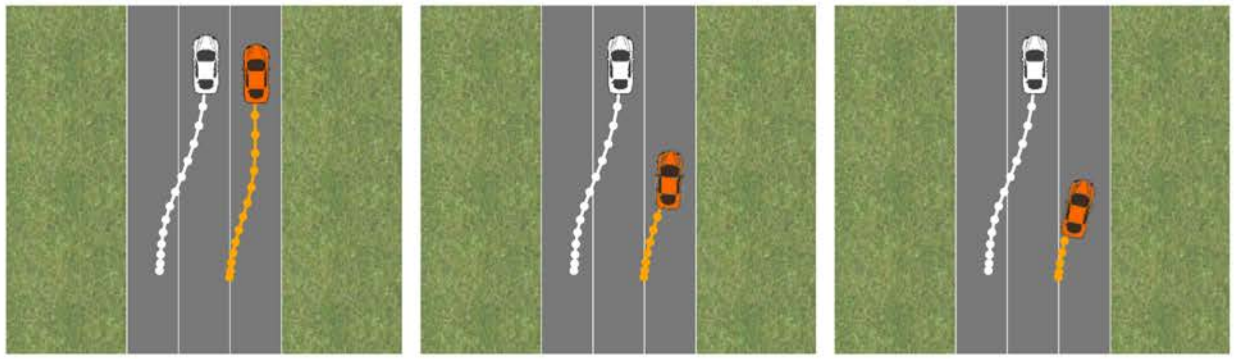


Figure 5.5: Car trajectories (orange) in a scenario obtained by optimizing different weights. On the left, we see the trajectories optimized based on the learned weight w^*, and the middle and right plots correspond to optimizing with w^1 (slightly perturbed w^*), and w^2 (highly perturbed w^*). The perturbations result in safe lane changes as well, but slower trajectories in this particular scenario.

The weights thus vary in distance or alignment with the learned weight, from identical to very different. We simply add a zero mean Gaussian noise to perturb w^*. For w^1, we add a Gaussian with standard deviation of $0.1 \times |w^*|$, and similarly with a standard deviation of $|w^*|$ for w^2. Our choice of perturbation is also affected by the test scenarios. In practice, we sample from possible w^1 and w^2 until the distance between the human trajectories is significant enough to create interesting scenarios that differentiate the weights (some simple scenarios have the car drive almost the same regardless of the reward function).

The 3 weights lead to 3 conditions. Figure 5.5 shows one of the test environments where the robot (white car) decides to change lanes, and the human driver can take any of the three depicted orange trajectories. Here, the trajectories correspond to the optimal human actions based on w^*, w^1, and w^2 from left to right. The learned reward function results in a longer and faster trajectory where the human driver changes lane. The perturbations also result in a safe lane change; however, they result in much slower trajectories.

Dependent Measures

For 10 predefined scenarios (initial state and actions of the other car $\mathcal{R}$), we generate the 3 trajectories corresponding to each condition. We ask users to rate each trajectory on a 7 point Likert scale in terms of how closely it matches their preference.

Hypothesis

H3. *Perturbation from the learned weights negatively impacts user rating: the learned weights outperform the perturbed weights, with the larger perturbation result in the smallest rating.*

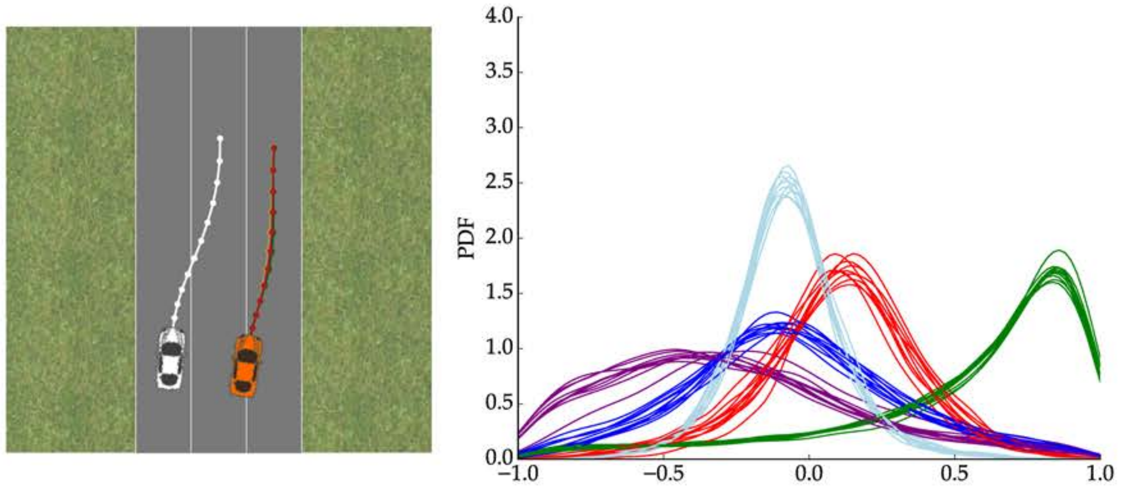


Figure 5.6: Here we show the human trajectories of all users for a specific scenario based on optimizing the learned reward function w^*. We plot the value of weights w^* for the five features for all users. This figure uses the same legend as Figure 5.3.

Subject Allocation

We recruited 10 participants (6 female, 4 male) in the age range of $26 - 65$. All owned drivers license with at least 8 years of driving experience. We ran our experiments using a 2D driving simulator (see Chapter 2.4), where we asked preference queries similar to Figure 5.6.

Analysis

We ran an ANOVA with perturbation amount (measured via our metric m relative to the learned weights) as a factor and user rating as a dependent measure. Supporting H3, we found a significant negative effect of perturbation (positive effect of similarity m) on the user rating ($F = 278.2$, $p < .0001$). This suggests the learned weights are useful in capturing desired driving behavior, and going away from them decreases performance. In Figure 5.7, we show the aggregate result of the $1 - 7$ rating across all scenarios and users. As shown in this figure, the highest rating in orange corresponds to the learned reward weights w^*, and our users preferred the slightly perturbed trajectories over the highly perturbed trajectories since the rating for w^1 is higher than w^2.

We found that, perhaps due to the fairly simple features we used, users tended to converge to similar weights, likely representing good driving in general. In Figure 5.6, we show the learned weight distribution for every feature is shown for all 10 users, and the resulting optimal trajectories for all our users. These have very high overlap, further suggesting that users converged to similar reward functions. In future work, we aim to look at a more expressive feature set that enables people to customize their

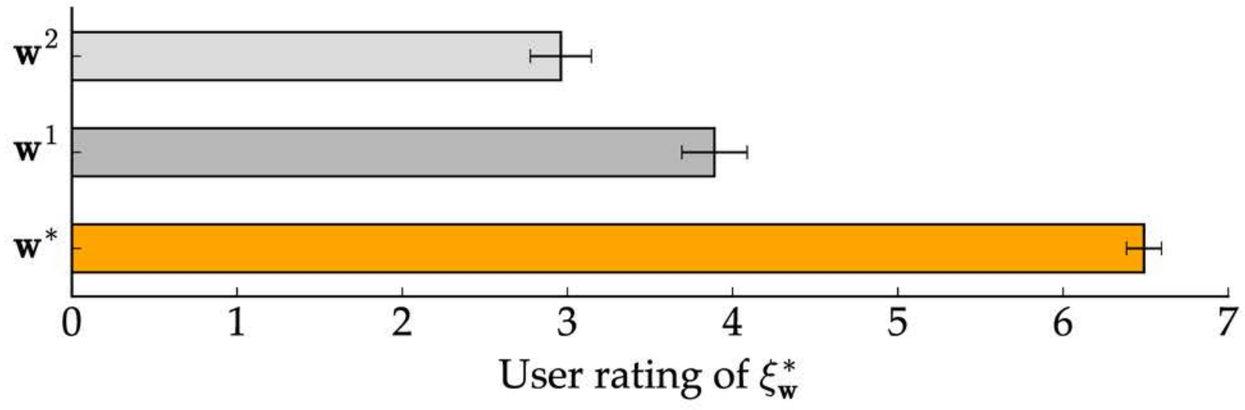


Figure 5.7: User ratings. The orange bar corresponds to the learned weights w^*, and the gray bars correspond to the perturbations of w^*. Our users preferred w^* over the slightly perturbed ones w^1, and preferred w^1 over the highly perturbed ones w^2.

car to their individual desired driving style, as opposed to getting the car to just do the one reasonable behavior possible (which is what might have happened in this case). Nonetheless, the result is encouraging because it shows that people were able to get to a behavior that they and other users liked.

5.7 Chapter Summary

We introduce an algorithm that can efficiently learn reward functions representing human preferences. While prior work relies on a discrete predefined set of queries or on passively learning the reward, our algorithm actively synthesizes preference queries: it optimizes in the continuous space of scenario parameters and trajectory pairs to identify a comparison to present to the human that will maximize the amount of expected volume that the answer will remove in the space of possible reward functions. We leverage continuous optimization to synthesize feasible queries, and assign a log-concave distribution over reward parameters to formulate an optimization-friendly criterion. We provide convergence guarantees for our algorithm, and compare it to a non-active and non-synthesis based approaches. Our results show that both active learning and synthesis are important. A user study suggests that the algorithm helps real end-users attain useful reward functions.

Limitations and Future Work. Our work is limited in many ways: we use a local optimization method for query synthesis which converges to local optima, we use an approximately rational model of the human when in fact real people might act differently, we assume that the robot has access to the right features, and we only consider one other agent in the world rather than multiple. Furthermore, our study shows that people can arrive at useful reward functions, such alignment with the reward leads to higher preference for the trajectory emerging from the learned weights, but it does not yet show that people can use this to personalize for their behavior – most users end up with very similar learned weights, and we believe this is because of the limitations of our features and our simulator. We plan to address these in the future.

Chapter 6

Falsification for Human-Robot Systems

In this chapter, we discuss a new approach for a rigorous analysis of human-robot systems that are based on *learned* models of human behavior. We focus on analyzing the robustness of a learning-based controller for human-robot systems where the human behavior deviates slightly from a learned model. As a motivating application, we consider autonomous controllers for vehicles that are based on the common and effective model predictive control (MPC) paradigm (see Chapter 2.1), in the receding horizon sense. In this setting, the autonomous controller computes its control actions for a horizon of N time units, while optimizing its reward function (which typically encodes safety and other objectives). For this purpose, it uses a dynamical model of both the system under control and its environment. Additionally, the environment contains human agents who are modeled as rational agents seeking to take actions that maximize their reward function. However, the human's *true* reward function is unknown to the autonomous controller. Therefore, the controller learns an approximation of the reward function from collected data, and uses this learned function in computing its actions. We refer to such a controller as an *interaction-aware controller* as described in the previous chapters (Chapter 3 and Chapter 4). Our goal is to find behaviors that exhibit "bugs", i.e., where the controller fails to meet its objectives. We refer to this goal as *falsification*.

Falsification of such human-robot systems faces some unique challenges. First, since the human model is learned from incomplete data, under specific assumptions and biases of the learning algorithm, and the true reward function is unknown, the learned model will be inaccurate. In such a setting, how can one bound the error in the learned model? Second, given an error bound, how can one systematically verify whether the controller will always achieve the desired objectives in an environment where the human agent(s) deviate from the learned model within that error bound?

Our approach addresses both challenges using optimization-based algorithmic methods. For the first, we provide a systematic and efficient methodology for estimating a bound between the values of the human's true reward function and the controller's estimate. Our methodology generates experiments to use in human subject trials during the learning process. For the second challenge, we present a novel encoding of the falsification problem from an optimization problem involving quantifiers over reward functions to a more straightforward quantifier-free optimization problem. Our contributions in this chapter are as follows:

- A formalization of the problem of falsifying controllers for human-robot systems with learned human models (Section 10.3);
- An encoding of the above problem into a more efficiently-solvable optimization problem (Section 10.3);
- An efficient (almost linear time) optimization-driven approach to estimating an error bound, δ, between the true human reward function and the controller's estimate (Section 6.3), and
- An experimental evaluation of our approach for autonomous vehicles using our car simulator (Section 7.4).

6.1 Running Example

We focus on falsification for human-robot systems, where similar to previous chapters $\mathcal{R}$ interacts with human agent $\mathcal{H}$ in a shared environment. Our goal is to verify if any of the actions of the human within a bound of a learned human model can possibly cause a violation of the desired specification, such as a collision between the two vehicles. For instance, consider an autonomous car (white car) driving on a road shared with a human-driven vehicle (orange car) (Figure 6.1). The autonomous car would normally optimize for reaching its destination, which in this example is to safely change lanes. The white car would choose its actions based on considering its interaction with the human-driven vehicle, and using a learned model of the orange car. Here, the white car decides to start the lane change maneuver because it has learned a model of the human that suggests the orange car would also optimize for her own safety, and thus will follow trajectory A.

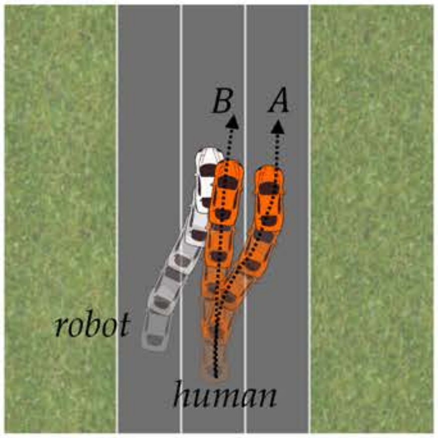


Figure 6.1: Based on a learned model of the human, the autonomous car (white) decides to change lane assuming the human-driven car (orange) follows trajectory A. In practice the orange car can possibly follow a perturbed model resulting in trajectory B.

Similar to all learned models, this model of the human (orange car) is learned under a set of assumptions from a limited training dataset. So the truth is *the actual human can possibly take actions that* *slightly differ from the robot's learned human model*. For example, the orange car might decide to choose a slightly perturbed trajectory such as trajectory B. As it is clear in Figure 6.1, such a perturbation can jeopardize satisfaction of the desired properties such as collision avoidance.

In this chapter, we show that these perturbations in models of learned human drivers exist, and we provide an efficient algorithm (almost linear) to learn a distribution for such possible perturbations using a query-based method. Further, we provide a method for finding the falsifying actions of the human agent (e.g. actions that result in trajectory B) within the perturbation bound.

6.2 Falsification in Interaction-Aware Control

Assuming a similar formalism as in Chapter 2.1, where the human-robot system follows the dynamics model:

$$x^{t+1} = f(x^t, u^t_{\mathcal{R}}, u^t_{\mathcal{H}}) \tag{6.1}$$

Let us remind you of the interaction-aware controller discussed in the previous chapters.

Problem 2 (Interaction-Aware Planner)**.** *Given a system as in equation* (6.1)*, an initial state $x^0 \in X$, a finite horizon N, and a robot reward function $R_{\mathcal{R}}$, compute:*

$$\mathbf{u}^*_{\mathcal{R}} = \arg\max_{\mathbf{u}_{\mathcal{R}}} R_{\mathcal{R}}(x^0, \mathbf{u}_{\mathcal{R}}, \mathbf{u}^*_{\mathcal{H}}(x^0, \mathbf{u}_{\mathcal{R}})) \tag{6.2}$$

Remark 2. *We emphasize that Problem 2 is only one method for computing the actions of the robot that we have introduced for concreteness. Our falsification approach in this chapter may be combined with any other technique for computing control actions over a finite horizon.*

Our falsification problem is to find out whether there exists a sequence of human actions that can lead to unsafe scenarios. Let $R_{\mathcal{H}}$, i.e., the sum of the learned reward functions of $\mathcal{H}$ over horizon N represent our human model. Our ultimate goal is to verify that the robot is resilient to model inaccuracies of $R_{\mathcal{H}}$ within a specific bound. Such inaccuracies usually exist due to two main factors: i) particular assumptions on the learning algorithm, ii) insufficiency of collected training data.

We have assumed humans are optimizers of a particular type of reward function $R_{\mathcal{H}}$, and such reward functions can be learned through various learning techniques (e.g., IRL as in Chapter 2.2). However, the human can follow a different reward function, which we call the true reward function $R^{\dagger}_{\mathcal{H}}$. $R^{\dagger}_{\mathcal{H}}$ can possibly have a different structure from $R_{\mathcal{H}}$, i.e., it might not even be a linear combination of a set of hand-coded features described in Chapter 2.2, or we (the designers) might have missed specifying a particular feature as part of ϕ. So the learning methods can possibly never converge to the true reward function $R^{\dagger}_{\mathcal{H}}$. Further, we might decide to learn $R_{\mathcal{H}}$ from a collection of human demonstrations, but in practice, the robot interacts with a specific human agent whose true reward function is $R^{\dagger}_{\mathcal{H}}$, which can be different from $R_{\mathcal{H}}$ due to variations amongst humans. So when it comes to interacting with different humans, the learned $R_{\mathcal{H}}$ might not perform as expected.

We are motivated to verify if Problem 2 can be solved in scenarios where the true human reward function $R^{\dagger}_{\mathcal{H}}$, unknown to us, deviates a bit from the learned function $R_{\mathcal{H}}$. We let δ be the bound between the distance of these two reward functions:

$$\forall(x^0, \mathbf{u}_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}),\ |R_{\mathcal{H}}(x^0, \mathbf{u}_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}) - R^{\dagger}_{\mathcal{H}}(x^0, \mathbf{u}_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}})| < \delta. \tag{6.3}$$

For brevity, we will slightly abuse notation and write this inequality as $|R_{\mathcal{H}} - R^{\dagger}_{\mathcal{H}}| < \delta$.

Problem 3 (Falsification of Interaction-Aware Controllers)**.** *For a parameter δ describing the bound between the human's true and learned reward functions ($R^{\dagger}_{\mathcal{H}}$ and $R_{\mathcal{H}}$), compute a falsifying input $\tilde{\mathbf{u}}_{\mathcal{H}}$:*

$$\begin{aligned} \tilde{\mathbf{u}}_{\mathcal{H}} = \arg\min_{\mathbf{u}_{\mathcal{H}}} \quad & R_{\mathcal{R}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}) \\ \textit{subject to} \quad & \exists R^{\dagger}_{\mathcal{H}} : \quad \mathbf{u}_{\mathcal{H}} = \arg\max_{\bar{\mathbf{u}}_{\mathcal{H}}} R^{\dagger}_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \bar{\mathbf{u}}_{\mathcal{H}}), \\ & |R_{\mathcal{H}} - R^{\dagger}_{\mathcal{H}}| < \delta. \end{aligned} \tag{6.4}$$

Here, $\mathbf{u}^*_{\mathcal{R}}$ is computed by solving the nested optimization in Problem 2, where $\mathbf{u}^*_{\mathcal{H}}(x^0, \mathbf{u}_{\mathcal{R}})$ is the true optimizer of the learned reward function $R_{\mathcal{H}}$.

Problem 3 looks for the worst case input provided by the human that violates the specification by minimizing the reward function of the robot $R_{\mathcal{R}}$, while still remaining in the set of possible human models, i.e., this input can optimize an alternative reward function $R^{\dagger}_{\mathcal{H}}$ that lies within δ distance of the learned reward function $R_{\mathcal{H}}$. We now show how we optimally solve for this falsifying input.

Theorem 2. *The optimization in Problem 3 is equivalent to solving the following:*

$$\begin{aligned} \tilde{\mathbf{u}}_{\mathcal{H}} = \arg\min_{\mathbf{u}_{\mathcal{H}}} \quad & R_{\mathcal{R}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}) \\ \textit{subject to} \quad & R_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}) > R^*_{\mathcal{H}} - 2\delta, \end{aligned} \tag{6.5}$$

where $R^*_{\mathcal{H}} = R_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}^*_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}))$ is a constant evaluated at $\mathbf{u}^*_{\mathcal{R}}$ and $\mathbf{u}^*_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}})$. Here, $\mathbf{u}^*_{\mathcal{R}}$ is computed by solving Problem 2, which depends on the function representing the optimal actions of the human $\mathbf{u}^*_{\mathcal{H}}(x^0, \mathbf{u}_{\mathcal{R}})$. Using this $\mathbf{u}^*_{\mathcal{R}}$, we then find $\mathbf{u}^*_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}) = \arg\max_{\mathbf{u}_{\mathcal{H}}} R_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}})$.

Proof. To show the equivalence between equation (6.4) and (6.5), we need to consider the equivalence of the constraints.

First, we show the constraint $\exists R^{\dagger}_{\mathcal{H}} : \mathbf{u}_{\mathcal{H}} = \arg\max_{\bar{\mathbf{u}}_{\mathcal{H}}} R^{\dagger}_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \bar{\mathbf{u}}_{\mathcal{H}})$ will imply the constraint in equation (6.5).

Since $\exists R^{\dagger}_{\mathcal{H}} : \mathbf{u}_{\mathcal{H}} = \arg\max_{\bar{\mathbf{u}}_{\mathcal{H}}} R^{\dagger}_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \bar{\mathbf{u}}_{\mathcal{H}})$, we conclude that $R^{\dagger}_{\mathcal{H}}$ evaluated at the optimum $\mathbf{u}_{\mathcal{H}}$ is greater than $R^{\dagger}_{\mathcal{H}}$ evaluated at any other sequence of human actions such as $\mathbf{u}^*_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}})$ (to simplify the notation, we simply let $\mathbf{u}^*_{\mathcal{H}} = \mathbf{u}^*_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}})$):

$$R^{\dagger}_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}) \geq R^{\dagger}_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}^*_{\mathcal{H}}). \tag{6.6}$$

Also, because of the bound δ, we can infer the following:

$$R_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}) > R^{\dagger}_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}) - \delta, \tag{6.7}$$

$$R^{\dagger}_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}^*_{\mathcal{H}}) > R_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}^*_{\mathcal{H}}) - \delta. \tag{6.8}$$

Equations (6.6) and (6.7) together will result in:

$$R_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}) > R^{\dagger}_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}^*_{\mathcal{H}}) - \delta. \tag{6.9}$$

Finally, equations (6.8) and (6.9) provide:

$$R_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}) > R_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}^*_{\mathcal{H}}) - 2\delta, \tag{6.10}$$

which is the same as the constraint in equation (6.5):

$$R_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}) > R^*_{\mathcal{H}} - 2\delta. \tag{6.11}$$

Now, to complete the proof, we need to show that this constraint will result in the constraint in equation (6.4), i.e., $\exists R^{\dagger}_{\mathcal{H}} : \mathbf{u}_{\mathcal{H}} = \arg\max_{\bar{\mathbf{u}}_{\mathcal{H}}} R^{\dagger}_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \bar{\mathbf{u}}_{\mathcal{H}})$. Our approach is to construct the following reward function $R^{\dagger}_{\mathcal{H}}$ so that at the optimal $\mathbf{u}_{\mathcal{H}}$ (i.e. $\mathbf{u}_{\mathcal{H}} = \arg\max_{\bar{\mathbf{u}}_{\mathcal{H}}} R^{\dagger}_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \bar{\mathbf{u}}_{\mathcal{H}})$), we will have:

$$R^{\dagger}_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}) = R_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}) + \delta, \tag{6.12}$$

and for any other human actions such as $\mathbf{u}'_{\mathcal{H}} \neq \mathbf{u}_{\mathcal{H}}$, this function is equal to:

$$R^{\dagger}_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}'_{\mathcal{H}}) = R_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}'_{\mathcal{H}}) - \delta. \tag{6.13}$$

Then, reordering the inequality in equation (6.4), and using equation (6.12) will result in:

$$\begin{aligned} R_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}) + \delta &> R^*_{\mathcal{H}} - \delta \\ \implies R^{\dagger}_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}) &> R^*_{\mathcal{H}} - \delta \end{aligned} \tag{6.14}$$

Also, by optimality of $R^*_{\mathcal{H}}$ and equation (6.13), we know for any $\mathbf{u}'_{\mathcal{H}} \neq \mathbf{u}_{\mathcal{H}}$:

$$R^{\dagger}_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}'_{\mathcal{H}}) = R_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}'_{\mathcal{H}}) - \delta \leq R^*_{\mathcal{H}} - \delta \tag{6.15}$$

Using the two results in equations (6.14) and (6.15), we conclude:

$$\begin{aligned} & R^{\dagger}_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}_{\mathcal{H}}) \geq R^{\dagger}_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \mathbf{u}'_{\mathcal{H}}) \\ \implies & \mathbf{u}_{\mathcal{H}} = \arg\max_{\bar{\mathbf{u}}_{\mathcal{H}}} R^{\dagger}_{\mathcal{H}}(x^0, \mathbf{u}^*_{\mathcal{R}}, \bar{\mathbf{u}}_{\mathcal{H}}). \end{aligned} \tag{6.16}$$

So for the $R^{\dagger}_{\mathcal{H}}$ we have constructed, the constraint in equation (6.4) holds. Since the two constraints in (6.4) and (6.5) are equivalent, Theorem 2 holds true. □

Through Theorem 2 we have shown we can solve equation (6.5), a much simpler optimization problem, instead of solving Problem 3 to find the falsifying actions of $\mathcal{H}$ for a fixed given bound δ.

6.3 Learning the Error Bound δ

In this section, we provide an efficient algorithm for computing a reasonable error bound δ, as defined in equation (6.3), which is used in solving Problem 3. We will first learn a distribution for the error:

$$\Delta(\xi) = R_{\mathcal{H}}(\xi) - R^{\dagger}_{\mathcal{H}}(\xi), \tag{6.17}$$

and then we upperbound $\Delta(\xi)$ by a constant factor δ using the distribution's quantiles. In the rest of this chapter, we use Δ instead of $\Delta(\xi)$ for brevity. So assuming that $\Delta \sim \mathcal{D}$ is a random variable derived from a distribution $\mathcal{D}$, our goal is to *efficiently* learn $\mathcal{D}$. We propose using a query based method that efficiently (almost linear time in the number of queries, i.e., $O(n \log n)$) finds the maximum likelihood estimate for $\mathcal{D}$.

Suppose, we query the human $\mathcal{H}$ whether she prefers trajectory ξ_A or ξ_B. Then, if the user decides ξ_A is preferred over ξ_B, we can conclude that the true reward function of trajectory ξ_A is higher than the true reward function of trajectory ξ_B, i.e.,

$$R^{\dagger}_{\mathcal{H}}(\xi_A) > R^{\dagger}_{\mathcal{H}}(\xi_B). \tag{6.18}$$

Note, since the trajectory is fully determined by $(x^0, \mathbf{u}_{\mathcal{H}}, \mathbf{u}_{\mathcal{R}})$, we alternatively can write $R_{\mathcal{H}}$ as a function of ξ, and using the error bound results as in:

$$R_{\mathcal{H}}(\xi_A) + \Delta_A > R_{\mathcal{H}}(\xi_B) + \Delta_B. \tag{6.19}$$

We also assume $\Delta_A, \Delta_B \sim \mathcal{D}$ are independent random variables, and so is their difference: $\chi = \Delta_A - \Delta_B$. We can rewrite this as:

$$\chi = \Delta_A - \Delta_B > R_{\mathcal{H}}(\xi_B) - R_{\mathcal{H}}(\xi_A), \tag{6.20}$$

where $R_{\mathcal{H}}$ is the learned reward function. Here, we simply let $R = R_{\mathcal{H}}(\xi_B) - R_{\mathcal{H}}(\xi_A)$.

Therefore, assuming humans are not noisy and always respond with a particular preference, every query from the human for two fixed set of trajectories ξ_A and ξ_B will results in either $\chi > R$ or $\chi < R$. Our goal in this section is to find the maximum likelihood distribution of $\chi \sim \bar{\mathcal{D}}$ by asking queries from the human for a number of times, and recovering the distribution of the true bound $\Delta \sim \mathcal{D}$.

Remark 3. *Here the human responses to the queries only provides an estimate of human's true reward function. These comparisons allow us to learn human's preferences, which can possibly be different from the true reward function she would optimize in practice. However, for the purpose of falsification in this chapter, the preference reward function is a reasonable estimate of $R^{\dagger}_{\mathcal{H}}$.*

Learning the Error Distribution of $\chi \sim \bar{\mathcal{D}}$

Now, we propose our algorithm to find the distribution $\bar{\mathcal{D}}$ based on any M given comparison queries $\chi_i > R_i$ or $\chi_i < R_i$. Suppose that $\chi_1, \dots, \chi_M$ are i.i.d. and drawn from some unknown distribution $\bar{\mathcal{D}}$. Given $R_1, \dots, R_M \in \mathbb{R}$, and the corresponding signs $s_1, \dots, s_M \in \{-1, +1\}$, representing the observations: $s_i(\chi_i - R_i) > 0$, i.e., $\chi_i > R_i$ for $s_i = +1$, and $\chi_i < R_i$ for $s_i = -1$, the goal is to find the maximum likelihood estimate of the distribution $\bar{\mathcal{D}}$.
Without loss of generality, we assume that R_i's are sorted, i.e. $R_1 < R_2 < \cdots < R_M$. Now let us define:

$$p_i = \mathbb{P}_{\chi \sim \bar{\mathcal{D}}}[\chi < R_i]. \tag{6.21}$$

Then, obviously since R_i's are sorted, p_i's would be sorted as well: $p_1 \le p_2 \le \cdots \le p_M$. The likelihood of our observations can be expressed in terms of p_i's and note that any increasing sequence of p_i's that lie in $[0,1]$ is valid. Our problem of estimating the distribution reduces to finding the increasing sequence of $p_1 \le \cdots \le p_M \in [0,1]$ that maximizes the log-likelihood of our observations. We now propose our method that finds p_i's by solving the following constrained optimization:

$$\begin{aligned} \max_{p_1,\dots,p_M} \quad & \log\Big(\prod_{i:s_i=+1}(1-p_i)\prod_{i:s_i=-1}p_i\Big) \\ \text{subject to} \quad & 0 \le p_1 \le p_2 \le \cdots \le p_M \le 1. \end{aligned} \tag{6.22}$$

Suppose $p_1^* \le \cdots \le p_M^*$ are the optimizers of equation (6.22). We derive some necessary conditions that $p_1^*, \dots, p_M^*$ need to satisfy, and conclude that these uniquely determine p_i^*.
We can partition p_i^* into contiguous maximal subsets of equal values. For example, $p_a^* = p_{a+1}^* = \cdots = p_b^*$ represents such a maximal contiguous subset, i.e., $p_{a-1}^* < p_a^*$ and $p_b^* < p_{b+1}^*$. We let q represent the common value of this contiguous subset.

Lemma 1. *In the above context, the value of q must be equal to:*

$$\frac{n_-^{[a,b]}}{n_-^{[a,b]} + n_+^{[a,b]}}, \tag{6.23}$$

where $n_+^{[a,b]} = |\{i : s_i = +1 \land i \in [a,b]\}|$ and $n_-^{[a,b]} = |\{i : s_i = -1 \land i \in [a,b]\}|$.

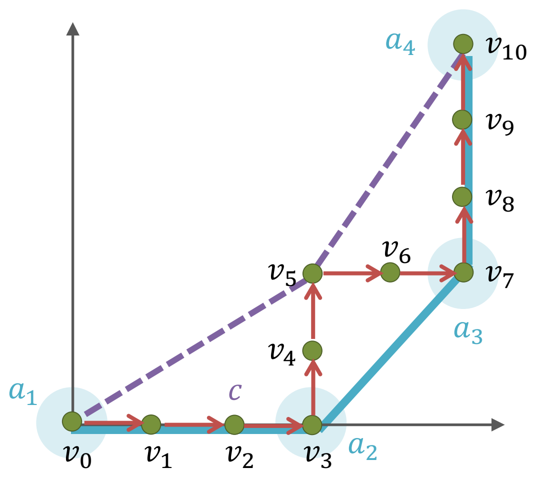


Figure 6.2: Visualization tool representing 10 queries from the human. We move right for positive queries ($s_i = +1$), and move up for negative ones ($s_i = -1$) over $\mathbb{Z}^2$. The convex hull of this plot provides our maximum likelihood estimate of the error distribution.

Proof. If we perturb q by a small amount in any direction, we would still be satisfying the constraint of monotonicity of p_i's. Further, the derivative of the log-likelihood with respect to q must be 0 at the optimal values $q = p_a^*$. This means that:

$$- \sum_{i:s_i=+1 \wedge i\in[a,b]} \frac{1}{1-q} + \sum_{i:s_i=-1 \wedge i\in[a,b]} \frac{1}{q} = 0, \tag{6.24}$$

which results in $q = \frac{n_-^{[a,b]}}{n_-^{[a,b]}+n_+^{[a,b]}}$. □

We visualize the values $n_+^{[a,b]}$ and $n_-^{[a,b]}$ using the graph in Figure 6.2: consider a path drawn on $\mathbb{Z}^2$, and let the path start from the origin $v_0 = (0,0)$. For each i in the sorted order of χ_i, if $s_i = -1$, move up one unit, and if $s_i = +1$, move to the right one unit, and call the new point v_i. Note, at the beginning runs, we would move to the right more frequently, as the sorted data makes $s_i = +1$ results more likely. Figure 6.2 shows an example of such a run. Then, between every two points v_{a-1} and v_b on the plot the difference in the y-direction is $\Delta y = n_-^{[a,b]}$, and similarly the difference in the x-direction is $\Delta x = n_+^{[a,b]}$. Thus, the optimal value of the log-likelihood for each maximal contiguous subset, e.g., $[a,b]$ is $q = \frac{\Delta y}{\Delta x+\Delta y}$, where $(\Delta x, \Delta y) = v_b - v_{a-1}$. We can think of $\frac{\Delta y}{\Delta x+\Delta y}$ as a type of slope, which we call L-slope. Note, the L-slope is an increasing function of the real slope (i.e., $\frac{\Delta y}{\Delta x}$). While the real slope lies in $[0,\infty]$, the L-slope has a monotonic relation to the real slope and maps it to $\frac{\Delta y}{\Delta y+\Delta x} \in [0,1]$.

So far, we have shown that if we know the maximal contiguous partitions of p_i's, the optimal values p_i^*'s are uniquely determined by the L-slopes. Thus, the main remaining question is how to find such maximal contiguous segments.

First, let's assume there are k maximal contiguous segments, and we represent them with $a_1, \dots, a_k$, where $a_1 = v_0$, and $a_k = v_M$, and all the other a_j's for $j \in \{1, \dots, k\}$ are equal to vertices v_i that partition the p_i's into maximal contiguous pieces. We use l_i to denote the L-slope between each a_i and a_{i+1}. Note that because of ordering of $p_1 \leq p_2 \leq \dots \leq p_M$, these L-slopes are also in an increasing order $l_1 \leq l_2 \leq \dots \leq l_{k-1}$. Therefore, these L-slopes create a convex graph, and as a result a_j's lie in a convex position.

Lemma 2. *In the above context, a_j's are precisely the set of vertices of the convex hull of the constructed graph (e.g. Figure 6.2).*

Proof. We prove this by contradiction. Assume that a_j's are not the convex hull of the graph. For example in Figure 6.2, this could happen with $a_1 = v_0, a_2 = v_5, a_3 = v_{10}$.

Because a_j's are not the convex hull, there exists an index i and a point on the graph c such that c lies under the line segment between a_i and a_{i+1} (e.g., let $c = v_2$ in the segment connecting $a_1 = v_0$ to $a_2 = v_5$).

Consider the probabilities corresponding to the line segment from a_i to a_{i+1}. These would be $p_r, p_{r+1}, \dots, p_s$, if $a_i = v_{r-1}$ and $a_{i+1} = v_s$; note that we have already shown all of them must be equal to the L-slope between a_i and a_{i+1}, i.e., $p_r = \dots = p_s = l_i$. We partition these probabilities into two contiguous segments $p_r, \dots, p_j$ and $p_{j+1}, \dots, p_s$, where j is the index for which $c = v_j$.

We perturb the first segment, $p_r, \dots, p_j$ by a small amount ϵ in the *negative* direction, and perturb the second segment, $p_{j+1}, \dots, p_s$ by ϵ in the *positive* direction. Since we decrease the first segment and increase the second, this does not violate our ordering constraint $p_1 \leq \dots \leq p_M$. Then, by a similar argument as in the proof of Lemma 1, we can conclude that this small perturbation increases the log-likelihood; in other words, the derivative of the log-likelihood with respect to the first segment is negative and the derivative with respect to the second segment is positive. We conclude that when a_j's form the optimal partition, no such point c can be found, and therefore the a_j's form the convex hull of the graph.

□

Theorem 3. *The optimization problem in equation (6.22) can be solved in $O(M \log(M))$ time.*

Proof. Sorting the R_i's takes $O(M \log(M))$ time. As we have shown in Lemma 2, the optimal partition is determined by the convex hull of the constructed graph. The graph can be constructed in linear time, and the convex hull can be computed in linear time as well. Finally, the values p_i's are determined by the L-slopes, each of which can be computed in $O(1)$ time. Thus the bottleneck is in the initial sort, and the whole computation takes $O(M \log(M))$ time. □

This method enables us to efficiently recover an estimate of the CDF of distribution $\bar{\mathcal{D}}$.

Remark 4. *In practice, the CDF might be well-approximated by a Gaussian. Finding the parameters of the Gaussian can again be formulated as a similar optimization in equation* (6.22)*, where the objective is still the log-likelihood of the observed data, but the variables are replaced by the parameters of the Gaussian.*

Recovering the Error Distribution of $\Delta \sim \mathcal{D}$

To recover the distribution of the error $\Delta \sim \mathcal{D}$, we need to make a further assumption that the distribution $\bar{\mathcal{D}}$ is symmetric. Assuming we successfully find the distribution of $\chi \sim \bar{\mathcal{D}}$, we can use moment generating functions $M_\chi(t) = \mathbb{E}[e^{t\chi}]$ to recover $\mathcal{D}$. Given

$$\chi = \Delta_A - \Delta_B, \quad \text{where } \chi \sim \bar{\mathcal{D}} \text{ and } \Delta_A, \Delta_B \sim \mathcal{D}: \tag{6.25}$$

$$M_\chi(t) = M_{\Delta_A - \Delta_B}(t) = M_{\Delta_A}(t) \cdot M_{-\Delta_B}(t) = M_{\Delta_A}(t) \cdot M_{\Delta_B}(-t). \tag{6.26}$$

Since Δ_A and Δ_B are drawn from the same distribution their moment generating function is the same. Further, Δ is drawn from a symmetric distribution, i.e., $M_\Delta(t) = M_\Delta(-t)$, so:

$$M_\chi(t) = M_\Delta(t)^2. \tag{6.27}$$

Assuming we have computed $\chi \sim \bar{\mathcal{D}}$, we now can use this relation to find the distribution of $\Delta \sim \mathcal{D}$.

Remark 5. *If we are only interested in solving equation* (6.5)*, we do not need to extract $\mathcal{D}$ from $\bar{\mathcal{D}}$. We can simply replace the 2δ bound by a number drawn from the quantiles of the distribution $\bar{\mathcal{D}}$. The reason for this is that even if we replace the assumption $|R^\dagger_\mathcal{H} - R_\mathcal{H}| \leq \delta$ by the weaker assumption:*

$$\forall \xi_A, \xi_B : |(R^\dagger_\mathcal{H}(\xi_A) - R^\dagger_\mathcal{H}(\xi_B)) - (R_\mathcal{H}(\xi_A) - R_\mathcal{H}(\xi_B))| \leq 2\delta, \tag{6.28}$$

Theorem 2 would still hold (with essentially the same proof). Note that the above quantity is simply: $|\chi| = |\Delta_A - \Delta_B|$, whose distribution is given by $\bar{\mathcal{D}}$.

