I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Marco Pavone, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

J Gerdes

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Mac Schwager

Approved for the Stanford University Committee on Graduate Studies.

Stacey F. Bent, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.
Abstract

Advances in the fields of artificial intelligence and machine learning have unlocked a new generation of robotic systems—“learning-enabled” robots that are designed to operate in unstructured, uncertain, and unforgiving environments, especially settings where robots are required to interact in close proximity with humans. However, as learning-enabled methods, especially “deep” learning, continue to become more pervasive throughout the autonomy stack, it also becomes increasingly difficult to ascertain the performance and safety of these robotic systems and explain their behavior, necessary prerequisites for their deployment in safety-critical settings. This dissertation develops methods drawing upon techniques from the field of formal methods, namely Hamilton-Jacobi (HJ) reachability and Signal Temporal Logic (STL), to complement a learning-enabled robot autonomy stack, thereby leading to safer and more robust robot behavior. The first part of this dissertation investigates the problem of providing safety assurance for human-robot interactions, safety-critical settings wherein robots must reason about the uncertainty in human behavior to achieve seamless interactions with humans. Specifically, we develop a two-step approach where we first develop a learning-based human behavior prediction model tailored towards proactive robot planning and decision-making, which we then couple with a reachability-based safety controller that minimally intervenes whenever the robot is near safety violation. The approach is validated through human-in-the-loop simulation as well as on an experimental vehicle platform, demonstrating clear connections between theory and practice. The second part of this dissertation examines the use of STL as a formal language to incorporate logical reasoning into robot learning. In particular, we develop a technique, named stlckg, that casts STL into the same computational language as deep neural networks. Consequently, by using stlckg to express designers’ domain expertise into a form compatible with neural networks, we can embed domain knowledge into learned components within the autonomy stack to provide additional levels of robustness and interpretability.
To my parents who made it possible for me to have the opportunity to learn and find my passion.
Acknowledgments

First and foremost, I would like to thank my advisor, Prof. Marco Pavone. I am eternally grateful for your patience, guidance, support, and faith in me, especially when I was at times doubting myself. You have provided me a home away from home, a place for me to grow, space for me to learn from mistakes, and time to find my direction. I am in awe of your dedication and pursuit of excellence as a researcher, teacher, and mentor, and am extremely blessed to have been able to learn from you. I also would like to thank Prof. Mac Schwager and Prof. Chris Gerdes for not only serving on my reading committee, but for their support, feedback, and discussions over the past four years. Moreover, their energy and passion for teaching continually reignite my passion and love for engineering and science. All of my interactions with you, whether it be in class, meetings, or just in passing, constantly fuels my excitement for the field.

My journey into grad school would not have existed without my undergraduate mentors and teachers whose passion sparked my desire to pursue more knowledge beyond the written curriculum. I want to thank Prof. Holger Dullin and Prof. Nalini Joshi for introducing me to the field of research. Prof. Ian Manchester, thank you for introducing the field of control theory in a way that piqued my interest. Also, to Prof. James Parkinson for teaching fundamental calculus concepts with such clarity, enthusiasm, and passion so that I looked forward to going into class each day and making all 13 weeks of Mon-Thurs 8am classes worth it.

I am so fortunate to have been able to work with such an amazing, smart, and hard-working group of people from who I constantly learn so much. None of this would have been possible without you all. In particular, I want to thank my co-authors, Edward Schmerling, Nikos Aréchiga, Boris Ivanovic, John Talbot, Mo Chen, Simon Schaefer, Xinrui Wang, Wolf Vollprecht, and Michelle Zhang, for their contributions, discussions, ideas, motivation, and ultimately making the work that much more enjoyable. Also to Benoit Landry, Amine Elhafṣî, Mingyu Wang, and Jon DeCastro for making the time to be my sounding board and providing a safe space where there is no such thing as a silly question. Also to Ben Hockman, Federico Rossi, and Nicoletta Bof for making my early years in the ASL welcoming, warm, and enjoyable. I stumbled into the ASL feeling like an impostor, but you always reminded me that I belonged there.

Half of my Ph.D. would not have been possible without the infectious enthusiasm and camaraderie
of the X1 team. It has been a huge honor and privilege to be able to learn from you, not just about cars, but especially what it means to work together as a team. I would also like to extend my gratitude to the rest of the DDL and CARS who have added more color and glitter to my time at Stanford. To the rest of the ASL, past, and present, the basement dwellers of Durand, the Aero/Astro students, staff, and faculty, thank you for your unwavering support.