Remark 6. *We select 2δ from the quantiles of $\bar{\mathcal{D}}$ in such a way that:* $\Pr_{\chi \sim \bar{\mathcal{D}}}(|\chi| > 2\delta) < \epsilon$*, where $\epsilon \in [0, 1]$ is the tolerated error ($1 - \epsilon$ is the confidence level).*

6.4 Case Studies

In this section, we demonstrate our falsification algorithm for two autonomous driving scenarios. We also efficiently learn an error distribution for human's reward function based on the proposed query-based method. We solved the constrained optimizations

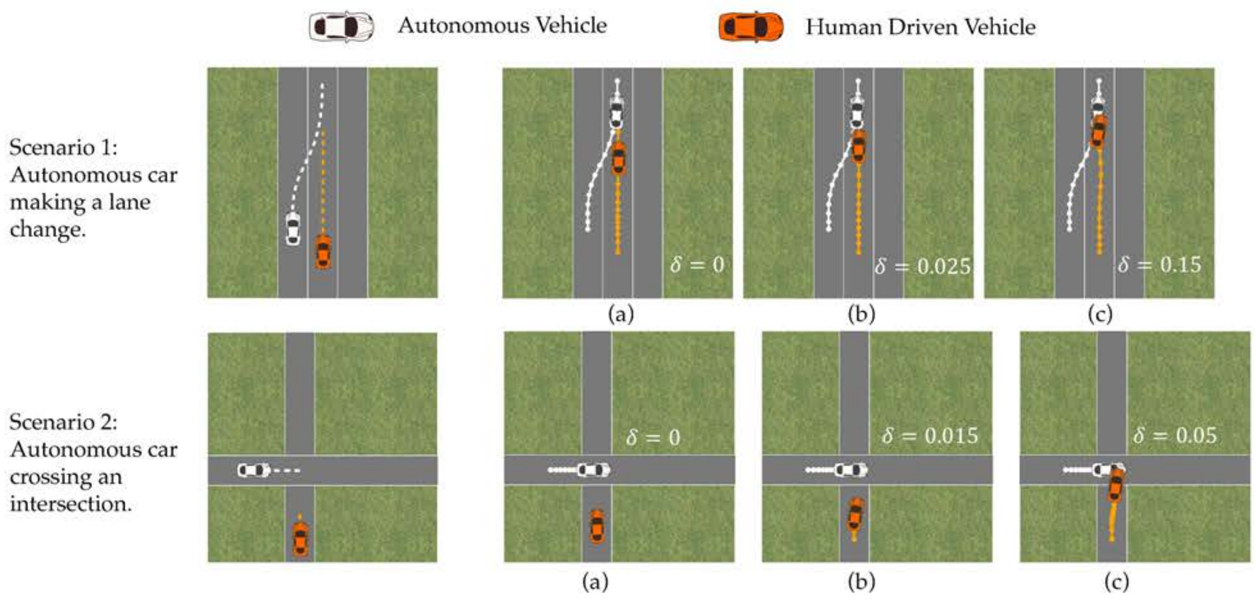


Figure 6.3: Falsification in driving scenarios. Each row represents a scenario, where the cars start from the initial positions shown on the left, then (a) shows the optimal actions of human based on $R_{\mathcal{H}}$. In (b) and (c), we illustrate the optimal actions of the vehicles if $\mathcal{H}$ follows a perturbed reward function. The perturbation in (b) is not enough for violation of the safety property; however, in (c) with enough perturbation the two vehicles collide.

using the package Ipopt [177]. Our falsification algorithm is solved in less than a second (0.25s on average) when run on a 2.3 GHz Intel Core i7 processor.

Both vehicles follow the same dynamics model introduced in Chapter 2.4. We illustrate the two case studies in Figure 6.3. The top row corresponds to the autonomous car (white car) changing lane (the running example), and the bottom row corresponds to the autonomous car crossing an intersection. For both cases, the reward function includes collision avoidance as a safety measure as part of its features (see Chapter 2.2).

Scenario 1: Falsifying Collision Avoidance in a Lane Change. In this scenario, the autonomous car (white car in Figure 6.3) optimizes for changing lanes, while avoiding collisions. Assuming that $\mathcal{H}$ fully follows the learned reward function $R_{\mathcal{H}}$, no collisions occur and the autonomous car safely changes lanes (Figure 6.3(a)). However, the orange car might not exactly follow $R_{\mathcal{H}}$, so for a perturbed $R_{\mathcal{H}}$ by $\delta = 0.025$, the two cars take a riskier maneuver (Figure 6.3(b)), and finally for a sufficiently large $\delta = 0.15$, we find a falsifying set of actions for the human (Figure 6.3(c)), which clearly ends in a collision. We also demonstrate $R_{\mathcal{R}}$ with respect to δ in Figure 6.4(a). Based on the value of the reward function, where $R_{\mathcal{R}} < 0$ corresponds to experiencing collisions. This criteria is similar to the robustness function introduced in Chapter 2.3, and we will discuss that in more details in the next part of this dissertation. The safety property of collision avoidance will be violated for any model of human $R^{\dagger}_{\mathcal{H}}$ that lies in a range that is at least $\delta = 0.11$ away

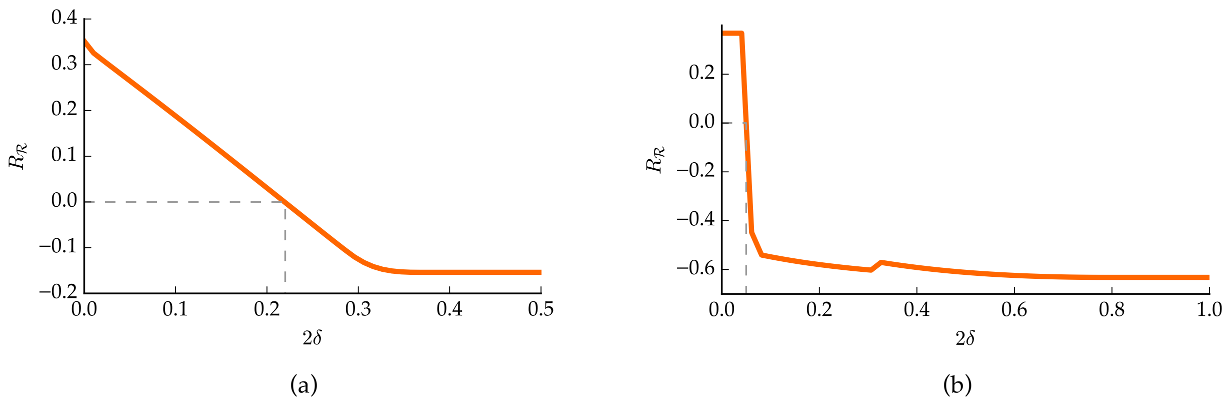


Figure 6.4: Robot Reward function of scenario 1 shown in (a), and scenario 2 in (b) with respect to δ. The violation threshold in (a) is $\delta = 0.11$, and in (b) is $\delta = 0.025$.

from $R_{\mathcal{H}}$.

Scenario 2: Falsifying Collision Avoidance in an Intersection. In this scenario, our goal is to find the falsifying actions for the human-driven car (orange car) when the autonomous car (white car) plans to cross an intersection as shown in Figure 6.3. Similar to the previous scenario, following $R_{\mathcal{H}}$, results in a collision-free outcome (Figure 6.3(b)). However, with a slightly perturbed reward function of the human with $\delta = 0.015$, the orange car gets closer to the autonomous car, and for a slightly higher $\delta = 0.05$ the two cars collide as shown in Figure 6.3(c). Similar to the previous case, we have shown the reward function $R_{\mathcal{R}}$ with respect to δ in Figure 6.4, which shows the violation threshold is $\delta = 0.025$. We hypothesize that the bumps in the reward function (Figure 6.4(b)) in this scenario are due to the system falling into local optima in solving the falsification problem.

Learning Human's Error. Using our query-based method, we learn a distribution for Δ, and find an appropriate bound δ. As shown in Figure 6.5(a), we have done 100 queries from $\mathcal{H}$ asking the human to compare two provided trajectories. The blue line in Figure 6.5(a) represents the walk over $\mathbb{Z}^2$ based on the sorted queries, where each horizontal line represents a positive query ($s_i = +1$), and each vertical line represents a negative query ($s_i = -1$) (similar to Figure 6.2). The orange line is the convex hull of the graph, which facilitates estimation of the distribution of Δ. In Figure 6.5(b), we have found the maximum likelihood estimate of the CDF of the distribution from the convex hull in Figure 6.5(a). Similarly, we can fit a Gaussian instead as shown in purple. Using the estimated CDF, we choose $\delta = 0.1$ for confidence level of $\epsilon = 0.74$ as discussed in Remark 6. Note, with this confidence level and bound $\delta = 0.1$, we would have violated the specification for the second case study (crossing an intersection), while still satisfying collision avoidance in the first scenario (lane change) based on the thresholds found in Figure 6.4.

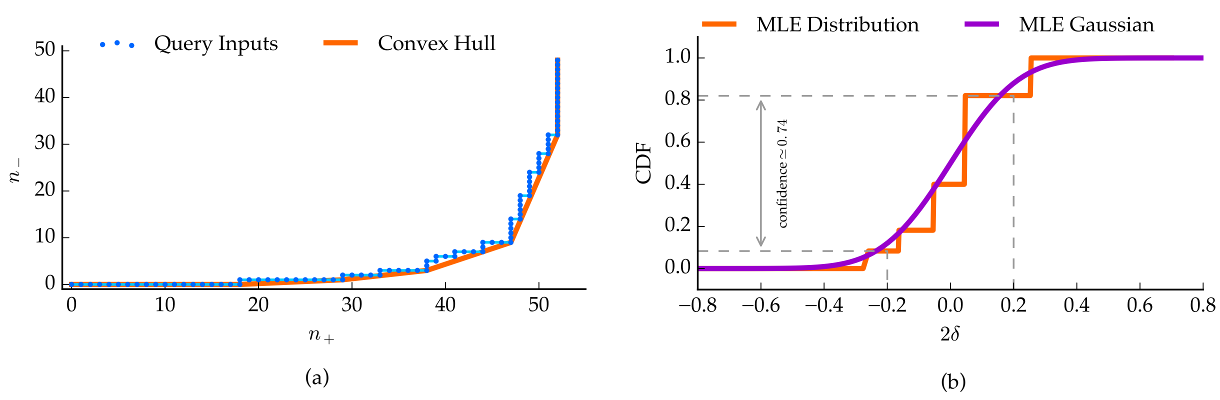


Figure 6.5: Learning Human's Error Distribution. In (a) we visualize the comparison queries from $\mathcal{H}$, and find the convex hull of the graph allowing us to find the CDF of the error distribution as shown in orange in (b). Using different quantiles of the CDF, we set an appropriate bound for δ.

6.5 Chapter Summary

As human-robot systems make their ways into our every day life, safety has become a core concern of the learning algorithms used by such systems. The correctness of controllers for human-robot systems rely on the accuracy of models of human behavior. In this chapter, we described a systematic methodology for analyzing the robustness of learning-based control of human-robot systems. We focus on the setting where human models are *learned* from data, with humans modeled as approximately rational agents optimizing their reward functions. We provide a novel optimization-driven approach to find small deviations in learned human behavior that lead to violation of desired (safety) objectives. Our approach is experimentally validated via simulation for the application of autonomous driving. For future work, we plan to expand the application of this methodology to a broader class of human behavior models and human-robot systems. We also note that our high-level approach might be more broadly applicable to algorithmically analyzing the behavior of other kinds of systems involving models learned from data.

Falsification and verification play an important role in safety of human-robot systems. However, falsification is not enough for the design of safe controllers for our human-robot systems. In the next part of this dissertation, we address the problem of synthesizing safe controllers from formal specifications by bringing ideas from the area of formal methods to address problems that arise in safe control of human-robot systems.

Part II

Safe Control

Chapter 7

Reactive Synthesis for Human-Robot Systems

The cost of incorrect operation in many human-robot systems can be very severe. Human factors are often the reason for failures or "near failures", as noted by several studies (e.g., [50, 101]). One alternative to control human-robot systems is to synthesize a fully autonomous controller from a high-level mathematical specification. The specification typically captures both assumptions about the environment and correctness guarantees that the controller must provide, and can be specified in a formal language such as Linear Temporal Logic (LTL) as introduced in Chapter 2.3. While this correct-by-construction approach looks very attractive, the existence of a fully autonomous controller that can satisfy the specification is not always guaranteed. For example, in the absence of adequate assumptions constraining its behavior, the environment can be modeled as being overly adversarial, causing the synthesis algorithm to conclude that no controller exists. Additionally, the high-level specification might abstract away from inherent physical limitations of the system, such as insufficient range of sensors, which must be taken into account in any real implementation. Thus, while full manual control puts too high a burden on the human operator, some element of human control is desirable. However, at present, there is no systematic methodology to synthesize a combination of human and autonomous control from high-level specifications. In this chapter, we address this limitation of the state of the art. Specifically, we consider the following question: Can we devise a controller that is mostly automatic and requires only occasional human interaction for correct operation? We formalize this problem of synthesis for human-robot systems and establish formal criteria for solving it.

A particularly interesting domain is that of autonomous or semiautonomous vehicles. Such systems, already capable of automating tasks such as lane keeping, navigating in stop-and-go traffic, and parallel parking, are being integrated into high-end automobiles. However, these emerging technologies also give rise to concerns over the safety of an ultimately driverless car. Recognizing the safety issues and the potential benefits of vehicle automation, the National Highway Traffic Safety Administration (NHTSA) has published a statement that provides descriptions and guidelines for the continual development of these technologies [127]. Particularly, the statement defines five levels of automation ranging from vehicles without any control systems automated (Level 0) to vehicles with full automation (Level 4). In this chapter, we focus on Level 3 which describes a mode of automation that requires only limited driver control:

> *"Level 3 - Limited Self-Driving Automation: Vehicles at this level of automation enable the driver to cede full control of all safety-critical functions under certain traffic or environmental conditions and in those conditions to rely heavily on the vehicle to monitor for changes in those conditions requiring transition back to driver control. The driver is expected to be available for occasional control, but with sufficiently comfortable transition time. The vehicle is designed to ensure safe operation during the automated driving mode."* [127]

Essentially, this mode of automation stipulates that the human driver can act as a fail-safe mechanism and requires the driver to take over control should something go wrong. The challenge, however, lies in identifying the complete set of conditions under which the human driver has to be notified ahead of time. Based on the NHTSA statement, we identify four important criteria required for a human-robot controller to achieve this level of automation. [1]

- *Efficient Monitoring*. The controller should be able to determine if human intervention is needed based on monitoring past and current information about the system and its environment.
- *Minimally Intervening*. The controller should only invoke the human operator when it is necessary, and does so in a minimally intervening manner.
- *Prescient*. The controller can determine if a specification may be violated ahead of time, and issues an advisory to the human operator in such a way that she has sufficient time to respond.
- *Safe Autonomy*. The controller should operate correctly until the point when human intervention is deemed necessary.

We further elaborate and formally define these concepts later in Section 10.3. In general, a human-robot controller, as shown in Figure 7.1 is a controller consists of three components: an automatic controller, a human operator, and an advisory control mechanism that orchestrates the switching between the auto-controller and the human operator.[2] In this setting, the auto-controller and the human operator can be viewed as

[1] The original terms used in [110] were slightly different.

[2] In this chapter, we do not consider explicit dynamics of the plant. Therefore it can also be considered as part of the environment.

two separate controllers, each capable of producing outputs based on inputs from the environment, while the advisory controller is responsible for determining precisely when the human operator should assume control while giving her enough time to respond.

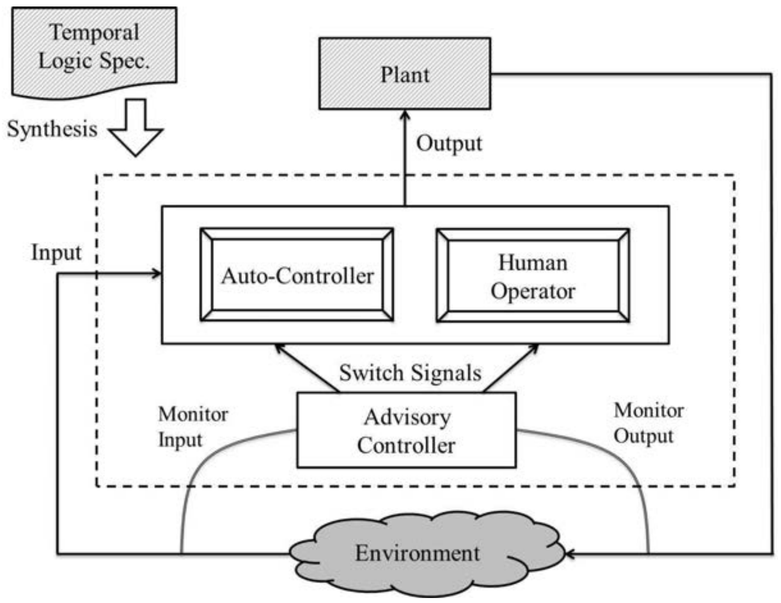
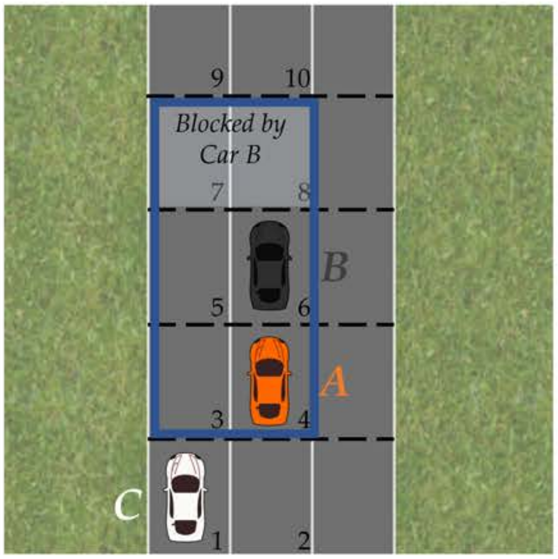


Figure 7.1: Human-Robot Controller: Component Overview

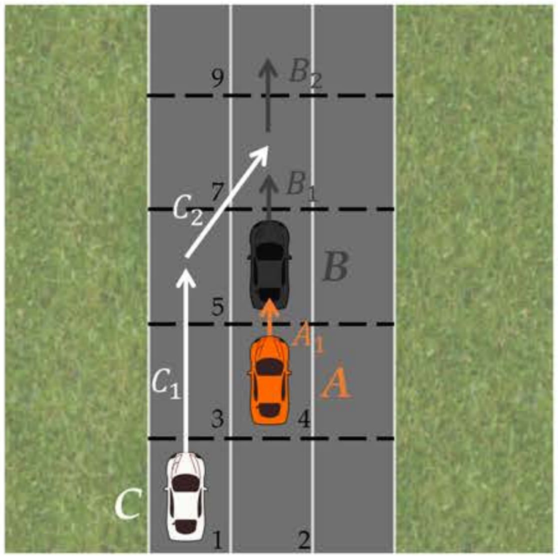
In this chapter, we study the construction of such controller in the context of *reactive synthesis* from LTL specifications. *Reactive synthesis* is the process of automatically synthesizing a discrete system (e.g., a finite-state Mealy transducer) that reacts to environment changes in such a way that the given specification (e.g., a LTL formula) is satisfied as briefly discussed in Chapter 2.3. There has been growing interest recently in the control and robotics communities (e.g., [185, 97]) to apply this approach to automatically generate embedded control software. This chapter has been in close collaboration with Wenchao Li, whose insights on using counterstrategy graphs for debugging enabled us to design the main technical algorithms of this work. In summary, the main contributions of this chapter are:

- A formalization of control of human-robot systems and the problem of synthesizing such controllers from high-level specifications, including four key criteria these controllers must satisfy.
- An algorithm for synthesizing controllers for human-robot systems that satisfy the afore-mentioned criteria.
- An application of the proposed technique to examples motivated by driver-assistance systems for automobiles.

7.1 Running Example



(a) A's Sensing Range.



(b) Failed to Follow.

Figure 7.2: Controller Synthesis – Car A Following Car B

Consider the example in Figure 7.2. Car A (orange vehicle) is the autonomous vehicle, car B (black vehicle) and C (white vehicle) are two other cars on the road. We assume that the road has been divided into discretized regions that encode all the legal transitions for the vehicles on the map, similar to the discretization setup used in receding horizon temporal logic planning [183]. The objective of car A is to *follow* car B. Note that car B and C are part of the *environment* and cannot be controlled. The notion of following can be stated as follows. We assume that car A is equipped with sensors that allows it to see two squares ahead of itself if its view is not obstructed, as indicated by the enclosed region by blue rectangle in Figure 7.2a. In this case, car B is blocking the view of car A, and thus car A can only see regions 3, 4, 5 and 6. Car A is said to be able to *follow* car B if it can always move to a position where it can see car B. Furthermore, we assume that car A and C can move at most 2 squares forward, but car B can move at most 1 square ahead, since otherwise car B can out-run or out-maneuver car A.

Given this objective, and additional safety rules such as cars not crashing into one another, our goal is to automatically synthesize a controller for car A such that:

- car A follows car B whenever possible;
- and in situations where the objective may not be achievable, *switches control* to the human driver while allowing *sufficient time* for the driver to respond and take control.

In general, it is not always possible to come up with a fully automatic controller that satisfies all requirements. Figure 7.2b illustrates such a scenario where car C blocks the view as well as the movement path of car A after two time steps. The arrow indicate the movements of the vehicles at time steps 1 and 2. Positions of a car X at time t is indicated by X_t. In this failure scenario, the autonomous vehicle needs to notify the human driver since it has lost track of car B.

Therefore, a human-robot control synthesis problem is tasked with producing an autonomous controller along with advisories for the human driver in situations where her attention is required. Our challenge, however, is to *identify the conditions that we need to monitor and notify the driver when they may fail*. In the next section, we discuss how human constraints such as response time can be simultaneously considered in the solution, and mechanisms for switching control between the auto-controller and the human driver.

7.2 Formal Model of Human-Robot Controller

Consider a Booleanized space over the input and output alphabet $\mathcal{X} = 2^X$ and $\mathcal{Y} = 2^Y$, where X and Y are two disjoint sets of variables representing inputs and outputs respectively, we model a discrete controller as a finite-state transducer. A finite-state (Mealy) transducer (FST) is a tuple $M = (Q, q^0, \mathcal{X}, \mathcal{Y}, \rho, \delta)$, where Q is the set of states, $q^0 \in Q$ is the initial state, $\rho : Q \times \mathcal{X} \to Q$ is the transition function, and $\delta : Q \times \mathcal{X} \to \mathcal{Y}$ is the output function. Given an input sequence $\boldsymbol{x} = x^0, x^1, \ldots$, a run of M is the infinite sequence $\boldsymbol{q} = q^0, q^1, \ldots$ of states such that $q^{k+1} = \rho(q^k, i^k)$ for all $k \geq 0$. The run $\boldsymbol{q}$ on $\boldsymbol{x}$ produces the word $M(\boldsymbol{x}) = \delta(q^0, x^0)\delta(q^1, x^1) \ldots$. The language of M is then denoted by the set $\mathcal{L}(M) = \{(x, y)^\omega \mid M(\boldsymbol{x}) = \boldsymbol{y}\}$.

To characterize correctness of M, we assume that we can label if a state is *unsafe* or not, by using a function $\mathcal{F} : Q \to \{\texttt{true}, \texttt{false}\}$, i.e. a state q is failure-prone if $\mathcal{F}(q) = \texttt{true}$. We elaborate on $\mathcal{F}$ later in Section 7.3.

Agents as Automata

We model two of the three agents in the human-robot controller shown in Figure **??**, the automatic controller *Auto* and the advisory controller $\mathcal{VC}$, as finite-state transducers (FSTs). The human operator can be viewed as another FST $\mathcal{H}$ that uses the same input and output interface as the auto-controller. The overall controller is then a composition of the models of $\mathcal{H}$, *Auto* and $\mathcal{VC}$.

We use a binary variable *auto* to denote the internal advisory signal that $\mathcal{VC}$ sends to both *Auto* and $\mathcal{H}$. Hence, $\mathcal{X}^{\mathcal{H}} = \mathcal{X}^{Auto} = \mathcal{X} \cup \{auto\}$, and $\mathcal{Y}^{\mathcal{VC}} = \{auto\}$. When $auto = \texttt{false}$, it means the advisory controller is requiring the human operator to take over control, and the auto-controller can have control otherwise.

We assume that the human operator (e.g., driver behind the wheel) can take control at any time by transitioning from a "non-active" state to an "active" state, e.g., by hitting a button on the dashboard or simply pressing down the gas pedal or the brake. When $\mathcal{H}$ is in the "active" state, the human operator essentially acts as the automaton that produces outputs to the plant (e.g., a car) based on environment inputs. We use a binary variable *active* to denote if $\mathcal{H}$ is in the "active" state. When $active =$ `true`, the output of $\mathcal{H}$ overwrites the output of *Auto*. The "overwrite" action happens when a sensor senses the human operator is in control, e.g., putting her hands on the wheel. Similarly, when $active =$ `false`, the output of the human-robot controller is the output of *Auto*. Note that even though the human operator is modeled as a FST here, since we do not have direct control of the human operator, it can in fact be any arbitrary relation mapping $\mathcal{X}$ to $\mathcal{Y}$.

Criteria for Human-Robot Controllers

One key distinguishing factor of a human-robot controller from traditional controller is the involvement of a human operator. Hence, human factors such as response time cannot be disregarded. In addition, we would like to minimize the need to engage the human operator. Based on the NHTSA statement, we derive four criteria for any effective human-robot controller, as stated below.

- *Monitoring.* An advisory *auto* is issued to the human operator under specific conditions. These conditions in turn need to be determined unambiguously *at runtime*, potentially based on history information but not predictions. In a reactive setting, this means we can use trace information only up to the point when the environment provides a next input from the current state.

- *Minimally intervening.* Our mode of interaction requires only selective human intervention. An intervention occurs when $\mathcal{H}$ transitions from the "non-active" state to the "active" state (we discuss mechanisms for suggesting a transition from "active" to "non-active" in Section 7.3, after prompted by the advisory signal *auto* being `false`). However, frequent transfer of control would mean constant attention is required from the human operator, thus nullifying the benefits of having the auto-controller. In order to reduce the overhead of human participation, we want to minimize a joint objective function $\mathcal{C}$ that combines two elements: (i) the *probability* that when *auto* is set to `false`, the environment will eventually force *Auto* into a failure scenario, and (ii) the *cost* of having the human operator taking control. We formalize this objective function in Section 7.3.

- *Prescient.* It may be too late to seek the human operator's attention when failure is imminent. We also need to allow extra time for the human to respond and study the situation. Thus, we require an advisory to be issued ahead of any failure scenario. In the discrete setting, we assume we are given a positive integer T representing human response time (which can be driver-specific), and require that *auto* is set to `false` at least T number of transitions ahead of a state (in *Auto*) that is *unsafe*.

- *Conditionally-Correct.* The auto-controller is responsible for correct operation as long as *auto* is set to `true`. Formally, if $auto = \texttt{true}$ when *Auto* is at a state q, then $\mathcal{F}(q) = \texttt{false}$. Additionally, when *auto* is set to `false`, the auto-controller should still maintain correct operation in the next $T - 1$ time steps, during or after which we assume the human operator take over control. Formally, if *auto* changes from `true` to `false` when *Auto* is at a state q, let $R_T(q)$ be the set of states reachable from q within $T - 1$ transitions, then $\mathcal{F}(q') = \texttt{false}$, $\forall q' \in R_T(q)$.

Problem 4 (Human-Robot Controller Synthesis Problem)**.** *Given a model of the system and its specification expressed in a formal language, synthesize a human-robot controller that is, by construction, monitoring, minimally intervening, prescient, and conditionally correct.*

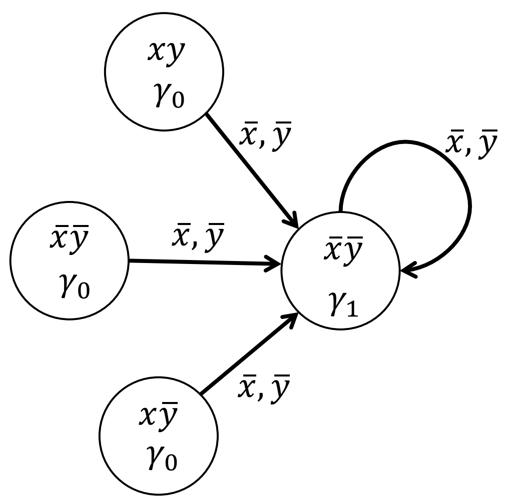
In this chapter, we study the synthesis of a human-robot controller in the setting of synthesis of reactive systems from Linear Temporal Logic (LTL) specifications. We give background on this setting in Chapter 2.3, and propose an algorithm for solving the Human-Robot Controller Synthesis problem in Section 7.3.

7.3 Human-Robot Controller Synthesis

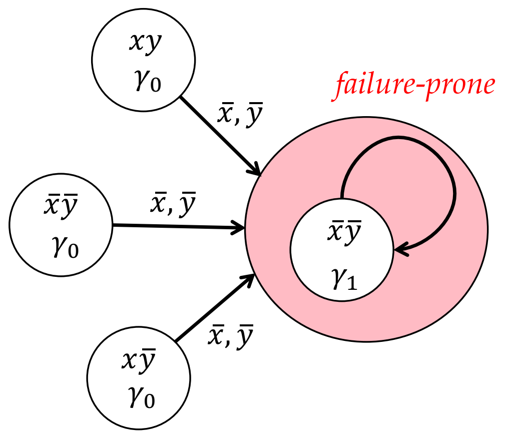
Given an unrealizable specification, a counterstrategy $\mathcal{S}^{env}$ exists for *env* which describes moves by *env* such that it can force a violation of the system guarantees. The key insight of our approach for synthesizing a human-robot controller is that we can synthesize an advisory controller that monitors these moves and prompts the human operator with sufficient time ahead of any danger. These moves are essentially assumptions on the environment under which the system guarantees can be ensured. When these assumptions are not violated (the environment may behave in a benign way in reality), the auto-controller fulfills the objective of the controller. On the other hand, if any of the assumptions is violated, as flagged by the advisory controller, then the control is safely switched to the human operator in a way that she can have sufficient time to respond. The challenge, however, is to decide when an advisory should be sent to the human operator, in a way that it is also *minimally intervening* to the human operator. We use the following example to illustrate our algorithm.

Example 1. *Consider $X = \{x\}$, $Y = \{y\}$ and the following LTL sub-formulas which together form $\psi = \psi^{env} \rightarrow \psi^{sys}$.*

- $\psi_f^{env} = \mathbf{G}\ (\mathbf{F}\, \neg x)$
- $\psi_t^{sys} = \mathbf{G}\ (\neg x \rightarrow \neg y)$
- $\psi_f^{sys} = \mathbf{G}\ (\mathbf{F}\, y)$



(a) Counterstrategy graph G_c for unrealizable specification ψ.



(b) Condensed graph $\hat{G}_c$ for G_c after contracting the strongly connected components.

Specification ψ is not realizable. Figure 7.3a shows the computed counterstrategy graph G_c. The literal $\bar{x}$ ($\bar{y}$) denotes the negation of the propositional variable x (y). The memory content is denoted by γ_i with γ_0 being the initial memory content. The three states on the left are the initial states. The literals on the edges indicate that the environment first chooses $\bar{x}$ and then the system chooses $\bar{y}$. (the system is forced to pick $\bar{y}$ due to ψ_t^{sys}). Observe that, according the counterstrategy, the system will be forced to pick $\bar{y}$ perpetually. Hence, the other system guarantee ψ_f^{sys} cannot be satisfied.

Weighted Counterstrategy Graph

Recall that a counterstrategy can be viewed as a discrete transition system or a directed graph G_c. We consider two types of *imminent* failures (violation of some system guarantee specification) described by G_c.

- **Safety violation.** For a node (state) $q_c^1 \in Q_c$, if there does not exist a node q_c^2 such that $(q_c^1, q_c^2) \in \rho_c$, then we say q_c^1 is *failure-imminent*. In this scenario, after *env* picks a next input according to the counterstrategy, *sys* cannot find a next output such that all of the (safety) guarantees are satisfied (some ψ_i^{sys} or ψ_t^{sys} is violated).
- **Fairness violation.** If a node q_c is part of a strongly connected component (SCC) in Q_c, then we say q_c is *failure-doomed*. For example, the node $(\bar{x}, \bar{y}, \gamma_1)$ in Figure 7.3a is a failure-doomed node. Starting from q_c, *env* can always pick inputs in such a way that the play is forced to get stuck in SCC. Clearly, all other states in SCC are also *failure-doomed*.

Now we make the connection of the labeling function $\mathcal{F}$ for a controller M to the counterstrategy graph G_c which describes behaviors that M should not exhibit. Consider an auto-controller M and a state q (represented by the assignment xy) in M. $\mathcal{F}(q) = \texttt{true}$ if and only if there exist some $q_c \in Q_c$ such that $\theta_c(q_c) = xy$ and q_c is either *failure-imminent* or *failure-doomed*. In practice, it is not always the case that the environment will behave in the most adversarial way. For example, a car in front may yield if it is blocking our path. Hence, even though the specification is not realizable, it is still important to assess, at any given state, whether it will actually lead to a violation. For simplicity, we assume that the environment will adhere to the counterstrategy once it enters a *failure-doomed* state.

We can convert G_c to its directed acyclic graph (DAG) embedding $\hat{G}_c = (\hat{Q}_c, \hat{Q}_c^0, \hat{\rho}_c)$ by contracting each SCC in G_c to a single node. Figure 7.3b shows the condensed graph $\hat{G}_c$ of G_c shown in Figure 7.3a. We use a surjective function $\hat{f} : Q_c \to \hat{Q}_c$ to describe the mapping of nodes from G_c to $\hat{G}_c$. We say a node $\hat{q} \in \hat{Q}_c$ is *failure-prone* if a node $q_c \in Q_c$ is either *failure-imminent* or *failure-doomed* and $\hat{f}(q_c) = \hat{q}$.

Recall from Section 7.2 that the notion of *minimally-intervening* requires the minimization of a cost function $\mathcal{C}$, which involves the *probability* that *auto* is set to `false`, Thus far, we have not associated any probabilities with transitions taken by the environment or the system. While our approach can be adapted to work with any assignment of probabilities, for ease of presentation, we make a particular choice in this chapter. Specifically, we assume that at each step, the environment picks a next-input uniformly at random from the set of possible *legal* actions (next-inputs) obtained from η^{env} given the current state. In Example 1 and correspondingly Figure 7.3a, this means that it is equally likely for *env* to choose $\bar{x}$ or x from any of the states. We use $c(q)$ to denote the total number of legal actions that the environment can take from a state q.

In addition, we take into account of the cost of having the human operator perform the maneuver instead of the auto-controller. In general, this cost increases with longer human engagement. Based on these two notions, we define ϖ, which assigns a weight to an edge $e \in \hat{Q}_c \times \hat{Q}_c$ in $\hat{G}_c$, recursively as follows. For an edge between $\hat{q}_i$ and $\hat{q}_j$,

$$\varpi(\hat{q}_i, \hat{q}_j) = \begin{cases} 1 & \text{if } \hat{q}_j \text{ is } \textit{failure-prone} \\ \frac{pen(\hat{q}_i) \times len(\hat{q}_i)}{c(\hat{q}_i)} & \text{Otherwise} \end{cases}$$

where $pen : \hat{Q}_c \to \mathbb{Q}^+$ is a user-defined penalty parameter[3], and $len : \hat{Q}_c \to \mathbb{Z}^+$ is the length (number of edges) of the shortest path from a node $\hat{q}_i$ to any failure-prone node in $\hat{G}_c$. Intuitively, a state far away from any failure-prone state is less likely to cause a failure since the environment would need to make multiple consecutive moves all in an adversarial way. However, if we transfer control at this state, the human operator will have to spend more time in control, which is not desirable for a human-robot system controller. Next, we describe how to use this edge-weighted DAG representation of a counterstrategy graph to derive a controller that satisfies the criteria established earlier.

[3] $pen(\hat{q}_i)$ should be chosen such that $\varpi(\hat{q}_i, \hat{q}_j) < 1$.

Counterstrategy-Guided Synthesis

Suppose we have a counterstrategy graph G_c that summarizes all possible ways for the environment to force a violation of the system guarantees. Consider an outgoing edge from a non-failure-prone node $\hat{q}$ in $\hat{G}_c$ (condensed graph of G_c), this edge encodes a particular condition where the environment makes a next-move given some last move made by the environment and the system. If some of these next-moves by the environment are disallowed, such that none of the failure-prone nodes are reachable from any initial state, then we have effectively eliminated the counterstrategy. This means that if we assert the negation of the corresponding conditions as additional ψ_t^{env} (environment transition assumptions), then we can obtain a realizable specification.

Formally, we mine assumptions of the form $\phi = \bigwedge_i (\mathbf{G}\,(a_i \rightarrow \neg \mathbf{X}\, b_i))$, where a_i is a Boolean formula describing a set of assignments over variables in $X \cup Y$, and b_i is a Boolean formula describing a set of assignments over variables in X.

Under the assumption ϕ, if $(\phi \wedge \psi^{env}) \rightarrow \psi^{sys}$ is realizable, then we can automatically synthesize an auto-controller that satisfies ψ. In addition, the key observation here is that mining ϕ is *equivalent to* finding a set of edges in $\hat{G}_c$ such that, if these edges are removed from $\hat{G}_c$, then none of the failure-prone nodes is reachable from any initial state. We denote such set of edges as E^{ϕ}, where each edge $e \in E^{\phi}$ corresponds to a conjunct in ϕ. For example, if we remove the three outgoing edges from the source nodes in Figure 7.3b, then the failure-prone node is not reachable. Removing these three edges correspond to adding the following environment assumption, which can be monitored at runtime.

$$\mathbf{G}\,((x \wedge y) \rightarrow \neg \mathbf{X}\,\bar{x}) \wedge \mathbf{G}\,((\bar{x} \wedge \bar{y}) \rightarrow \neg \mathbf{X}\,\bar{x}) \wedge \mathbf{G}\,((x \wedge \bar{y}) \rightarrow \neg \mathbf{X}\,\bar{x})$$

Human factors play an important role in the design of a controller for human-robot systems. The criteria established for a controller in Section 7.2 also require it to be *prescient* and *minimally intervening*. Hence, we want to mine assumptions that reflect these criteria as well. The notion of *prescient* essentially requires that none of the failure-prone nodes is reachable from a non-failure-prone node with less than T steps (edges). The weight function ϖ introduced earlier can be used to characterize the cost of a failing assumption resulting in the advisory controller prompting the human operator to take over control (by setting *auto* to `false`). Formally, we seek E^{ϕ} such that the total cost of switching control $\sum_{e \in E^{\phi}} \varpi(e)$ is minimized.

We can formulate this problem as a *s-t* min-cut problem for directed acyclic graphs. Given $\hat{G}_c$, we first compute the subset of nodes $\hat{Q}_c^T \subseteq \hat{Q}_c$ that are backward reachable within $T-1$ steps from the set of failure-prone nodes (when $T = 1$, $\hat{Q}_c^T$ is the set of failure-prone node). We assume that $\hat{Q}_c^0 \cap \hat{Q}_c^T = \emptyset$. Next, we remove the set of nodes $\hat{Q}_c^T$ from $\hat{G}_c$ and obtain a new graph $\hat{G}_c^T$. Since $\hat{G}_c^T$ is again a DAG, we have a set of source nodes and a set of terminal nodes. Thus, we can formulate a *s-t* min-cut problem by adding a new source node that has an outgoing edge (with a sufficiently large weight) to each of the source nodes and a new terminal node that has an incoming edge (with a sufficiently large weight) from each of the terminal nodes. This *s-t* min-cut problem

Algorithm 2 Counterstrategy-Guided Human-Robot Controller Synthesis

```
Input: GR(1) specification $\psi = \psi^{env} \to \psi^{sys}$.
Input: $T$ : parameter for minimum human response time.
Output: $Auto$ and $\mathcal{VC}$. Our controller is then a composition of $Auto$, $\mathcal{VC}$ and $\mathcal{H}$.
  if $\psi$ is realizable then
    Synthesize transducer $M \models \psi$ (using standard LTL synthesis);
    Controller := $M$ (fully autonomous).
  else
    Generate $G_c$ from $\psi$ (assume a single $G_c$; otherwise the algorithm is performed iteratively);
    Generate the DAG embedded $\hat{G}_c$ from $G_c$.
    Reduce $\hat{G}_c$ to $\hat{G}_c^T$;
    Assign weights to $\hat{G}_c$ using $\varphi$; by removing $\hat{Q}_c^T$ – nodes that are within $T-1$ steps of any failure-prone node;
    Formulate a s-t min-cut problem with $\hat{G}_c^T$;
    Solve the s-t min-cut problem to obtain $E^{\phi}$;
    Add assumptions $\phi$ to $\psi$ to obtain the new specification $\psi_{new} := (\phi \wedge \psi^{env}) \to \psi^{sys}$;
    Synthesize $Auto$ so that $M \models \psi_{new}$;
    Synthesize $\mathcal{VC}$ as a (stateless) monitor that outputs $auto =$ false if $\phi$ is violated.
  end if
```

can be easily solved by standard techniques [40]. The overall approach is summarized in Algorithm 2.

Theorem 4. *Given a LTL specification ψ and a response time parameter T, Algorithm 2 is guaranteed to either produce a fully autonomous controller satisfying ψ, or a human-robot controller, modeled as a composition of an auto-controller Auto, a human operator and an advisory controller $\mathcal{VC}$, that is* monitoring, prescient *with parameter T,* minimally intervening[4] *with respect to the cost function $f_C = \sum_{e \in E^{\phi}} \varpi(e)$, and* conditionally correct[5].

Proof. (Sketch) When ψ is realizable, a fully autonomous controller is synthesized and unconditionally satisfies ψ. Now consider that case when ψ is not realizable.

The HuIL controller is *monitoring* as ϕ only comprises a set of environment transitions up to the next environment input.

It is *prescient* by construction. The *auto* flag advising the human operator to take over control is set to `false` *precisely* when ϕ is violated. When ϕ is violated, it corresponds to the environment making a next-move from the current state q according to some edge $e = (\hat{q}_i, \hat{q}_j) \in E^{\phi}$. Consider any $q_c \in Q_c$ such that $\hat{f}(q_c) = \hat{q}_i$, $\theta_c(q^c) = q$. Since $\hat{q}_i \notin \hat{Q}_c^T$ by

[4]We assume the counterstrategy we use to mine the assumptions is an optimal one – it forces a violation of the system guarantees as quickly as possible.

[5]We assume that all failure-prone nodes are at least T steps away from any initial node.

the construction of $\hat{G}_c^T$, $\hat{q}_i$ is at least T transitions away from any *failure-prone* state in $\hat{G}_c$. This means q_c must also be at least T transitions away from any *failure-imminent* state or *failure-doomed* state in Q_c. Hence, by the definition of $\mathcal{F}$ with respect to a *failure-doomed* or *failure-doomed* state in Section 7.3, q is (and *auto* is set) at least T transitions ahead of any state that is *unsafe*.

The controller is also *conditionally correct*. By the same reasoning as above, for any state $q' \in R_T(q)$, $\mathcal{F}(q') = \texttt{false}$, i.e. q' is *safe*.

Finally, since *auto* is set to `false` precisely when ϕ is violated, and ϕ in turn is constructed based on the set of edges E^ϕ, which minimizes the cost function $f_C = \sum_{e \in E^\phi} \varpi(e)$, the controller is *minimally-intervening* with respect to the cost function f_C. $\square$

Switching from Human Operator to Auto-Controller

Once control has been transferred to the human operator, when should the human yield control to the autonomous controller again? One idea is for the controller to continually monitor the environment after the human operator has taken control, checking if a state is reached from which the auto-controller can ensure that it satisfies the specification (under assumption ϕ), and then the advisory controller can signal a driver telling her that the auto-controller is ready to take back control. We note that alternative approaches may exist and we plan to investigate this further in future work.

7.4 Experimental Results

Our algorithm is implemented as an extension to the GR(1) synthesis tool RATSY [30]. We now discuss the car-following example (as shown in Section 7.1) here. The implementation can be found here: `http://verifun.eecs.berkeley.edu/tacas14/`

Recall the car-following example shown in Section 7.1. We describe some of the more interesting specifications below and their corresponding LTL formulas. p_A, p_B, p_C are used to denote the positions of car A, B and C respectively.

- Any position can be occupied by at most one car at a time (no crashing):

$$\mathbf{G}\left(p_A = x \rightarrow (p_B \neq x \wedge p_C \neq x)\right)$$

  where x denotes a position on the discretized space. The cases for B and C are similar, but they are part of ψ_{env}.

- Car A is required to follow car B:

$$\mathbf{G}\left((v_{AB} = \texttt{true} \wedge p_A = x) \rightarrow \mathbf{X}\left(v_{AB} = \texttt{true}\right)\right)$$

  where $v_{AB} = \texttt{true}$ if and only if car A can see car B.

- Two cars cannot cross each other if they are right next to each other. For example, when $p_C = 5$, $p_A = 6$ and $p'_C = 8$ (in the next cycle), $p'_A \neq 7$. In LTL,

$$\mathbf{G}\left(((p_C = 5) \wedge (p_A = 6) \wedge (\mathbf{X}\, p_C = 8)) \rightarrow (\mathbf{X}\,(p_A \neq 7))\right)$$

The other specifications can be found in the link at the beginning of this section. Observe that car C can in fact force a violation of the system guarantees in one step under two situations – when $p_C = 5, p_B = 8$ and $p_A = 4$, or $p_C = 5, p_B = 8$ and $p_A = 6$. Both are situations where car C is blocking the view of car A, causing it to lose track of car B. The second failure scenario is illustrated in Figure 7.2b.

Applying our algorithm to this (unrealizable) specification with $T = 1$, we obtain the following assumption ϕ.

$$\begin{aligned}\phi = \mathbf{G}\left(((p_A = 4) \wedge (p_B = 6) \wedge (p_C = 1)) \rightarrow \neg\mathbf{X}\,((p_B = 8) \wedge (p_C = 5))\right) \bigwedge \\ \mathbf{G}\left(((p_A = 4) \wedge (p_B = 6) \wedge (p_C = 1)) \rightarrow \neg\mathbf{X}\,((p_B = 6) \wedge (p_C = 3))\right) \bigwedge \\ \mathbf{G}\left(((p_A = 4) \wedge (p_B = 6) \wedge (p_C = 1)) \rightarrow \neg\mathbf{X}\,((p_B = 6) \wedge (p_C = 5))\right)\end{aligned}$$

In fact, ϕ corresponds to three possible evolutions of the environment from the initial state. In general, ϕ can be a conjunction of conditions at different time steps as *env* and *sys* progress. The advantage of our approach is that it can produce ϕ such that we can synthesize an auto-controller that is guaranteed to satisfy the specification if ϕ is not violated, together with an advisory controller that prompts the driver (at least) T (T = 1 in this case) time steps ahead of a potential failure when ϕ is violated.

7.5 Chapter Summary

In this chapter, we've discussed the problem of reactive synthesis from temporal logic specifications [141, 138]. Similar to [97], we have synthesized a discrete controller from temporal logic specifications.

Wongpiromsarn et al. [183] consider a receding horizon framework to reduce the synthesis problem to a set of simpler problems for a short horizon. We further discuss a receding horizon variant of our problem in the future chapters. Livingston et al. [112, 113] exploit the notion of locality that allows "patching" a nominal solution. They update the local parts of the strategy as new data accumulates allowing incremental synthesis. The patching algorithm considers changes in the game graph, and identifies the affected nodes for each system goal and modifies the graph locally. In our approach, we do not start with a synthesized game graph, and we mine assumptions from counterstrategies. Also, we consider environment assumptions that are not only limited to the topological changes. The key innovation in this chapter is that our work is the first to consider synthesizing interventions that combine an autonomous controller with a human operator.

Our work is inspired by the recent works on assumption mining. Chatterjee et al. [37] construct a minimal environment assumption by removing edges from the game graph to ensure safety assumptions, then compute liveness assumptions to put additional fairness constraints on the remaining edges. Finding this minimal set is NP-Hard. Our approach is based on removing edges from the counter-strategy; therefore, the environment assumptions are easier to mine, and more practical as they can be mapped to monitoring real sensors. The flexibility of choosing which edge to remove gives us a measure of conservativeness in our control. Removing edges closer to the source node prevents any violations in the future moves and removing edges towards the sink node is less conservative. Li et al. [109] and later Alur et al. [8] use a counterstrategy-guided approach to mine environment assumptions for GR(1) specifications. We adapt this approach to the synthesis of controllers for human-robot systems.

Finally, human's reaction time while driving is an important consideration in this chapter. This time is based on human's reaction time while driving, which is driver-specific. The value of reaction time can range from 1 to 2.5 seconds for different tasks and drivers [169].

One limitation of the current approach is the use of an explicit counterstrategy graph (due to weight assignment). We plan to explore symbolic algorithms in the future. Further, we plan to synthesize a systematic intervention scheme that closely combines models of operators, and provides formal correctness guarantees about it.

Chapter 8

Reactive Synthesis from Signal Temporal Logic

In this chapter, our goal is to design controllers that satisfy desired temporal logic specifications despite a potentially adversarial environment or human models. These synthesized controllers are robust to uncertain environment or human actions.

As we have seen in the previous chapter, temporal logics is a valuable tool for controller synthesis. Approaches using temporal logic can be broadly categorized based on if they use a discrete abstraction of the system, and if the environment is assumed to be deterministic.

Using discrete abstraction enables the construction of a discrete supervisory controller, which can then construct a hybrid controller. These discrete abstraction techniques address both deterministic [94, 133] and adversarial environments [53, 186]. In contrast, techniques that avoid discrete abstraction include sampling-based methods [88], and mixed-integer linear programming encodings of temporal logic specifications [86, 89, 100, 182, 147].

In this chapter, we plan to close the gap between these two techniques by addressing controller synthesis under reactive environments through mixed-integer linear programming encodings. The ideas and results of this chapter was done in close collaboration with Vasumathi Raman, Alexandre Donze, Richard Murray, and Sanjit Seshia. Specifically, Vasumathi Raman's ideas for the mixed integer encoding of signal temporal logic, and her previous work [146] has provided us with an insight that led to the algorithms and results of this chapter.

We adopt a *counterexample-guided inductive synthesis* [166] approach to synthesize a controller satisfying reactive specifications. Inductive synthesis refers to the automated generation of a system from input-output examples, using each new example to iteratively refine the hypothesis about the system until convergence. In Counterexample-Guided Inductive Synthesis (CEGIS), the examples are mostly counterexamples discovered while trying to verify correctness of the current guess. CEGIS thus relies primarily on a validation engine to validate candidates produced at intermediate iterations, which can produce counterexamples for use in the next iteration. Automated synthesis of systems using CEGIS and the closely related Counterexample-Guided Abstraction Refinement (CEGAR) paradigm has been widely studied in various contexts [38, 8, 84].

The specification language adopted here is STL [119], which allows the specification of temporal properties of real-valued signals, and has been applied to the analysis of hybrid dynamical systems from various application domains such as analog and mixed signal circuits, systems biology or Cyber-Physical Systems (CPS) (see Chapter 2.3). We exploit the quantitative semantics of STL to compute the robustness of satisfaction in the validation engine for our CEGIS approach to reactive synthesis.

A key advantage of temporal logic over, e.g., domain-specific languages based on propositional logic, is that it enables the expression of properties of infinite traces. We would therefore like to synthesize controllers that can run indefinitely, and satisfy infinite-horizon properties. However, in this chapter we use a Receding Horizon Control approach introduced in Chapter 2.1. This not only reduces computational complexity, but also improves robustness with respect to exogenous disturbances and modeling uncertainties by allowing new information to be incorporated as it becomes available [126].

The connection between Receding Horizon Control (RHC) and control synthesis from STL specifications are discussed in previous work [147], where the authors specify desired properties of the system using a STL formula, and synthesize control strategies such that the system satisfies that specification, while using a receding horizon approach. Raman et al. present automatically-generated Mixed Integer Linear Program (MILP) encodings for STL specifications, extending the Bounded Model Checking (BMC) paradigm for finite discrete systems [28] to STL. These encodings can be used not only to generate open-loop control signals that satisfy finite and infinite horizon STL properties, but also to generate signals that maximize quantitative (robust) satisfaction. In this chapter, we show how the robustness-based encoding can be used to produce a validation engine that synthesizes counterexamples to guide a CEGIS approach to reactive synthesis.

Abbas et al. [1] exploit the quantitative semantics of Metric Temporal Logic (MTL) to design a framework for specification-guided testing of stochastic cyber-physical systems. Leveraging results from stochastic optimization, they frame the verification of properties on such systems as a global optimization problem of minimizing the expected robustness of satisfaction. While we work with nondeterministic systems rather than stochastic systems, our CEGIS approach leverages a similar idea when finding an adversarial environment input that minimizes the robustness of satisfaction.

Receding horizon control that satisfies temporal logic specifications in adversarial settings has been considered before in the context of Linear Temporal Logic (LTL) [186], where the authors propose a framework utilizing discrete abstractions to synthesize supervisory controllers for specifications in the GR(1) subset of LTL. In this work, feasibility of the global specification is determined via symbolic checks on a series of pre-defined smaller problems, and strategies extracted as needed. In contrast, we do not require an *a priori* defined finite set of sub-problems. Our approach also extends synthesis capabilities to a wider class of temporal logic specifications and environments than [68, 23], and avoids potentially expensive computations of a finite state abstraction of the system as in [43] and [186].

The key contribution of this chapter is a CEGIS approach to controller synthesis for cyber-physical systems subject to signal temporal logic (STL) specifications, operating in and interacting with potentially adversarial nondeterministic environments or human agents. Specific features of our approach include:

- We leverage the previously-proposed encoding of STL specifications as mixed integer-linear constraints [147], and solve a counterexample-guided series of optimization problems to yield a satisfying control sequence.
- Our framework can be used in a receding horizon fashion to fulfill properties over unbounded horizons.
- We present experimental results using a case study of controller synthesis for an autonomous driving scenario in the presence of adversarial agents; simulation results in this domain illustrate the effectiveness of our methodology.

8.1 MILP Encoding for Controller Synthesis

In order to synthesize a run that satisfies a STL formula φ, we add STL constraints to a MILP formulation of the control synthesis problem as in [147]. We first represent the system trajectory as a finite sequence of states satisfying the model dynamics in equation (2.7). Then, we encode the formula φ with a set of MILP constraints; our encoding produces a MILP as long as the functions μ that define the predicates π^{μ} in φ are linear or affine.

Constraints on system evolution

The system constraints encode valid finite (horizon-N) trajectories for a dynamical system–these constraints hold if and only if the trajectory $\xi(x^0, \mathbf{u}^N, \mathbf{w}^N)$ satisfies (2.7). A typical situation is when the discrete dynamics f_d is linear. In that case, the constraints on system evolution are of the form

$$\begin{array}{rcl} x^1 & = & Ax^0 + B_u u^0 + B_w w^0 \\ x^2 & = & Ax^1 + B_u u^1 + B_w w^1 \\ & \dots & \\ x^N & = & Ax^{N-1} + B_u u^{N-1} + B_w w^{N-1} \end{array}$$

STL constraints

The robustness of satisfaction of the STL specification, as defined in Chapter 2.3, provides a natural objective for the MILP defined above, either in the absence of, or as a complement to domain-specific objectives on runs of the system.

As described in Chapter 2.3, the robustness of a STL specification φ can be computed recursively on the structure of the formula. Moreover, since max and min operations can be expressed in a MILP formulation using additional binary variables, this does not add complexity to the encoding (although the additional variables make it more computationally expensive in practice). For a given formula φ, we introduce a variable ρ_k^φ, and an associated set of MILP constraints such that $\rho_k^\varphi > 0$ if and only if φ holds at time t_k. Given φ, we recursively generate MILP constraints for every subformula of φ, such that ρ_0^φ determines whether φ holds in the initial state. For example, enforcing $\rho_k^\varphi = \rho^\varphi(\mathbf{x}, t_k)$ gives us this property. The reader is referred to [147] for details of this encoding. The advantage of this *robustness-based* encoding is that it allows us to maximize or minimize the value of ρ_0^φ, obtaining a trajectory that maximizes or minimizes the robustness of satisfaction.

The union of the STL constraints and system constraints yields a MILP, enabling us to check feasibility and find a solution when possible using a MILP solver; for further details and examples see [147]. Given an objective function on runs of the system, we can also find an optimal trajectory that satisfies the STL specification. The robustness provides a natural objective for this MILP, either in the absence of, or as a complement to domain-specific objectives on runs of the system.

Mixed integer-linear programs are NP-hard, and hence impractical when the dimensions of the problem grow. We present the computational costs of the above encoding in terms of the number of variables and constraints in the resulting MILP. If P is the set of predicates used in the formula and $|\varphi|$ is the length (i.e. the number of operators), then $O(N \cdot |P|) + O(N \cdot |\varphi|)$ continuous variables are introduced. In addition, $O(N)$ binary variables are introduced for every instance of a Boolean operator, i.e., $O(N \cdot |\varphi|)$ Boolean variables.

The dimensionality of the discrete-time dynamical system affects the size of the constructed MILP linearly via the constraints encoding system evolution (more precisely, through the size of the set of predicates P). However, given the efficiency of modern MILP solvers, there is no evidence that a linear increase in the problem size would in practice lead to more than a linear increase in computational time for a solution. Methods based on abstraction or discretization of the state space, on the other hand, are much more likely to have exponential complexity with respect to system dimensionality. We note, however, that our approach is more sensitive to the size of the specifications, and in particular to the nesting degree of temporal operators. We report on the scalability of our approach in Section 8.5.

8.2 STL Reactive Synthesis Problem

We address the problem of synthesizing control inputs for a system operating in the presence of potentially adversarial, *a priori* uncertain external inputs or disturbances. The controllers we produce will provide guarantees for specifications of the form $\varphi \doteq \varphi_e \Rightarrow \varphi_s$, where φ_e places assumptions on the external environment, and φ_s specifies desired guarantees on the plant behavior. In this chapter, φ_e refers exclusively to properties of signals $\mathbf{w} \in W^\omega$, whereas φ_s refers to properties of $\mathbf{x} \in \mathcal{X}^\omega$ and $\mathbf{u} \in U^\omega$.

We now formally state the synthesis problem for reactive controllers subject to STL specifications of the form above, and its receding horizon formulation.

Problem 5 (STL Reactive Synthesis). *Given a discrete dynamical system:*

$$x^{t+1} = f(x^t, u^t, w^t), \tag{8.1}$$

initial state x^0, trajectory length N, STL formula φ and cost function J, compute:

$$\begin{array}{lll} \underset{\mathbf{u}^N}{\operatorname{argmin}} & \underset{\mathbf{w}^N \in \{\mathbf{w} \in W^N \mid \mathbf{w} \models \varphi_e\}}{\max} & J(\xi(x^0, \mathbf{u}^N, \mathbf{w}^N)) \\ \\ s.t. & \forall \mathbf{w}^N \in W^N, & \xi(x^0, \mathbf{u}^N, \mathbf{w}^N) \models \varphi \end{array}$$

Problem 6 (Receding Horizon Reactive Synthesis). *Given a system of the form in equation (8.1), initial state x^0, STL formula φ and cost function J, at each time step k, compute:*

$$\begin{array}{lll} \underset{\mathbf{u}^{H,k}}{\operatorname{argmin}} & \underset{\mathbf{w}^{H,k} \in \{\mathbf{w} \in W^H \mid \mathbf{w} \models \varphi_e\}}{\max} & J(\xi(x^k, \mathbf{u}^{H,k}, \mathbf{w}^{H,k})) \\ \\ s.t. & \forall \mathbf{w} \in W^\omega, & \xi(x^0, \mathbf{u}, \mathbf{w}) \models \varphi, \end{array}$$

where H is a finite horizon provided as a user input or selected in some other fashion, $\mathbf{u}^{H,k}$ is the horizon-H control input computed at each time step and $\mathbf{u} = u_0^{H,0} u_0^{H,1} u_0^{H,2} \ldots$.

In Sections 8.3 and 8.4, we present both a finite-trajectory solution to Problem 5, and a solution to Problem 6 for a large class of STL formulas. A key component of our solution is to use the previously presented encoding of STL specifications as MILP constraints [147] in combination with MILP constraints representing the system dynamics to efficiently solve the resulting constrained optimization problem.

8.3 Counterexample-Guided Finite Horizon Synthesis

We propose a solution to Problem 5 using a counterexample guided inductive synthesis (CEGIS) procedure. We first consider bounded STL properties φ, bounded by $N \in \mathbb{N}$.

Algorithm 3 CEGIS Algorithm for Problem 5

```
1: Input: $\xi, x_0, N, \varphi, J$
2: Let $\mathbf{w}^0 = (w_1^0, w_2^0, ...w_{N-1}^0)$, s.t. $\mathbf{w}^N \models \varphi_e$
3: $W_{cand} = \{\mathbf{w}^0\}$
4: while True do
5:
       $$\mathbf{u}^0 \leftarrow \underset{\mathbf{u} \in U^N}{\text{argmin}} \quad \max_{\mathbf{w}^0 \in W_{cand}} J(\xi(x^0, \mathbf{u}, \mathbf{w}^0)) \quad \text{s.t. } \forall \mathbf{w}^0 \in W_{cand},\ \xi(x^0, \mathbf{u}, \mathbf{w}^0) \models \varphi_s,$$
6:     if $\mathbf{u}^0$ == null then
7:         Return INFEASIBLE
8:     end if
9:
       $$\mathbf{w}^1 \leftarrow \text{argmin}_{\mathbf{w} \in W^N} \quad \rho^{\varphi}(\xi(x^0, \mathbf{u}^0, \mathbf{w}), 0) \quad \text{s.t. } \mathbf{w}^1 \models \varphi_e$$
10:    if $\rho^{\varphi}(\xi(x^0, \mathbf{u}^0, \mathbf{w}^1)) > 0$ then
11:        Return $\mathbf{u}^0$
12:    else
13:        $W_{cand} \leftarrow W_{cand} \cup \{\mathbf{w}^1\}$
14:    end if
15: end while
```

Once we have this scheme for synthesizing control for finite trajectories satisfying bounded specifications, we will use a receding horizon scheme for infinite trajectories.

We now describe the steps of Algorithm 3 in detail. In Step 2, we choose an initial instance $\mathbf{w}^0$ of an environment that satisfies the specification φ_e. We do so using the open-loop synthesis algorithm for bounded-time STL described in [147]. Our initial set of candidate environment inputs is a singleton, $W_{cand} = \{\mathbf{w}^0\}$ (Step 3). Then, in Step 5, we compute the optimal control input $\mathbf{u}^0$ with respect to this environment, such that the system specification φ_s is satisfied; this step also uses the solution in [147]. If the problem in Step 5 is infeasible, we know that there is a control input $\mathbf{w}^0 \in W_{cand}$ against which no control input can satisfy φ, so we can stop and return (Step 7). Otherwise, in Step 9, we find an environment $\mathbf{w}^1$ that satisfies φ_e, but also minimizes the robustness of satisfaction of φ for the control input $\mathbf{u}^0$. Essentially, this step tries to find an environment that falsifies the specification φ when the control input $\mathbf{u}^0$ is used. If the minimum robustness $\rho^{\varphi}(\xi(x^0, \mathbf{u}^0, \mathbf{w}^1))$ thus computed is positive, this implies $\forall \mathbf{w} \in W^N\ \xi(x^0, \mathbf{u}^0, \mathbf{w}) \models \varphi$, so we can return the control input $\mathbf{u}^0$ as our result in Step 11. Otherwise, we have generated a counterexample to $\mathbf{u}^0$ being the desired control input, i.e. an environment $\mathbf{w}^1$ that falsifies φ when $\mathbf{u}^0$ is used. We use this counterexample to guide our inductive synthesis in Step 13, by adding it to the set of environments to be considered in the next iteration.

We then resume execution of the while loop from Step 4.

Theorem 5. *If Algorithm 3 returns $\mathbf{u}^N \in U^N$, then $\forall \mathbf{w}^N \in W^N$, $\xi(x^0, \mathbf{u}^N, \mathbf{w}^N) \models \varphi$. If Algorithm 3 returns* `INFEASIBLE`*, then Problem 5 is infeasible.*

Note that Algorithm 3 does not fully solve Problem 5, because it does not always ensure cost-optimality of $\mathbf{u}^N$ with respect to all disturbances $\mathbf{w}^N \in W^N$ — the returned $\mathbf{u}^N$ is optimal with respect to a specific set of disturbances $W_{cand} \subseteq W^N$.

Since $|W_{cand}|$ grows by 1 at every iteration of the `while` loop, the MILP in Step 5 grows linearly with the number of iterations, as we duplicate constraints for each new counterexample. If W is finite, W_{cand} will converge, and Algorithm 3 is sound and complete.

If W is infinite, Algorithm 3 may never terminate. We can choose to execute a maximum number of iterations of the while loop before declaring the problem infeasible. However, there is actually more information available to us. In fact, we can solve the robust control problem (i.e. Problem 5) to within an arbitrary tolerance by sampling a finite number of independent and identically distributed (i.i.d.) $\mathbf{w}^N$. Esfahani et al. show the feasibility and performance bound of this approach for a class of non-convex MILP problems [49]. They prove that for this specific class of programs, the quality of the solution after drawing M i.i.d. samples can be lower-bounded. Here, we first show that we can have a similar bound by i.i.d. sampling, and then prove that Algorithm 3 performs strictly better than i.i.d. sampling. We first formulate Problem 5 as a feasibility problem by moving the objective to the set of constraints. This transformation facilitates defining a relaxed variant of Problem 5.

Problem 7. *Given a system of the form in equation (8.1), initial state x^0, trajectory length N, STL formula φ, cost function J, and threshold t compute:*

$$
\begin{aligned}
&\underset{\mathbf{u}^N}{\operatorname{argmin}} && 0 \\
&\textit{s.t.} && \forall \mathbf{w}^N \in W^N, \quad \xi(x^0, \mathbf{u}^N, \mathbf{w}^N) \models \varphi, \\
& && \forall \mathbf{w}^N \in \{\mathbf{w} \in W^N | \mathbf{w} \models \varphi_e\} \quad J(\xi(x^0, \mathbf{u}^N, \mathbf{w}^N)) \leq t.
\end{aligned}
$$

Now, we define an ϵ-relaxation of Problem 7, where each constraint is satisfied within an ϵ threshold.

Problem 8 (Relaxed STL Reactive Synthesis). *Given a system of the form in equation (8.1), initial state x^0, trajectory length N, STL formula φ, cost function J, and threshold t, compute:*

$$
\begin{aligned}
&\underset{\mathbf{u}^N}{\operatorname{argmin}} && 0 \\
&\textit{subject to} && \forall \mathbf{w}^N \in W^N, \quad \xi(x^0, \mathbf{u}^N, \mathbf{w}^N) \models_\epsilon \varphi, \\
& && \forall \mathbf{w}^N \in \{\mathbf{w} \in W^N | \mathbf{w} \models \varphi_e\} \quad J(\xi(x^0, \mathbf{u}^N, \mathbf{w}^N)) \leq t + \epsilon,
\end{aligned}
\tag{8.2}
$$

where the relaxed constraint $\xi(x^0, \mathbf{u}^N, \mathbf{w}^N) \models_\epsilon \varphi$ represents the ϵ-satisfaction of the STL formula φ, which is equivalent to $\rho^\varphi(\xi(x^0, \mathbf{u}^N, \mathbf{w}^N)) > -\epsilon$.

We define a new robustness function that takes into account both the objective J and the constraints φ:

$$\rho^{\varphi}_{t,J}(\xi(x^0,\mathbf{u}^N,\mathbf{w}^N)) = \min(\rho^{\varphi}(\xi(x^0,\mathbf{u}^N,\mathbf{w}^N)), t - J(\xi(x^0,\mathbf{u}^N,\mathbf{w}^N)). \tag{8.3}$$

The solution to Problem 8 can be found by slightly modifying Algorithm 3, where satisfaction of formulas are replaced by ϵ-satisfaction for the feasibility problem. We call this modified algorithm the *Relaxed CEGIS Algorithm*.

Algorithm 4 Relaxed CEGIS Algorithm for Problem 8

```
1:  **Input:** $\xi, x_0, N, \varphi, J, t$
2:  Let $\mathbf{w}^0 = (w^0_1, w^0_2, ...w^0_{N-1})$, s.t. $\mathbf{w}^N \models \varphi_e$
3:  $W_{cand} = \{\mathbf{w}^0\}$
4:  **while** True **do**
5:
        $$\begin{aligned} \mathbf{u}^0 \leftarrow \underset{\mathbf{u}\in U^N}{\text{argmin}} \quad & 0 \\ \text{subject to} \quad & \forall \mathbf{w}^0 \in W_{cand},\ \xi(x^0,\mathbf{u},\mathbf{w}^0) \models \varphi_s, \\ & \forall \mathbf{w}^0 \in W_{cand},\ J(\xi(x^0,\mathbf{u},\mathbf{w}^0)) \leq t, \end{aligned} \tag{8.4}$$
6:      **if** $\mathbf{u}^0 ==$ null **then**
7:          **return** INFEASIBLE
8:      **end if**
9:
        $$\begin{aligned} \mathbf{w}^1 \leftarrow \underset{\mathbf{w}\in W^N}{\text{argmin}} \quad & \rho^{\varphi}_{t,J}(\xi(x^0,\mathbf{u}^0,\mathbf{w}),0) \\ \text{subject to} \quad & \mathbf{w}^1 \models \varphi_e \end{aligned} \tag{8.5}$$
10:     **if** $\rho^{\varphi}_{t,J}(\xi(x^0,\mathbf{u}^0,\mathbf{w}^1)) > -\epsilon$ **then**
11:         **return** $\mathbf{u}^0$
12:     **else**
13:         $W_{cand} \leftarrow W_{cand} \cup \{\mathbf{w}^1\}$
14:     **end if**
15: **end while**
```

Remark 7. *The robustness function of an STL formula ρ^{φ} is Lipschitz continuous, and its Lipschitz constant L_{φ} can be computed from the Lipschitz constant of the atomic propositions of φ by simply noting that minimum and maximum do not increase the Lipschitz constant. For example, let $\varphi = \mathbf{G}_{[0,\infty)}(s(t) > 5) \wedge \mathbf{F}_{[0,\infty)}(2s(t) > 20)$, then the Lipschitz constant L_{φ} is the maximum of the Lipschitz constants of the atomic predicates, i.e., $L_{\varphi} = \max(1,2) = 2$.*

Remark 8. *Under the assumption that the objective function J is Lipschitz continuous, with Lipschitz constant L_J, we define the Lipschitz constant of $\rho^{\varphi}_{t,J}$ to be the maximum of L_φ and L_J, i.e., $L = \max(L_J, L_\varphi)$.*

Lemma 3. *After running the Relaxed CEGIS Algorithm for k time steps, the pairwise distance between $\mathbf{w}^i$ and $\mathbf{w}^j \in \{\mathbf{w}^1, \dots, \mathbf{w}^k\}$ is at least $\frac{\epsilon}{L}$, where L is the Lipschitz constant of the robustness function $\rho^{\varphi}_{t,J}$.*

Proof. We use a proof by contradiction method for Lemma 3. Without loss of generality, we assume that $i < j$. Let $\mathbf{u}^j$ be the control input selected at iteration j, and $\mathbf{w}^i$ be the disturbance input selected at iteration i. Then, we have $\rho^{\varphi}_{t,J}(\xi(x^0, \mathbf{u}^j, \mathbf{w}^i)) > 0$. Further, we assume the distance between $\mathbf{w}^i$ and $\mathbf{w}^j$ is less than $\frac{\epsilon}{L}$, i.e., $|\mathbf{w}^i - \mathbf{w}^j| < \frac{\epsilon}{L}$. Therefore, by Lipschitz continuity of $\rho^{\varphi}_{t,J}$, we conclude:

$$\rho^{\varphi}_{t,J}(\xi(x^0, \mathbf{u}^j, \mathbf{w}^i)) > \frac{-\epsilon}{L} \times L = -\epsilon \tag{8.6}$$

This contradicts our assumption, since if $\rho^{\varphi}_{t,J}(\xi(x^0, \mathbf{u}^j, \mathbf{w}^i)) > -\epsilon$, Step 10 of the Relaxed CEGIS Algorithm 4 would have returned $\mathbf{u}^j$. □

Theorem 6. *If W_{cand} is an $\frac{\epsilon}{L}$ cover of W^N, then solving Problem 7 (or equivalently Problem 5), where W^N is replaced by W_{cand} provides a solution to the Relaxed STL Reactive Synthesis problem (Problem 8).*

Proof. Suppose solving Problem 7 with a finite number of $\mathbf{w} \in W_{cand}$ returns $\mathbf{u}^*$. Since W_{cand} is an $\frac{\epsilon}{L}$ cover of W^N, for every $\mathbf{w} \in W^N$, there exists $\mathbf{w}' \in W_{cand}$ and $|\mathbf{w} - \mathbf{w}'| < \frac{\epsilon}{L}$. Since $\mathbf{u}^*$ is a solution of Problem 7, we have $\rho^{\varphi}_{t,J}(\xi(x^0, \mathbf{u}^*, \mathbf{w}')) > 0$. By Lipschitz continuity:

$$|\mathbf{w} - \mathbf{w}'| < \frac{\epsilon}{L} \to |\rho^{\varphi}_{t,J}(\xi(x^0, \mathbf{u}^*, \mathbf{w})) - \rho^{\varphi}_{t,J}(\xi(x^0, \mathbf{u}^*, \mathbf{w}'))| < \epsilon. \tag{8.7}$$

Therefore, $\rho^{\varphi}_{t,J}(\xi(x^0, \mathbf{u}^*, \mathbf{w})) > -\epsilon$ for any $\mathbf{w} \in W^N$, which provides a solution to the Relaxed STL Reactive Synthesis problem (Problem 8). □

Theorem 7. *Let $W^N = [0,1]^N$. Then, an $\frac{\epsilon}{L}$ cover of W^N is obtained with $\Theta((\frac{L}{\epsilon})^N \log(\frac{L}{\epsilon}))$ i.i.d. samples.*

Proof. We first prove that we can lower bound the number of i.i.d. samples by $\Omega((\frac{L}{\epsilon})^N \log(\frac{L}{\epsilon}))$. We divide the cube $[0,1]^N$ into $(\frac{L}{2\epsilon})^N$ subcubes of side length $\frac{2\epsilon}{L}$, where the center of each subcube is $\frac{\epsilon}{L}$ far from the side of the subcube. To cover the centers of the subcubes, we need to have selected at least one point from each subcube. Therefore, the problem reduces to the *coupon collector's problem* [125], where the number of coupons is the number of subcubes, i.e., $\Theta((\frac{L}{\epsilon})^N)$ (for a constant N). Then, the expected number of samples required to cover the centers of the subcubes is $\Omega((\frac{L}{\epsilon})^N \log(\frac{L}{\epsilon}))$ with i.i.d. sampling. So this implies that to find an $\frac{\epsilon}{L}$ cover of W^N we need $\Omega((\frac{L}{\epsilon})^N \log(\frac{L}{\epsilon}))$.

Now, we show that this bound is also an upper bound. We now choose the subcubes with side length $\frac{\epsilon}{\sqrt{N}\cdot L}$. The diameter of such a subcube is $\frac{\epsilon}{L}$. Therefore, if we pick one point from each subcube, these points form an $\frac{\epsilon}{L}$ cover. Then, using the coupon collector's problem, we only need $O((\frac{L}{\epsilon})^N \log(\frac{L}{\epsilon}))$ samples.

As we have shown both the upper and lower bounds, we only need $\Theta((\frac{L}{\epsilon})^N \log(\frac{L}{\epsilon}))$ samples to cover the cube $[0,1]^N$. □

Remark 9. *If $W^N = [0,1]^N$ and W_{cand} is selected by i.i.d. sampling , then $O((\frac{L}{\epsilon})^N \log(\frac{L}{\epsilon}))$ samples are enough to ensure that the solution to Problem 5 with W^N replaced with W_{cand} is a solution of Problem 8.*

Theorem 8. *The Relaxed CEGIS Algorithm 4 requires at most $O((\frac{L}{\epsilon})^N)$ samples (under the assumption that $W^N = [0,1]^N$) to terminate.*

Proof. We can divide the cube $W^N = [0,1]^N$ to subcubes of side length $\frac{\epsilon}{\sqrt{N}\cdot L}$. By Lemma 3, we only select one $\mathbf{w}^i \in W_{cand}$ for each subcube during the CEGIS loop. Therefore, the number of samples is at most the number of subcubes, i.e., $O((\frac{L}{\epsilon})^N)$. □

Theorem 9. *Any solution returned by Algorithm 4 is a feasible solution for Problem 8. Further, if Problem 7 has a feasible solution, Algorithm 4 will never return infeasible.*

Proof. Step 10 of Algorithm 4 enforces that $\rho^{\varphi}_{t,J}(\xi(x^0, \mathbf{u}^0, \mathbf{w}^1)) > -\epsilon$; therefore, any solution returned satisfies the feasibility of Problem 8. In addition, if Problem 7 is feasible, Step 5 of Algorithm 4 can never return infeasible. □

Remark 10. *If W^N is bounded, then Algorithm 4 terminates in a finite number of steps. This is because a bounded W^N can be covered by balls of diameter $\frac{\epsilon}{L}$, and W_{cand} cannot take more than one point in each of these balls.*

Remark 11. *One might be discouraged by the seemingly modest improvement of the Relaxed CEGIS Algorithm 4 over i.i.d. sampling. In fact, one might think that this improvement is offset by the more complicated and time-consuming way of choosing $\mathbf{w}$. However, we solve the optimization problem of choosing $\mathbf{u}$ by reducing the problem to a MILP whose size scales linearly in the size of W_{cand}. Therefore, keeping this length of W_{cand} as small as possible is beneficial.*

In practice, solving the problem in Step 5 becomes expensive as W_{cand} grows, in particular because the objective is now non-linear. While state-of-the-art MILP solvers e.g. Gurobi[1] handle nonlinear objective functions efficiently, we can preserve the difficulty of the problem at each iteration by setting $W_{cand} = \{\mathbf{w}^1\}$ in Step 13 instead of growing the set of candidates. This breaks completeness even for finite sets W, since we may oscillate between two disturbances, but preserves soundness with respect to the satisfaction of φ, while allowing faster solutions at each iteration of the loop.

[1]http://www.gurobi.com/

In the case study described in Section 8.5, we find that a few number of iterations through the while loop suffices to either find a satisfying control input or render the problem infeasible.

8.4 Receding Horizon Controller Synthesis in an Adversarial Environment

In this section, we will describe a solution to Problem 6 by adding STL constraints to a receding horizon control framework. At each step t of the computation, we will employ the CEGIS approach in Section 8.3 to find a finite trajectory of fixed horizon length H, such that the trajectory accumulated over time satisfies φ.

Note that this problem is relatively simple for bounded-time STL formulas φ, as described in [147]. Here the length of the horizon H is chosen to be at least the bound of formula φ. Then, at time step 0, we synthesize control $\mathbf{u}^{H,0}$ using the formulation in Section 8.3, and execute only the first time step $u_0^{H,0}$; we then observe $w_0^{H,0}$ and x_1. Then at the next step, we solve for $\mathbf{u}^{H,1}$, while constraining the values of $u_0^{H,1} = u_0^{H,0}, w_0^{H,1} = w_0^{H,0}$ in the MILP, and retaining the STL constraints on the trajectory up to time H. Keeping track of the history in this manner ensures that the formula is satisfied over the length-H prefix of the trajectory, while solving for $\mathbf{u}^{H,t}$ at every time step t.

Suppose that we have a specification $\psi = \mathbf{G}\varphi$, where φ is a bounded-time formula with bound H. In this case, we can stitch together trajectories of length H using a receding horizon approach to produce an infinite computation that satisfies the STL formula. At each step of the receding horizon computation, we search for a finite trajectory of horizon length $2H$, keeping track of the past values and robustness constraints necessary to determine satisfaction of ψ at every time step in the trajectory.

First we define a procedure:

$$\texttt{CEGIS}^*(\xi, x^0, N, \psi = \mathbf{G}\varphi, J, \mathbf{P}^H, \mathbf{u}^k_{old}) \tag{8.8}$$

that takes additional inputs $\mathbf{P} = \{P^0, P^1, ..., P^{H-1}\}$ and $\mathbf{u}^k_{old} = u^k_{old_0} u^k_{old_1} ... u^k_{old_{k-1}}$, and is identical to Algorithm 3, except that the optimization problem in Step 5 is solved with the added constraints:

$$\begin{aligned} &\rho^{\varphi}(\xi(x^0, \mathbf{u}, \mathbf{w}^0), i) > P_i \quad \forall i \in [0, H-1] \\ &u_i = u^k_{old_i} \qquad\qquad\qquad\quad\; \forall i < k \end{aligned} \tag{8.9}$$

Algorithm 3 (CEGIS) solves reactive synthesis for bounded horizon formulas. We are designing Algorithm 5 to deal with unbounded formulas, by invoking bounded-horizon synthesis at each time step. To ensure soundness of this infinite-horizon synthesis algorithm, some history needs to be carried forth from one horizon to another to ensure consistency between the newly synthesized inputs and those produced in previous steps.

This is achieved by the two additional arguments of `CEGIS`* and the corresponding added constraints. The first constraint enforces satisfaction of φ at all time steps $i \in [0, H-1]$. The second constraint fixes the first k values of the newly computed input to values computed in the previous time step.

Given `CEGIS`*, we define a receding horizon control procedure as in Algorithm 5. At each time step, we compute control inputs over a horizon of $2H$.

Algorithm 5 RHC Algorithm for Problem 6

```
1:  **Input:** $\xi, x^0, \psi = \mathbf{G}\varphi, J$
2:  Let $M$ be a large positive constant.
3:  Let $H$ be the bound of $\varphi$.
4:  Set $P^0 = 0$ and $P^i = -M \quad \forall\, 0 < i \leq H$.
5:  Compute $\mathbf{u}^0 = u_0^0 u_1^0 .... u_{2H-1}^0$ as:
        $\mathbf{u}^0 \leftarrow \texttt{CEGIS}^*(\xi, x^0, 2H, \mathbf{G}_{[0,H]}\varphi, J, \mathbf{P}^H, \emptyset)$
6:  **for** $k = 1; k < H; k = k+1$ **do**
7:      Set $\mathbf{u}_{old}^k = u_0^0 u_1^1 u_2^2 ... u_{k-1}^{k-1}$.
8:      Set $P^i = 0$ for $0 \leq i \leq k$, $P^i = -M \quad \forall k < i \leq H$.
9:      Compute $\mathbf{u}^k = u_0^k u_1^k .... u_{2H-1}^k$ as:
            $\mathbf{u}^k \leftarrow \texttt{CEGIS}^*(\xi, x^k, 2H, \mathbf{G}_{[0,H]}\varphi, J, \mathbf{P}^H, \mathbf{u}_{old}^k)$
10: **end for**
11: **while** True **do**
12:     Set $\mathbf{u}_{old}^k = u_1^{k-1} u_2^{k-1} u_3^{k-1} ... u_H^{H-1}$.
13:     Set $P^i = 0$ for $0 \leq i \leq H$.
            $\mathbf{u}^k \leftarrow \texttt{CEGIS}^*(\xi, x^k, 2H, \mathbf{G}_{[0,H]}\varphi, J, \mathbf{P}^H, \mathbf{u}_{old}^k)$
14:     $k = k+1$
15: **end while**
```

Algorithm 5 has two phases, a *transient* phase (Lines 4-10) and a *stationary* phase (Lines 11-14). The transient phase applies until an initial control sequence of length H has been computed, and the stationary phase follows. In the transient phase, the number of stored previous inputs ($\mathbf{u}_{old}^k$) as well as the number of time steps at which formula φ is enforced (i.e. time steps for which $P_i = 0$) grows by one at each iteration, until they both attain a maximum of H at iteration H. Every following iteration uses a window of size H for stored previous inputs, and sets all $P_i = 0$. The size-H window of previously-computed inputs advances forward one step in time at each iteration after step H. In this manner, we keep a record of the previously computed inputs required to ensure satisfaction of φ up to H time steps in the past.