I am also grateful to have friends that kept me sane and reminded me to enjoy the world beyond the concrete walls of the Durand basement. Tess, from the first day we met, there has never been a boring second with you. Thank you for keeping things interesting, and having the wildest stories to share, and being my source of entertainment. Amine, I know you were forced to work with me when we were TAs for AA274, but I am glad you stuck around after. Thank you for playing infinity hours of computer games with me. To my other Stanford friends, Hannah, Kyle, Jessie, Anthony, and Jean who remind me of life beyond the basement, and remember to drag me out into the sunshine to have lunch.

Outside of Stanford, Michael, Nathan, Alexey, Rachel, and Hayoung. Thank you for entertaining me and for providing many excuses to eat ridiculous amounts of food in one sitting. Sylvia Herbert— I am so thankful for all the support, advice, and encouragement you have given me and I can’t wait to hang out again at another international conference. Andrea Bajcsy, dood. I can’t begin to explain how much I enjoyed working with you during our internship, and I can’t believe we only really got to know each other and work together in the final stretches of our Ph.D. We will forever be as thick as thieves. Living away from family is difficult, but Allison and Jared, living with you for the past few years have made it so much easier. I am so blessed to have found such generous, caring, and warm people to live with, and it is a relief to know that I will be coming home to a stress-free place (with delicious baked goods) after a long day in the lab. Of course, special mention goes to the furry pals, Leo and Riley, for keeping me honest when working from home and providing endless supplies of cuddles.

Friends from Sydney! Ragib, Suda, Matt, Saweng, Felix, Jack, Nick, and Nikita, you have provided my undergraduate learning experience with laughter, color, and joy. Without you, the thought of pursuing further education and applying to grad school would have never entered my mind. Letitia, you have been my best friend since I was six, and we have watched each other grow. You help me keep true to myself, and help me embrace my geekier side. Mel, you always keep me grounded and taught me to embrace who I am, stick to what I believe in, and not take any nonsense from anyone.

Saving the best for last—to my loving parents, Dantes and Iris. None of this would be possible without your unwavering love and support. It has not been easy living so far away from home, but I can always feel your warmth and love more than 7000 miles away. Words cannot begin to express how much I love you and how proud I am to be your daughter. To Boson, you constantly bring joy to my world, and your presence as my computer wallpaper has kept me motivated all these years.
Contents

Abstract v
Acknowledgments vii

1 Introduction 1
  1.1 Dissertation Contributions .............................................. 2
  1.2 Structure .............................................................................. 4