Theorem 10. *Let $\mathbf{u}^*$ be the infinite sequence of control inputs generated by setting $u^*_k = u^k_H$, where $\mathbf{u}^k = u^k_0 u^k_1 ... u^k_{2H-1}$ is the control input sequence of length $2H$ generated by Algorithm 5 at time t_k. Then $\forall \mathbf{w} \in W^\omega,\ \xi(x^0, \mathbf{u}^*, \mathbf{w}) \models \psi$.*

Proof. Since H is the bound of φ, the satisfaction of φ at time k is established by the control inputs $u^*_k \dots u^*_{k+H-1}$.

At time $k+H$,

$$\begin{aligned}
\mathbf{u}^{k+H}_{old} &= u^{k+H}_0 u^{k+H}_1 u^{k+H}_2 ... u^{k+H}_{H-1} \\
&= u^{k+H-1}_1 u^{k+H-1}_2 u^{k+H-1}_3 ... u^{k+H-1}_{H-1} u^{k+H-1}_H \\
&= u^{k+H-2}_2 u^{k+H-2}_3 u^{k+H-2}_4 ... u^{k+H-2}_H u^{k+H-1}_H \\
&= \cdots \\
&= u^k_H u^{k+1}_H u^{k+2}_H ... u^{k+H-1}_H \\
&= u^*_k u^*_{k+1} u^*_{k+2} ... u^*_{k+H-1}
\end{aligned}$$

and so all the inputs required to determine satisfaction of φ at time k have been fixed. Moreover, if $\mathbf{u}^{k+H}$ is successfully computed, then by the correctness of Algorithm 3, $\mathbf{u}^{k+H}_{old}$ has the property that $\forall \mathbf{w}^H \in W^H,\ \xi(x^t, \mathbf{u}^{k+H}_{old}, \mathbf{w}^H) \models \varphi$. Since $u^k_H u^{k+1}_H u^{k+2}_H ... u^{k+H-1}_H = \mathbf{u}^{k+H}_{old}$, we see that $\forall \mathbf{w}^H \in W^H,\ \xi(x^k, u^*_k \dots u^*_{k+H-1}, \mathbf{w}^H) \models \varphi$.

It follows that $\forall \mathbf{w}^\omega \in W^\omega,\ \xi(x^0, \mathbf{u}^*, \mathbf{w}) \models \varphi$. □

We have therefore shown how a control input can be synthesized for infinite sequences satisfying ψ, by repeatedly synthesizing control for sequences of length $2H$. A similar approach applies for formulas $\mathbf{F}\varphi$ and $\varphi\ \mathbf{U}\ \psi$, where φ, ψ are bounded-time.

8.5 Autonomous Driving in Nondeterministic Environments

We now consider the problem of controlling an autonomous vehicle operating in the presence of other, potentially adversarial vehicles.

In this example, two moving vehicles approach an intersection, which they must cross. We let the red car in Figure 8.1 be the *ego* vehicle (the vehicle we control), and the black car be part of the environment. We define the state space using a simplified 6-dimensional model, with the position of the two vehicles $((x^{\text{ego}}, y^{\text{ego}}), (x^{\text{adv}}, y^{\text{adv}}))$ and the velocity of the two ($v^{\text{ego}} = \dot{y}^{\text{ego}}, v^{\text{adv}} = \dot{x}^{\text{adv}}$) in ms^{-1} as state variables, and the acceleration ($a^{\text{ego}} = \dot{v}^{\text{ego}}$) of the ego vehicle as a single input. The disturbance is the acceleration of the

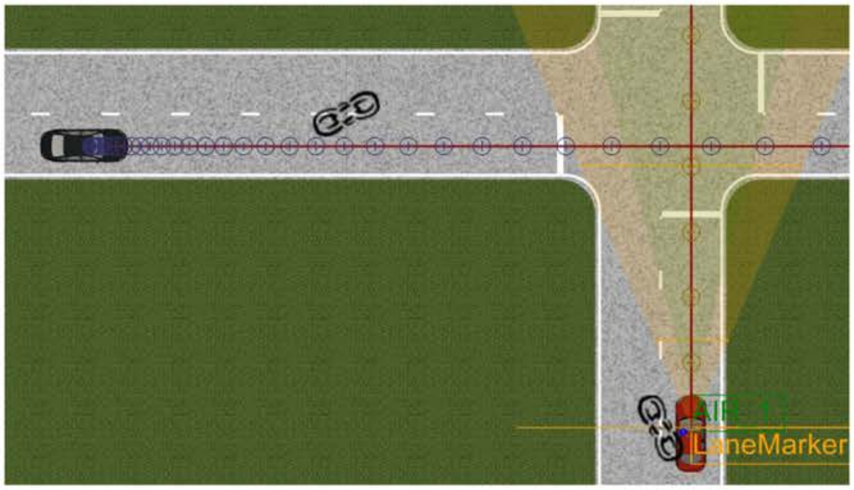


Figure 8.1: Two vehicles crossing an intersection simultaneously. The red car is the *ego* vehicle, which we control. The black car is part of the adversarial environment.

environment vehicle ($a^{\text{adv}} = \dot{v}^{\text{adv}}$), which is allowed to take values in a bounded range. Thus:

$$\mathbf{x}_k = \begin{bmatrix} x_k^{\text{ego}} & y_k^{\text{ego}} & x_k^{\text{adv}} & y_k^{\text{adv}} & v_k^{\text{ego}} & v_k^{\text{adv}} \end{bmatrix}^\top \tag{8.10}$$

$$\mathbf{u} = a_k^{\text{ego}} \quad \mathbf{w} = a_k^{\text{adv}} \tag{8.11}$$

We assume each vehicle has the dynamics of a double integrator:

$$\begin{bmatrix} \dot{x}^{\text{ego}} \\ \dot{y}^{\text{ego}} \\ \dot{v}^{\text{ego}} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x^{\text{ego}} \\ y^{\text{ego}} \\ v^{\text{ego}} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \mathbf{u} \tag{8.12}$$

$$\begin{bmatrix} \dot{x}^{\text{adv}} \\ \dot{y}^{\text{adv}} \\ \dot{v}^{\text{adv}} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x^{\text{adv}} \\ y^{\text{adv}} \\ v^{\text{adv}} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \mathbf{w} \tag{8.13}$$

Our specification in this example is that there should be no collisions at the intersection between the two vehicles, and that the ego vehicle's speed should be close to $1ms^{-1}$. Here the disturbance $\mathbf{w}$ is the acceleration of the adversary, whose value is assumed to be close to a reference value, $\mathbf{w}^{\text{ref}}$. We use the following STL formulas:

$$\varphi_e = \mathbf{G}(|\mathbf{w} - \mathbf{w}^{\text{ref}}| < 0.1)$$
$$\varphi_s = \mathbf{G}(|y_k^{\text{ego}} - x_k^{\text{adv}}| < 2) \rightarrow \mathbf{G}_{[0,2]}(|v_k^{\text{ego}}| < 0.1)$$

The formula φ_s specifies that whenever y_k^{ego} is close to x_k^{adv}, i.e. within the range of $2m$, the ego vehicle should come to a stop ($|v_k^{\text{ego}}| < 0.1$) for a short period of time (2s). Figure 8.1, shows that the two vehicles will be close only when they are in the vicinity of the intersection. We expect the ego vehicle to stop at the intersection in order to allow the adversary to cross. In addition, we optimize the following cost function, which encourages the ego vehicle's speed to be close to $1ms^{-1}$.

$$J(\xi(x^k, \mathbf{u}^H, \mathbf{w}^H)) = \sum_{l=0}^{H-1} ||v_{k+l}^{\text{ego}} - 1|| \tag{8.14}$$

Figure 8.2 illustrates the result of applying Algorithm 5 to synthesize control inputs for the ego vehicle. The first plot shows the position of the two vehicles, x_k^{adv} and y_k^{ego} (in m). The ego vehicle starts with a negative value on its y-axis $y_0^{\text{ego}} < 0$, and the adversary starts with a positive x-value $x_0^{\text{adv}} > 0$. Here the origin represents the middle of the intersection: at any time k if $y_k^{\text{ego}} = x_k^{\text{adv}} = 0$, the two cars have collided. The synthesized control input should therefore avoid such a collision, and the two vehicles should not be at location 0 or its vicinity ($|y_k^{\text{ego}} - x_k^{\text{adv}}| < 2$) at the same time.

As seen in the first and second subplots in Figure 8.2, at time $t = 8s$, the ego vehicle stops at its current position in order to avoid collision with the adversary car. The vehicle proceeds after a short stop to let the adversary pass. The third subplot shows the velocity of the two vehicles, and the fourth plot represents the acceleration. Notice that the velocity of the ego vehicle stabilizes at $1ms^{-1}$ at most times as long as it avoids any collisions. The accelerations shown in the fourth plot include the control input synthesized using Algorithm 5, and the disturbance, i.e., the acceleration of the adversary.

8.6 Chapter Summary

The main contribution of this chapter is a CEGIS procedure for synthesis of reactive controllers for systems satisfying STL specifications. We further showed our CEGIS technique can be used in a receding horizon control framework, and we synthesized strategies for systems that must satisfy a set of desired STL specifications. Our approach uses an optimization framework with a given domain-specific cost function in the presence of adversarial environments. We have also presented experimental results for synthesizing controllers in a simplified driving scenario, where the controller satisfies the desired properties despite nondeterministic adversarial environments.

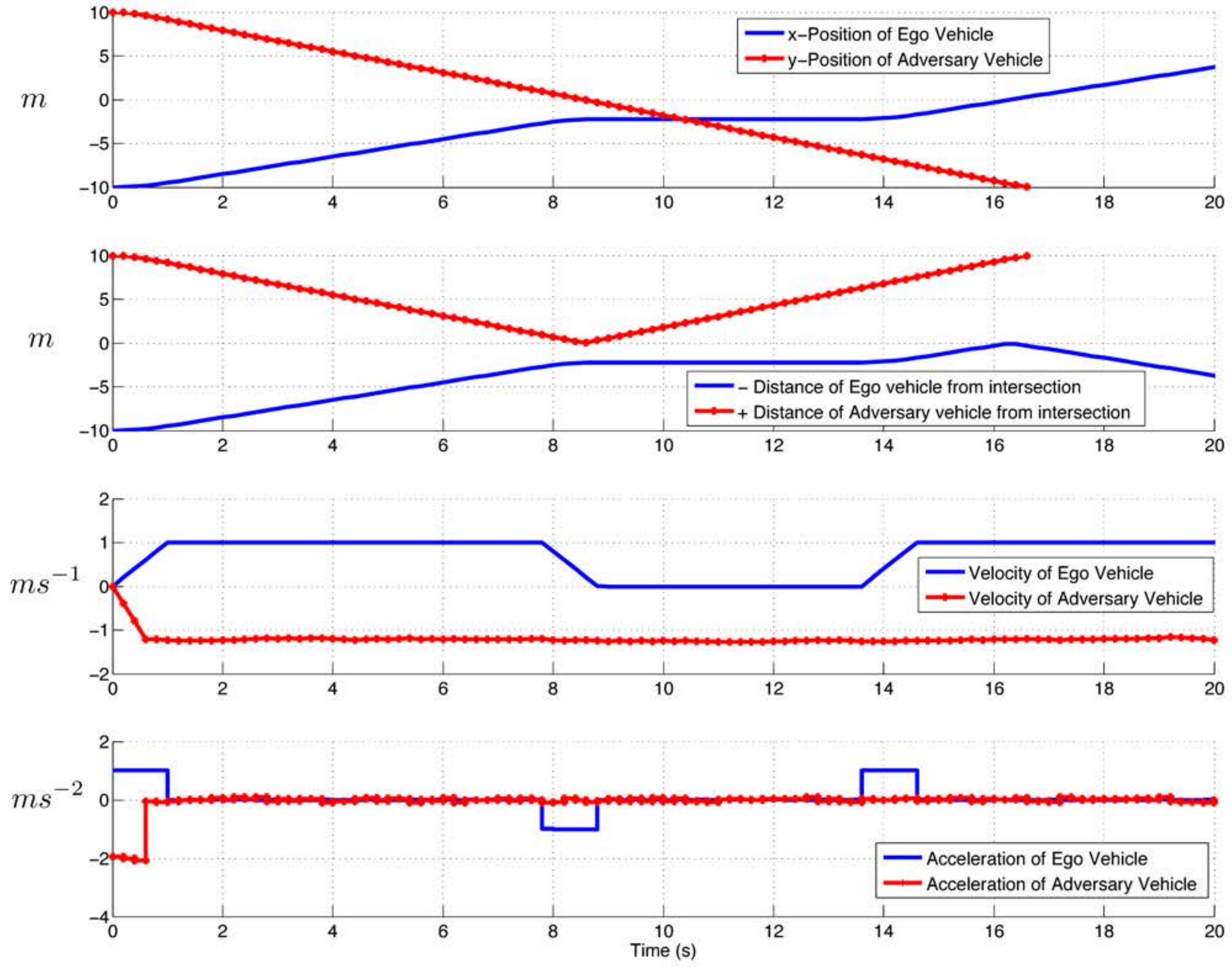


Figure 8.2: Plot of position, velocity and acceleration of the ego and adversary vehicles. The second plot from the top shows the distance (in m) of each vehicle from the intersection. While the adversary vehicle drives straight through the intersection at a constant speed, the ego vehicle stops at t=8s, when it is around 2.5m from the intersection, then resumes moving around t=16s, thus avoiding collision.

Chapter 9

Safe Control under Uncertainty

Achieving safe control under *uncertain* models for human-robot systems is challenging. For example, any safe control strategy for quadcopters need to incorporate predictive information about wind gusts and any associated uncertainty in such predictions. Similarly, in the case of autonomous driving, the controller needs a probabilistic predictive model about the other vehicles on the road to avoid collisions. Without a model of uncertainty that characterizes all possible outcomes, there is no hope in giving guarantees about the safety of the synthesized controller.

The field of machine learning (ML) has a rich set of tools that can characterize uncertainties. Specifically, Bayesian graphical models [85] have been popular in modeling uncertainties arising in scenarios common to robotics. For example, one of the common strategies is to build classifiers or predictors based on acquired sensor data. It is appealing to consider such predictors in achieving safe control of dynamical systems. However, it is almost impossible to guarantee a prediction system that works perfectly at all times. Consequently, we need to devise control methodologies aware of such limitations imposed by the ML systems. Specifically, we need to build a framework that is capable of achieving safe control by being aware of when the prediction system would work or fail.

In this chapter, we propose a framework for achieving safe control, when machine learning models are employed to make predictions based on sensed signals. The heart of our framework is the novel *Probabilistic Signal Temporal Logic* (PrSTL) that allows us to express safety constraints by considering the predictive models and their associated uncertainties. This logic allows specifications that embed Bayesian classification methods via *probabilistic predicates* that take random variables as parameters, thereby resulting in a powerful framework that can reason about safety under uncertainty. One of the main advantages of using Bayesian classification models is the fact that the predictions provided are full distributions associated with the quantity of interest as opposed to a point estimate. For example, a classical machine learning method might just provide a value for wind speed; however, under the Bayesian paradigm we would be recovering an entire probability distribution over all possible wind profiles. Finally, another distinguishing aspect of our framework is that these probabilistic predicates are adaptive: as the system sees more and more data, the inferred distribution over the latent variables of interest can change leading to change in the predicates themselves.

Previous efforts for achieving safe control either operate under deterministic environments or model uncertainty only as part of the dynamics of the system [72]. These approaches lack clear connections to various sources of uncertainty present in the environment. Specifically, there is no clear understanding of how uncertainty arising due to sensing and classification could be incorporated while reasoning about safe controllers.

Over the years researchers have proposed different approaches for safe control of dynamical systems. Designing controllers under reachability analysis and safe learning are well-studied methods that allow specifying safety and reachability properties, while learning the optimal strategy online [122, 121, 67, 13, 5]. However, finding the reachable set is computationally expensive, which makes these approaches impractical for most interesting tasks. Controller synthesis under temporal specifications such as Linear Temporal Logic (LTL) allows expressing interesting properties of the system and environment, e. g., safety, liveness, response, stability, and has shown promising results [139, 97, 87, 184, 88, 140]. However, synthesis for LTL requires time and space discretization, which suffers from the curse of dimensionality. Also, while such approaches are effective at high level planning, they are unsuitable for lower level control of dynamical systems. Recently, synthesis for Signal Temporal Logic (STL), which allows real-valued, dense-time properties have been studied in receding horizon settings [148, 89, 65]. One downside of specifying properties in STL or LTL is that the properties of the system and environment have to be expressed deterministically. Full knowledge of the exact parameters and bounds of the specification is an unrealistic assumption for most robotics applications, where the system interacts with uncertain environments. Related to our work is the paradigm of Markov Logic Networks [150] that aim to induce probability distributions over possible worlds by considering weighted logical formulae. However, it is not clear how such networks can be used for controller synthesis. The proposed framework instead considers formulae parameterized by random variables, thereby inducing probability distribution over the set of possible formulae. Further, we show how such formalism can be embedded in a receding horizon MPC for controller synthesis. In robust control, uncertainty is modeled as part of the dynamics, and the optimal strategy is found for the worst case disturbance, which can be a conservative assumption [96, 178]. More recently, the uncertainty is modeled in a chance constrained framework showing promising results for urban autonomous driving [105, 176, 29, 36]. Considering uncertainties while satisfying temporal logic requirements has recently been explored for controller synthesis and verification [157, 168, 60, 61, 144, 83]. Leahy et al. maximize information gain in a distributed setting to reduce the uncertainty over belief of every state; however, the uncertainty is not considered as part of the specification [103]. To best of our knowledge, none of the previous studies consider scenarios, where the uncertainty and confidence in properties are originated from sensors, predictors and classifiers, and are formalized as part of the property.

In this chapter, we aim to alleviate these issues by defining a probabilistic logical specification framework that has the capacity to reason about safe control strategies by embedding various predictions and their associated uncertainty. Specifically, our contributions in this chapter are:

- Framework for safe control under uncertainty.
- Formally define PrSTL, a logic for expressing probabilistic properties that can embed Bayesian graphical models.
- Solve a receding horizon control problem to satisfy PrSTL specifications using Mixed Integer SDPs.
- A toolbox implementing the framework and experiments in autonomous driving and control of quadrotors.

9.1 Bayesian Classification to Model Uncertainty:

Probability theory provides a natural way to represent uncertainty in the environment and recent advances in machine learning have relied on Bayesian methods to infer distributions over latent phenomenon of interest [64, 85]. We focus on Bayesian classifiers, which unlike other optimization based methods, provide entire distributions over the predictions. Such predictive distributions characterize the uncertainty present in the system and are crucial for achieving safe control. Formally, given a set of training points $\mathbf{X}_L = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, with observations $\mathbf{t}_L = \{t_1, \dots, t_n\}$, where $t_i \in \{+1, -1\}$, we are interested in finding a hyperplane $\mathbf{w}$ that separates the points belonging to the two classes according to $\text{sgn}(\mathbf{w}^T\mathbf{x})$. Under the Bayesian paradigm, we look for the distribution:

$$\begin{aligned} &p(\mathbf{w}|\mathbf{X}_L, \mathbf{t}_L) = p(\mathbf{w}) \cdot p(\mathbf{t}_L|\mathbf{X}_L, \mathbf{w}) = \\ &p(\mathbf{w}) \prod_i p(t_i|\mathbf{w}, \mathbf{x}_i) = p(\mathbf{w}) \prod_i \mathbb{I}\left[\text{sgn}(\mathbf{w}^T\mathbf{x}_i) = t_i\right]. \end{aligned} \tag{9.1}$$

The first line in the above equation stems from the Bayes rule, and the second line simply exploits the fact that given the classifier $\mathbf{w}$ the labels for each of the points in the dataset are independent. The expression $\mathbb{I}[\cdot]$ is an indicator function, which evaluates to 1, when the condition inside the brackets holds. Thus, equation (9.1) starts from a prior $p(\mathbf{w})$ over the classifiers and eventually by incorporating the training data points, infers a posterior distribution over the set of all the classifiers that respect the observed labels and the points. Given these statistical dependencies among the various variables, Bayesian inference techniques [120, 21, 12] aim to infer $p(\mathbf{w}|\mathbf{X}_L, \mathbf{t}_L)$ as a Gaussian distribution $\mathcal{N}(\mathbf{w}; \bar{\mathbf{w}}, \Sigma)$. Linear classification of a test data point $\mathbf{w}^T\mathbf{x}$ results in a Gaussian distribution of the prediction with the mean $\mathbf{w}^T\mathbf{x}$ and the variance $\mathbf{x}^T\Sigma\mathbf{x}$. Similarly, for the case of Bayesian linear regression the same procedure can be followed, albeit with continuous target variables $t \in \mathbb{R}$. These Bayesian linear classifiers and regressors are a fairly rich class of models, and have similar or better representation capabilities as kernel machines [179]. In this chapter, we specifically aim to incorporate such rich family of classification models for safe control.

9.2 Probabilistic Controller Synthesis Problem

We propose *Probabilistic Signal Temporal Logic* (PrSTL) that enables expression of uncertainty over the latent variables via probabilistic specifications. The key idea is to incorporate random variables in predicates, and then express temporal and Boolean operations on such predicates. The proposed logic provides an expressive framework for defining safety conditions under a wide variety of uncertainties, including ones that arise from application of machine learning classifiers.

The core ingredient in this chapter is the insight that when the uncertainty over the random variable is reasoned out in a Bayesian framework, we can use the inferred probability distributions to efficiently derive constraints from the PrSTL specifications. We provide a novel solution for synthesizing controllers for dynamical systems given different PrSTL properties. An interesting aspect of this framework is that the PrSTL formulae can evolve at every step. For example, a classifier associated with the dynamical system can continue to learn with time, thereby changing the inferred probability distributions on the latent random variables.

Probabilistic Signal Temporal Logic

PrSTL supports probabilistic temporal properties on real-valued, dense-time signals. Specifically, $(\xi, t) \models \varphi$ denotes the signal ξ satisfies the PrSTL formula φ at time t. We introduce the notion of a probabilistic atomic predicate $\lambda_{\alpha_t}^{\epsilon_t}$ of a PrSTL formula that is parameterized with a time-varying random variable α_t drawn from a distribution $p(\alpha_t)$:

$$(\xi, t) \models \lambda_{\alpha_t}^{\epsilon_t} \iff P(\lambda_{\alpha_t}(\xi(t)) < 0) > 1 - \epsilon_t. \tag{9.2}$$

Similar to STL (see Chapter 2.3), with a slight abuse of notation, we let $\lambda_{\alpha_t}^{\epsilon_t}$ denote the probabilistic atomic predicate, and λ_{α_t} be a function of the signal $\xi(t)$. Here, $P(\cdot)$ represents the probability of the event, and $1 - \epsilon_t$ defines the *tolerance* level in satisfaction of the probabilistic properties. The parameter $\epsilon_t \in [0, 0.5]$ is a small time-varying positive number and represents the threshold on satisfaction probability of $\lambda_{\alpha_t}(\xi(t)) < 0$. Small values of ϵ_t favors high tolerance satisfaction of formulas, which also facilitates solving the controller synthesis problem as discussed later in this section. A signal $\xi(t)$ satisfies the PrSTL predicate $\lambda_{\alpha_t}^{\epsilon_t}$ with confidence $1 - \epsilon_t$ if and only if:

$$\int_{\alpha_t} \mathbb{I}[\lambda_{\alpha_t}(\xi(t)) < 0]\, p(\alpha_t)\, d\alpha_t > 1 - \epsilon_t. \tag{9.3}$$

Here, $\mathbb{I}[\cdot]$ is an indicator function, and the equation marginalizes out the random variable α_t with the probability density $p(\alpha_t)$. The truth value of the PrSTL predicate $\lambda_{\alpha_t}^{\epsilon_t}$ is equivalent to satisfaction of the probabilistic constraint in equation (9.2). Computing such integrals as in equation (9.3) for general distributions is computationally difficult; however, there are many parameterized distributions (e.g., Gaussian and other members of the exponential family) for which there exists either a closed form solution or efficient numerical procedures.

Note that $\lambda_{\alpha_t}(\xi(t))$ is a stochastic function of the signal ξ at time t and expresses a model of the uncertainty in environment based on the observed signals. As the system evolves and observes more data about the environment, the distribution over the random variable α_t changes over time, thereby leading to an adaptive PrSTL predicate. The PrSTL formula consists of Boolean and temporal operations over their predicates. We recursively define the syntax of PrSTL as:

$$\varphi ::= \lambda_{\alpha_t}^{\epsilon_t} \mid \tilde{\neg}\lambda_{\alpha_t}^{\epsilon_t} \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \mathbf{G}_{[a,b]}\psi \mid \varphi\, \mathbf{U}_{[a,b]}\psi \mid \mathbf{F}_{[a,b]}\psi.$$

Here, φ is a PrSTL formula, which is built upon predicates $\lambda_{\alpha_t}^{\epsilon_t}$ defined in equation (9.2), *propositional formulae* φ composed of the predicates and Boolean operators such as $\wedge$ (and), $\tilde{\neg}$ (negation), and *temporal operators* on φ such as **G** (globally), **F** (eventually) and **U** (until). Note, that in these operations the PrSTL predicates can have different probabilistic parameters, i. e., α_t and ϵ_t. In addition, satisfaction of the PrSTL formulae for each of the Boolean and temporal operations based on the predicates is defined as:

$$\begin{array}{lcl}
(\xi,t) \models \lambda_{\alpha_t}^{\epsilon_t} & \Leftrightarrow & P(\lambda_{\alpha_t}(\xi(t)) < 0) > 1-\epsilon_t \\
(\xi,t) \models \tilde{\neg}\lambda_{\alpha_t}^{\epsilon_t} & \Leftrightarrow & P(-\lambda_{\alpha_t}(\xi(t)) < 0) > 1-\epsilon_t \\
(\xi,t) \models \varphi \wedge \psi & \Leftrightarrow & (\xi,t) \models \varphi \wedge (\xi,t) \models \psi \\
(\xi,t) \models \varphi \vee \psi & \Leftrightarrow & (\xi,t) \models \varphi \vee (\xi,t) \models \psi \\
(\xi,t) \models \mathbf{G}_{[a,b]}\varphi & \Leftrightarrow & \forall t' \in [t+a,t+b], (\xi,t') \models \varphi \\
(\xi,t) \models \mathbf{F}_{[a,b]}\varphi & \Leftrightarrow & \exists t' \in [t+a,t+b], (\xi,t') \models \varphi \\
(\xi,t) \models \varphi\ \mathbf{U}_{[a,b]}\ \psi & \Leftrightarrow & \exists t' \in [t+a,t+b] \text{ s.t. } (\xi,t') \models \psi \\
 & & \wedge \forall t'' \in [t,t'], (\xi,t'') \models \varphi.
\end{array}$$

Remark 12. *PrSTL does not follow the* law of excluded middle, *while it follows the* law of noncontradiction. *This means it will never be the case for a formula and its negation* $\tilde{\neg}$, *as defined above, to both satisfy a specification at the same time. However, there exists situations, where they both can violate a specification.*

Remark 13. *The PrSTL framework reduces to STL, when the distribution* $p(\alpha_t)$ *is a Dirac distribution. A Dirac or a point distribution over* α_t *enforces* $\lambda_{\alpha_t}(\xi(t)) < 0$ *to be deterministic and equivalent to an STL predicate* μ *defined in Chapter 2.3.*

Controller Synthesis for Probabilistic STL

We now formally define the controller synthesis problem in the MPC framework with PrSTL specifications.

Problem 9. *Given a hybrid dynamical system as in equation* (2.3)*, initial state x^0, PrSTL formula φ, and cost function $J(\xi^H)$ defined over a finite horizon trajectory ξ^H, find:*

$$\underset{\mathbf{u}^H}{\textit{argmin}}\, J(\xi^H(x^0, \mathbf{u}^H)) \quad \textit{subject to} \quad \xi^H(x^0, \mathbf{u}^H) \models \varphi. \tag{9.4}$$

Problem (9) formulates a framework for finding a control strategy $\mathbf{u}^H$ that optimizes a given cost function, and satisfies a PrSTL formula. Finding the best strategy for this optimization given only deterministic PrSTL formulae, where α_t is drawn from a Dirac distribution is the same as solving a set of mixed integer linear constraints. We now show how the optimization is solved for the general case of PrSTL. Specifically, we provide full solution for Gaussian distributions, where the optimization reduces to mixed integer semi-definite programs.

Mixed Integer Constraints

We first discuss how every PrSTL formula generates a set of integer constraints. Given a PrSTL formula, we introduce two integer variables for every time step t, p_{t}^{φ} and $q_{\mathrm{t}}^{\varphi} \in \{0, 1\}$, which correspond to the truth value of the PrSTL formula and its negation respectively. These variables enforce satisfaction of the formula φ as:

$$(p_{\mathrm{t}}^{\varphi} = 1) \rightarrow (\xi, t) \models \varphi \text{ and } (q_{\mathrm{t}}^{\varphi} = 1) \rightarrow (\xi, t) \models \tilde{\neg}\varphi \tag{9.5}$$

The formula φ holds true if $p_{\mathrm{t}}^{\varphi} = 1$, and its negation $\tilde{\neg}\varphi$ (defined in Section 9.2) holds true if $q_{\mathrm{t}}^{\varphi} = 1$. Due to the definition of negation, and Remark 12, these implications are only one-way, i. e., there always exist a satisfiable solution when p_{t}^{φ} and q_{t}^{φ} are zero. Using both integer variables, we define the constraints required for logical and temporal operations of PrSTL on p_{t}^{φ} and q_{t}^{φ}. It is important to note that p_{t}^{φ} and q_{t}^{φ} are not functions of the truth value of the formula φ, so their values are not meant to be uniquely determined. Instead, the integer variables enforce the truth value of the formula φ. We refer to them as *truth value enforcers*:

- **Negation** $(\varphi = \tilde{\neg}\psi)$: $p_{\mathrm{t}}^{\varphi} \leq q_{\mathrm{t}}^{\psi}$ and $q_{\mathrm{t}}^{\varphi} \leq p_{\mathrm{t}}^{\psi}$
- **Conjunction** $(\varphi = \wedge_{i=1}^{N} \psi_i)$: $p_{\mathrm{t}}^{\varphi} \leq p_{\mathrm{t}}^{\psi_i}$ and $q_{\mathrm{t}}^{\varphi} \leq \sum_{i=1}^{N} q_{\mathrm{t}}^{\psi_i}$
- **Disjunction** $(\varphi = \vee_{i=1}^{N} \psi_i)$: $\varphi = \tilde{\neg} \wedge_{i=1}^{N} \tilde{\neg}\psi_i$
- **Globally** $(\varphi = \mathbf{G}_{[a,b]}\psi)$: $p_{\mathrm{t}}^{\varphi} \leq p_{t'}^{\psi} \quad \forall t' \in [t + a, \min(t + b, H-1)]$,
  $q_{\mathrm{t}}^{\varphi} \leq \sum_{t'=t+a}^{t+b} q_{t'}^{\psi}$ (Only for $t < H - b$).

- **Eventually** $(\varphi = \mathbf{F}_{[a,b]}\psi) : \varphi = \tilde{\neg}\, \mathbf{G}_{[a,b]} \tilde{\neg}\psi.$
- **Unbounded Until** $(\varphi = \psi_1\, \tilde{\mathbf{U}}_{[0,\infty)}\psi_2) : \bigvee_{t=0}^{H-1} \left((\mathbf{G}_{[0,t]}\psi_1) \wedge (\mathbf{G}_{[t,t]}\psi_2)\right) \vee \mathbf{G}_{[0,H-1]}\psi_1$
- **Bounded Until** $(\varphi = \psi_1\, \mathbf{U}_{[a,b]}\psi_2) : \varphi = \mathbf{G}_{[0,a]}\psi_1 \wedge \mathbf{F}_{[a,b]}\psi_2 \wedge \mathbf{G}_{[a,a]}(\psi_1 \tilde{\mathbf{U}}_{[0,\infty)}\psi_2)$

Here, we have shown how p_{t}^{φ} and q_{t}^{φ} are defined for every logical property such as *negation*, *conjunction*, and *disjunction*, and every temporal property such as *globally*, *eventually*, and *until*. We use $\tilde{\mathbf{U}}$ to refer to unbounded until with infinite time interval, and $\mathbf{U}$ for bounded until.

While synthesizing controllers for PrSTL formulae in an MPC scheme, we sometimes are required to evaluate satisfaction of the formula outside of the horizon range. For instance, a property $\mathbf{G}_{[a,b]}\varphi$ might need to be evaluated beyond H for some $t' \in [t+a, t+b]$. In such cases, our proposal is to act optimistically, i. e., we assume the formula holds true for the time steps outside the horizon of *globally* operator, and similarly assume the formula does not hold true for the negation of the *globally* operator. This optimism is evident in formulating the truth value enforcers of the *globally* operator, and based on that, it is specified for other temporal properties. With the recursive definition of PrSTL, and the above encoding, the truth value enforcers of every PrSTL formula is defined using a set of integer inequalities involving a composition of the truth value enforcers of the inner predicates.

Satisfaction of PrSTL predicates

We have defined the PrSTL predicate $\lambda_{\alpha_t}^{\epsilon_t}$ for a general function, $\lambda_{\alpha_t}(\xi(t))$ of the signal ξ at time t. This function allows a random variable $\alpha_t \sim p(\alpha_t)$ to be drawn from any distribution at every time step. The general problem of controller synthesis satisfying the PrSTL predicates is computationally difficult since the evaluation of the predicates boils down to computing the integration in equation (9.3). Consequently, to solve the problem in equation (9.4), we need to enforce a structure on the predicates of φ. In this section, we explore the linear-Gaussian structure of the predicates that appears in many real-world scenarios, and show how it translates to Mixed Integer SDPs. Formally, if $\varphi = \lambda_{\alpha_t}^{\epsilon_t}$ is only a single predicate, the optimization in equation (9.4) will reduce to:

$$\begin{aligned} \underset{\mathbf{u}^H}{\operatorname{argmin}} \quad & J(\xi^H(x^0, \mathbf{u}^H)) \\ \text{subject to} \quad & (\xi, t) \models \lambda_{\alpha_t}^{\epsilon_t} \qquad \forall t \in \{0, \ldots, H-1\}. \end{aligned} \tag{9.6}$$

This optimization translates to a chance constrained problem [25, 31, 175, 105, 29] at every time step of the horizon, based on the definition of PrSTL predicates in equation (9.2):

$$\begin{aligned} \underset{\mathbf{u}^H}{\operatorname{argmin}} \quad & J(\xi^H(x^0, \mathbf{u}^H)) \\ \text{subject to} \quad & P(\lambda_{\alpha_t}(\xi(t)) < 0) > 1 - \epsilon_t \\ & \forall t \in \{0, \ldots, H-1\}. \end{aligned} \tag{9.7}$$

An important challenge with such chance constrained optimization is there are no guarantees that equation (9.7) is convex. The convexity of the problem depends on the structure of the function λ_{α_t}, and the distribution $p(\alpha_t)$. It turns out that the problem takes a simple convex form when λ_{α_t} is linear-Gaussian, i. e., the random variable α_t comes from a Gaussian distribution and the function itself is linear in α_t:

$$\lambda_{\alpha_t}(\xi(t)) = {\alpha_t}^\top \xi_x(t) = {\alpha_t}^\top x^t, \quad \alpha_t \sim \mathcal{N}(\mu_t, \Sigma_t). \tag{9.8}$$

It is easy to show that for this structure of λ_{α_t}, i.e., a weighted sum of the states with Gaussian weights α_t, the chance constrained optimization in equation (9.7) is convex [171, 91]. Specifically, the optimization problem can be transformed to a second-order cone program (SOCP). First, consider a normally distributed random variable $\nu \sim \mathcal{N}(0,1)$, and its cumulative distribution function (CDF) $\Phi = \int_{-\infty}^{z} \frac{1}{\sqrt{2\pi}} e^{\frac{-t^2}{2}} dt$. Then, the chance constrained optimization reduces to SOCP:

$$\begin{aligned}
&P(\lambda_{\alpha_t}(\xi(t)) < 0) > 1 - \epsilon_t \Leftrightarrow P(\alpha_t^\top x^t < 0) > 1 - \epsilon_t \Leftrightarrow \\
&P(\nu < \frac{-\mu_t^\top x^t}{(x^t)^\top \Sigma_t x^t}) > 1 - \epsilon_t \Leftrightarrow \int_{-\infty}^{\frac{-\mu_t^\top x^t}{(x^t)^\top \Sigma_t x^t}} \frac{1}{\sqrt{2\pi}} e^{\frac{-t^2}{2}} dt > 1 - \epsilon_t \\
&\Leftrightarrow \Phi(\frac{\mu_t^\top x^t}{(x^t)^\top \Sigma_t x^t}) < \epsilon_t \Leftrightarrow \mu_t^\top x^t - \Phi^{-1}(\epsilon_t)||\Sigma_t^{1/2} x^t||_2 < 0.
\end{aligned}$$

In this formulation, $\mu_t^\top x^t$ is the linear term, where μ_t is the mean of the random variable α_t at every time step, and $||\Sigma_t^{1/2} x^t||_2$ is the l_2-norm representing a quadratic term, where Σ_t is the variance of α_t. This quadratic term is scaled by $\Phi^{-1}(\epsilon_t)$, the inverse of the Normal CDF function, which is negative for small values of $\epsilon_t \leq 0.5$. Thus, every chance constraint can be reformulated as a SOCP, and as a result with a convex cost function $J(\xi^H)$, we can efficiently solve the following convex optimization for every predicate of PrSTL:

$$\begin{aligned}
\underset{\mathbf{u}^H}{\text{minimize}} \quad & J(\xi^H(x^0, \mathbf{u}^H)) \\
\text{subject to} \quad & \mu_t^\top x^t - \Phi^{-1}(\epsilon_t)||\Sigma_t^{1/2} x^t||_2 < 0 \\
& \forall t \in \{0, \dots, H-1\}.
\end{aligned} \tag{9.9}$$

Assuming a linear-Gaussian form of the function, we generate the SOCP above, and translate it to a semi-definite program (SDP) by introducing auxiliary variables [31]. We use this SDP that solves the problem in equation (9.6) with a single constraint $\varphi = \lambda_{\alpha_t}^{\epsilon_t}$ as a building block, and use it multiple times to handle complex PrSTL formulae. Specifically, any PrSTL formula can be decomposed to its predicates by recursively introducing integer variables that correspond to the truth value enforcers of the formula at every step as discussed in Section 9.2.

We would like to point out that assuming linear-Gaussian form of λ_{α_t} is not too restrictive. The linear-Gaussian form subsumes the case of Bayesian linear classifiers, and consequently the framework can be applied to a wide variety of scenarios where a classification or regression function needs to estimate quantities of interest that are critical for safety. Furthermore, the framework is applicable to all random variables whose distributions exhibit unimodal behavior and aligned with the large law of numbers. For non-Gaussian random variables, there are many approximate inference procedures that effectively estimate the distributions as Gaussian.

Convex Subset of PrSTL

As discussed in Section 9.2, at the predicate level of φ, we create a chance constrained problem for predicates $\lambda_{\alpha_t}^{\epsilon_t}$. These predicates of the PrSTL formulae can be reformulated as a semi-definite program, where the predicates are over intersections of cone of positive definite matrices with affine spaces. Semi-definite programs are special cases of convex optimization; consequently, solving Problem 9, only for PrSTL predicates is a convex optimization problem. Note that in Section 9.2 we introduced integer variables for temporal and Boolean operators of the PrSTL formula. Construction of such integer variables increases the complexity of Problem 9, and results in a mixed integer semi-definite program (MISDP). However, we are not always required to create integer variables. Therefore, we define *Convex PrSTL* as a subset of PrSTL formulae that can be solved without constructing integer variables.

Definition 2. *Convex PrSTL is a subset of PrSTL such that it is recursively defined over the predicates by applying Boolean conjunctions, and the globally temporal operator. Satisfaction of a convex PrSTL formulae is defined as:*

$$\begin{array}{lcl} (\xi, t) \models \lambda_{\alpha_t}^{\epsilon_t} & \Leftrightarrow & P(\lambda_{\alpha_t}(\xi(t)) < 0) > 1 - \epsilon_t \\ (\xi, t) \models \varphi \wedge \psi & \Leftrightarrow & (\xi, t) \models \varphi \wedge (\xi, t) \models \psi \\ (\xi, t) \models \mathbf{G}_{[a,b]}\varphi & \Leftrightarrow & \forall t' \in [t+a, t+b], (\xi, t') \models \varphi \end{array}$$

Theorem 11. *Given a convex PrSTL formula φ, a hybrid dynamical system as defined in equation* (2.3)*, and an initial state x^0; the controller synthesis problem (Problem 9) is convex under a convex cost function J.*

Proof. We have shown that the predicates of φ, i. e., $\lambda_{\alpha_t}^{\epsilon_t}$ create a set of convex constraints. The Boolean conjunction of convex programs are also convex; therefore, $\varphi \wedge \psi$ result in convex constraints. In addition, the *globally* operator is defined as a set of finite conjunctions over its time interval: $\mathbf{G}_{[a,b]}\varphi = \bigwedge_{t=a}^{b} \varphi_t$. Thus, the *globally* operator retains the convexity property of the constraints. Consequently, Problem 9, with a convex PrSTL constraint φ is a convex program. □

Theorem 11 allows us to efficiently reduce the number of integer variables required for solving Problem 9. We only introduce integer variables when disjunctions, eventually, or until operators appear in the PrSTL constraints. Even when a formula is not completely

Algorithm 6 Controller Synthesis with PrSTL Formulae

```
1: Inputs: $f, x^0, H, \tau, J, \varphi$
2: Let $\tau = [t_1, t_2]$ is the time interval of interest.
3: $\texttt{past} \leftarrow$ Initialize($t_1$)
4: for $t = t_1$: $dt$: $t_2$ do
5:     $f_{\text{lin}} = \text{linearize}(f, \xi(t))$
6:     $\alpha_t \leftarrow$ Update Distributions($\alpha_{t-dt}$, sense($\xi_x(t)$))
7:     $\varphi \leftarrow \varphi(\alpha_t, \epsilon_t)$
8:     $\texttt{C}_{\texttt{PrSTL}} = \text{MISDP}(\varphi)$
9:     $\texttt{C} = \texttt{C}_{\texttt{PrSTL}} \wedge f_{\text{lin}} \wedge [\, \xi(\max(t_1, t-H) \cdots t-dt) = \texttt{past}\,]$
10:    $\mathbf{u}^H = \text{optimize}\big(J(\xi^H), \texttt{C}\big)$
11:    $x_{t+1} = f(x_t, u_t)$
12:    $\texttt{past} \leftarrow [\texttt{past}\ \xi(t)]$
13:    Drop the first element of past if len(past) $> \frac{H}{dt}$
14: end for
```

part of the Convex PrSTL, integer variables are introduced only for the non-convex segments.

We show our complete method of controlling dynamical systems in uncertain environments in Algorithm 6. At the first time step t_1, we run an open-loop control algorithm to populate past in line 2. We then run the closed-loop algorithm, finding the optimal strategy at every time step of the time interval $\tau = [t_1, t_2]$. In the closed-loop algorithm, we linearize the dynamics at the current local state and time in line 4, and then update the distributions over the random variables in the PrSTL formula based on new sensor data in line 5. Then, we update the PrSTL formulae, based on the updated distributions. If there are any other dynamic parameters that change at every time step, they can also be updated in line 6. In line 7, we generate the mixed integer constraints in $\texttt{C}_{\texttt{PrSTL}}$, and populate C with all the constraints including the PrSTL constraints, linearized dynamics, and enforcing to keep the past horizon's trajectory. Note that we do not construct integer variables if the formula is Convex PrSTL. Then, we call the finite horizon optimization algorithm under the cost function $J(\xi^H)$, and the constraints C in line 9, which provides a length H strategy $\mathbf{u}^H$. We advance the state with the first element of $\mathbf{u}^H$, and update the previous horizon's history in past. The size of this problem does not grow, and keeping the past horizon history is crucial in satisfaction of fairness properties, e. g., enforcing a signal to oscillate. We continue running this loop and synthesizing controllers for all times in interval τ. Algorithm 6 solves MISDP problems, which is NP-hard. However, the actual runtime depends on the size of the MISDP, which is linear in the number of predicates and operators in the PrSTL specification. For convex PrSTL, the complexity is the same as an SDP, which is cubic in the number of constraints [104].

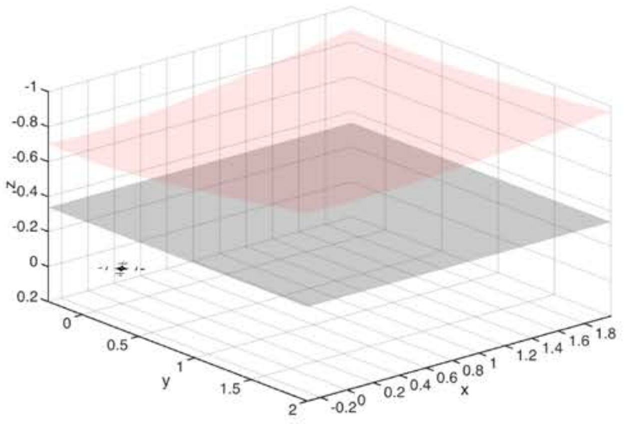


Figure 9.1: A quadrotor starting a trajectory from the origin. The gray surface represents the ceiling, and the pink surface is the quadrotors belief of the ceiling's location is based on the current sensor data.

9.3 Experimental Results

We implemented our controller synthesis algorithm for PrSTL formulae as a Matlab toolbox available at:
`https://github.com/dsadigh/CrSPrSTL`.
We uses Yalmip [115] and Gurobi [73] as the optimization engines. In all of our examples discussed below, the optimization at every step is completed in less than 2 seconds on a 2.3 GHz Intel Core i7 processor with 16 GB RAM.

Quadrotor Control

Controlling quadrotors in dynamic uncertain environments is a challenging task due to different sources of uncertainty, e.g., the position of the obstacles estimated based on classification methods, distributions over wind or battery profiles. We show how to characterize different models of uncertainty over time, and then find an optimal strategy using the framework.

Control in an Uncertain Environments

We first demonstrate obstacle avoidance for a quadrotor following the dynamics described in Chapter 2.4. Figure 9.1, shows the quadrotor at its initial position $(0,0,0)$, and its objective is to reach the coordinates $(1,1,0)$ smoothly. If we let $z = 0$ represent the ground level ($z < 0$ is above the ground level), the objective of the quadrotor is to take off and travel the distance, and then land on the ground at the destination coordinate through the following:

$$J(\xi^H) = \sum_{t=0}^{H-1} ||(x^t, y^t, z^t) - (1,1,0)||_2^2 + c||(\phi^t, \theta^t, \psi^t)||_2^2.$$

Here, we penalize the l_2-norm of the Euler angles, which enforces a resulting smooth trajectory. Besides initializing the state and control input at zero, we bound the control inputs via the following deterministic PrSTL formulae (here only shown for roll, similar form follows for pitch and thrust):

$$\varphi_{\text{roll}} = \mathbf{G}_{[0,\infty)}(||u_1|| \leq 0.3) \text{ Bound on Roll Input}$$

In Figure 9.1, the gray surface is a ceiling that the quadrotor should not collide with as it is taking off and landing at the final position. However, the quadrotor does not have a full knowledge of where the ceiling is exactly located. We define a sensing mechanism for the quadrotor, which consists of a meshgrid of points around the body of the quadrotor. As the system moves in the space, a Bayesian binary classifier is updated by providing a single label -1 (no obstacles present) or 1 (obstacle present) for each of the sensed points.

The Bayesian classifier is the same as the Gaussian Process based method described in Section 9.1 and has the linear-Gaussian form. Applying this classifier results in a Gaussian distribution for every point in the 3D-space. We define our classifier with confidence $1 - \epsilon_t = 0.95$, as the stochastic function $\lambda^{0.05}_{\alpha_t}(\xi(t)) = \alpha_t^\top [x^t \;\; y^t \;\; z^t]$. Here, x^t, y^t, z^t are the coordinates of the sensing points, and $\alpha_t \sim \mathcal{N}(\mu_t, \Sigma_t)$ is the Gaussian weight inferred over time using the sensed data. We define a time-varying probabilistic constraint that needs to be held at every step as its value changes over time. Our constraint specifies that given a classifier based on the sensing points parameterized by α_t, we would enforce the quadrotor to stay within a safe region (defined by the classifier) with probability $1 - \epsilon_t$ at all times. Thus the probabilistic formula is:

$$\varphi_{\text{classifier}} = \quad \mathbf{G}_{[0.1,\infty)}\big(P(\alpha_t^\top [x^t \;\; y^t \;\; z^t] < 0) > 0.95\big).$$

We enforce this probabilistic predicate at $t \in [0.1, \infty)$, which verifies the property starting from a small time after the initial state, so the quadrotor has gathered some sensor data. In Figure 9.1, the pink surface represents the second order cone created based on $\varphi_{\text{classifier}}$, at every step characterized by:

$$\mu_t^\top \begin{bmatrix} x^t & y^t & z^t \end{bmatrix} - \Phi^{-1}(0.05)||\Sigma_t^{1/2} \begin{bmatrix} x^t & y^t & z^t \end{bmatrix}||_2 < 0.$$

Note that the surface shown in Figure 9.1, at the initial time step is not an accurate estimate of where the ceiling is, and it is based on a distribution learned from the initial values of the sensors. Thus, if the quadrotor was supposed to follow this estimate without updating, it would have collided with the ceiling. In Figure 9.2, the blue path represents the trajectory the quadrotor has already taken, and the dotted green line is the future planned trajectory based on the current state of the classifier. The dotted green trajectory at the initial state goes through the ceiling since the belief of the location of the ceiling is incorrect; however, the trajectory is modified at every step as the Bayesian inference updates the distribution over the classifier. As shown in Figure 9.2, the pink surface changes at every time step, and the learned parameters μ_t, and Σ_t are updated, so the quadrotor safely reaches the final position. We solve the optimization using our toolbox, with $dt = 0.03$, and horizon of $H = 20$. These choices describe a common setting for solving MPC problems. We emphasize that some of the constraints are time-varying, and we need to update them at every step of the optimization. We similarly update the dynamics at every step, since we locally linearize the dynamics around the current position of the quadrotor.

Control under Battery Constraints

Next we demonstrate a scenario that addresses safety, when there is uncertainty around the battery level. The battery level is a stochastic variable due to uncertain environment and usage factors, such as radio communications, etc. Our goal is to derive a safe strategy that considers such stochasticity. We start by augmenting the logarithm of battery level (b^t) to the state space of the system discussed above. Thus, the quadrotor is a 13 dimensional system, where the first 12 states follow the same order and system dynamics as before and $\mathbf{x}^t(13) = \log(b^t)$. In our model, the state of $\log(b^t)$ evolves with the negative thrust value: $\frac{d \log(b^t)}{dt} = -|u_4|$. We enforce the same constraints to bound the control as earlier, and the objective of the quadrotor is to start from the origin and reach the top diagonal corner of the space with coordinates $(1, 1, -0.9)$ smoothly.

We then consider adding a safety invariant that would limit high altitude flights if the battery level is low. The following formulae describe these constraints:

- $\phi := \mathbf{F}_{[0,0.3]}(z^t \leq -0.1)$ encodes that eventually in the next 0.3 s, the quadrotor will fly above a threshold level of -0.1 m.
- $\psi := P(\log(b^t) + \mathcal{N}(0, t\sigma^2) \geq b_{\min}) \geq 1 - \epsilon_t$ represents the constraint that the quadrotor has to be confident that the logarithm of its battery state perturbed by a time-varying variance is above $b_{\min}$.

Now, we can combine the two formulae to specify the condition that the quadrotor needs to fly low if the battery state is low:

$$\varphi_{\text{battery}} := \mathbf{G}_{[0,\infty)}(\phi \to \psi). \tag{9.10}$$

The $\to$ in the formula φ_{battery} is the implication relation, and intuitively states that anytime the quadrotor flies above the threshold then it means that there is sufficient battery reserve.

We synthesize a controller for the specifications with $\epsilon_t = 0.2$. The trajectory of the quadrotor is shown in Figure 9.3. Figure 9.3a corresponds to $\sigma = 0$, i. e., the battery state changes deterministically, and Figure 9.3b, corresponds to $\sigma = 10$, when the quadrotor is more cautious about the state of the battery. Note the safe trajectory does not pass the -0.1 m height level whenever the confidence in the battery level is low.

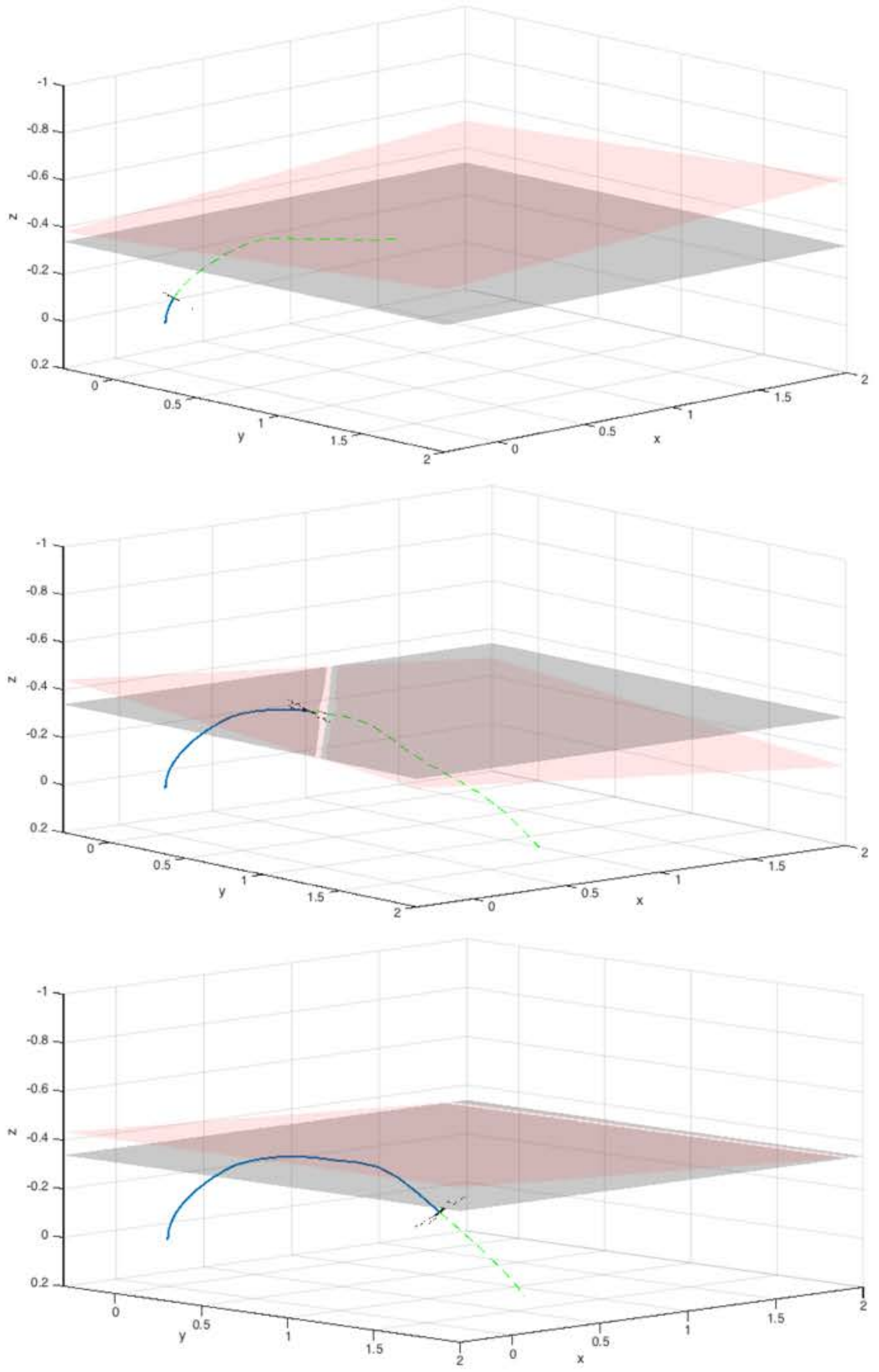
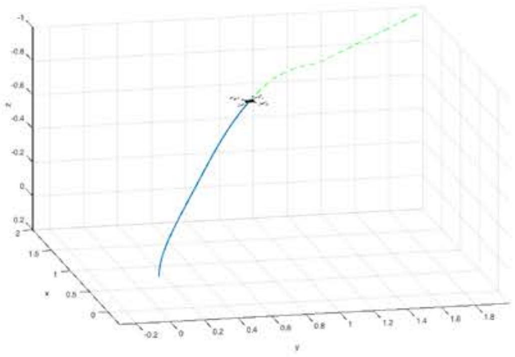


Figure 9.2: The quadrotor in Fig. 9.1 taking the optimal trajectory to reach the goal, while avoiding collisions with the ceiling. The figures from the top correspond to $t = 0.18$ s $t = 0.63$ s, and $t = 1.02$ s.


(a) Deterministic state: $\sigma = 0$.

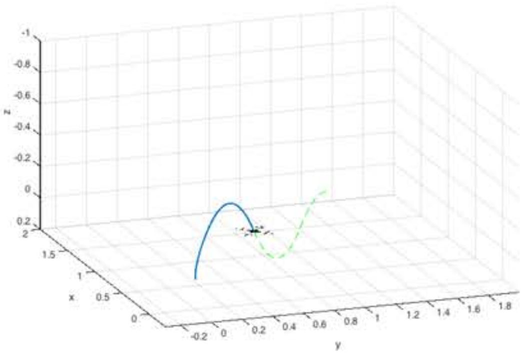

(b) Probabilistic state: $\sigma = 10$.

Figure 9.3: Quadrotor flying to reach a goal while being confident in its battery level. On the right, the quadrotor has low confidence in its battery state, so it avoids flying higher than $z = -0.1$ m.

Autonomous Driving

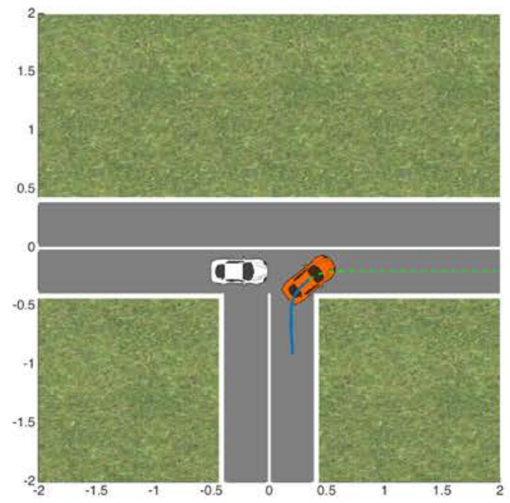
We now consider an autonomous driving scenario. We use the dynamics model introduced in Chapter 2.4.

Figure 9.4, shows a scenario for an autonomous vehicle making a right turn at a signalized intersection. Here the orange car is the *ego* vehicle that is autonomous, and the white car as the *environment* vehicle. Our goal is to find a strategy for the ego vehicle (orange), so it makes a safe right turn when the traffic light is red, while yielding to the oncoming traffic (white).

Note the ego vehicle only has a probabilistic model of the velocity of the environment car. You can think of this environment model as a simple driver model. We refer to the states of the environment vehicle as: $[x^{\text{env}} \;\; y^{\text{env}} \;\; \theta^{\text{env}} \;\; v^{\text{env}}]^\top$. To synthesize a safe strategy, we need to ensure collision avoidance despite the uncertainty in the estimates of the states. We define collision avoidance as the following PrSTL property:

$$\begin{aligned}
\varphi_{\text{crash}} = \mathbf{G}_{[t_0,\infty)} \Big(\;& P\big(x^t - (x^{\text{env}}_{t_0} + s_{t_0,x}(t-t_0)) \geq \delta\big) \;\; \geq 1 - \epsilon_t \\
\vee \;\; & P\big(x^t - (x^{\text{env}}_{t_0} + s_{t_0,x}(t-t_0)) \leq -\delta\big) \;\; \geq 1 - \epsilon_t \\
\vee \;\; & P\big(y^t - (y^{\text{env}}_{t_0} + s_{t_0,y}(t-t_0)) \geq \delta\big) \;\; \geq 1 - \epsilon_t \\
\vee \;\; & P\big(y^t - (y^{\text{env}}_{t_0} + s_{t_0,y}(t-t_0)) \leq -\delta\big) \;\; \geq 1 - \epsilon_t \;\Big)
\end{aligned}$$

Here, φ_{crash} consists of a global operator at all times over the disjunction of four PrSTL predicates. The four probabilistic predicates encode all possible crash scenarios between the two vehicles. The minimum permissible distance between the vehicles is represented as δ that generates the four disjunctions on the predicates. The estimate of the distance between x and y coordinates of the two vehicles is encoded in each predicate as the difference between the coordinates of the ego vehicle, and the propagated coordinates of


(a) A deterministic model of the white car's velocity with $\sigma = 0$.

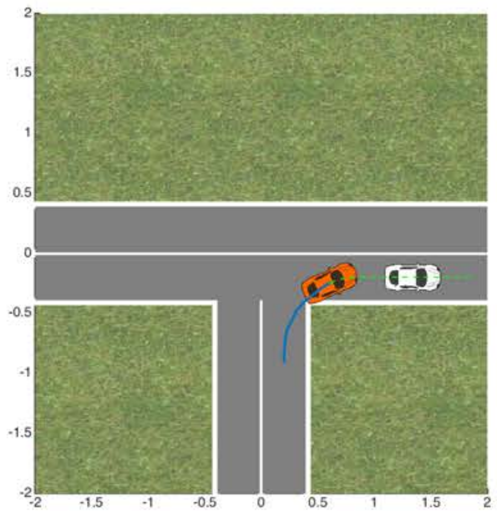

(b) A probabilistic model of the white car's velocity with $\sigma = 0.4$.

Figure 9.4: Orange car making a right turn at a signalized intersection. On the right, the strategy computed performs a safer trajectory with a probabilistic model of the environment, where it first waits for the white car to pass. The blue lines show the trajectory followed by the ego vehicle, and the dotted green line is the future planned trajectory.

the environment vehicle based on the value of its velocity. We assume the magnitude of the velocity is distributed as a Gaussian distribution $v_t^{\text{env}} \sim \mathcal{N}(\bar{v}_t{}^{\text{env}}, \sigma^2)$. Then the vector of Gaussian random variables $[s_{t_0,x}, s_{t_0,y}]^T = v_{t_0}^{\text{env}}[\cos(\theta_{t_0}^{\text{env}}), \sin(\theta_{t_0}^{\text{env}}]^T$ consists of projections of the velocity v_t^{env} on the x and y directions based on the current heading $\theta_{t_0}^{\text{env}}$. The predicates in φ_{crash} take the familiar linear Gaussian form representing the coordinates of the ego vehicle, and are parameterized by the random variable characterizing the velocity of the environment vehicle.

We use $dt = 0.1$ s as sampling time, $H = 20$ as the horizon, $\sigma = 0.4$, $\delta = 0.4$ and $\epsilon_t = 0.2$. We synthesize a strategy for the autonomous vehicle by solving Problem 9, and following the steps in Algorithm 6. The trajectory generated is shown by the solid blue line in Figure 9.4. The dotted green line is the future trajectory computed by the MPC scheme. In Figure 9.4a, the ego vehicle has a deterministic model of the environment vehicle as $\sigma = 0$; therefore, it turns right before letting the environment vehicle pass. However, as shown in Figure 9.4b, for $\sigma = 0.4$, and $\epsilon_t = 0.2$, the ego vehicle is not confident enough in avoiding collisions, so it acts in a conservative manner and waits for the environment car to pass first before turning right.

9.4 Chapter Summary

We presented a framework to achieve safe control under uncertainty. The key contributions include defining PrSTL, a logic for expressing probabilistic properties that can embed Bayesian graphical models. We also show how to synthesize receding horizon controllers under PrSTL specifications that express Bayesian linear classifiers. Further, the resulting logic adapts as more data is observed with the evolution of the system. We demonstrate the approach by synthesizing safe strategies for a quadrotor and an autonomous vehicle traveling in uncertain environments.

The approach extends easily to distributions other than Gaussians via Bayesian approximate inference techniques [120, 21] that can project distributions to the Gaussian densities. Future work includes extending controller synthesis for arbitrary distributions via sampling based approaches; we are also exploring using the proposed framework for complex robotic tasks that need to invoke higher level planning algorithms. We further plan to combine controller synthesis from PrSTL with sampling based task planners that can be guaranteed to be safe under probabilistic specifications.

Chapter 10

Diagnosis and Repair for Synthesis from Signal Temporal Logic

A major challenge to the adoption of synthesis from temporal logic in practice is the difficulty of writing formal specifications – writing temporal logic specifications is not easy and is error-prone, even for experts. Specifications that are poorly stated, incomplete, or inconsistent can produce synthesis problems that are unrealizable (no controller exists for the provided specification), intractable (synthesis is computationally too hard), or lead to solutions that fail to capture the designer's intent. In this chapter, we present an algorithmic approach to reduce the specification burden for controller synthesis from temporal logic specifications, focusing on the case where the original specification is unrealizable.

Logical specifications can be provided in multiple ways. One approach is to provide *monolithic* specifications, combining within a single formula constraints on the environment with desired properties of the system under control. In many cases, a system specification can be conveniently provided as a contract, to distinguish the responsibilities of the system under control (guarantees) from the assumptions on the external, possibly adversarial environment [135, 136]. In such a scenario, an unrealizable specification can be made realizable by either *"weakening" the guarantees* or *"tightening" the assumptions*. In fact, when a specification is unrealizable, it could be either because the environment assumptions are too weak, or the requirements are too strong, or a combination of both. Finding the "problem" with the specification manually can be a tedious and time-consuming process, nullifying the benefits of automatic synthesis. Further, in the *reactive* setting, when the environment is adversarial, finding the right assumptions a priori can be difficult. Thus, given an unrealizable logical specification, there is a need for tools that localize the cause of unrealizability to (hopefully small) parts of the formula, and provide suggestions for repairing the formula in an "optimal" manner.

The problem of diagnosing and repairing formal requirements has received its share of attention in the formal methods community. Ferrère *et al.* perform diagnosis on faulty executions of systems with specifications expressed in Linear Temporal Logic (LTL) and

Metric Temporal Logic (MTL) [59]. They identify the cause of unsatisfiability of these properties in the form of prime implicants, which are conjunctions of literals, and map the failure of a specification to the failure of these prime implicants. Similar syntax-tree based definitions of unsatisfiable cores for LTL were presented by Schuppan [162]. In the context of synthesis from LTL specifications, Raman *et al.* [145] address the problem of categorizing the causes of unrealizability, and how to detect them in high-level robot control specifications. The use of counter-strategies to debug unrealizable cores in a set of specifications or derive new environment assumptions for synthesis has also been explored [109, 95, 9, 110]. Our approach, based on exploiting information already available from off-the-shelf optimization solvers, is similar to the one adopted by Nuzzo *et al.* [134] to extract unsatisfiable cores for Satisfiability Modulo Theories (SMT) solving.

In this chapter, we address the problem of diagnosing and repairing specifications formalized in *Signal Temporal Logic (STL)*. Our work is conducted in the setting of automated synthesis from STL using optimization methods in a Model Predictive Control (MPC) framework similar to the previous chapters [147, 148]. In this approach to synthesis, both the system dynamics and the STL requirements encoded as mixed integer constraints on variables modeling the dynamics of the system and its environment. Controller synthesis is then formulated as an optimization problem to be solved subject to these constraints [147]. In the reactive setting, this approach proceeds by iteratively solving a combination of optimization problems using a *Counterexample-Guided Inductive Synthesis* (CEGIS) scheme (see Chapter 8) [148]. In this context, an unrealizable STL specification leads to an infeasible optimization problem. We leverage the ability of existing Mixed Integer Linear Programming (MILP) solvers to localize the cause of infeasibility to so-called *Irreducibly Inconsistent Systems* (IIS). Our algorithms use the IIS to localize the cause of unrealizability to the relevant parts of the STL specification. Additionally, we give a method for generating a *minimal set of repairs* to the STL specification such that, after applying those repairs, the resulting specification is realizable. The set of repairs is drawn from a suitably defined space that ensures that we rule out vacuous and other unreasonable adjustments to the specification. Specifically, in this chapter, we focus on the numerical parameters in a formula, since their specification is often the most tedious and error-prone part. Our algorithms are sound and complete, i. e., they provide a correct diagnosis, and always terminate with a reasonable specification that is realizable using the chosen synthesis method, when such a repair exists in the space of possible repairs. Recall that Chapter 7 discussed the design of an intervention framework to systematically transfer control to the human while synthesizing control strategies from LTL. In a similar manner, in this chapter our diagnosis and repair algorithms provide a way of detecting the source of unrealizability in the reactive synthesis problem from STL (discussed in Chapter 8). Likewise, this can then be used to design interventions in a shared control framework for a human-robot system.

The problem of infeasibility in constrained predictive control schemes has also been widely addressed in the literature, e.g., by adopting robust MPC approaches, soft constraints, and penalty functions [92, 163, 24]. Rather than tackling general infeasibility issues in MPC, our focus is on providing tools to help debug the controller specification at design time. However, the deployment of robust or soft-constrained MPC approaches can also benefit from our techniques. Our use of MILP does not restrict our method to linear dynamical systems; indeed, we can handle constrained linear and piecewise affine systems, Mixed Logical Dynamical (MLD) systems [23], and certain differentially flat systems.

10.1 Mixed Integer Linear Program Formulation

We described the approach of using Model Predictive Control in Chapter 2.1. When f, the dynamics of the system, as defined in (2.3), is nonlinear, we assume optimization is performed at each MPC step after locally linearizing the system dynamics. For example, at time $t = k$, the linearized dynamics around the current state and time are used to compute an optimal strategy $\mathbf{u}_*^H$ over the time interval $[k, k + H - 1]$. Only the first component of $\mathbf{u}_*^H$ is, however, applied to the system, while a similar optimization problem is solved at time $k + 1$ to compute a new optimal control sequence along the interval $[k + 1, k + H]$ for the model linearized around $t = k + 1$. While the global optimality of MPC is not guaranteed, the technique is frequently used and performs well in practice.

In this chapter, we use STL to express temporal constraints on the environment and system runs for MPC. We then translate a STL specification into a set of mixed integer linear constraints, as further detailed below [147, 148]. Given a formula φ to be satisfied over a finite horizon H, the associated optimization problem has the form:

$$\begin{aligned} \underset{\mathbf{u}^H}{\text{minimize}} \quad & J(\xi(x^0, \mathbf{u}^H)) \\ \text{subject to} \quad & \xi(x^0, \mathbf{u}^H) \models \varphi, \end{aligned} \tag{10.1}$$

which extracts a control strategy $\mathbf{u}^H$ that minimizes the cost function $J(\xi)$ over the finite-horizon trajectory ξ, while satisfying the STL formula φ at time step 0. In a closed-loop setting, we compute a fresh $\mathbf{u}^H$ at every time step $i \in \mathbb{N}$, replacing x^0 with x^i in (10.1) [147, 148].

While equation (10.1) applies to systems without uncontrolled inputs, a more general formulation can be provided to account for an uncontrolled disturbance input $\mathbf{w}^H$ that can act, in general, adversarially as discussed in Chapter 8. To provide this formulation, we assume that the specification is given in the form of a STL *assume-guarantee (A/G) contract* [135, 136] $\mathcal{C} = (V, \varphi_e, \varphi \equiv \varphi_e \to \varphi_s)$, where V is the set of variables, φ_e captures the assumptions (admitted behaviors) over the (uncontrolled) environment inputs w, and φ_s describes the guarantees (promised behaviors) over all the system variables. A game-theoretic formulation of the controller synthesis problem can then be represented as a *minimax* optimization problem:

$$\begin{aligned} \underset{\mathbf{u}^H}{\text{minimize}} \quad & \underset{\mathbf{w}^H \in \mathcal{W}^e}{\text{maximize}} \quad J(\xi(x^0, \mathbf{u}^H, \mathbf{w}^H)) \\ \text{subject to} \quad & \forall \mathbf{w}^H \in \mathcal{W}^e \quad \xi(x^0, \mathbf{u}^H, \mathbf{w}^H) \models \varphi, \end{aligned} \tag{10.2}$$

where we aim to find a strategy $\mathbf{u}^H$ that minimizes the worst case cost $J(\xi)$ over the finite horizon trajectory, under the assumption that the disturbance signal $\mathbf{w}^H$ acts adversarially. We use $\mathcal{W}^e$ in (10.2) to denote the set of disturbances that satisfy the environment specification φ_e, i.e., $\mathcal{W}^e = \{\mathbf{w} \in \mathcal{W}^H | \mathbf{w} \models \varphi_e\}$.

To solve the control problems in (10.1) and (10.2) the STL formula φ can be translated into a set of mixed integer constraints, thus reducing the optimization problem to a *Mixed Integer Program* (MIP), as long as the system dynamics can also be translated into mixed integer constraints. Specifically, in this chapter similar to Chapter 8, we consider control problems that can be encoded as *Mixed Integer Linear Programs* (MILP).

The MILP constraints are constructed recursively on the structure of the STL specification as in [147, 148], and express the robust satisfaction value of the formula. A first set of variables and constraints capture the robust satisfaction of the atomic predicates of the formula. To generate the remaining constraints, we traverse the parse tree of φ from the leaves (associated with the atomic predicates) to the root node (corresponding to the robustness satisfaction value of the overall formula ρ^{φ}), adding variables and constraints that obey the quantitative semantics discussed in Chapter 2.3.

Recall from Chapter 2.3 that the robustness value of subformulae with temporal and Boolean operators is expressed as the *min* or *max* of the robustness values of the operands over time. We discuss here the encoding of the *min* operator as an example. To encode $p = \min(\rho^{\varphi_1}, \dots, \rho^{\varphi_n})$, we introduce Boolean variables z^{φ_i} for $i \in \{1, \dots, n\}$ and MILP constraints:

$$\begin{gathered} p \leq \rho^{\varphi_i}, \sum_{i=1 \dots n} z^{\varphi_i} \geq 1 \\ \rho^{\varphi_i} - (1 - z^{\varphi_i})M \leq p \leq \rho^{\varphi_i} + (1 - z^{\varphi_i})M \end{gathered} \tag{10.3}$$

where M is a constant selected to be much larger than $|\rho^{\varphi_i}|$ for all i, and $i \in \{1, \dots, n\}$. The above constraints ensure that $z^{\varphi_i} = 1$ and $p = \rho^{\varphi_i}$ only if ρ^{φ_i} is the minimum over all i. For *max*, we replace $\leq$ by $\geq$ in the first constraint of (10.3). Finally, we choose ρ^{φ} as the cost function of the resulting MILP, meaning that our controllers aim at maximizing the robustness of satisfaction of the specification. Note that the robustness function is analogous to the robot reward function $R_{\mathcal{R}}$ discussed in Part I; both encode a quantitative measure for satisfaction of the desired properties, and both are optimized to find the desirable controllers for the robot.

We solve the resulting MILP with an off-the-shelf solver. If the receding horizon scheme is feasible, then the controller synthesis problem is *realizable*, i.e., the algorithm returns a controller that satisfies the specification and optimizes the objective. However, if the MILP is infeasible, the synthesis problem is *unrealizable*. In this case, the failure to

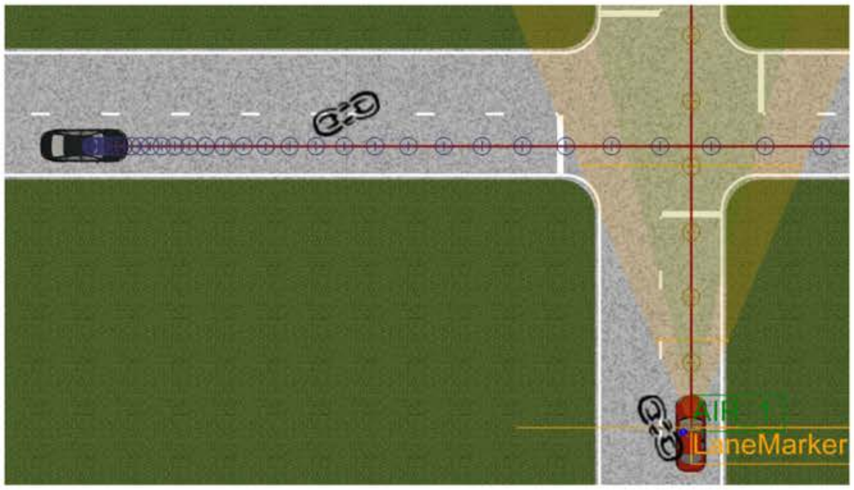


Figure 10.1: Vehicles crossing an intersection. The red car is the *ego* vehicle, while the black car is part of the environment.

synthesize a controller may well be attributed to just a portion of the STL specification. In the rest of the chapter, we discuss how infeasibility of the MILP constraints can be used to infer the "cause" of failure and, consequently, diagnose and repair the original STL specification.

10.2 Running Example

To illustrate our approach, we introduce a running example from the autonomous driving domain. We refer the readers to the example introduced in Chapter 8.5. As shown in Figure 10.1, we consider a scenario in which two moving vehicles approach an intersection. The red car, labeled the *ego* vehicle, is the vehicle under control, while the black car is part of the external environment and may behave, in general, adversarially. The state space and dynamics of this example is introduced in Chapter 8.5. We assume the ego vehicle is initialized at the coordinates $(0, -1)$ and the adversary vehicle is initialized at $(-1, 0)$. We further assume all the units in this example follow the metric system. We would like to design a controller for the ego vehicle to satisfy a STL specification under some assumptions on the external environment, and provide diagnosis and feedback if the specification is infeasible. We discuss the following three scenarios.

Example 2 (Collision Avoidance). *The specification is to avoid a collision between the ego and the adversary vehicle. We assume the adversary vehicle's acceleration is fixed at all times, i.e.,* $a_t^{\text{adv}} = 2$*, while the initial velocities are* $v_0^{\text{adv}} = 0$ *and* $v_0^{\text{ego}} = 0$*. We encode our requirements* *using the formula $\varphi := \varphi_1 \wedge \varphi_2$, where φ_1 and φ_2 are defined as follows:*

$$\begin{aligned}\varphi_1 &= \mathbf{G}_{[0,\infty)} \neg\big((-0.5 \le y_t^{\text{ego}} \le 0.5) \wedge (-0.5 \le x_t^{\text{adv}} \le 0.5)\big),\\ \varphi_2 &= \mathbf{G}_{[0,\infty)} \big(1.5 \le a_t^{\text{ego}} \le 2.5\big).\end{aligned} \tag{10.4}$$

We prescribe bounds on the system acceleration, and state that both cars should never be confined together within a box of width 1 *around the intersection* $(0, 0)$ *to avoid a collision.*

Example 3 (Non-adversarial Race)**.** *We discuss a race scenario, in which the* ego *vehicle must increase its velocity to exceed* 0.5 *whenever the adversary's initial velocity exceeds* 0.5*. We then formalize our requirement as a contract (ψ_e, $\psi_e \to \psi_s$), where ψ_e are the assumptions made on the environment and ψ_s are the guarantees of the system provided the environment satisfies the assumptions. Specifically:*

$$\begin{aligned}\psi_e &= (v_0^{\text{adv}} \ge 0.5),\\ \psi_s &= \mathbf{G}_{[0,\infty)}(-1 \le a_t^{\text{ego}} \le 1) \wedge \mathbf{G}_{[0.2,\infty)}(v_t^{\text{ego}} \ge 0.5).\end{aligned} \tag{10.5}$$

The initial velocities are $v_0^{\text{adv}} = 0.55$ and $v_0^{\text{ego}} = 0$, while the environment vehicle's acceleration is $a_t^{\text{adv}} = 1$ at all times. We also require the acceleration to be bounded by 1*.*

Example 4 (Adversarial Race)**.** *We discuss another race scenario, in which the environment vehicle acceleration a_t^{adv} is no longer fixed, but can vary up to a maximum value of* 2*. Initially, $v_0^{\text{adv}} = 0$ and $v_0^{\text{ego}} = 0$ hold. Under these assumptions, we would like to guarantee that the velocity of the* ego *vehicle exceeds* 0.5 *if the speed of the adversary vehicle exceeds* 0.5*, while maintaining an acceleration in the* $[-1, 1]$ *range. Altogether, we capture the requirements above via a contract (ϕ_w, $\phi_w \to \phi_s$), where:*

$$\begin{aligned}\phi_w &= \mathbf{G}_{[0,\infty)}\big(0 \le a_t^{\text{adv}} \le 2\big),\\ \phi_s &= \mathbf{G}_{[0,\infty)}\big((v_t^{\text{adv}} > 0.5) \to (v_t^{\text{ego}} > 0.5)\big) \wedge \big(|a_t^{\text{ego}}| \le 1\big).\end{aligned} \tag{10.6}$$

10.3 Diagnosis and Repair Problem

In this section, we define the problems of specification diagnosis and repair in the context of controller synthesis from STL. We assume that the discrete-time system dynamics f_d, the initial state x^0, the STL specification φ, and a cost function J are given. The *controller synthesis* problem, denoted $\mathcal{P} = (f_d, x^0, \varphi, J)$, is to solve (10.1) (when φ is a monolithic specification of the desired system behavior) or (10.2) (when φ represents a contract between the system and the environment).