I Safety Assurance for Human-Robot Interactions 5

2 Multimodal Probabilistic Model-based Planning for HRI 8
  2.1 Introduction .............................................................................. 8
  2.2 Related Work ........................................................................... 10
  2.3 Problem Formulation ................................................................. 11
    2.3.1 Interaction Dynamics .......................................................... 12
    2.3.2 Robot Goal ....................................................................... 12
    2.3.3 Traffic Weaving Scenario ..................................................... 13
  2.4 Multimodal Human Behavior Prediction Model ....................... 14
    2.4.1 Conditional Variational Autoencoder ................................... 14
    2.4.2 Interaction-aware Human Behavior Prediction ................. 16
  2.5 Robot Policy Construction ........................................................ 18
  2.6 Traffic-weaving Case-study ....................................................... 18
    2.6.1 Data Collection .................................................................. 19
    2.6.2 Generative Model for Human Driver Actions ................. 20
    2.6.3 Robot Policy Construction ............................................... 21
    2.6.4 Human-in-the-loop trials ............................................... 22
  2.7 Summary .................................................................................. 23
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Reachability-based Safety Assurance</td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td>Introduction</td>
<td>25</td>
</tr>
<tr>
<td>3.2</td>
<td>Related Work</td>
<td>26</td>
</tr>
<tr>
<td>3.3</td>
<td>Contributions</td>
<td>28</td>
</tr>
<tr>
<td>3.4</td>
<td>Hamilton-Jacobi Backward Reachability Analysis</td>
<td></td>
</tr>
<tr>
<td>3.4.1</td>
<td>Forward Reachability Analysis</td>
<td>29</td>
</tr>
<tr>
<td>3.4.2</td>
<td>Backward Reachability Analysis</td>
<td>29</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Hamilton-Jacobi Reachability Analysis</td>
<td>30</td>
</tr>
<tr>
<td>3.4.4</td>
<td>Optimal HJI Collision Avoidance Control</td>
<td>34</td>
</tr>
<tr>
<td>3.4.5</td>
<td>Safety-Preserving HJI Control</td>
<td>34</td>
</tr>
<tr>
<td>3.5</td>
<td>Minimally-Interventional Safety Controller</td>
<td>35</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Interaction Planner</td>
<td>36</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Tracking Controller</td>
<td>36</td>
</tr>
<tr>
<td>3.5.3</td>
<td>Safety-Preserving HJI Control Constraint</td>
<td>37</td>
</tr>
<tr>
<td>3.6</td>
<td>Dynamics: Pairwise Vehicle-Vehicle Interaction</td>
<td></td>
</tr>
<tr>
<td>3.6.1</td>
<td>Robot Vehicle Dynamics</td>
<td>38</td>
</tr>
<tr>
<td>3.6.2</td>
<td>Human Vehicle Dynamics</td>
<td>39</td>
</tr>
<tr>
<td>3.6.3</td>
<td>Relative Human-Robot Dynamics</td>
<td>40</td>
</tr>
<tr>
<td>3.6.4</td>
<td>MPC Tracking Dynamics</td>
<td>41</td>
</tr>
<tr>
<td>3.7</td>
<td>The MPC+HJI Trajectory Tracking Controller</td>
<td></td>
</tr>
<tr>
<td>3.7.1</td>
<td>Optimization Problem</td>
<td>42</td>
</tr>
<tr>
<td>3.7.2</td>
<td>Adding Static Obstacles</td>
<td>44</td>
</tr>
<tr>
<td>3.7.3</td>
<td>Reachability Computation</td>
<td>45</td>
</tr>
<tr>
<td>3.8</td>
<td>Traffic-Weaving Experiments</td>
<td></td>
</tr>
<tr>
<td>3.8.1</td>
<td>Experimental Set-up</td>
<td>46</td>
</tr>
<tr>
<td>3.8.2</td>
<td>Experimental Results</td>
<td>46</td>
</tr>
<tr>
<td>3.8.3</td>
<td>Adding Static Obstacles</td>
<td>50</td>
</tr>
<tr>
<td>3.8.4</td>
<td>Baseline Comparison</td>
<td>52</td>
</tr>
<tr>
<td>3.8.5</td>
<td>1/10-Scale Human-Driven Vehicle</td>
<td>53</td>
</tr>
<tr>
<td>3.9</td>
<td>Discussion</td>
<td>53</td>
</tr>
<tr>
<td>3.10</td>
<td>Summary</td>
<td>55</td>
</tr>
<tr>
<td>4</td>
<td>Extensions to Safe Multi-agent Interactions</td>
<td>59</td>
</tr>
<tr>
<td>4.1</td>
<td>Introduction</td>
<td>59</td>
</tr>
<tr>
<td>4.2</td>
<td>Related Work</td>
<td>61</td>
</tr>
<tr>
<td>4.3</td>
<td>Multi-Agent HJ Reachability Controller</td>
<td>62</td>
</tr>
<tr>
<td>4.4</td>
<td>Reachability-Based Reward Augmentation</td>
<td>64</td>
</tr>
<tr>
<td>4.5</td>
<td>Case Study: Highway Driving</td>
<td>66</td>
</tr>
</tbody>
</table>
II Logical Reasoning in Robot Learning