If synthesis fails, the *diagnosis* problem is, intuitively, to return an explanation in the form of a subset of the original problem constraints that are already infeasible when taken alone. The *repair* problem is to return a "minimal" set of changes to the specification that would render the resulting controller synthesis problem feasible. To diagnose and repair a STL formula, we focus on its sets of atomic predicates and time intervals of the temporal operators. We then start by providing a definition of the *support* of its atomic predicates, i.e., the set of times at which the value of a predicate affects satisfiability of the formula, and define the set of allowed repairs.

Definition 3 (Support). *The* support *of a predicate μ in a STL formula φ is the set of times t such that $\mu(\xi(t))$ appears in φ.*

For example, given $\varphi = \mathbf{G}_{[6,10]}(x^t > 0.2)$, the support of predicate $\mu = (x^t > 0.2)$ is the time interval $[6, 10]$. We can compute the support of each predicate in φ by traversing the parse tree of the formula from the root node to the leaves, which are associated with the atomic predicates. The support of the root of the formula is $\{0\}$ by definition. While parsing φ, new nodes are created and associated with the Boolean and temporal operators in the formula. Let κ and δ be the subsets of nodes associated with the Boolean and bounded temporal operators, respectively, where $\delta = \delta_{\mathbf{G}} \cup \delta_{\mathbf{F}} \cup \delta_{\mathbf{U}}$. The support of the predicates can then be computed by recursively applying the following rule for each node i in the parse tree:

$$\sigma_i = \begin{cases} \sigma_r \text{ if } r \in \kappa \\ \sigma_r + I_r \text{ if } r \in \delta_{\mathbf{G}} \cup \delta_{\mathbf{F}} \\ [\sigma_r^{lb}, \sigma_r^{ub} + I_r^{ub}] \text{ if } r \in \delta_{\mathbf{U}}, \end{cases} \tag{10.7}$$

where r is the parent of i, σ_r is the support of r, and I_i is the interval associated with i when i corresponds to a temporal operator. We denote as $I_1 + I_2$ the Minkowski sum of the sets I_1 and I_2, and as I^{lb} and I^{ub}, respectively, the lower and upped bounds of interval I.

Definition 4 (Allowed Repairs). *Let Φ denote the set of all possible STL formulae. A* repair action *is a relation $\gamma : \Phi \to \Phi$ consisting of the union of the following:*

- *A* predicate repair *returns the original formula after modifying one of its atomic predicates μ to μ^*. We denote this sort of repair by $\varphi[\mu \mapsto \mu^*] \in \gamma(\varphi)$;*
- *A* time interval repair *returns the original formula after replacing the interval of a temporal operator. This is denoted $\varphi[\Delta_{[a,b]} \mapsto \Delta_{[a^*,b^*]}] \in \gamma(\varphi)$ where $\Delta \in \{\mathbf{G}, \mathbf{F}, \mathbf{U}\}$.*

Repair actions can be composed to get a *sequence of repairs* $\Gamma = \gamma_n(\gamma_{n-1}(\dots(\gamma_1(\varphi))\dots))$. Given a STL formula φ, we denote as $\mathtt{REPAIR}(\varphi)$ the set of all possible formulae obtained through compositions of allowed repair actions on φ. Moreover, given a set of atomic predicates $\mathcal{D}$ and a set of time intervals $\mathcal{T}$, we use $\mathtt{REPAIR}_{\mathcal{T},\mathcal{D}}(\varphi) \subseteq \mathtt{REPAIR}(\varphi)$ to denote the set of repair actions that act only on predicates in $\mathcal{D}$ or time intervals in $\mathcal{T}$. We are now ready to provide the formulation of the problems addressed in the chapter, both in terms of diagnosis and repair of a *monolithic* specification φ (*general diagnosis and repair*) and an A/G contract $(\varphi_e, \varphi_e \to \varphi_s)$ (*contract diagnosis and repair*).

Problem 10 (General Diagnosis and Repair). *Given a controller synthesis problem $\mathcal{P} = (f_d, x^0, \varphi, J)$ such that* (10.1) *is infeasible, find:*

- *A set of atomic predicates $\mathcal{D} = \{\mu_1, \dots, \mu_d\}$ or time intervals $\mathcal{T} = \{\tau_1, \dots, \tau_d\}$ of the original formula φ,*
- *$\varphi' \in \texttt{REPAIR}_{\mathcal{T},\mathcal{D}}(\varphi)$,*

such that $\mathcal{P}' = (f_d, x^0, \varphi', J)$ is feasible, and the following minimality conditions hold:

- (predicate minimality) *if φ' is obtained by predicate repair*[1]*, $s_i = \mu_i^* - \mu_i$ for $i \in \{1, \dots, d\}$, $s_{\mathcal{D}} = (s_1, \dots, s_d)$, and $||\cdot||$ is a norm on $\mathbb{R}^d$, then*
$$\nexists\, (\mathcal{D}', s_{\mathcal{D}'}) \quad \text{s.t.} \quad ||s_{\mathcal{D}'}|| \leq ||s_{\mathcal{D}}|| \tag{10.8}$$
and $\mathcal{P}'' = (f_d, x^0, \varphi'', J)$ is feasible, with $\varphi'' \in \texttt{REPAIR}_{\mathcal{D}'}(\varphi)$.
- (time interval minimality) *if φ' is obtained by time interval repair, $\mathcal{T}^* = \{\tau_1^*, \dots, \tau_l^*\}$ are the non-empty repaired intervals, and $||\tau||$ is the length of interval τ:*
$$\nexists\, \mathcal{T}' = \{\tau_1', \dots, \tau_l'\}, \text{ s.t. } \exists i \in \{1, \dots, l\}, ||\tau_i^*|| \leq ||\tau_i'|| \tag{10.9}$$
and $\mathcal{P}'' = (f_d, x^0, \varphi'', J)$ is feasible, with $\varphi'' \in \texttt{REPAIR}_{\mathcal{T}'}(\varphi)$.

Problem 11 (Contract Diagnosis and Repair). *Given a controller synthesis problem $\mathcal{P} = (f_d, x^0, \varphi \equiv \varphi_e \to \varphi_s, J)$ such that* (10.2) *is infeasible, find:*

- *Sets of atomic predicates $\mathcal{D}_e = \{\mu_1^e, \dots, \mu_d^e\}$, $\mathcal{D}_s = \{\mu_1^s, \dots, \mu_{\tilde{d}}^s\}$ or sets of time intervals $\mathcal{T}_e = \{\tau_1^e, \dots, \tau_l^e\}$,$\mathcal{T}_s = \{\tau_1^s, \dots, \tau_{\tilde{l}}^s\}$, respectively, of the original formulas φ_e and φ_s,*
- *$\varphi_e' \in \texttt{REPAIR}_{\mathcal{T}_e,\mathcal{D}_e}(\varphi_e)$, $\varphi_s' \in \texttt{REPAIR}_{\mathcal{T}_s,\mathcal{D}_s}(\varphi_s)$,*

such that $\mathcal{P}' = (f_d, x^0, \varphi', J)$ is feasible, $\mathcal{D} = \mathcal{D}_e \cup \mathcal{D}_s$, $\mathcal{T} = \mathcal{T}_e \cup \mathcal{T}_s$, and $\varphi' \equiv \varphi_e' \to \varphi_s'$ satisfies the minimality conditions of Problem (10).

In the following sections, we discuss our solution to the above problems.

10.4 Monolithic Specifications

The scheme adopted to diagnose inconsistencies in the specification and provide constructive feedback to the designer is pictorially represented in Figure 10.2. In this section we find a solution for Problem 10, as summarized in Algorithm 7. Given a problem

[1]For technical reasons, our minimality conditions are predicated on a single type of repair being applied to obtain φ'.

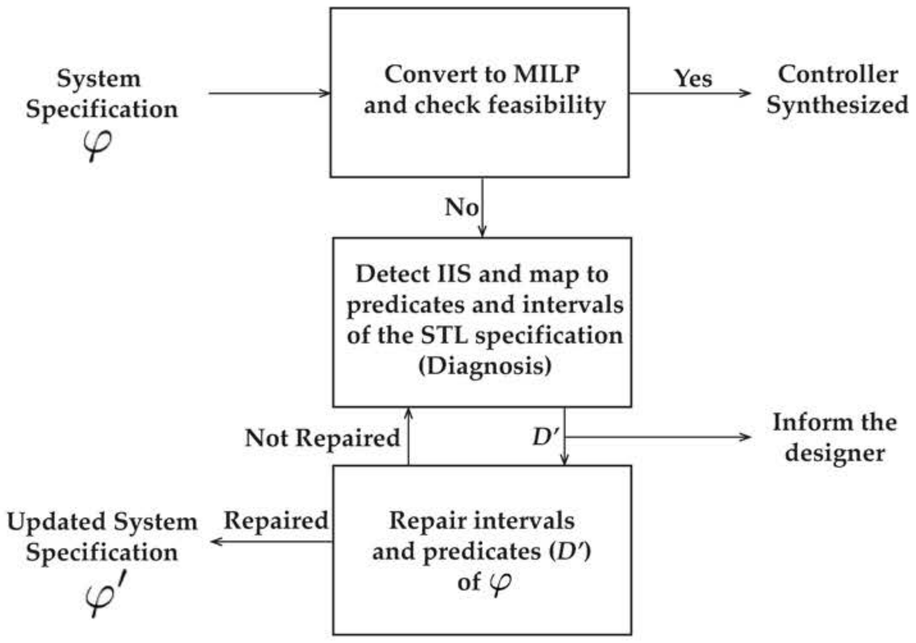

System Specification φ
Convert to MILP and check feasibility
Yes
Controller Synthesized
No
Detect IIS and map to predicates and intervals of the STL specification (Diagnosis)
Not Repaired
D'
Inform the designer
Updated System Specification φ'
Repaired
Repair intervals and predicates (D') of φ

Figure 10.2: Diagnosis and repair flow diagram.

$\mathcal{P}$, defined as in Section 10.3, `GenMILP` reformulates (10.1) in terms of the following MILP:

$$\begin{aligned} \underset{\mathbf{u}^H}{\text{minimize}} \quad & J(\xi) \\ \text{subject to} \quad & f_i^{\text{dyn}} \le 0 \qquad i \in \{1,\dots,m_d\} \\ & f_k^{\varphi} \le 0 \qquad k \in \{1,\dots,m_s\}, \end{aligned} \tag{10.10}$$

where f^{dyn} and f^{φ} are mixed integer linear constraint functions over the states, outputs, and inputs of the finite horizon trajectory ξ^H associated, respectively, with the system dynamics and the STL specification φ. We let (J, C) represent this MILP, where J is the objective, and C is the set of constraints. If problem (10.10) is infeasible, we iterate between diagnosis and repair phases until the repaired feasible specification φ' is obtained. We let $\mathcal{D}$ and I denote, respectively, the set of predicates returned by the diagnosis procedure, and the constraints corresponding to those predicates.

Optionally, we support an interactive repair mechanism, where the designer provides a set of *options* that prioritize which predicates to modify (`UserInput` procedure) and get converted into a set of weights λ (`ModifyConstraints` routine). The designer (human) can then leverage this weighted-cost variant of the problem to distinguish between "hard" constraints, i.e., constraints that should never be violated in the controller synthesis problem, and "soft" constraints, i.e., constraints whose violation or perturbation is admitted within a predefined margin. In the following, we detail the implementation of the `Diagnosis` and `Repair` routines.

Algorithm 7 Diagnosis Repair

```
 1: Input: $\mathcal{P}$
 2: Output: $\mathbf{u}^H$, $\mathcal{D}$, $repaired$, $\varphi'$
 3: $(J, C) \leftarrow$ GenMILP($\mathcal{P}$), $repaired \leftarrow 0$
 4: $\mathbf{u}^H \leftarrow$ Solve($J, C$)
 5: if $\mathbf{u}^H = \emptyset$ then
 6:     $\mathcal{D} \leftarrow \emptyset$, $\mathcal{S} \leftarrow \emptyset$, $I \leftarrow \emptyset$, $\mathcal{M} \leftarrow (0, C)$
 7:     while $repaired = 0$ do
 8:         $(\mathcal{D}', \mathcal{S}', I') \leftarrow$ Diagnosis($\mathcal{M}$, $\mathcal{P}$)
 9:         $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}'$, $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{S}'$, $I \leftarrow I \cup I'$
10:         $options \leftarrow$ UserInput($\mathcal{D}'$)
11:         $\lambda \leftarrow$ ModifyConstraints($I'$, $options$)
12:         $(repaired, \mathcal{M}, \varphi') \leftarrow$ Repair($\mathcal{M}, I', \lambda, \mathcal{S}, \varphi$)
13:     end while
14:     $\mathbf{u}^H \leftarrow$ Solve($J$, $\mathcal{M}.C$)
15: end if
```

Algorithm 8 Diagnosis

```
1: Input: $\mathcal{M}$, $\mathcal{P}$
2: Output: $\mathcal{D}$, $\mathcal{S}$, $I'$
3: $I_C \leftarrow$ IIS($\mathcal{M}$)
4: $(\mathcal{D}, \mathcal{S}) \leftarrow$ ExtractPredicates($I_C$, $\mathcal{P}$)
5: $I' \leftarrow$ ExtractConstraints($\mathcal{M}$, $\mathcal{D}$)
```

Diagnosis

Our diagnosis procedure is summarized in Algorithm 8. `Diagnosis` receives as inputs the controller synthesis problem $\mathcal{P}$ and an associated MILP formulation $\mathcal{M}$. $\mathcal{M}$ can either be the *feasibility problem* corresponding to the original problem (10.10), or a relaxation of it. This feasibility problem has the same constraints as (10.10) (possibly relaxed) but constant cost. Formally, we provide the following definition of relaxed constraint and relaxed optimization problem.

Definition 5 (Relaxed Problem). *We say that a constraint $f' \leq 0$ is a* relaxed version *of $f \leq 0$ if $f' = (f - s)$ for some slack variable $s \in \mathbb{R}^+$. In this case, we also say that $f \leq 0$ "is relaxed to" $f' \leq 0$. An optimization problem $\mathcal{O}'$ is a* relaxation *of another optimization problem $\mathcal{O}$ if it is obtained from $\mathcal{O}$ by relaxing at least one of its constraints.*

When $\mathcal{M}$ is infeasible, we rely on the capability of state-of-the-art MILP solvers to provide an *Irreducibly Inconsistent System* (IIS) [73, 39] of constraints I_C, defined as follows.

Definition 6 (Irreducibly Inconsistent System). *Given a feasibility problem $\mathcal{M}$ with constraint set C, an* Irreducibly Inconsistent System *I_C is a subset of constraints $I_C \subseteq C$ such that: (i) the optimization problem $(0, I_C)$ is infeasible; (ii) $\forall\, c \in I_C$, problem $(0, I_C \setminus \{c\})$ is feasible.*

In other words, an IIS is an infeasible subset of constraints that becomes feasible if any single constraint is removed. For each constraint in I_C, `ExtractPredicates` traces back the STL predicate(s) originating it, which will be used to construct the set $\mathcal{D} = \{\mu_1, \ldots, \mu_d\}$ of STL atomic predicates in Problem 10, and the corresponding set of support intervals $\mathcal{S} = \{\sigma_1, \ldots, \sigma_d\}$ (adequately truncated to the current horizon H) as obtained from the STL syntax tree. $\mathcal{D}$ will be used to produce a relaxed version of $\mathcal{M}$ as further detailed in Section 10.4. For this purpose, the procedure also returns the subset I of all the constraints in $\mathcal{M}$ that are associated with the predicates in $\mathcal{D}$.

Repair

The diagnosis procedure isolates a set of STL atomic predicates that jointly produce a reason of infeasibility for the synthesis problem. For repair, we are instead interested in how to modify the original formula to make the problem feasible. The repair procedure is summarized in Algorithm 9. We formulate relaxations of the feasibility problem $\mathcal{M}$ associated with problem (10.10) by using *slack variables*.

Let f_i, $i \in \{1, \ldots, m\}$ denote both of the categories of constraints f^{dyn} and f^{φ} in the feasibility problem $\mathcal{M}$. We reformulate $\mathcal{M}$ into the following *slack feasibility problem*:

$$\begin{aligned} \underset{\mathbf{s} \in \mathbb{R}^{|I|}}{\text{minimize}} \quad & ||\mathbf{s}|| \\ \text{subject to} \quad & f_i - s_i \leq 0 \qquad i \in \{1, \ldots, |I|\} \\ & f_i \leq 0 \qquad i \in \{|I|+1, \ldots, m\} \\ & s_i \geq 0 \qquad i \in \{1, \ldots, |I|\}, \end{aligned} \tag{10.11}$$

where $\mathbf{s} = s_1 \ldots s_{|I|}$ is a vector of slack variables corresponding to the subset of optimization constraints I, as obtained after the latest call of `Diagnosis`. Not all the constraints in the original optimization problem (10.10) can be modified. For instance, the designer will not be able to arbitrarily modify constraints that can directly affect the dynamics of the system, i. e., constraints encoded in f^{dyn}. Solving problem (10.11) is equivalent to looking for a set of slacks that make the original control problem feasible while minimizing a suitable norm $||\cdot||$ of the slack vector. In most of our application examples, we choose the l_1-norm, which tends to provide sparser solutions for $\mathbf{s}$, i.e., nonzero slacks for a smaller number of constraints. However, other norms can also be used, including weighted norms based on the set of weights λ. If problem (10.11) is feasible, `ExtractFeedback` uses the solution $\mathbf{s}^*$ to repair the original infeasible specification φ. Otherwise, the infeasible problem is subjected to another round of diagnosis to retrieve further constraints to relax. In what follows, we provide details on the implementation of `ExtractFeedback`.

Algorithm 9 Repair

```
1: Input: $\mathcal{M}$, $I$, $\lambda$, $\mathcal{S}$, $\varphi$
2: Output: $repaired$, $\mathcal{M}$, $\varphi$
3: $\mathcal{M}.J \leftarrow \mathcal{M}.J + \lambda^\top s_I$
4: for $c$ in $I$ do
5:     if $\lambda(c) > 0$ then
6:         $\mathcal{M}.C(c) \leftarrow \mathcal{M}.C(c) + s_c$
7:     end if
8: end for
9: $(repaired, \mathbf{s}^*) \leftarrow$ Solve($\mathcal{M}.J$, $\mathcal{M}.C$)
10: if $repaired = 1$ then
11:     $\varphi \leftarrow$ ExtractFeedback($\mathbf{s}^*$,$\mathcal{S}$,$\varphi$)
12: end if
```

Based on the encoding discussed in Section 2.1, the constraints in $\mathcal{M}$ capture, in a recursive fashion, the robust satisfaction of the STL specification as a function of its subformulae, from φ itself to the atomic predicates μ_i. To guarantee satisfaction of a Boolean operation in φ at time t, we must be able to perturb, in general, all the constraints associated with its operands, i.e., the children nodes of the corresponding Boolean operator in the parse tree of φ, at time t. Similarly, to guarantee satisfaction of a temporal construct at time t, we must be able to perturb the constraints associated with the operands of the corresponding operator at all times in their support.

By recursively applying this line of reasoning, we can then conclude that, to guarantee satisfaction of φ, it is sufficient to introduce slacks to all the constraints associated with all the diagnosed predicates in $\mathcal{D}$ over their entire support. For each $\mu_i \in \mathcal{D}$, $i \in \{1, \dots, d\}$, let $\sigma_i = [\sigma_i^{lb}, \sigma_i^{ub}]$ be its support interval.

The set of slack variables $\{s_1, \dots, s_{|I|}\}$ in (10.11) can then be seen as the set of variables $s_{\mu_i,t}$ used to relax the constraints corresponding to each diagnosed predicate $\mu_i \in \mathcal{D}$ at time t, for all $t \in \{\max\{0, \sigma_i^{lb}\}, \dots, \min\{H-1, \sigma_i^{ub}\}\}$ and $i \in \{1, \dots, d\}$.

If a minimum norm solution $\mathbf{s}^*$ is found for (10.11), then the slack variables $\mathbf{s}^*$ can be mapped to a set of *predicate repairs* $s_{\mathcal{D}}$, as defined in Problem 10, as follows. The slack vector $\mathbf{s}^*$ in Algorithm 9 consists of the set of slack variables $\{s^*_{\mu_i,t}\}$, where $s^*_{\mu_i,t}$ is the variable added to the optimization constraint associated with an atomic predicate $\mu_i \in \mathcal{D}$ at time t, $i \in \{1, \dots, d\}$. We set

$$\forall\, i \in \{1, \dots, d\}\ s_i = \mu_i^* - \mu_i = \max_{t \in \{\sigma_{i,l}, \cdots, \sigma_{i,u}\}} s^*_{\mu_i,t}, \tag{10.12}$$

where H is the time horizon for (10.10), $s_{\mathcal{D}} = \{s_1, \dots, s_d\}$, $\sigma_{i,l} = \max\{0, \sigma_i^{lb}\}$, and $\sigma_{i,u} = \min\{H-1, \sigma_i^{ub}\}$.

To find a set of *time-interval repairs*, we proceed, instead, as follows:

1. The slack vector $\mathbf{s}^*$ in Algorithm 9 consists of the set of slack variables $\{s^*_{\mu_i,t}\}$, where $s^*_{\mu_i,t}$ is the variable added to the optimization constraint associated with an atomic predicate $\mu_i \in \mathcal{D}$ at time t. For each $\mu_i \in \mathcal{D}$, with support interval σ_i, we search for the largest time interval $\sigma'_i \subseteq \sigma_i$ such that the slack variables $s^*_{\mu_i,t}$ for $t \in \sigma'_i$ are 0. If $\mu_i \notin \mathcal{D}$, then we set $\sigma'_i = \sigma_i$.

2. We convert every temporal operator in φ into a combination of **G** (timed or untimed) and untimed **U** by using the following transformations:

$$\mathbf{F}_{[a,b]}\psi = \neg\mathbf{G}_{[a,b]}\neg\psi,$$

$$\psi_1\mathbf{U}_{[a,b]}\psi_2 = \mathbf{G}_{[0,a]}(\psi_1\mathbf{U}\ \psi_2) \wedge \mathbf{F}_{[a,b]}\psi_2,$$

   where **U** is the untimed (unbounded) *until* operator. Let $\hat{\varphi}$ be the new formula obtained from φ after applying these transformations[2].

3. The nodes of the parse tree of $\hat{\varphi}$ can then be partitioned into three subsets, ν, κ, and δ, respectively associated with the *atomic predicates*, *Boolean operators*, and *temporal operators* $(\mathbf{G}, \mathbf{U})$ in $\hat{\varphi}$. We traverse this parse tree from the leaves (atomic predicates) to the root and recursively define for each node i a new support interval σ^*_i as follows:

$$\sigma^*_i = \begin{cases} \sigma'_i & \text{if } i \in \nu \\ \bigcap_{j \in C(i)} \sigma^*_j & \text{if } i \in \kappa \cup \delta_{\mathbf{U}} \\ \sigma^*_{C(i)} & \text{if } i \in \delta_{\mathbf{G}} \end{cases} \tag{10.13}$$

   where $C(i)$ denotes the set of children of node i, while $\delta_{\mathbf{G}}$ and $\delta_{\mathbf{U}}$ are, respectively, the subsets of nodes associated with the **G** and **U** operators. We observe that the set of children for a **G** operator node is a singleton. Therefore, with some abuse of notation, we also use $C(i)$ in (10.13) to denote a single node in the parse tree.

4. We define the interval repair $\hat{\tau}_j$ for each (timed) temporal operator node j in the parse tree of $\hat{\varphi}$ as $\hat{\tau}_j = \sigma^*_j$. If $\hat{\tau}_j$ is empty for some j, no time-interval repair is possible. Otherwise, we map back the set of intervals $\{\hat{\tau}_j\}$ into a set of interval repairs $\mathcal{T}^*$ for the original formula φ according to the transformations in step 2 and return $\mathcal{T}^*$.

We provide an example of predicate repair below, while time interval repair is exemplified in Section 10.5.

[2]While the second transformation introduces a new interval $[0, a]$, its parameters are directly linked to the ones of the original interval $[a, b]$ (now inherited by the **F** operator) and will be accordingly processed by the repair routine.

time	0	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8
s_{l1}	0	0	0	0	0	-0.26	0	0	0	0
s_{u2}	0	0	0	0	0	0	-0.07	0	0	0

Table 10.1: Slack values over a single horizon, for $\Delta t = 0.2$ and $H = 10$.

Example 5 (Collision Avoidance)**.** *We diagnose the specifications introduced in Example 2. To formulate the synthesis problem, we assume a horizon $H = 10$ and a discretization step $\Delta t = 0.2$. The system is found infeasible at the first MPC run, and Diagnosis detects the infeasibility of $\varphi_1 \wedge \varphi_2$ at time $t = 6$. Intuitively, given the allowed range of accelerations for the ego vehicle, both cars end up entering the forbidden box at the same time.*

Algorithm 7 chooses to repair φ_1 by adding slacks to all of its predicates, such that $\varphi_1' = (-0.5 - s_{l1} \leq y_t^{\text{ego}} \leq 0.5 + s_{u1}) \wedge (-0.5 - s_{l2} \leq x_t^{\text{adv}} \leq 0.5 + s_{u2})$. Table 10.1 shows the optimal slack values at each t, while s_{u1} and s_{l2} are set to zero at all t.

We can then conclude that the specification replacing φ_1 with φ_1'

$$\varphi_1' = \mathbf{G}_{[0,\infty)} \neg \big((-0.24 \leq y_t^{\text{ego}} \leq 0.5) \wedge (-0.5 \leq x_t^{\text{adv}} \leq 0.43) \big) \tag{10.14}$$

is feasible, i.e., the cars will not collide, but the original requirement was overly demanding.

Alternatively, the user can choose to run the repair procedure on φ_2 and change its predicate as $(1.5 - s_l \leq a_t^{\text{ego}} \leq 2.5 + s_u)$. In this case, we decide to stick with the original requirement on collision avoidance, and tune, instead, the control "effort" to satisfy it. Under the assumption of constant acceleration (and bounds), the slacks will be the same at all t. We then obtain $[s_l, s_u] = [0.82, 0]$, which ultimately turns into $\varphi_2' = \mathbf{G}_{[0,\infty)}(0.68 \leq a_t^{\text{ego}} \leq 2.5)$. The ego *vehicle should then slow down to prevent entering the forbidden box at the same time as the other car. This latter solution is, however, suboptimal with respect to the l_1-norm selected in this example when both repairs are allowed.*

Our algorithm offers the following guarantees, for which a proof is reported below.

Theorem 12 (Soundness)**.** *Given a controller synthesis problem $\mathcal{P} = (f_d, x^0, \varphi, J)$, such that* (10.1) *is infeasible at time t, let $\varphi' \in \texttt{REPAIR}_{\mathcal{D},\mathcal{T}}(\varphi)$ be the repaired formula returned from Algorithm 7 without human intervention, for a given set of predicates $\mathcal{D}$ or time interval $\mathcal{T}$. Then, $\mathcal{P}' = (f_d, x^0, \varphi', J)$ is feasible at time t and (φ', $\mathcal{D}$, $\mathcal{T}$) satisfy the minimality conditions in Problem 10.*

Proof (Theorem 12). Suppose $\mathcal{M}$ is the MILP encoding of $\mathcal{P}$ as defined in (10.10), φ' is the repaired formula, and $\mathcal{D}$ the set of diagnosed predicates, as returned by Algorithm 7. We start by discussing the case of predicate repair.

We let $\mathcal{M}'$ be the MILP encoding of $\mathcal{P}'$ and $\mathcal{D}^* \subseteq \mathcal{D}$ be the set of predicates that are fixed to provide φ', i.e., such that $s = (\mu^* - \mu) \neq 0$, with $\mu \in \mathcal{D}$. Algorithm 7 modifies $\mathcal{M}$ by introducing a slack variable $s_{\mu,t}$ into each constraint associated with an atomic predicate μ in $\mathcal{D}$ at time t. Such a transformation leads to a feasible MILP $\mathcal{M}''$ and an optimal slack set $\{s^*_{\mu,t} | \mu \in \mathcal{D}, t \in \{0, \ldots, H-1\}\}$.

We now observe that $\mathcal{M}'$ and $\mathcal{M}''$ are both relaxations of $\mathcal{M}$. In fact, we can view $\mathcal{M}'$ as a version of $\mathcal{M}$ in which only the constraints associated with the atomic predicates in $\mathcal{D}^*$ are relaxed. Therefore, each constraint having a nonzero slack variable in $\mathcal{M}''$ is also relaxed in $\mathcal{M}'$. Moreover, by (10.12), the relaxed constraints in $\mathcal{M}'$ are offset by the largest slack value over the horizon H. Then, because $\mathcal{M}''$ is feasible, $\mathcal{M}'$, and subsequently $\mathcal{P}'$, are feasible.

We now prove that (φ',$\mathcal{D}$) satisfy the predicate minimality condition of Problem 10. Let $\tilde{\varphi}$ be any formula obtained from φ after repairing a set of predicates $\tilde{\mathcal{D}}$ such that the resulting problem $\tilde{\mathcal{P}}$ is feasible. We recall that, by Definition 6, at least one predicate in $\mathcal{D}$ generates a conflicting constraint and must be repaired for $\mathcal{M}$ to become feasible. Then, $\tilde{\mathcal{D}} \cap \mathcal{D} \neq \emptyset$ holds. Furthermore, since Algorithm 7 iterates by diagnosing and relaxing constraints until feasibility is achieved, $\mathcal{D}$ contains all the predicates that can be responsible for the infeasibility of φ. In other words, Algorithm 7 finds all the IISs in the original optimization problem and allows relaxing any constraint in the union of the IISs. Therefore, repairing any predicate outside of $\mathcal{D}$ is redundant: a predicate repair set that only relaxes the constraints associated with predicates in $\bar{\mathcal{D}} = \tilde{\mathcal{D}} \cap \mathcal{D}$, by the same amount as in $\tilde{\varphi}$, and sets to zero the slack variables associated with predicates in $\mathcal{D} \setminus \bar{\mathcal{D}}$ is also effective and exhibits a smaller slack norm. Let $s_{\bar{\mathcal{D}}}$ be such a repair set and $\bar{\varphi}$ the corresponding repaired formula. $s_{\bar{\mathcal{D}}}$ and $s_{\mathcal{D}}$ can then be seen as two repair sets on the same predicate set. However, by the solution of Problem (10.11), we are guaranteed that $s_{\mathcal{D}}$ has minimum norm; then, $||s_{\mathcal{D}}|| \leq ||s_{\bar{\mathcal{D}}}||$ will hold for any such formulas $\bar{\varphi}$, and hence $\tilde{\varphi}$.

We now consider the MILP formulation $\mathcal{M}'$ associated with $\mathcal{P}'$ and φ' in the case of time-interval repairs. For each atomic predicate $\mu_i \in \mathcal{D}$, for $i \in \{1, \ldots, |\mathcal{D}|\}$, $\mathcal{M}'$ includes only the associated constraints evaluated over time intervals σ_i' for which the slack variables $\{s_{\mu_i,t}\}$ are zero. Such a subset of constraints is trivially feasible. All the other constraints enforcing the satisfaction of Boolean and temporal combinations of the atomic predicates in φ' cannot cause infeasibility with these atomic predicate constraints, or the associated slack variables $\{s_{\mu_i,t}\}$ would be non-zero. So, $\mathcal{M}'$ is feasible.

To show that (φ',$\mathcal{T}$) satisfy the minimality condition in Problem 10, we observe that, by the transformations in step 2 of the time-interval repair procedure, φ is logically equivalent to a formula $\hat{\varphi}$ which only contains *untimed* **U** and *timed* **G** operators. Moreover, $\hat{\varphi}$ and φ have the same interval parameters. Therefore, if the proposed repair set is minimal for $\hat{\varphi}$, this will also be the case for φ.

We now observe that Algorithm 7 selects, for each atomic predicate $\mu_i \in \mathcal{D}$ the largest interval σ_i' such that the associated constraints are feasible, i.e., their slack variables are zero after norm minimization[3]. Because feasible intervals for Boolean combinations

[3]Because we are not directly maximizing the sparsity of the slack vector, time-interval minimality is to be interpreted with respect to slack norm minimization. Directly maximizing the number of zero slacks is

of atomic predicates are obtained by intersecting these maximal intervals, and then propagated to the temporal operators, the length of the intervals of each **G** operator in $\hat{\varphi}$, hence of the temporal operators in φ, will also be maximal. $\square$

Theorem 13 (Completeness)**.** *Assume the controller synthesis problem $\mathcal{P} = (f_d, x^0, \varphi, J)$ results in* (10.1) *being infeasible at time t. If there exist a set of predicates $\mathcal{D}$ or time-intervals $\mathcal{T}$ such that there exists $\Phi \subseteq \texttt{REPAIR}_{\mathcal{D},\mathcal{T}}(\varphi)$ for which $\forall\ \phi \in \Phi$, $\mathcal{P}' = (f_d, x^0, \phi, J)$ is feasible at time t and (ϕ, $\mathcal{D}$, $\mathcal{T}$) are minimal in the sense of Problem 10, then Algorithm 7 returns a repaired formula φ' in Φ.*

Proof (Theorem 13). We first observe that Algorithm 7 always terminates with a feasible solution φ' since the set of MILP constraints to diagnose and repair is finite. We first consider the case of predicate repairs. Let $\mathcal{D}$ be the set of predicates modified to obtain $\phi \in \Phi$ and $\mathcal{D}'$ the set of diagnosed predicates returned by Algorithm 7. Then, by Definition 6 and the iterative approach of Algorithm 7, we are guaranteed that $\mathcal{D}'$ includes all the predicates responsible for inconsistencies, as also argued in the proof of Theorem 12. Therefore, we conclude $\mathcal{D} \subseteq \mathcal{D}'$. $s_{\mathcal{D}}$ and $s_{\mathcal{D}'}$ can then be seen as two repair sets on the same predicate set. However, by the solution of Problem (10.11), we are guaranteed that $s_{\mathcal{D}'}$ has minimum norm; then, $||s_{\mathcal{D}'}|| \leq ||s_{\mathcal{D}}||$ will hold, hence $\varphi' \in \Phi$.

We now consider the case of time-interval repair. If a formula $\phi \in \Phi$ repairs a set of intervals $\mathcal{T} = \{\tau_1, \ldots, \tau_l\}$, then there exists a set of constraints associated with atomic predicates in φ which are consistent in $\mathcal{M}$, the MILP encoding associated with ϕ, and make the overall problem feasible. Then, the relaxed MILP encoding $\mathcal{M}'$ associated with φ after slack norm minimization will also include a set of predicate constraints admitting zero slacks over the same set of time intervals as in $\mathcal{M}$, as determined by $\mathcal{T}$. Since these constraints are enough to make the entire problem $\mathcal{M}$ feasible, this will also be the case for $\mathcal{M}'$. Therefore, our procedure for time-interval repair terminates and produces a set of non-empty intervals $\mathcal{T}' = \{\tau_1', \ldots, \tau_l'\}$. Finally, because Algorithm 7 finds the longest intervals for which the slack variables associated with each atomic predicate are zero, we are also guaranteed that $||\tau_i'|| \geq ||\tau_i||$ for all $i \in \{1, \ldots, l\}$, as also argued in the proof of Theorem 12. We can then conclude that $\varphi' \in \Phi$ holds. $\square$

In the worst case, Algorithm 7 solves a number of MILP problem instances equal to the number of atomic predicates in the STL formula. While the complexity of solving a MILP is NP-hard, the actual runtime depends on the size of the MILP, which is $O(H \cdot |\varphi|)$, where H is the length of the horizon and $|\varphi|$ is the number of predicates and operators in the STL specification.

also possible but computationally more intensive.

10.5 Contracts

In this section, we consider specifications provided in the form of a contract $(\varphi_e, \varphi_e \to \varphi_s)$, where φ_e is a STL formula expressing the assumptions, i.e., the set of behaviors assumed from the environment, while φ_s captures the guarantees, i.e., the behaviors promised by the system in the context of the environment. To repair contracts, we can capture tradeoffs between assumptions and guarantees in terms of minimization of a weighted norm of slacks. We describe below our results for both non-adversarial and adversarial environments.

Non-Adversarial Environment

For a contract, we make a distinction between controlled inputs u_t and uncontrolled (environment) inputs w_t of the dynamical system. In this section we assume that the environment signal $\mathbf{w}^H$ can be predicted over a finite horizon and set to a known value for which the controller must be synthesized. With $\varphi \equiv \varphi_e \to \varphi_s$, equation (10.2) reduces to:

$$\begin{array}{ll} \underset{\mathbf{u}^H}{\text{minimize}} & J(\xi(x^0, \mathbf{u}^H, \mathbf{w}^H)) \\ \text{subject to} & \xi(x^0, \mathbf{u}^H, \mathbf{w}^H) \models \varphi. \end{array} \tag{10.15}$$

Because of the similarity of Problem (10.15) and Problem (10.1), we can then diagnose and repair a contract using the methodology illustrated in Section 10.4. However, to reflect the different structure of the specification, i.e., its partition into assumption and guarantees, we adopt a weighted sum of the slack variables in Algorithm 7, allocating different weights to predicates in the assumption and guarantee formulae. We can then provide the same guarantees as in Theorems 12 and 13, where $\varphi \equiv \varphi_e \to \varphi_s$ and the minimality conditions are stated with respect to the weighted norm.

Example 6 (Non-adversarial Race)**.** *We consider Example 3 with the same discretization step $\Delta t = 0.2$ and horizon $H = 10$ as in Example 2. The MPC scheme results infeasible at time 1. In fact, we observe that ψ_e is true as $v_0^{\text{adv}} \geq 0.5$. Since $v_1^{ego} = 0.2$, the predicate $\psi_{s2} = \mathbf{G}_{[0.2,\infty)}(v_t^{\text{ego}} \geq 0.5)$ in ψ_s is found to be failing. As in Section 10.4, we can modify the conflicting predicates in the specification by using slack variables as follows: $v_t^{\text{adv}} + s_e(t) \geq 0.5$ (assumptions) and $v_t^{\text{ego}} + s_s(t) \geq 0.5$ (guarantees). However, we also assign a set of weights to the assumption (λ_e) and guarantee (λ_s) predicates, our objective being $\lambda_e|s_e| + \lambda_s|s_s|$. By setting $\lambda_s > \lambda_e$, we encourage modifications in the assumption predicate, thus obtaining $s_e = 0.06$ at time 0 and zero otherwise, and $s_s = 0$ at all times. We can then set $\psi_e' = (v_0^{\text{adv}} \geq 0.56)$, which falsifies ψ_e' so that $\psi_e' \to \psi_s$ is satisfied. Alternatively, by setting $\lambda_s < \lambda_e$, we obtain the slack values in Table 10.2, which lead to the following predicate repair: $\psi_{s2}' = \mathbf{G}_{[0.2,\infty)}(v_t^{\text{ego}} \geq 0.2)$.*

We can also modify the time interval of the temporal operator associated with ψ_{s2} to repair the overall specification. To do so, Algorithm 7 uses the parse tree of $\psi_e \to \psi_s$ in Figure 10.3. For any of the leaf node predicates μ_i, $i \in \{1,2,3\}$, we get a support $\sigma_i = [0,9]$, which is only

time	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8
s_s	0.31	0.11	0	0	0	0	0	0	0

Table 10.2: Slack variables used in Example 3 and 6.