5 Backpropagation Through STL Specifications

5.1 Introduction

5.2 Signal Temporal Logic (STL)

5.2.1 Signals

5.2.2 Syntax and Semantics

5.2.3 Graphical Structure of STL Formulas

5.3 \texttt{stlcg}

5.3.1 Computation Graphs

5.3.2 Computation Graphs of STL Operators

5.3.3 Calculating Robustness with Computation Graphs

5.3.4 Smooth Approximations to \texttt{stlcg}

5.4 Applications of \texttt{stlcg}

5.4.1 Motion Planning With STL Constraints

5.4.2 Parametric STL for Behavioral Clustering

5.4.3 Robustness-Aware Neural Networks

5.5 Summary

6 Control Synthesis With STL Specifications

6.1 Introduction

6.2 Related Work

6.2.1 Temporal Logic Synthesis

6.2.2 Temporal Logic in Learning-Based Approaches

6.3 Problem Formulation

6.4 STL Control Synthesis

6.4.1 Overview

6.4.2 Trajectory-Feedback Controller Architecture

6.4.3 Learning the Control Parameters

6.4.4 Deploying the Controller Online

6.5 Experiments

6.5.1 Case-Study Set-Up
List of Tables

2.1 Fraction of human-human trials where the car starting in left lane passes in front. . 20

4.1 Safety and control statistics for different planner and safety controller configurations. The values in the $\text{TTC} \geq 3$, $\text{BTN} \leq 1$, and $\text{STN} \leq 1$ column represent the fraction of samples that satisfy the inequality. Our proposed method is in bold (Row 3). . . . . 72

4.2 Performance statistics for different planner and safety controller configurations. Our proposed method is in bold (Row 3). . . . . . . . . . . . . . . . . . . . . . . 72

5.1 Hidden and output states of $\hat{q}_1(s > 0)$ for a signal $s = 1, 1, 2, 3, 1$ and different intervals $I$. The signal is fed into the computation graph backward. . . . . . . . . . . 87

A.1 X1 parameters relevant to defining the equations of motion in (3.8). . . . . . . . . . 122

A.2 Parameter values relevant for the MPC tracking optimization in (3.13). . . . . . . . . . 123
List of Figures

1.1 Robots from the past, present, and future. Past: Large automated machines designed to perform repetitive and precise tasks in highly controlled environments. Present: Off-the-shelf “selfie” drones that can capture anyone's adventure. Future: Autonomous vehicles that can perform safe and complex decision-making in highly dynamic and uncertain environments. ........................................ 1

2.1 A traffic weaving scenario whereby two cars initially side-by-side must swap lanes in a short amount of time and distance, emulating a highway on/off-ramp. .................. 9

2.2 Probabilistic graphical model of a conditional variational autoencoder. ............... 14

2.3 Illustration of a multimodal distribution over future human trajectories conditioned on interaction history and candidate future robot trajectory. The distribution changes as the candidate future robot trajectory changes. ........................................ 16

2.4 A CVAE-based multimodal human behavior prediction model with LSTM networks for the encoder and decoder. The dashed lines represent components that are used only during training, while components with solid lines are used in both training and testing. ................................................................. 17

2.5 Left: Vires VTD driving simulation setup used to gather a dataset of 1105 pairwise human-human traffic weaving interactions (Right: drivers are required to swap lanes before the highway cutoff; who shall pass whom is ambiguous at the start). ........ 19