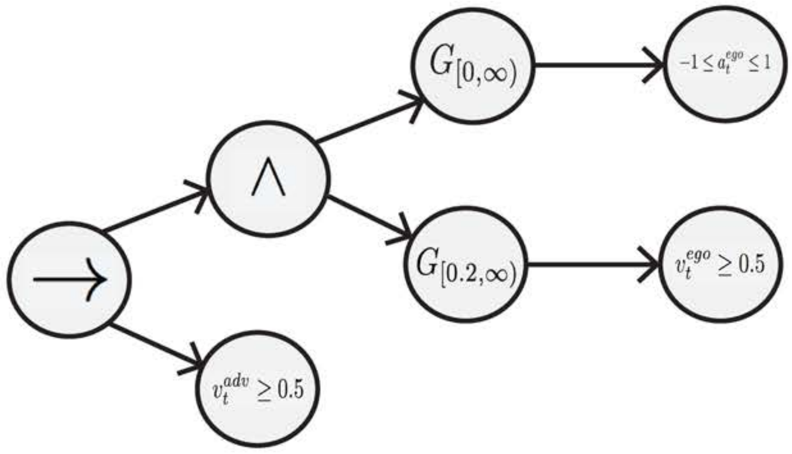


Figure 10.3: Parse tree of $\psi \equiv \psi_e \rightarrow \psi_s$ used in Example 3 and 6.

limited by the finite horizon H. Then, based on the slack values in Table 10.2, we can conclude $\sigma_1' = \sigma_2' = [0, 9]$ (the optimal slack values for these predicates are always zero), while $\sigma_3' = [3, 9]$. For the given syntax tree, we also have $\sigma_1^* = \sigma_1'$, $\sigma_2^* = \sigma_2'$, and $\sigma_3^* = \sigma_3'$ for the temporal operator nodes that are parent nodes of μ_1, μ_2, and μ_3, respectively. Since none of the above intervals is empty, a time interval repair is indeed possible by modifying the time interval of the parent node of μ_3, thus achieving $\tau_3^* = \sigma_3^*$. This leads to the following proposed sub-formula $\psi_{s2}' = \mathbf{G}_{[0.6,\infty)}(v_t^{\text{ego}} \geq 0.5)$. In this example, repairing the specification over the first horizon is enough to guarantee controller realizability in the future. We can then keep the upper bound of the **G** *operator to infinity.*

Adversarial Environment

When the environment can behave adversarially, the control synthesis problem assumes the structure in (10.2). Specifically, in this chapter, we allow w_t to lie in an interval $[w_{\min}, w_{\max}]$ at all times; this corresponds to the STL formula $\varphi_w = \mathbf{G}_{[0,\infty)}(w_{\min} \leq w_t \leq w_{\max})$. We decompose a specification φ of the form $\varphi_w \wedge \varphi_e \rightarrow \varphi_s$, representing the contract, as $\varphi \equiv \varphi_w \rightarrow \psi$, where $\psi \equiv (\varphi_e \rightarrow \varphi_s)$. Our diagnosis and repair method is summarized in Algorithm 10.

We first check the satisfiability of the control synthesis problem by examining whether

Algorithm 10 Diagnose Repair Adversarial

```
 1: Input: $\mathcal{P}$
 2: Output: $\mathbf{u}^H$, $\mathcal{P}'$
 3: $(J, C) \leftarrow$ GenMILP($\mathcal{P}$)
 4: $(\mathbf{u}_0^H, \mathbf{w}_0^H, sat) \leftarrow$ CheckSAT($J, C$)
 5: if $sat$ then
 6:     $\mathcal{W}^*_{cand} \leftarrow$ SolveCEGIS($\mathbf{u}_0^H, \mathcal{P}$)
 7:     $\mathcal{W}_{cand} \leftarrow \mathcal{W}^*_{cand}$
 8:     while $\mathcal{W}_{cand} \neq \emptyset$ do
 9:         $\mathcal{P}_w \leftarrow$ RepairAdversarial($\mathcal{W}_{cand}, \mathcal{P}$)
10:         $\mathcal{W}_{cand} \leftarrow$ SolveCEGIS($\mathbf{u}_0^H, \mathcal{P}_w$)
11:     end while
12:     $\mathcal{W}_{cand} \leftarrow \mathcal{W}^*_{cand}$, $\mathcal{P}_\psi \leftarrow \mathcal{P}$
13:     while $\mathcal{W}_{cand} \neq \emptyset$ do
14:         $\mathcal{P}_\psi \leftarrow$ DiagnoseRepair($\mathcal{P}_\psi$)
15:         $\mathcal{W}_{cand} \leftarrow$ SolveCEGIS($\mathbf{u}_0^H, \mathcal{P}_\psi$)
16:     end while
17:     $\mathcal{P}' \leftarrow$ FindMin($\mathcal{P}_w, \mathcal{P}_\psi$)
18: end if
```

there exists a pair of $\mathbf{u}^H$ and $\mathbf{w}^H$ for which problem (10.2) is feasible (`CheckSAT` routine):

$$\begin{aligned} \underset{\mathbf{u}^H, \mathbf{w}^H}{\text{minimize}} \quad & J(\xi(x^0, \mathbf{u}^H, \mathbf{w}^H)) \\ \text{subject to} \quad & \xi(x^0, \mathbf{u}^H, \mathbf{w}^H) \models \varphi \\ & \mathbf{w}^H \models \varphi_w \wedge \varphi_e. \end{aligned} \tag{10.16}$$

If problem (10.16) is unsatisfiable, we can use the techniques introduced in Section 10.4 and 10.5 to diagnose and repair the infeasibility. Therefore, in the following, we assume that (10.16) is satisfiable, hence there exist $\mathbf{u}_0^H$ and $\mathbf{w}_0^H$ that solve (10.16).

To check realizability, we use the following CEGIS loop (SolveCEGIS routine) [148]. By first fixing the control trajectory to $\mathbf{u}_0^H$, we find the worst case disturbance trajectory $\mathbf{w}_1^H$ that minimizes the robustness value of φ by solving the following problem:

$$\begin{aligned} \underset{\mathbf{w}^H}{\text{minimize}} \quad & \rho^\varphi(\xi(x^0, \mathbf{u}^H, \mathbf{w}^H), 0) \\ \text{subject to} \quad & \mathbf{w}^H \models \varphi_e \wedge \varphi_w \end{aligned} \tag{10.17}$$

with $\mathbf{u}^H = \mathbf{u}_0^H$. The optimal $\mathbf{w}_1^H$ from (10.17) will falsify the specification if the resulting robustness value is below zero[4]. If this is the case, we look for a $\mathbf{u}_1^H$ which solves (10.15)

[4]A tolerance ρ_{min} can be selected to accommodate approximation errors, i.e., $\rho^\varphi(\xi(x^0, \mathbf{u}_0^H, \mathbf{w}_1^H), 0) < \rho_{min}$.

with the additional restriction of $\mathbf{w}^H \in \mathcal{W}_{cand} = \{\mathbf{w}_1^H\}$. If this step is feasible, we once again attempt to find a worst-case disturbance sequence $\mathbf{w}_2^H$ that solves (10.17) with $\mathbf{u}^H = \mathbf{u}_1^H$: this is the counterexample-guided inductive step. At each iteration i of this CEGIS loop, the set of candidate disturbance sequences $\mathcal{W}_{cand}$ expands to include $\mathbf{w}_i^H$. If the loop terminates at iteration i with a successful $\mathbf{u}_i^H$ (one for which the worst case disturbance $\mathbf{w}_i^H$ in (10.17) has positive robustness), we conclude that the formula φ is realizable.

The CEGIS loop may not terminate if the set $\mathcal{W}_{cand}$ is infinite. We, therefore, run it for a maximum number of iterations. If SolveCEGIS fails to find a controller sequence prior to the timeout, then (10.15) is infeasible for the current $\mathcal{W}_{cand}$, i.e., there is no control input that can satisfy φ for all disturbances in $\mathcal{W}_{cand}$. We conclude that the specification is not realizable (or, equivalently, the contract is inconsistent). While this infeasibility can be repaired by modifying ψ based on the techniques in Section 10.4 and Section 10.5, an alternative solution is to repair φ_w by minimally pruning the bounds on w_t (RepairAdversarial routine).

To do so, given a small tolerance $\epsilon \in \mathbb{R}^+$, we find

$$w_u = \max_{\substack{\mathbf{w}_i^H \in \mathcal{W}_{cand} \\ t \in \{0,\ldots,H-1\}}} w_{i,t} \qquad w_l = \min_{\substack{\mathbf{w}_i^H \in \mathcal{W}_{cand} \\ t \in \{0,\ldots,H-1\}}} w_{i,t} \tag{10.18}$$

and define $s_u = w_{\max} - w_u$ and $s_l = w_l - w_{\min}$. We then use s_u and s_l to update the range for w_t in φ_w to a maximal interval $[w'_{\min}, w'_{\max}] \subseteq [w_{\min}, w_{\max}]$ and such that at least one $\mathbf{w}_i^H \in \mathcal{W}_{cand}$ is excluded. Specifically, if $s_u \le s_l$, we set $[w'_{\min}, w'_{\max}] = [w_{\min}, w_u - \epsilon]$; otherwise we set $[w'_{\min}, w'_{\max}] = [w_l + \epsilon, w_{\max}]$. The smaller the value of ϵ, the larger the resulting interval.

Finally, we use the updated formula φ'_w to run `SolveCEGIS` again until a realizable control sequence $\mathbf{u}^H$ is found. For improved efficiency, the linear search proposed above to find the updated bounds $w'_{\min}$ and $w'_{\max}$ can be replaced by a binary search. Moreover, in Algorithm 10, assuming a predicate repair procedure, `FindMin` provides the solution with minimum slack norm between the ones repairing ψ and φ_w.

Example 7 (Adversarial Race). *We consider the specification in Example 4. For the same horizon as in the previous examples, after solving the satisfiability problem, for the fixed $\mathbf{u}_0^H$, the CEGIS loop returns $a_t^{\mathrm{adv}} = 2$ for all $t \in \{0, \ldots, H-1\}$ as the single element in $\mathcal{W}_{cand}$ for which no controller sequence can be found. We then choose to tighten the environment assumptions to make the controller realizable, by shrinking the bounds on a_t^{adv} by using Algorithm 10 with $\epsilon = 0.01$. After a few iterations, we finally obtain $w'_{\min} = 0$ and $w'_{\max} = 1.24$, and therefore $\phi'_w = \mathbf{G}_{[0,\infty)}(0 \le a_t^{\mathrm{adv}} \le 1.24)$.*

To account for the error introduced by ϵ, given $\varphi' \in \mathtt{REPAIR}_{\mathcal{D},\mathcal{T}}(\varphi)$, we say that ($\varphi'$, $\mathcal{D}$, $\mathcal{T}$) are ϵ-minimal if the magnitudes of the predicate repairs (predicate slacks) or time-interval repairs differ by at most ϵ from a minimal repair in the sense of Problem

11. Assuming that SolveCEGIS terminates before reaching the maximum number of iterations[5], the following theorems state the properties of Algorithm 10.

Theorem 14 (Soundness)**.** *Given a controller synthesis problem $\mathcal{P} = (f_d, x^0, \varphi, J)$, such that* (10.2) *is infeasible at time t, let $\varphi' \in \texttt{REPAIR}_{\mathcal{D},\mathcal{T}}(\varphi)$ be the repaired formula returned from Algorithm 10 for a given set of predicates $\mathcal{D}$ or time interval $\mathcal{T}$. Then, $\mathcal{P}' = (f_d, x^0, \varphi', J)$ is feasible at time t and (φ', $\mathcal{D}$, $\mathcal{T}$) is ϵ-minimal.*

Proof (Theorem 14). We recall that $\varphi \equiv \varphi_w \to \psi$. Moreover, Algorithm 10 provides the solution with minimum slack norm between the ones repairing ψ and ϕ_w in the case of predicate repair. Then, when $\psi = \varphi_e \to \varphi_s$ is modified using Algorithm 7, soundness is guaranteed by Theorem 12 and the termination of the CEGIS loop. On the other hand, assume Algorithm 10 modifies the atomic predicates in ϕ_w. Then, the RepairArdversarial routine and (10.18), together with the termination of the CEGIS loop, assure that φ_w is also repaired in such a way that the controller is realizable, and ϵ-optimal (i.e., the length of the bounding box around w_t differs from the maximal interval length by at most ϵ), which concludes our proof.

□

Theorem 15 (Completeness)**.** *Assume the controller synthesis problem $\mathcal{P} = (f_d, x^0, \varphi, J)$ results in* (10.2) *being infeasible at time t. If there exist a set of predicates $\mathcal{D}$ and time-intervals $\mathcal{T}$ such that there exists $\Phi \subseteq \texttt{REPAIR}_{\mathcal{D},\mathcal{T}}(\varphi)$ for which $\forall\ \phi \in \Phi$, $\mathcal{P}' = (f_d, x^0, \phi, J)$ is feasible at time t and (ϕ, $\mathcal{D}$, $\mathcal{T}$) is ϵ-minimal, then Algorithm 10 returns a repaired formula φ' in Φ.*

Proof (Theorem 15). As discussed in the proof of Theorem 14, if Algorithm 10 modifies $\psi = \varphi_e \to \varphi_s$ using Algorithm 7, completeness is guaranteed by Theorem 13 and the termination of the CEGIS loop. On the other hand, let us assume there exists a minimum norm repair for the atomic predicates of φ_w, which returns a maximal interval $[w'_{\min}, w'_{\max}] \subseteq [w_{\min}, w_{\max}]$. Then, given the termination of the CEGIS loop, by repeatedly applying (10.18) and RepairAdversarial, we produce a predicate repair such that the corresponding interval $[w''_{\min}, w''_{\max}]$ makes the control synthesis realizable and is maximal within an error bounded by ϵ (i.e., its length differs by at most ϵ from the one of the maximal interval $[w'_{\min}, w'_{\max}]$). Hence, $\varphi' \in \Phi$ holds. □

10.6 Case Studies

We developed the toolbox DiaRY (Diagnosis and Repair for sYnthesis)[6] implementing our algorithms. DiaRY uses Yalmip [114] to formulate the optimization problems and

[5]Under failing assumptions, Algorithm 10 terminates with `UNKNOWN`.
[6]https://github.com/shromonag/DiaRY

Gurobi [73] to solve them. It interfaces to different synthesis tools, e.g., BluSTL[7] and CrSPrSTL[8]. Here, we summarize some of the results of DiaRY for diagnosis and repair.

Autonomous Driving

We consider the problem of synthesizing a controller for an autonomous vehicle in a city driving scenario. We analyze the following two tasks: (i) changing lanes on a busy road; (ii) performing an unprotected left turn at a signalized intersection. We use the dynamics model introduced in Chapter 2.4. To determine the control strategy, we linearize the overall system dynamics around the initial state at each run of the MPC, which is completed in less than 2 s on a 2.3-GHz Intel Core i7 processor with 16-GB memory. We further impose the following constraints on the *ego* vehicle (i.e., the vehicle under control): (i) a minimum distance must be established between the *ego* vehicle and other cars on the road to avoid collisions; (ii) the *ego* vehicle must obey the traffic lights; (iii) the *ego* vehicle must stay within its road boundaries.

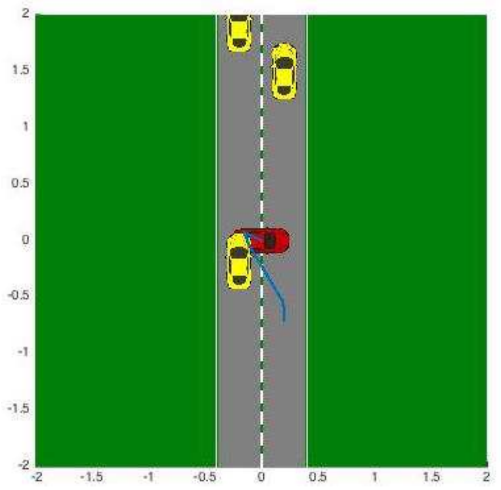
Lane Change

We consider a lane change scenario on a busy road as shown in Figure 10.4a. The *ego* vehicle is in red. *Car 1* is at the back of the left lane, *Car 2* is in the front of the left lane, while *Car 3* is on the right lane. The states of the vehicles are initialized as follows: $x_0^{\text{Car 1}} = [-0.2 \;\; -1.5 \;\; \frac{\pi}{2} \;\; 0.5]^\top$, $x_0^{\text{Car 2}} = [-0.2 \;\; 1.5 \;\; \frac{\pi}{2} \;\; 0.5]^\top$, $x_0^{\text{Car 3}} = [0.2 \;\; 1.5 \;\; \frac{\pi}{2} \;\; 0]^\top$, and $x_0^{\text{ego}} = [0.2 \;\; -0.7 \;\; \frac{\pi}{2} \;\; 0]^\top$. The control inputs for *ego* and *Car 3* are initialized at $[0 \;\; 0]^\top$; the ones for *Car 1* and *Car 2* are set to $u_0^{\text{Car 1}} = [0 \;\; 1]^\top$ and $u_0^{\text{Car 2}} = [0 \;\; -0.25]^\top$. The objective of *ego* is to safely change lane, while satisfying the following requirements:

$$\begin{aligned} \varphi_{\text{str}} &= \mathbf{G}_{[0,\infty)}(|u_1| \leq 2) && \text{Steering Bounds} \\ \varphi_{\text{acc}} &= \mathbf{G}_{[0,\infty)}(|u_2| \leq 1) && \text{Acceleration Bounds} \\ \varphi_{\text{vel}} &= \mathbf{G}_{[0,\infty)}(|v| \leq 1) && \text{Velocity Bounds} \end{aligned} \tag{10.19}$$

The solid blue line in Figure 10.4 is the trajectory of *ego* as obtained from our MPC scheme, while the dotted green line is the future trajectory pre-computed for a given horizon at a given time. MPC becomes infeasible at time $t = 1.2$ s when the no-collision requirement is violated, and a possible collision is detected between the *ego* vehicle and *Car 1* before the lane change is completed (Fig. 10.4a). Our solver takes 2 s, out of which 1.4 s are needed to generate all the IISs, consisting of 39 constraints. To make the system feasible, the proposed repair increases both the acceleration bounds and the velocity bounds on

[7]https://github.com/BluSTL/BluSTL
[8]https://github.com/dsadigh/CrSPrSTL


(a)

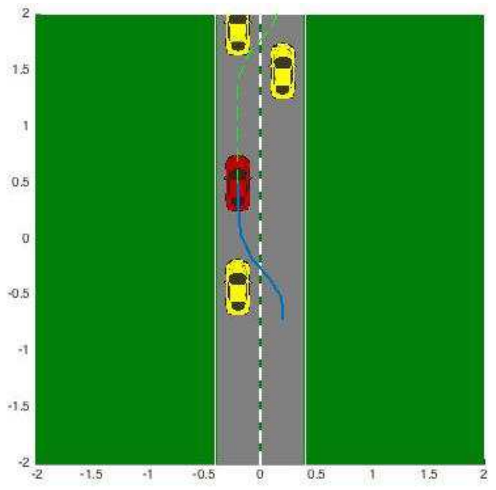

(b)

Figure 10.4: Changing lane is infeasible at $t = 1.2$ s in (a) and is repaired in (b).

the *ego* vehicle as follows:

$$\begin{aligned} \varphi_{\text{acc}}^{\text{new}} &= \mathbf{G}_{[0,\infty)}(|u_2| \leq 3.5) \\ \varphi_{\text{vel}}^{\text{new}} &= \mathbf{G}_{[0,\infty)}(|v| \leq 1.54). \end{aligned} \tag{10.20}$$

When replacing the initial requirements φ_{acc} and φ_{vel} with the modified ones, the revised MPC scheme allows the vehicle to travel faster and safely complete a lane change maneuver, without risks of collision, as shown in Figure 10.4b.

Unprotected Left Turn

In the second scenario, we would like the ego vehicle to perform an unprotected left turn at a signalized intersection, where the ego vehicle has a green light and is supposed to yield to oncoming traffic, represented by the yellow cars crossing the intersection in Figure 10.5. The environment vehicles are initialized at the states $x_0^{\text{Car 1}} = [-0.2\ \ 0.7\ \ -\frac{\pi}{2}\ \ 0.5]^\top$ and $x_0^{\text{Car 2}} = [-0.2\ \ 1.5\ \ -\frac{\pi}{2}\ \ 0.5]^\top$, while the ego vehicle is initialized at $x_0^{\text{ego}} = [0.2\ \ -0.7\ \ \frac{\pi}{2}\ \ 0]^\top$. The control input for each vehicle is initialized at $[0\ \ 0]^\top$. Moreover, we use the same bounds as in (10.19).

The MPC scheme becomes infeasible at $t = 2.1$ s. The solver takes 5 s, out of which 2.2 s are used to generate the IISs, including 56 constraints. As shown in Figure 10.5a, the ego vehicle yields in the middle of intersection for the oncoming traffic to pass. However, the traffic signal turns red in the meanwhile, and there is no feasible control input for the ego vehicle without breaking the traffic light rules. Since we do not allow modifications to the traffic light rules, the original specification is repaired again by increasing the bounds

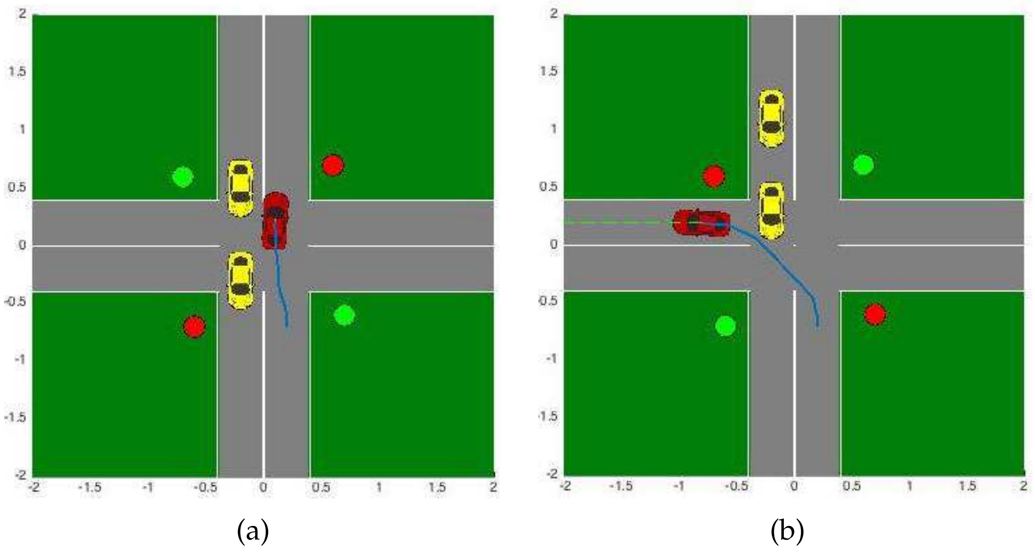


Figure 10.5: Left turn becomes infeasible at time $t = 2.1$ s in (a) and is repaired in (b).

on acceleration and velocity, thus obtaining:

$$\begin{aligned} \varphi_{\text{acc}}^{\text{new}} &= \mathbf{G}_{[0,\infty)}\big(|u_2| \leq 11.903\big) \\ \varphi_{\text{vel}}^{\text{new}} &= \mathbf{G}_{[0,\infty)}\big(|v| \leq 2.42\big). \end{aligned} \tag{10.21}$$

As shown by the trajectory in Figure 10.5b, under the assumptions and initial conditions of our scenario, higher allowed velocity and acceleration make the ego vehicle turn before the oncoming cars get close or cross the intersection.

Quadrotor Control

We assume a quadrotor dynamical model similar to the one discussed in Chapter 2.4.

Our goal is to synthesize a strategy for the quadrotor to travel from a starting position $[x^0, y^0, z^0] = [0, 0, -0.4]$ with zero roll, pitch, and yaw angles, i.e., $[\phi^0, \theta^0, \psi^0] = [0, 0, 0]$, to a destination $[x^d, y^d, z^d] = [1, 1, -0.1]$, still with zero roll, pitch, and yaw. All the other elements in the state vector and the control input are initialized at zero. We define the following constraints on the quadrotor:

$$\begin{aligned} &\varphi_{\text{h}} = \mathbf{G}_{[0,\infty)}(-1.1 \leq z \leq 0) && \text{Height of Flight Bounds} \\ &\varphi_{\text{roll}} = \mathbf{G}_{[0,\infty)}(|u_1| \leq 0.3) && \text{Roll Bounds} \\ &\varphi_{\text{pitch}} = \mathbf{G}_{[0,\infty)}(|u_2| \leq 0.3) && \text{Pitch Bounds} \\ &\varphi_{\text{thr}} = \mathbf{G}_{[0,\infty)}(0 \leq u_4 \leq 6.5) && \text{Thrust Bounds.} \end{aligned} \tag{10.22}$$

Our MPC scheme becomes infeasible at time $t = 0.675$ s because φ_{h} and φ_{roll} are both violated. Given the bounds on the control inputs, the trajectory is not guaranteed to lie in

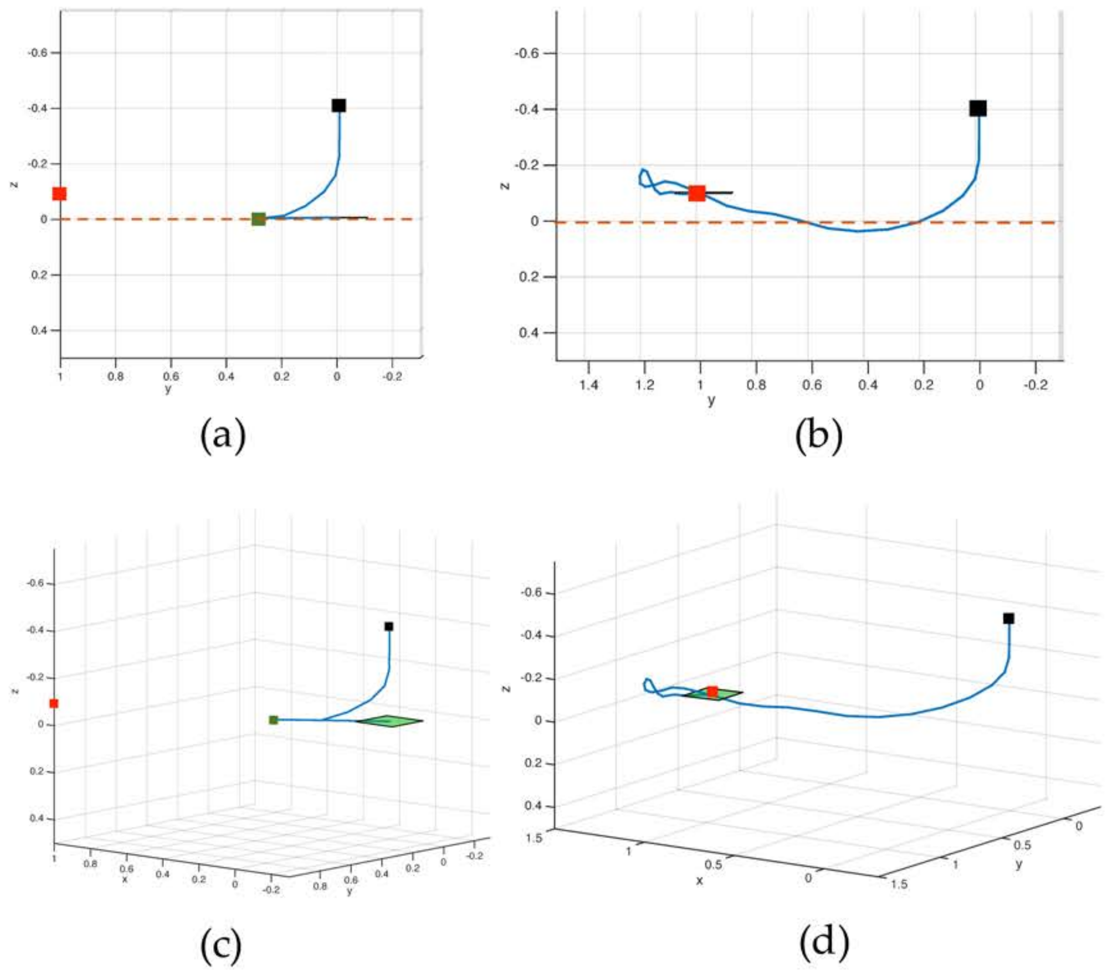


Figure 10.6: Diagnosis and repair of infeasibilities in quadrotor control. The top and bottom figures show, respectively, a 2D and 3D projection of the trajectory (blue line) of the quadrotor, represented as a green rectangle. The black square marks the initial position while the red square marks the goal. As shown in Fig. (a) and (c), the original specification becomes infeasible at time $t = 0.675$, which is marked by a green square along the trajectory, when the quadrotor hits the boundary represented by the dotted red line. After updating the specification, controller synthesis becomes feasible, as shown in Fig. (b) and (d), where the quadrotor reaches the final position, at the cost of passing through the red dotted line.

the desired region. This is visualized in Figure 10.6 (a) and (c), respectively showing a two-dimensional and three-dimensional projection of the quadrotor trajectory. The solid blue line shows the path computed by the MPC framework and taken by the quadrotor, aiming at traveling from the origin, marked by a black square, to the target, marked by a red square. Because of the bounds on the control inputs, the quadrotor touches the boundary of the allowed region (along the z axis) at $t = 0.675$ s.

The solver takes less than 0.1 s to generate the IIS, including 32 constraints, out of which 11 constraints are associated with the predicates in φ_h and φ_roll. Our algorithm adds a slack of 1.58 to the upper bound on z in φ_h. We then modify φ_h to:

$$\varphi_\text{h}^\text{new} = \mathbf{G}_{[0,\infty)}(-1.1 \leq z \leq 1.58), \tag{10.23}$$

thus allowing the quadrotor to violate the upper bound on the vertical position during the maneuver. The new specification makes the problem realizable, and the resulting trajectory is shown in blue in Figure 10.6 (b) and (d). We can view the above slack as the estimated margin from the boundary needed for the quadrotor to complete the maneuver based on the linearized model of the dynamics. As apparent from Figure 10.6b, such a margin is much larger than the space actually used by the quadrotor to complete its maneuver. A better estimate can be achieved by using a finer time interval for linearizing the dynamics and executing the controller. While the solution above provides the minimum slack norm for the given linearization, it is still possible to notify DiaRY that φ_h must be regarded as a hard constraint, e.g., used to mark a rigid obstacle. In this case, DiaRY tries to relax the bounds on the control inputs to achieve feasibility.

10.7 Chapter Summary

We presented a set of algorithms for diagnosis and repair of STL specifications in the setting of controller synthesis for hybrid systems using a model predictive control scheme. Given an unrealizable specification, our algorithms can detect possible reasons for infeasibility and suggest repairs to make it realizable. We showed the effectiveness of our approach on the synthesis of controllers for several applications. As future work, we plan to investigate techniques that better leverage the structure of the STL formulae and extend to a broader range of environment assumptions in the adversarial setting.

Chapter 11

Final Words

We envision a world where autonomous systems are interacting with us seamlessly. Robots are integrated into our society, and are cognizant of their interaction with each other, with the humans, and the environment they live in. They are also capable of reasoning about their actions, and they plan safe strategies that humans can trust and rely on.

This thesis has been a key step towards the goal of understanding and designing a framework for modeling the interaction between humans and other agents, which is crucial for correct-by-construction design and analysis of human-robot systems. In this chapter, we would like to conclude this dissertation by reflecting on our approach and discussing some of the limitations and future directions.

We have focused on two main control approaches in this work: *interaction-aware control*, and *safe control*.

Our work in *interaction-aware control* requires models of humans and the interaction between the human and autonomy as we have discussed in Chapter 3, Chapter 4, and Chapter 5. In Chapter 6, we have addressed the initial steps in increasing trust in human models by studying individual variations from learned human models. However, there are still many avenues to explore in designing better and more accurate models of humans in safety-critical human-robot systems such as approaches in behavioral economics to model human's bounded rationality [66]. Similarly, we have used *active* methods in learning human's reward functions for both information gathering and comparison-based learning (Chapter 4, Chapter 5). Such active learning techniques enable us to only extract informative data and quickly converge to the desired outcome. In the future, we plan to utilize such methods in automatic testing and validation of human-robot systems.

Our work in safe control, addresses correct-by-construction control from temporal logic specifications in reactive, continuous, and probabilistic settings (Chapter 7,Chapter 8, and Chapter 9). However, as more learning components are used in control and estimation of robotics systems, our formal techniques need to be modified to address safety guarantees in such settings. Chapter 9 is only a first step towards this direction, and future work in control and verification needs to address the challenges in safe and verified AI. We discuss some of these challenges in the next section. Further, in Chapter 10, we studied the problem of diagnosing and repairing infeasible specifications. Similarly, as more data-driven and learned components appear in our systems, we are required to design algorithms that are capable of diagnosis and repair with such learned components, which is in the direction of addressing various challenges in explainable and accountable AI.

11.1 Challenges in Safe and Interactive AI

Systems that heavily use AI, henceforth referred to as *AI-based systems*, have had a significant impact in society in domains that include healthcare, transportation, social networking, e-commerce, education, etc. This growing societal-scale impact has brought with it a set of risks and concerns including errors in AI software, cyber-attacks, and safety of AI-based systems [153, 42, 10]. As we have seen in this dissertation, the AI-based systems have had an incredible impact on the design and control of human-robot systems. We address some of the challenges that need to be addressed in safe and interactive AI for human-robot systems. Some of the material in this section appeared in [164], and we refer the reader to that paper for further details.

Environment Modeling

In the traditional success stories for formal verification, such as verifying cache coherence protocols or device drivers, the *interface* between the system and its environment is well defined. Moreover, while the environment itself may not be known, it is usually acceptable to model it as a *non-deterministic* process subject to constraints specified in a suitable logic or automata-based formalism. Typically such an environment model is "over-approximate", meaning that it may include more environment behaviors than are possible.

We see systems based on AI or machine learning (ML) as being quite different. Consider an autonomous vehicle operating in rush-hour traffic in an urban environment. It may be impossible even to precisely define the interface between the system and environment (i.e., to identify the variables/features of the environment that must be modeled), let alone to model all possible behaviors of the environment. Even if the interface is known, non-deterministic or over-approximate modeling is likely to produce too many spurious bug reports, rendering the verification process useless in practice.

Similarly, for human-robot systems, human agents are a key part of the environment and/or system. Researchers have attempted modeling humans as non-deterministic or stochastic processes with the goal of verifying the correctness of the overall system [152, 157]. Given the variability and uncertainty in human behavior, a data-driven approach based on machine learning is usually necessary. Such an approach, in turn, is sensitive to the quality of data. For example, the technique of inverse reinforcement learning [128] can be used for learning the reward function of human agents [2, 188]. However, accuracy of the learned reward function depends on the expressivity of the hand-coded features by the designer and the amount and variety of the data collected. In order to achieve *verified AI* for such human-in-the-loop systems, we need to address the limitations of the current human modeling techniques and provide guarantees about their *prediction accuracy and convergence*. When learned models are used, one must represent any *uncertainty in the learned parameters* as a first-class entity in the model, and take that into account in verification and control. The challenge, then, is to come up with a method of environment modeling that allows one to provide provable guarantees on the system's behavior even when there is considerable uncertainty about the environment.

Formal Specification

Formal verification critically relies on having a formal specification – a precise, mathematical statement of what the system is supposed to do. However, the challenge of coming up with a high-quality formal specification is well known, even in application domains in which formal verification has found considerable success (see, e.g., [22]).

This challenge is only exacerbated in AI-based systems. Consider a module in an autonomous vehicles that performs object recognition, distinguishing humans from other objects. What is the specification for such a module? How might it differ from the specifications used in traditional applications of formal methods? What should the specification language be, and what tools can one use to construct a specification?

Thus, we need to find an effective method to specify desired and undesired properties of systems that use AI- or ML-based components.

Modeling Systems that Learn

In most traditional applications of formal verification, the system is precisely known: it is a C program, or a circuit described in a hardware description language. The system modeling problem is primarily concerned with reducing the size of the system to a more tractable representation by abstracting away irrelevant details.

AI-based systems lead to a very different challenge for system modeling. A major challenge is the use of *machine learning*, where the system evolves as it encounters new data and new situations. Modeling a deep neural network that has been trained on millions of data points can be challenging enough even if one "freezes" the training process: new abstraction techniques will be necessary. Additionally, the verification procedure must account for future changes in the learner as new data arrives. New techniques must be devised to formally model components based on machine learning.

Generating Training Data

Formal methods has proved effective for the systematic generation of test data in various settings including simulation-based verification of circuits (e.g., [93]) and finding security exploits in commodity software (e.g., [15]). In these cases, even though the end result is not a proof of correctness of the system, the generated tests raise the level of assurance in the system's correctness. Can the testing of AI-based systems leverage formal methods in a similar manner?

We have seen that various machine learning algorithms can fail under small adversarial perturbations [130, 57, 123, 70]. Learning algorithms promise to generalize from data, but such simple perturbations that fool the algorithms create concerns regarding their use in safety-critical applications such as autonomous driving. Such small perturbations might be even unrecognizable to humans, but drive the algorithm to misclassify the perturbed data. Recent efforts have started to combine ideas from formal methods and machine learning to address problems such as compositional falsification of machine learning components [48]. There are still many challenges left to devise techniques based on formal methods to systematically generate training and testing data for ML-based components.

Active Test Scenario Generation

Efficiently and actively generating test scenarios is also a key factor in verification of human-robot systems. For example, in the context of autonomous driving, today's automated cars measure the reliability of autonomy by the number of miles driven. However, the quality of driven miles is more important than its quantity. Using the power of formal methods through model checking and counterexample-based techniques, one challenge is to automatically generate representative test scenarios for human-robot systems in order to detect the interesting corner cases that potentially result in accidents.

Scalability of Verification and Optimization Engines

A constant question asked of formal verification is whether it can scale up to handle industrial designs. Much progress has been made in this regard, especially in the area of hardware verification, where formal methods are a standard part of the design flow.

However, in systems that use AI or ML, the scalability challenge is even greater. In addition to the scale of systems as measured by traditional metrics (dimension of state space, number of components, etc.), the *types* of components can be much more complex. For instance, in (semi-)autonomous driving, autonomous vehicles and their controllers need to be modeled as *hybrid systems* combining both discrete and continuous dynamics. Moreover, agents in the environment (humans, other vehicles) may need to be modeled as *probabilistic processes*. Finally, the requirements may involve not only traditional Boolean specifications on safety and liveness, but also *quantitative requirements* on system robustness and performance.

On a similar note, many of the performance measures or reward functions discussed in this dissertation are nonlinear and non-convex. Another challenge is to design scalable nonlinear optimization techniques that enable real-time computation for synthesizing safe and interaction-aware controllers.

Human's Bounded Rationality

Designing a framework for interaction between humans and autonomy requires illustrative learned human models. Under the assumption that humans are approximately rational, we have used the principle of maximum entropy to learn their reward functions [159, 158, 160]. However, in many extreme events (e.g., car accidents), human actions cannot be explained as agents who optimize a rational reward function. Understanding and modeling such extreme cases is an essential component in providing guarantees about human-robot systems. We believe exploring ideas from behavioral economics can facilitate designing systems that are aware of humans' bounded rationality [66]. This potentially further enables inference of probability of loss in such events and provides a systematic design of insurance mechanisms for semiautonomous systems (e.g., autonomous vehicles driving on roads shared with humans).

Human Variations

Data-driven human models such as human reward functions are usually constructed based on large datasets that give access to a single model of rationality. However, humans vary in how they handle different situations, and we cannot fit a single model to all humans. In safety-critical scenarios, we need to be able to quantize and measure how humans can differ from the fitted model. As a step towards verified human modeling, we have constructed an efficient (almost linear) algorithm to quantize and learn a distribution over the variations from the fitted model by querying individual humans on actively generated scenarios (see Chapter 6). However, more work is required in robustness analysis of human models in order to be able to use these results in control. Our work in Chapter 6 finds a sequence of falsifying actions for humans; however, one challenge is to utilize this falsifying sequence in an online manner to close the loop by synthesizing resilient controllers to human variations.

Shared Transportation Networks for Human-Robot Systems

Going beyond single agent systems, another challenging problem is to address human-robot systems at the system level, and explore its fundamental limits. Imagine a transportation network with fleets of autonomous and human-driven vehicles. If there are no human-driven vehicles, a central authority can plan routes for vehicles so as to minimize the total cost. Moreover, if there are no autonomous vehicles, vehicles follow an equilibrium, which might have a suboptimal total cost. The cost of an equilibrium can be worse than the optimum by a factor known as the price of anarchy. As systems move towards semiautonomy, it is important to study scenarios that fall in between the two extremes. Scenarios in which some vehicles (autonomous) can be routed by a central authority and the rest follow an equilibrium (that depends on the choices of the central authority). The study of these scenarios requires combining optimization techniques (for the central authority) with game theoretic ideas (for human-driven vehicles).

11.2 Closing Thoughts

Our work presented in this dissertation is only a step that attempts to bring ideas from robotics, control theory, and formal methods to address the problems in the design, control, learning, and verification of human-robot systems. As discussed in this chapter, there are still many existing challenges that need to be addressed for safe and interactive autonomy. We envision a strong collaboration between learning and control, human-robot interaction, and formal methods communities to take advantage of both data-driven and model-based techniques in an effort to seamlessly integrate safe and reliable human-robot systems in our society.

Bibliography

[1] H. Abbas et al. “Robustness-Guided Temporal Logic Testing and Verification for Stochastic Cyber-Physical Systems”. In: *Proc. of IEEE International Conference on CYBER Technology in Automation, Control, and Intelligent Systems*. 2014 (cit. on p. 108).

[2] P. Abbeel and A. Y. Ng. “Apprenticeship learning via inverse reinforcement learning”. In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 2004, p. 1 (cit. on p. 168).

[3] P. Abbeel and A. Y. Ng. “Exploration and apprenticeship learning in reinforcement learning”. In: *Proceedings of the 22nd international conference on Machine learning*. ACM. 2005, pp. 1–8 (cit. on pp. 3, 17, 29, 31).

[4] N. Ailon and M. Mohri. “Preference-based learning to rank”. In: *Machine Learning* 80.2-3 (2010), pp. 189–211 (cit. on p. 62).

[5] A. K. Akametalu et al. “Reachability-based safe learning with gaussian processes”. In: *2014 IEEE 53rd Annual Conference on Decision and Control*, pp. 1424–1431 (cit. on p. 124).

[6] B. Akgun et al. “Keyframe-based learning from demonstration”. In: *International Journal of Social Robotics* 4.4 (2012), pp. 343–355 (cit. on p. 62).

[7] R. Akrour, M. Schoenauer, and M. Sebag. “April: Active preference learning-based reinforcement learning”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2012, pp. 116–131 (cit. on p. 62).

[8] R. Alur, S. Moarref, and U. Topcu. “Counter-strategy guided refinement of GR(1) temporal logic specifications”. In: *Proceedings of the 13th Conference on Formal Methods in Computer-Aided Design (FMCAD’13)*. 2013, pp. 26–33 (cit. on pp. 106, 108).

[9] R. Alur, S. Moarref, and U. Topcu. “Counter-strategy guided refinement of GR(1) temporal logic specifications”. In: *Formal Methods in Computer-Aided Design*. 2013 (cit. on p. 141).

[10] D. Amodei et al. “Concrete Problems in AI Safety”. In: *arXiv preprint arXiv:1606.06565* (2016) (cit. on p. 167).

[11] G. Andrew and J. Gao. "Scalable training of L1-regularized log-linear models". In: *Proceedings of the 24th international conference on Machine learning*. ACM. 2007, pp. 33–40 (cit. on pp. 31, 66).

[12] C. Andrieu et al. "An introduction to MCMC for machine learning". In: *Machine learning* 50.1-2 (2003), pp. 5–43 (cit. on p. 125).

[13] A. Aswani et al. "Provably safe and robust learning-based model predictive control". In: *Automatica* 49.5 (2013), pp. 1216–1226 (cit. on p. 124).

[14] R. J. Aumann, M. Maschler, and R. E. Stearns. *Repeated games with incomplete information*. MIT press, 1995 (cit. on p. 28).

[15] T. Avgerinos et al. "Automatic exploit generation". In: *Commun. ACM* 57.2 (2014), pp. 74–84 (cit. on p. 169).

[16] M. Awais and D. Henrich. "Human-robot collaboration by intention recognition using probabilistic state machines". In: *Robotics in Alpe-Adria-Danube Region (RAAD), 2010 IEEE 19th International Workshop on*. IEEE. 2010, pp. 75–80 (cit. on p. 43).

[17] C. L. Baker, R. Saxe, and J. B. Tenenbaum. "Action understanding as inverse planning". In: *Cognition* 113.3 (2009), pp. 329–349 (cit. on p. 43).

[18] T. Bandyopadhyay et al. "Intention-aware motion planning". In: *Algorithmic Foundations of Robotics X*. Springer, 2013, pp. 475–491 (cit. on pp. 5, 43).

[19] F. Bastien et al. *Theano: new features and speed improvements*. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop. 2012 (cit. on p. 32).

[20] C. Basu et al. "Do you want your autonomous car to drive like you?" In: *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE. 2017 (cit. on p. 62).

[21] M. J. Beal. *Variational algorithms for approximate Bayesian inference*. University of London, 2003 (cit. on pp. 125, 139).

[22] I. Beer et al. "Efficient Detection of Vacuity in ACTL Formulas". In: *Formal Methods in System Design* 18.2 (2001), pp. 141–162 (cit. on p. 168).

[23] A. Bemporad and M. Morari. "Control of systems integrating logic, dynamics, and constraints". In: *Automatica* 35.3 (1999), pp. 407–427. DOI: 10.1016/S0005-1098(98)00178-2. URL: http://dx.doi.org/10.1016/S0005-1098(98)00178-2 (cit. on pp. 108, 142).

[24] A. Bemporad and M. Morari. "Robust model predictive control: A survey". In: *Robustness in identification and control*. Springer, 1999, pp. 207–226 (cit. on p. 141).

[25] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust optimization*. Princeton University Press, 2009 (cit. on p. 129).

[26] J. Bergstra et al. "Theano: a CPU and GPU Math Expression Compiler". In: *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation. Austin, TX, June 2010 (cit. on p. 32).

[27] D. S. Bernstein et al. “The complexity of decentralized control of Markov decision processes”. In: *Mathematics of operations research* 27.4 (2002), pp. 819–840 (cit. on p. 28).

[28] A. Biere et al. “Symbolic Model Checking without BDDs”. In: *Tools and Algorithms for Construction and Analysis of Systems, 5th International Conference, TACAS '99, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'99, Amsterdam, The Netherlands, March 22-28, 1999, Proceedings*. 1999, pp. 193–207. DOI: 10.1007/3-540-49059-0_14. URL: http://dx.doi.org/10.1007/3-540-49059-0_14 (cit. on p. 108).

[29] L. Blackmore, M. Ono, and B. C. Williams. “Chance-constrained optimal path planning with obstacles”. In: *2011 IEEE Transactions on Robotics* 27.6 (), pp. 1080–1094 (cit. on pp. 124, 129).

[30] R. Bloem et al. “RATSY - A New Requirements Analysis Tool with Synthesis.” In: *CAV'10*. 2010, pp. 425–429 (cit. on p. 104).

[31] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004 (cit. on pp. 129, 130).

[32] D. Braziunas. “Computational approaches to preference elicitation”. In: *Department of Computer Science, University of Toronto, Tech. Rep* (2006) (cit. on p. 62).

[33] K. Brinker, J. Fürnkranz, and E. Hüllermeier. *Label ranking by learning pairwise preferences*. Tech. rep. Technical Report TUD-KE-2007-01, Knowledge Engineering Group, TU Darmstadt, 2007 (cit. on p. 62).

[34] J. R. Büchi. “Symposium on decision problems: On a decision method in restricted second order arithmetic”. In: *Studies in Logic and the Foundations of Mathematics* 44 (1966), pp. 1–11 (cit. on p. 19).

[35] E. F. Camacho and C. B. Alba. *Model predictive control*. Springer Science & Business Media, 2013 (cit. on pp. 16, 29).

[36] A. Carvalho et al. “Stochastic Predictive Control of Autonomous Vehicles in Uncertain Environments”. In: *12th International Symposium on Advanced Vehicle Control*. 2014 (cit. on p. 124).

[37] K. Chatterjee, T. A. Henzinger, and B. Jobstmann. “Environment Assumptions for Synthesis”. In: *Proceedings of the 19th international conference on Concurrency Theory*. CONCUR '08. Toronto, Canada: Springer-Verlag, 2008, pp. 147–161. DOI: http://dx.doi.org/10.1007/978-3-540-85361-9_14. URL: http://dx.doi.org/10.1007/978-3-540-85361-9_14 (cit. on p. 106).

[38] K. Chatterjee et al. “Counterexample-guided Planning”. In: *UAI '05, Proceedings of the 21st Conference in Uncertainty in Artificial Intelligence, Edinburgh, Scotland, July 26-29, 2005*. 2005, pp. 104–111. URL: http://uai.sis.pitt.edu/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=1214&proceeding_id=21 (cit. on p. 108).

[39] J. W. Chinneck and E. W. Dravnieks. “Locating minimal infeasible constraint sets in linear programs”. In: *ORSA Journal on Computing* 3.2 (1991), pp. 157–168 (cit. on p. 149).

[40] M.-C. Costa, L. Léocart, and F. Roupin. “Minimal multicut and maximal integer multiflow: A survey”. In: *European Journal of Operational Research* 162.1 (2005), pp. 55–69 (cit. on p. 103).

[41] C. Daniel et al. “Active Reward Learning.” In: *Robotics: Science and Systems*. 2014 (cit. on p. 64).

[42] T. G. Dietterich and E. J. Horvitz. “Rise of concerns about AI: reflections and directions”. In: *Communications of the ACM* 58.10 (2015), pp. 38–40 (cit. on p. 167).

[43] X. C. Ding, M. Lazar, and C. Belta. “LTL receding horizon control for finite deterministic systems”. In: *Automatica* 50.2 (2014), pp. 399–408. DOI: `10.1016/j.automatica.2013.11.030`. URL: `http://dx.doi.org/10.1016/j.automatica.2013.11.030` (cit. on p. 109).

[44] M. Dissanayake et al. “A solution to the simultaneous localization and map building (SLAM) problem”. In: *Robotics and Automation, IEEE Transactions on* 17.3 (2001), pp. 229–241 (cit. on p. 8).

[45] A. Donzé and O. Maler. “Robust Satisfaction of Temporal Logic over Real-Valued Signals”. In: *Formal Modeling and Analysis of Timed Systems - 8th International Conference, FORMATS 2010, Klosterneuburg, Austria, September 8-10, 2010. Proceedings*. 2010, pp. 92–106. DOI: `10.1007/978-3-642-15297-9_9`. URL: `http://dx.doi.org/10.1007/978-3-642-15297-9_9` (cit. on p. 21).

[46] A. D. Dragan. “Robot Planning with Mathematical Models of Human State and Action”. In: *arXiv preprint arXiv:1705.04226* (2017) (cit. on p. 28).

[47] A. D. Dragan and S. S. Srinivasa. *Formalizing assistive teleoperation*. MIT Press, July, 2012 (cit. on p. 43).

[48] T. Dreossi, A. Donzé, and S. A. Seshia. “Compositional Falsification of Cyber-Physical Systems with Machine Learning Components”. In: *NASA Formal Methods Symposium*. Springer. 2017, pp. 357–372 (cit. on p. 169).

[49] P. M. Esfahani, T. Sutter, and J. Lygeros. “Performance bounds for the scenario approach and an extension to a class of non-convex programs”. In: *IEEE Transactions on Automatic Control* 60.1 (2015), pp. 46–58 (cit. on p. 113).

[50] F. A. A. (FAA). *The Interfaces between Flight Crews and Modern Flight Systems*. `http://www.faa.gov/avr/afs/interfac.pdf`. 1995 (cit. on p. 93).

[51] G. E. Fainekos and G. J. Pappas. "Robust Sampling for MITL Specifications". In: *Formal Modeling and Analysis of Timed Systems, 5th International Conference, FORMATS 2007, Salzburg, Austria, October 3-5, 2007, Proceedings*. 2007, pp. 147–162. DOI: `10.1007/978-3-540-75454-1_12`. URL: `http://dx.doi.org/10.1007/978-3-540-75454-1_12` (cit. on p. 23).

[52] G. E. Fainekos and G. J. Pappas. "Robustness of temporal logic specifications for continuous-time signals". In: *Theor. Comput. Sci.* 410.42 (2009), pp. 4262–4291. DOI: `10.1016/j.tcs.2009.06.021`. URL: `http://dx.doi.org/10.1016/j.tcs.2009.06.021` (cit. on p. 21).

[53] G. E. Fainekos et al. "Temporal logic motion planning for dynamic robots". In: *Automatica* 45.2 (2009), pp. 343–352. DOI: `10.1016/j.automatica.2008.08.008`. URL: `http://dx.doi.org/10.1016/j.automatica.2008.08.008` (cit. on p. 107).

[54] P. Falcone et al. "Integrated braking and steering model predictive control approach in autonomous vehicles". In: *Advances in Automotive Control*. Vol. 5. 1. 2007, pp. 273–278 (cit. on p. 8).

[55] P. Falcone et al. "MPC-based yaw and lateral stabilisation via active front steering and braking". en. In: *Vehicle System Dynamics* 46.sup1 (Sept. 2008), pp. 611–628 (cit. on p. 8).

[56] P. Falcone et al. "Predictive Active Steering Control for Autonomous Vehicle Systems". In: *IEEE Transactions on Control Systems Technology* 15.3 (May 2007), pp. 566–580 (cit. on p. 8).

[57] A. Fawzi, O. Fawzi, and P. Frossard. "Analysis of classifiers' robustness to adversarial perturbations". In: *arXiv preprint arXiv:1502.02590* (2015) (cit. on p. 169).

[58] A. Fern et al. "A Decision-Theoretic Model of Assistance." In: *IJCAI*. 2007, pp. 1879–1884 (cit. on pp. 5, 43, 46).

[59] T. Ferrère, O. Maler, and D. Nickovic. "Trace Diagnostics Using Temporal Implicants". In: *Proc. Int. Symp. Automated Technology for Verification and Analysis*. 2015 (cit. on p. 141).

[60] J. Fu and U. Topcu. "Computational methods for stochastic control with metric interval temporal logic specifications". In: *arXiv:1503.07193* (2015) (cit. on p. 124).

[61] J. Fu and U. Topcu. "Integrating active sensing into reactive synthesis with temporal logic constraints under partial observations". In: *arXiv:1410.0083* (2014) (cit. on p. 124).

[62] D. Fudenberg and J. Tirole. "Game theory, 1991". In: *Cambridge, Massachusetts* 393 (1991) (cit. on p. 29).

[63] J. Fürnkranz et al. "Preference-based reinforcement learning: a formal framework and a policy iteration algorithm". In: *Machine learning* 89.1-2 (2012), pp. 123–156 (cit. on p. 62).

[64] A. Gelman et al. *Bayesian data analysis*. Vol. 2. Taylor & Francis, 2014 (cit. on p. 125).

[65] S. Ghosh et al. “Diagnosis and repair for synthesis from signal temporal logic specifications”. In: *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*. ACM. 2016, pp. 31–40 (cit. on pp. 12, 124).

[66] G. Gigerenzer and R. Selten. *Bounded rationality: The adaptive toolbox*. MIT press, 2002 (cit. on pp. 166, 170).

[67] J. H. Gillula and C. J. Tomlin. “Guaranteed safe online learning via reachability: tracking a ground target using a quadrotor”. In: *2012 IEEE International Conference on Robotics and Automation*, pp. 2723–2730 (cit. on p. 124).

[68] E. A. Gol and M. Lazar. “Temporal logic model predictive control for discrete-time systems”. In: *Proceedings of the 16th international conference on Hybrid systems: computation and control, HSCC 2013, April 8-11, 2013, Philadelphia, PA, USA*. 2013, pp. 343–352. DOI: `10.1145/2461328.2461379`. URL: `http://doi.acm.org/10.1145/2461328.2461379` (cit. on p. 108).

[69] D. Golovin and A. Krause. “Adaptive submodularity: Theory and applications in active learning and stochastic optimization”. In: *Journal of Artificial Intelligence Research* 42 (2011), pp. 427–486 (cit. on p. 69).

[70] I. J. Goodfellow, J. Shlens, and C. Szegedy. “Explaining and harnessing adversarial examples”. In: *arXiv preprint arXiv:1412.6572* (2014) (cit. on p. 169).

[71] A. Gray et al. “Robust predictive control for semi-autonomous vehicles with an uncertain driver model”. In: *Intelligent Vehicles Symposium (IV), 2013 IEEE*. IEEE. 2013, pp. 208–213 (cit. on p. 8).

[72] M. Green and D. J. Limebeer. *Linear robust control*. Courier Corporation, 2012 (cit. on p. 124).

[73] *Gurobi Optimizer.* [Online]: `http://www.gurobi.com/` (cit. on pp. 133, 149, 161).

[74] H. Haario, E. Saksman, and J. Tamminen. “An adaptive Metropolis algorithm”. In: *Bernoulli* (2001), pp. 223–242 (cit. on p. 67).

[75] E. A. Hansen, D. S. Bernstein, and S. Zilberstein. “Dynamic programming for partially observable stochastic games”. In: *AAAI*. Vol. 4. 2004, pp. 709–715 (cit. on p. 28).

[76] T. Hedden and J. Zhang. “What do you think I think you think?: Strategic reasoning in matrix games”. In: *Cognition* 85.1 (2002), pp. 1–36 (cit. on pp. 3, 29).

[77] C. Hermes et al. “Long-term vehicle motion prediction”. In: *2009 IEEE Intelligent Vehicles Symposium*. 2009, pp. 652–657 (cit. on p. 8).

[78] R. Holladay et al. “Active Comparison Based Learning Incorporating User Uncertainty and Noise”. In: () (cit. on p. 62).

[79] H. Huang et al. “Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering”. In: *2009 IEEE International Conference on Robotics and Automation*. IEEE. 2009, pp. 3277–3282 (cit. on p. 24).

[80] A. Jain et al. “Learning preferences for manipulation tasks from online coactive feedback”. In: *The International Journal of Robotics Research* (2015) (cit. on p. 62).

[81] S. Javdani, J. A. Bagnell, and S. Srinivasa. “Shared Autonomy via Hindsight Optimization”. In: *arXiv preprint arXiv:1503.07619* (2015) (cit. on pp. 5, 43, 46).

[82] S. Javdani et al. “Efficient touch based localization through submodularity”. In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE. 2013, pp. 1828–1835 (cit. on p. 28).

[83] S. Jha and V. Raman. “Automated synthesis of safe autonomous vehicle control under perception uncertainty”. In: *NASA Formal Methods Symposium*. Springer. 2016, pp. 117–132 (cit. on p. 124).

[84] X. Jin et al. “Mining requirements from closed-loop control models”. In: *Proceedings of the 16th international conference on Hybrid systems: computation and control, HSCC 2013, April 8-11, 2013, Philadelphia, PA, USA*. 2013, pp. 43–52. DOI: `10.1145/2461328.2461337`. URL: `http://doi.acm.org/10.1145/2461328.2461337` (cit. on p. 108).

[85] M. Jordan. “Learning in graphical models (adaptive computation and machine learning)”. In: (1998) (cit. on pp. 123, 125).

[86] S. Karaman and E. Frazzoli. “Linear temporal logic vehicle routing with applications to multi-UAV mission planning”. In: *International Journal of Robust and Nonlinear Control* 21.12 (2011), pp. 1372–1395. DOI: `10.1002/rnc.1715`. URL: `http://dx.doi.org/10.1002/rnc.1715` (cit. on p. 107).

[87] S. Karaman and E. Frazzoli. “Linear temporal logic vehicle routing with applications to multi-UAV mission planning”. In: *International Journal of Robust and Nonlinear Control* 21.12 (2011), pp. 1372–1395 (cit. on p. 124).

[88] S. Karaman and E. Frazzoli. “Sampling-based motion planning with deterministic ¯-calculus specifications”. In: *Proceedings of the 48th IEEE Conference on Decision and Control, CDC 2009, combined withe the 28th Chinese Control Conference, December 16-18, 2009, Shanghai, China*. 2009, pp. 2222–2229. DOI: `10.1109/CDC.2009.5400278`. URL: `http://dx.doi.org/10.1109/CDC.2009.5400278` (cit. on pp. 107, 124).

[89] S. Karaman and E. Frazzoli. “Vehicle Routing Problem with Metric Temporal Logic Specifications”. In: *Proceedings of the 47th IEEE Conference on Decision and Control, CDC 2008, December 9-11, 2008, Cancún, México*. 2008, pp. 3953–3958. DOI: `10.1109/CDC.2008.4739366`. URL: `http://dx.doi.org/10.1109/CDC.2008.4739366` (cit. on pp. 107, 124).

[90] A. Karbasi, S. Ioannidis, et al. “Comparison-based learning with rank nets”. In: *arXiv preprint arXiv:1206.4674* (2012) (cit. on p. 62).

[91] S. Kataoka. "A stochastic programming model". In: *Econometrica: Journal of the Econometric Society* (1963), pp. 181–196 (cit. on p. 130).

[92] E. C. Kerrigan and J. M. Maciejowski. "Soft constraints and exact penalty functions in model predictive control". In: *Control Conference, Cambridge*. 2000 (cit. on p. 141).

[93] N. Kitchen and A. Kuehlmann. "Stimulus generation for constrained random simulation". In: *Proceedings of the 2007 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE Press. 2007, pp. 258–265 (cit. on p. 169).

[94] M. Kloetzer and C. Belta. "A Fully Automated Framework for Control of Linear Systems from Temporal Logic Specifications". In: *IEEE Trans. Automat. Contr.* 53.1 (2008), pp. 287–297. DOI: `10.1109/TAC.2007.914952`. URL: `http://dx.doi.org/10.1109/TAC.2007.914952` (cit. on p. 107).

[95] R. Könighofer, G. Hofferek, and R. Bloem. "Debugging formal specifications: a practical approach using model-based diagnosis and counterstrategies". In: *STTT* 15.5-6 (2013), pp. 563–583 (cit. on p. 141).

[96] M. V. Kothare, V. Balakrishnan, and M. Morari. "Robust constrained model predictive control using linear matrix inequalities". In: *Automatica* 32.10 (1996), pp. 1361–1379 (cit. on p. 124).

[97] H. Kress-Gazit, G. Fainekos, and G. Pappas. "Temporal-Logic-Based Reactive Mission and Motion Planning". In: *IEEE Transactions on Robotics* 25.6 (Dec. 2009), pp. 1370–1381. DOI: `10.1109/TRO.2009.2030225` (cit. on pp. 95, 105, 124).

[98] M. Kuderer, S. Gulati, and W. Burgard. "Learning driving styles for autonomous vehicles from demonstration". In: *Proceedings of the IEEE International Conference on Robotics & Automation (ICRA), Seattle, USA*. Vol. 134. 2015 (cit. on pp. 17, 31).

[99] D. Kulic and E. A. Croft. "Affective state estimation for human–robot interaction". In: *Robotics, IEEE Transactions on* 23.5 (2007), pp. 991–1000 (cit. on p. 43).

[100] Y. Kwon and G. Agha. "LTLC: Linear Temporal Logic for Control". In: *Hybrid Systems: Computation and Control, 11th International Workshop, HSCC 2008, St. Louis, MO, USA, April 22-24, 2008. Proceedings*. 2008, pp. 316–329. DOI: `10.1007/978-3-540-78929-1_23`. URL: `http://dx.doi.org/10.1007/978-3-540-78929-1_23` (cit. on p. 107).

[101] L. T. Kohn and J. M. Corrigan and M. S. Donaldson, editors. *To Err is Human: Building a Safer Health System*. Tech. rep. National Academy Press. Washington, DC: A report of the Committee on Quality of Health Care in America, Institute of Medicine, 2000 (cit. on p. 93).

[102] C.-P. Lam, A. Y. Yang, and S. S. Sastry. "An efficient algorithm for discrete-time hidden mode stochastic hybrid systems". In: *Control Conference (ECC), 2015 European*. IEEE. 2015, pp. 1212–1218 (cit. on pp. 43, 46).

[103] K. Leahy et al. "Distributed information gathering policies under temporal logic constraints". In: *2015 IEEE 54th Annual Conference on Decision and Control (CDC)*, pp. 6803–6808 (cit. on p. 124).

[104] Y. T. Lee, A. Sidford, and S. C.-w. Wong. "A faster cutting plane method and its implications for combinatorial and convex optimization". In: *arXiv preprint arXiv:1508.04874* (2015) (cit. on p. 132).

[105] D. Lenz, T. Kessler, and A. Knoll. "Stochastic model predictive controller with chance constraints for comfortable and safe driving behavior of autonomous vehicles". In: *2015 IEEE Intelligent Vehicles Symposium*, pp. 292–297 (cit. on pp. 124, 129).

[106] J. Leonard et al. "A perception-driven autonomous urban vehicle". In: *Journal of Field Robotics* 25.10 (2008), pp. 727–774 (cit. on p. 8).

[107] S. Levine and V. Koltun. "Continuous inverse optimal control with locally optimal examples". In: *arXiv preprint arXiv:1206.4617* (2012) (cit. on pp. 3, 17, 18, 29, 31, 48, 64).

[108] J. Levinson et al. "Towards fully autonomous driving: Systems and algorithms". In: *2011 IEEE Intelligent Vehicles Symposium (IV)*, pp. 163–168 (cit. on p. 8).

[109] W. Li, L. Dworkin, and S. A. Seshia. "Mining Assumptions for Synthesis". In: *Proceedings of the Ninth ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*. July 2011, pp. 43–50 (cit. on pp. 106, 141).

[110] W. Li et al. "Synthesis for Human-in-the-Loop Control Systems". In: *Proceedings of the 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Apr. 2014, pp. 470–484 (cit. on pp. 6, 12, 94, 141).

[111] M. Liebner et al. "Driver intent inference at urban intersections using the intelligent driver model". In: *Intelligent Vehicles Symposium (IV), 2012 IEEE*. IEEE. 2012, pp. 1162–1167 (cit. on p. 43).

[112] S. C. Livingston, R. M. Murray, and J. W. Burdick. "Backtracking temporal logic synthesis for uncertain environments". In: *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2012, pp. 5163–5170 (cit. on p. 105).

[113] S. C. Livingston et al. "Patching task-level robot controllers based on a local μ-calculus formula". In: (2012) (cit. on p. 105).

[114] J. Löfberg. "YALMIP: A Toolbox for Modeling and Optimization in MATLAB". In: *Proceedings of the CACSD Conference*. Taipei, Taiwan, 2004. URL: `http://users.isy.liu.se/johanl/yalmip` (cit. on p. 160).

[115] J. Löfberg. "YALMIP: A toolbox for modeling and optimization in MATLAB". In: *2004 IEEE International Symposium on Computer Aided Control Systems Design*. IEEE. 2004, pp. 284–289 (cit. on p. 133).

[116] M. Lopes, F. Melo, and L. Montesano. "Active learning for reward estimation in inverse reinforcement learning". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2009, pp. 31–46 (cit. on p. 64).

[117] L. Lovász and S. Vempala. "Fast algorithms for logconcave functions: Sampling, rounding, integration and optimization". In: *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. IEEE. 2006, pp. 57–68 (cit. on p. 67).

[118] B. Luders, M. Kothari, and J. P. How. "Chance constrained RRT for probabilistic robustness to environmental uncertainty". In: *AIAA guidance, navigation, and control conference (GNC), Toronto, Canada*. 2010 (cit. on p. 8).

[119] O. Maler and D. Nickovic. "Monitoring Temporal Properties of Continuous Signals". In: *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems, Joint International Conferences on Formal Modelling and Analysis of Timed Systems, FORMATS 2004 and Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT 2004, Grenoble, France, September 22-24, 2004, Proceedings*. 2004, pp. 152–166. DOI: `10.1007/978-3-540-30206-3_12`. URL: `http://dx.doi.org/10.1007/978-3-540-30206-3_12` (cit. on p. 108).

[120] T. P. Minka. "A family of algorithms for approximate Bayesian inference". PhD thesis. Massachusetts Institute of Technology, 2001 (cit. on pp. 125, 139).

[121] I. Mitchell, A. Bayen, and C. J. Tomlin. "A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games". In: *2005 IEEE Transactions on Automatic Control* 50.7 (), pp. 947–957 (cit. on p. 124).

[122] I. Mitchell and C. J. Tomlin. "Level set methods for computation in hybrid systems". In: *Hybrid Systems: Computation and Control*. Springer, 2000, pp. 310–323 (cit. on p. 124).

[123] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. "DeepFool: a simple and accurate method to fool deep neural networks". In: *arXiv preprint arXiv:1511.04599* (2015) (cit. on p. 169).

[124] M. Morari et al. *Model predictive control*. Prentice Hall Englewood Cliffs, NJ, 1993 (cit. on p. 15).

[125] R. Motwani and P. Raghavan. *Randomized algorithms*. Chapman & Hall/CRC, 2010 (cit. on p. 115).

[126] R. M. Murray et al. "Online Control Customization via Optimization-Based Control". In: *Software-Enabled Control*. John Wiley & Sons, Inc., 2005, pp. 149–174. DOI: `10.1002/047172288X.ch9`. URL: `http://dx.doi.org/10.1002/047172288X.ch9` (cit. on p. 108).

[127] National Highway Traffic Safety Administration. *Preliminary Statement of Policy Concerning Automated Vehicles*. May 2013 (cit. on p. 94).

[128] A. Y. Ng and S. J. Russell. "Algorithms for Inverse Reinforcement Learning". In: *Proceedings of the Seventeenth International Conference on Machine Learning (ICML)*. 2000, pp. 663–670 (cit. on p. 168).

[129] A. Y. Ng, S. J. Russell, et al. "Algorithms for inverse reinforcement learning." In: *Proceedings of the 17th international conference on Machine learning*. 2000, pp. 663–670 (cit. on pp. 3, 17, 29, 31).

[130] A. Nguyen, J. Yosinski, and J. Clune. "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2015, pp. 427–436 (cit. on p. 169).

[131] T.-H. D. Nguyen et al. "Capir: Collaborative action planning with intention recognition". In: *arXiv preprint arXiv:1206.5928* (2012) (cit. on p. 43).

[132] S. Nikolaidis et al. "Formalizing Human-Robot Mutual Adaptation via a Bounded Memory Based Model". In: *Human-Robot Interaction*. Mar. 2016 (cit. on p. 46).

[133] P. Nuzzo et al. "A Contract-Based Methodology for Aircraft Electric Power System Design". In: *IEEE Access* 2 (2014), pp. 1–25. DOI: `10.1109/ACCESS.2013.2295764`. URL: `http://dx.doi.org/10.1109/ACCESS.2013.2295764` (cit. on p. 107).

[134] P. Nuzzo et al. "CalCS: SMT Solving for Non-linear Convex Constraints". In: *IEEE Int. Conf. Formal Methods in Computer-Aided Design*. 2010 (cit. on p. 141).

[135] P. Nuzzo et al. "A Contract-Based Methodology for Aircraft Electric Power System Design". In: *IEEE Access* 2 (2014), pp. 1–25. DOI: `10.1109/ACCESS.2013.2295764` (cit. on pp. 140, 142).

[136] P. Nuzzo et al. "A Platform-Based Design Methodology with Contracts and Related Tools for the Design of Cyber-Physical Systems". In: *Proc. IEEE* 103.11 (Nov. 2015) (cit. on pp. 140, 142).

[137] S. Patil et al. "Scaling up gaussian belief space planning through covariance-free trajectory optimization and automatic differentiation". In: *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 515–533 (cit. on p. 28).

[138] N. Piterman and A. Pnueli. "Synthesis of reactive(1) designs". In: *In Proc. Verification, Model Checking, and Abstract Interpretation (VMCAI'06*. Springer, 2006, pp. 364–380 (cit. on p. 105).

[139] N. Piterman, A. Pnueli, and Y. Sa′ar. "Synthesis of reactive (1) designs". In: *Verification, Model Checking, and Abstract Interpretation, 2006*. Springer. 2006, pp. 364–380 (cit. on pp. 6, 124).

[140] E. Plaku and S. Karaman. "Motion planning with temporal-logic specifications: Progress and challenges". In: *AI Communications* Preprint (), pp. 1–12 (cit. on p. 124).

[141] A. Pnueli and R. Rosner. "On the synthesis of a reactive module". In: *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. POPL '89. Austin, Texas, United States: ACM, 1989, pp. 179–190. DOI: `http://doi.acm.org/10.1145/75277.75293`. URL: `http://doi.acm.org/10.1145/75277.75293` (cit. on pp. 20, 105).

[142] A. Pnueli. "The Temporal Logic of Programs". In: *18th Annual Symposium on Foundations of Computer Science (FOCS)*. 1977, pp. 46–57 (cit. on p. 19).

[143] S. Prentice and N. Roy. "The belief roadmap: Efficient planning in belief space by factoring the covariance". In: *The International Journal of Robotics Research* (2009) (cit. on p. 28).

[144] A. Puggelli et al. "Polynomial-Time Verification of PCTL Properties of MDPs with Convex Uncertainties". In: *Proceedings of the 25th International Conference on Computer-Aided Verification (CAV)*. July 2013 (cit. on p. 124).

[145] V. Raman and H. Kress-Gazit. "Explaining Impossible High-Level Robot Behaviors". In: *IEEE Trans. Robotics* 29 (2013) (cit. on p. 141).

[146] V. Raman et al. "Model predictive control with signal temporal logic specifications". In: *2014 IEEE 53rd Annual Conference on Decision and Control*, pp. 81–87 (cit. on pp. 7, 11, 107).

[147] V. Raman et al. "Model predictive control with signal temporal logic specifications". In: *53rd IEEE Conference on Decision and Control, CDC 2014, Los Angeles, CA, USA, December 15-17, 2014*. 2014, pp. 81–87. DOI: `10.1109/CDC.2014.7039363`. URL: `http://dx.doi.org/10.1109/CDC.2014.7039363` (cit. on pp. 107–112, 117, 141–143).

[148] V. Raman et al. "Reactive Synthesis from Signal Temporal Logic Specifications". In: *18th International Conference on Hybrid Systems: Computation and Control*. 2015 (cit. on pp. 124, 141–143, 158).

[149] V. Raman et al. "Reactive synthesis from signal temporal logic specifications". In: *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*. ACM. 2015, pp. 239–248 (cit. on pp. 7, 8, 11).

[150] M. Richardson and P. Domingos. "Markov logic networks". In: *Machine learning* 62.1-2 (2006), pp. 107–136 (cit. on p. 124).

[151] C. Rothkopf and C. Dimitrakakis. "Preference elicitation and inverse reinforcement learning". In: *Machine Learning and Knowledge Discovery in Databases* (2011), pp. 34–48 (cit. on p. 63).

[152] J. Rushby. "Using Model Checking to Detect Automation Surprises". In: *Reliability Engineering and System Safety* 75.2 (2002), pp. 167–177 (cit. on p. 167).

[153] S. Russell et al. "Letter to the Editor: Research Priorities for Robust and Beneficial Artificial Intelligence: An Open Letter". In: *AI Magazine* 36.4 (2015) (cit. on p. 167).

[154] D. Sadigh and A. Kapoor. "Safe control under uncertainty with probabilistic signal temporal logic". In: *Proceedings of Robotics: Science and Systems, AnnArbor, Michigan* (2016) (cit. on pp. 7, 8, 11).

[155] D. Sadigh, S. S. Sastry, and S. A. Seshia. "Falsification for Human-Robot Systems". In: *Advances in Neural Information Processing Systems (NIPS) (under review)*. 2017 (cit. on pp. 6, 11).

[156] D. Sadigh et al. "Active Preference-Based Learning of Reward Functions". In: *Proceedings of the Robotics: Science and Systems Conference (RSS)*. 2017 (cit. on pp. 6, 10, 11).

[157] D. Sadigh et al. "Data-driven probabilistic modeling and verification of human driver behavior". In: *Formal Verification and Modeling in Human-Machine Systems* (2014) (cit. on pp. 124, 167).

[158] D. Sadigh et al. "Information gathering actions over human internal state". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2016, pp. 66–73 (cit. on pp. 5, 10, 11, 170).

[159] D. Sadigh et al. "Planning for autonomous cars that leverages effects on human actions". In: *Proceedings of the Robotics: Science and Systems Conference (RSS)*. 2016 (cit. on pp. 5, 9–11, 44, 64, 170).

[160] D. Sadigh et al. "Planning for Cars that Coordinate with People: Leveraging Effects on Human Actions for Planning and Active Information Gathering over Human Internal State". In: *Autonomous Robots (AURO) (under review)*. 2016 (cit. on pp. 5, 9–11, 28, 170).

[161] S. Safra. "On the Complexity of ω-automata". In: *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*. SFCS '88. 1988, pp. 319–327 (cit. on p. 20).

[162] V. Schuppan. "Towards a Notion of Unsatisfiable Cores for LTL". In: *Fundamentals of Software Engineering*. 2009 (cit. on p. 141).

[163] P. O. Scokaert and J. B. Rawlings. "Feasibility issues in linear model predictive control". In: *AIChE Journal* 45.8 (1999), pp. 1649–1659 (cit. on p. 141).

[164] S. A. Seshia, D. Sadigh, and S. S. Sastry. "Towards verified artificial intelligence". In: *arXiv preprint arXiv:1606.08514* (2016) (cit. on p. 167).

[165] M. Shimosaka, T. Kaneko, and K. Nishi. "Modeling risk anticipation and defensive driving on residential roads with inverse reinforcement learning". In: *2014 IEEE 17th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2014, pp. 1694–1700 (cit. on pp. 17, 31).

[166] A. Solar-Lezama et al. "Combinatorial sketching for finite programs". In: *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2006, San Jose, CA, USA, October 21-25, 2006*. 2006, pp. 404–415. DOI: `10.1145/1168857.1168907`. URL: `http://doi.acm.org/10.1145/1168857.1168907` (cit. on pp. 7, 107).

[167] H. Sugiyama, T. Meguro, and Y. Minami. "Preference-learning based Inverse Reinforcement Learning for Dialog Control." In: *INTERSPEECH*. 2012, pp. 222–225 (cit. on p. 62).

[168] M. Svoreňová et al. "Temporal logic control for stochastic linear systems using abstraction refinement of probabilistic games". In: *18th International Conference on Hybrid Systems: Computation and Control*. ACM. 2015, pp. 259–268 (cit. on p. 124).

[169] T. J. Triggs, W. G. Harris, et al. "Reaction time of drivers to road stimuli". In: (1982) (cit. on p. 106).

[170] C. Urmson et al. "Autonomous driving in urban environments: Boss and the urban challenge". In: *Journal of Field Robotics* 25.8 (2008), pp. 425–466 (cit. on p. 8).

[171] C. Van de Panne and W. Popp. "Minimum-cost cattle feed under probabilistic protein constraints". In: *Management Science* 9.3 (1963), pp. 405–430 (cit. on p. 130).

[172] T. Van Kasteren et al. "Accurate activity recognition in a home setting". In: *Proceedings of the 10th international conference on Ubiquitous computing*. ACM. 2008, pp. 1–9 (cit. on p. 43).

[173] H. P. Vanchinathan et al. "Explore-exploit in top-n recommender systems via gaussian processes". In: *Proceedings of the 8th ACM Conference on Recommender systems*. ACM. 2014, pp. 225–232 (cit. on p. 47).

[174] R. Vasudevan et al. "Safe semi-autonomous control with enhanced driver modeling". In: *American Control Conference (ACC), 2012*. IEEE. 2012, pp. 2896–2903 (cit. on p. 8).

[175] M. P. Vitus and C. J. Tomlin. "A probabilistic approach to planning and control in autonomous urban driving". In: *2013 IEEE 52nd Annual Conference on Decision and Control (CDC)*, pp. 2459–2464 (cit. on pp. 8, 129).

[176] M. P. Vitus. "Stochastic Control Via Chance Constrained Optimization and its Application to Unmanned Aerial Vehicles". PhD thesis. Stanford University, 2012 (cit. on p. 124).

[177] A. Wächter and L. T. Biegler. "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". In: *Mathematical programming* 106.1 (2006), pp. 25–57 (cit. on p. 89).

[178] Y. Wang, L. Xie, and C. E. de Souza. "Robust control of a class of uncertain nonlinear systems". In: *Systems & Control Letters* 19.2 (1992), pp. 139–149 (cit. on p. 124).

[179] C. K. Williams and C. E. Rasmussen. "Gaussian processes for machine learning". In: *the MIT Press* 2.3 (2006), p. 4 (cit. on p. 126).

[180] A. Wilson, A. Fern, and P. Tadepalli. "A bayesian approach for policy learning from trajectory preference queries". In: *Advances in neural information processing systems*. 2012, pp. 1133–1141 (cit. on p. 62).

[181] C. Wirth, J. Fürnkranz, and G. Neumann. "Model-Free Preference-Based Reinforcement Learning". In: *Thirtieth AAAI Conference on Artificial Intelligence*. 2016 (cit. on p. 63).

[182] E. M. Wolff, U. Topcu, and R. M. Murray. "Optimization-based trajectory generation with linear temporal logic specifications". In: *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*. 2014, pp. 5319–5325. DOI: `10.1109/ICRA.2014.6907641`. URL: `http://dx.doi.org/10.1109/ICRA.2014.6907641` (cit. on p. 107).

[183] T. Wongpiromsarn, U. Topcu, and R. Murray. "Receding Horizon Temporal Logic Planning". In: *IEEE Transactions on Automatic Control* 57.11 (2012), pp. 2817–2830 (cit. on pp. 96, 105).

[184] T. Wongpiromsarn, U. Topcu, and R. Murray. "Receding Horizon Temporal Logic Planning". In: *IEEE Transactions on Automatic Control* 57.11 (2012), pp. 2817–2830. DOI: `10.1109/TAC.2012.2195811` (cit. on p. 124).

[185] T. Wongpiromsarn, U. Topcu, and R. Murray. "Receding horizon temporal logic planning for dynamical systems". In: *Decision and Control, 2009 held jointly with the 2009 28th Chinese Control Conference. CDC/CCC 2009. Proceedings of the 48th IEEE Conference on*. Dec. 2009, pp. 5997–6004 (cit. on p. 95).

[186] T. Wongpiromsarn, U. Topcu, and R. M. Murray. "Receding Horizon Temporal Logic Planning". In: *IEEE Trans. Automat. Contr.* 57.11 (2012), pp. 2817–2830. DOI: `10.1109/TAC.2012.2195811`. URL: `http://dx.doi.org/10.1109/TAC.2012.2195811` (cit. on pp. 107–109).

[187] B. D. Ziebart. "Modeling purposeful adaptive behavior with the principle of maximum causal entropy". In: (2010) (cit. on p. 17).

[188] B. D. Ziebart et al. "Maximum Entropy Inverse Reinforcement Learning." In: *AAAI*. 2008, pp. 1433–1438 (cit. on pp. 3, 17, 29, 31, 46, 64, 168).

[189] B. D. Ziebart et al. "Planning-based prediction for pedestrians". In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2009, pp. 3931–3936 (cit. on p. 43).