2.6 Predictions (50 samples) from three different models of human future longitudinal acceleration actions conditioned on the available interaction history (solid lines) and future robot actions (dashed blue line). The dashed red line represents the true future action sequence taken by the human. The line color of the prediction corresponds to different $z$ values, if applicable. Eval NLL denotes negative log-likelihood prediction loss over a fixed validation set of interaction trials (lower is better). Not Pictured: basic LSTM prediction model (no multimodality, with $N_z = K_z = M_{GMM} = 1$, Eval NLL = -9.70). ................................................................. 20
2.7 4096 candidate robot action sequences (dependent on current robot state and action in
progress) scored each planning loop. Lane boundaries are marked as dotted horizontal
lines; the visible branching corresponds to the robot’s decision to target a particular
lane during each time window. .................................................. 21
2.8 Example of considerations the robot makes when choosing the best candidate action
sequence. Left: the best-scoring sequence is the robot first waiting then planning to
eventually accelerate since it has time to wait and see if the human will accelerate
or slow down. Center: a candidate action sequence showing a braking maneuver
that predicts the human likely to accelerate (to pass in front) in response. Right: a
candidate robot future action sequence showing a hard acceleration maneuver which
predicts the human likely to decelerate (to let the robot pass) in response. However,
these sequences (center and right) incur a higher control cost penalty $J_s$ than waiting
(left). ................................................................. 23
2.9 Interaction action sequences and $(s, \tau)$-trajectories of the robot (blue) and a human
driver (red) who tries to cut in front. Top Left: robot chooses to accelerate, causing
the human to brake to change lanes near the end of the road. Top Right: both the
human and robot accelerate at first, but the human is more persistent and cuts off
the robot. Bottom: Lateral and longitudinal positions of the human and robot car,
showcasing successful autonomous human-in-the-loop traffic-weaving maneuvers. . . 24
3.1 Illustration of forward and backward reachability, and how they are commonly used
to ensure safe planning and controls. The robot and human are depicted in red and
blue respectively. .......................................................... 28
3.2 Slices of the target set and BRT of a simple example system described in Example 1. 33
3.3 Decision-making and control stack for human-robot pairwise vehicle interactions. A
safety-ensuring control constraint, derived from HJ reachability computed and cached
offline is incorporated into a model predictive controller’s tracking optimization prob-
lem. A high-level interaction planner produces nominal trajectories for the robot car
and the low-level safe tracking controller executes controls that minimally deviate
from the planner’s choice if the vehicles approach the set of unsafe relative states. . . 35
3.4 Schematic of the single track model (bicycle model) tracking a path. This is the
dynamics model used to describe the robot vehicle. .................................. 38
3.5 Schematic of the dynamically-extended unicycle model. This is the dynamics model
used to describe the human vehicle. .................................................. 40
3.6 X1: a steer-by-wire experimental vehicle platform. The 1/10-scale RC car has a mast
that is visible to LiDARs. ....................................................... 45
3.7 Controller comparison on experimental data where the robot car uses the MPC+HJI controller. Left: Simulations of using the MPC, MPC+HJI, and switching controllers when $V(x_{rel}(t))$ first drops below $\epsilon = 0.05$ are shown and are compared against the executed trajectory (from experiments) and the desired trajectory (from the interaction planner). Right: The corresponding evolutions of $V(x_{rel}(t))$. Bottom: Illustration of the traffic-weaving interaction. The human-controlled car (blue car) carelessly drives into the path of the robot car (red car), causing the robot car to react by swerving. The transparency of the rectangles corresponds to the speed of the car (higher transparency corresponds to higher speed).

3.8 Controller comparison on experimental data where the robot car uses the switching controller. Left: Simulations of using the MPC, MPC+HJI, and switching controllers when $V(x_{rel}(t))$ first drops below $\epsilon = 0.05$ are shown and are compared against the executed trajectory (from experiments) and the desired trajectory (from the interaction planner). Right: The corresponding evolutions of $V(x_{rel}(t))$. Bottom: Illustration of the traffic-weaving interaction. The human-controlled car (blue car) carelessly drives into the path of the robot car (red car), causing the robot car to react by swerving. The transparency of the rectangles corresponds to the speed of the car (higher transparency corresponds to higher speed).

3.9 Safety and efficiency trade-offs for different control strategies.

3.10 Comparison of control sequences from using MPC lateral error state constraints, robot-wall HJI control constraint, and no constraints to avoid a static road boundary. The plot begins when the robot car is planning autonomously.

3.11 Trajectory comparison between using lateral error constraints (top), a robot-wall HJI control constraint (middle), and using no wall constraints (bottom). The human car is in the left (top) lane and is constant across all three trials, and the robot car is in the right (bottom) lane. The transparency of the car corresponds to the speed of the car (higher transparency corresponds to higher speed).

3.12 Comparison between the MPC+HJI controller (top) and a baseline approach not based on any reachability analysis (bottom) for the task of dynamic obstacle avoidance. The trajectory of the human car starting in the left (top) lane is visualized in gray; the trajectory of the robot car starting in the right (bottom) lane is visualized in color. In each close-up view, “error bars” accompanying each snapshot of the robot illustrate the first ($k = 1$) lateral state constraint of the MPC problem at that step. To help correspond the positions of the cars over the trials, each pair of matching-shape markers represents a common time instant.
3.13 Time-lapse of pairwise vehicle interaction: X1 with 1/10-scale human-driven RC car.

The RC car (green trajectory) nudges into X1 (red trajectory), which swerves gently to avoid. Inset: The value function over time of the X1-human driven RC car experiment.

4.1 A robot car (red) boxed-in by other externally controlled (e.g., human-driven) cars (blue). If all blue cars behave adversarially, collision is inevitable.

4.2 Due to the curse of dimensionality of HJ reachability, we consider a pairwise decomposition between the robot and each human agent. For each human-robot pair, we consider the HJ value function and corresponding safety-preserving control set.

4.3 Illustration of optimized control using (4.2) and (4.3) under different settings for $U_{\text{multi}}(x_{\text{rel}})$. 

4.4 Snapshot from the highway simulation environment developed in [2]. The robot car (red) is tasked to drive as fast as possible through the traffic but avoid collision with other cars (blue).

4.5 The trade-off between safety (1\textsuperscript{st} percentile TTC) and performance (mean speed) computed from samples taken over ten-second intervals across each episode. Three standard deviation ellipses are shown.

5.1 An illustration of a parse tree (left) and a computation graph (middle) for an STL formula $\Diamond \Box (\phi \land \psi)$ where $\phi = w \geq 0.9$ and $\psi = x \leq 1.0$. On the right is a visualization generated from the stlcg toolbox. The blue nodes represent input signals ($x$ and $w$) into the computation graph, the green nodes represent parameters of the predicates, and the orange nodes represent STL operations.

5.2 Computation graph of the mathematical operation $z = x + y$.

5.3 An illustration the computation graph representing the STL robustness formula for (a) predicates, (b) negation, (c) implies, (d) and, and (e) or. Blue nodes denote input variables, green nodes denote fixed parameter values, and orange nodes denote mathematical operations.

5.4 A schematic of an unrolled recurrent computation graph for the $\Diamond$ (eventually) or $\Box$ (always) temporal operators which differs only in the operation (max or min) used in the recurrent cells, depicted as orange circles. The dashed lines indicate the decomposition/reconstruction of the input/output signal via the time dimension.

5.5 Motion planning with STL specifications solved using stlcg.

5.6 Control effort along the trajectory for all cases with $\theta = \Box \|u\|_2 \leq 0.8$.

5.7 Computation time of different methods used to solve a pSTL problem. This was computed using a 3.0GHz octocore AMD Ryzen 1700 CPU and a Titan X (Pascal) GPU.
5.8 A simple supervised learning problem without (left) and with (right) STL regularization. The output is required to satisfy $\phi = \Box_{[1,2.75]}(s > 0.48 \land s < 0.52)$ despite the training data being noisy and violating $\phi$. ................................. 94

5.9 Architecture of a neural network enhanced with STL regularization in its loss function. ................................. 94

5.10 Schematic of a rolled-out RNN network illustrating how to leverage stlcg to improve long-term predictions with access to only short-term data. ................................. 95

5.11 Comparison of a sequence-to-sequence prediction model without (left) and with (right) robustness regularization. With robustness regularization, the model achieves desired long-term behavior $\phi = \Diamond \Box_{[0,1]}(s > 0.4 \land s < 0.6)$ despite having access to only short-term behaviors during training. ................................. 96

5.12 Architecture of a latent space model leveraging stlcg for added structure. ................................. 96

5.13 A VAE example leveraging stlcg to provide more structure into the latent space representation. Left: Training data used to train a VAE with a discrete latent space. Middle: Generated outputs from a VAE model leveraging stlcg. Right: Generated outputs from a standard VAE without leveraging stlcg. ................................. 97

6.1 We use STL and expert demonstrations to synthesize a trajectory-feedback controller satisfying a desired spatio-temporal specification. The controller consists of a LSTM network and an environment encoding which are optimized offline via an adversarial training strategy. Online, an adaptation scheme refines the controller to robustify against disturbances. ................................. 101

6.2 Neural network architecture of a trajectory-feedback STL controller. The environment $e \in \mathcal{E}$ is encoded into a context vector $c_E$ via a CNN, and then is transformed into the initial hidden state $h_0$ for the LSTM via an MLP. The past trajectory $\tau_p$, is passed through the LSTM to obtain $h_{p+1}$. Future states and controls are generated by propagating through the LSTM and MLP, and then integrating through the system dynamics (with disturbance $d_t$). Note: The dynamics depend on state but the arrows are not shown to reduce visual clutter. The LSTM is unrolled for visualization purposes. ................................. 105

6.3 Environment with an initial state, coverage, obstacle, and goal region. Each channel of the environment image $e$ corresponds to each region type. ................................. 110

6.4 Comparison of trajectories generated from models trained using different values of $\gamma$ and from a human expert. ................................. 112

6.5 Distribution of robustness values (512 samples) corresponding to policies trained with different $\gamma$ values (left) and different adversarial training iterations $K$ (right). No online adaptation is used in computing the robustness value. Orange bar indicates the median, and the green triangle denotes the mean. ................................. 112
6.6 Trajectories generated from different online control strategies. Our proposed method \((TF^*)\) satisfies the STL specification and is robust against disturbances. The corresponding robustness values are (higher is better): Open-loop: -1.08, Baseline: 0.27, \(TF: -0.29, \ TF^*-1: 0.13, \ TF^*-3: 0.42\).

6.7 Comparison of robustness (left) and computation time to compute a control input at each time step (right). 100 random initial states and environments were used. The simulations were ran using a 3.0GHz octocore AMD Ryzen 1700 CPU and a Titan X (Pascal) GPU.

6.8 Trajectory of a car-like robot satisfying STL specification \(\psi\) defined in (6.7). Obstacles, initial states set, and goal set are denoted by the pink, blue and red regions respectively.
Chapter 1

Introduction

Over the past couple of decades, the term “robots” is no longer synonymous with science-fiction. In the beginning, robots only existed in the confines of cages on the factory floor—they were large automated machines designed to perform precise and repetitive motions in highly controlled environments, such as large robotic manipulators used in large-scale assembly lines (see Figure 1.1 left). Now, there are commercial robots that anyone can purchase and fit in the palms of one’s hand, such as “selfie” drones that can follow the user on an outdoor adventure (see Figure 1.1 middle), or a robotic vacuum cleaner that cleans one’s living room. Although physically and functionally different from the robots operating on the factory floor, these present-day robots are similar in that they can only perform relatively structured tasks—tasks with clearly specified goals and expectations on their surroundings (e.g., a robotic vacuum that learns a static map of an apartment) which may be leveraged to simplify decision making.

Figure 1.1: Robots from the past, present, and future. Past: Large automated machines designed to perform repetitive and precise tasks in highly controlled environments. Present: Off-the-shelf “selfie” drones that can capture anyone’s adventure. Future: Autonomous vehicles that can perform safe and complex decision-making in highly dynamic and uncertain environments.

Today, researchers and industry are working towards the next wave of robots: robots that operate in dynamic, stochastic, and unstructured environments, such as settings where a robot must interact with humans. For instance, there has been huge progress made in the field of autonomous driving (see Figure 1.1 right), robotic manipulators that aid humans in manufacturing tasks, or robots...
that can assist the elderly. This new generation of robots is a stark departure from the previous and current generation of robots primarily because these new generation robots are expected to (i) operate in highly dynamic and uncertain environments, (ii) interact with human agents, and (iii) perform a variety of tasks under many different settings.

Much of the progress towards the next wave of robots has been made possible by the incredible advancements in computing, artificial intelligence, and machine learning. In particular, deep learning has enabled researchers and alike to develop data-driven algorithms that can learn complex patterns and relationships from high-dimensional data and generalize them to new inputs, thereby enabling systems to perform complex tasks. Consequently, there has been a plethora of powerful deep learning techniques that bring us closer to the next generation of robotic systems—deep learning has opened up new ways for a robot to perceive and understand the world around it (e.g., [3]), reason about the uncertainty in the environment (e.g., [4]), and move around in an unstructured world (e.g., [5]). Unfortunately, as deep learning becomes more pervasive throughout the autonomy stack, it also becomes increasingly difficult to ascertain the performance and safety of these robotic systems and explain their behavior, necessary prerequisites for their deployment in safety-critical settings. Consequently, a core challenge is ensuring the safety and trustworthiness of learning-enabled robotic systems, especially those that interact in close proximity with humans.

This dissertation represents a culmination of efforts towards the goal of building trust in robot autonomy along two axes: (i) developing safety assurance for robots interacting with humans or human counterparts, and (ii) incorporating interpretability and structure in learning-based techniques to provide a level of transparency into otherwise opaque deep learned components.

1.1 Dissertation Contributions

The goal of this dissertation is to harness advances in learning-empowered robot autonomy and unite them with the rigor and assurances provided by formal methods to develop powerful yet safe and trustworthy autonomous systems. To this end, the contributions of this dissertation fall into two main categories.

I. Safety assurance for model-based probabilistic planning for human-robot interactions. The contributions of this part examine how to provide safety assurance in a principled manner for probabilistic planning problems that require a robot to reason about uncertainty in human behaviors and complex interaction dynamics between agents. We approach this problem from the context of autonomous driving wherein a self-driving car (a “robot”) is required to predict the behavior of a human-driven car (a “human”) and plan optimal yet safe trajectories that account for possible responses by the human. First, we learn the complex and stochastic interaction dynamics between a human and robot agent for proactive planning and decision-making. Specifically, we develop a model-based probabilistic robot planner that leverages
1.1. DISSERTATION CONTRIBUTIONS

a multimodal probability distribution over future action sequences of human drivers conditioned on the present interaction history as well as candidate future robot action sequences. Second, understanding that the probabilistic and data-driven nature of our approach may lead to cases where the human’s actions defy the robot planner’s expectation, we devise a minimally-interventional safety controller operating in concert with a probabilistic interaction planner. We leverage a formal verification technique, namely Hamilton-Jacobi (HJ) reachability, to design a safety controller that will modify a nominal controller only when the system is near possible inevitable collision and only to the extent necessary without unduly impacting planning performance. The complete planning and control stack is validated on a full-scale steer-by-wire vehicle with human-in-the-loop experiments and we demonstrate that our approach achieves a good balance between safety and efficiency compared to traditional reactive control techniques. Third, we extend our minimally-interventional safety controller to multi-agent settings and investigate how HJ reachability can be utilized beyond its traditional usage within the control module to provide consistent notions of safety throughout the autonomy stack.

II. Infusing logical reasoning into robot learning. The contributions of this part examine how to infuse logical reasoning within gradient-based techniques that make up components of the decision-making and control stack. As many robots that we are interested in within this dissertation interact with humans, there are rules, or spatio-temporal specifications, that govern how the interactions should play out. For instance, road rules that govern how (self-driving) vehicles should operate on the road. In this dissertation, we express spatio-temporal specifications using Signal Temporal Logic (STL), a formal logic language that is capable of translating specifications written in natural language into a concise mathematical representation. A special property of STL is that it is equipped with a notion of robustness, a measure of how much an STL specification is satisfied or violated. One of our core contributions in this dissertation is the development of stlcg, a technique that expresses STL robustness formulas as computation graphs. By doing so, we can leverage highly-optimized modern automatic differentiation tools that are readily (and freely) available to the public and thereby bridging the gap between gradient-based methods with spatio-temporal logic. Using stlcg, we can incorporate spatio-temporal considerations within gradient-based techniques such as trajectory optimization and neural network training for behavior prediction and policy construction. As a result, we can provide inductive bias in the form of spatio-temporal specifications which can aid in improving data efficiency and performance. We provide several examples to demonstrate the flexibility of stlcg, and take a deep dive into a controller synthesis problem subject to spatio-temporal constraints.
1.2 Structure

The rest of this dissertation is structured as follows.

**Chapter 2**: We develop a data-driven learning-based model capable of capturing multimodal uncertainty characterizing human-human interactions to use in planning and decision-making for robots interacting with humans. We couple the model with a planner to construct a proactive model-based probabilistic planning algorithm tailored towards human-robot interactions. In addition to detailing the core modeling and planning framework, we also introduce our definition of an interactive scenario and outline key considerations when designing a robot planning algorithm. We also perform human-in-the-loop experiments in simulation to demonstrate the efficacy of this work.

**Chapter 3**: With the understanding that a probabilistic planner such as the one developed in Chapter 2 may make sub-optimal decisions due to unmodeled stochasticity or limitations in the planning algorithm, we leverage HJ reachability theory to provide safety assurance for the planning and control stack. When a planner is faced with an unexpected situation and is in a potentially unsafe state, a reachability-based safety controller will compute minimally-interventional evasive maneuvers to help a robot stay safe without unduly impacting planning performance. In addition to describing the core technical details of our proposed algorithm, we also provide a self-contained introduction to HJ reachability and perform human-in-the-loop experiments on a full-scale vehicle to demonstrate the efficacy and practical implications of this work.

**Chapter 4**: We extend the work developed in Chapter 3 to multi-agent scenarios and also elevate the use of HJ reachability to the planning level to provide a shared notion of safety throughout the planning and control stack. We showcase the benefits of having consistent notions of safety throughout the planning and control stack in a multi-agent highway driving scenario.

**Chapter 5**: We shift gears and circle back to the idea of modeling interactions—we focus on the idea of incorporating logical structure as a form of inductive bias in how we model interactions. In this chapter, we introduce the concept of STL as a formal language to express spatio-temporal specifications that can be useful for describing interactions and task specifications. We then present the technical underpinnings of stlcg, a computational technique that translates STL specifications into computation graphs such that STL can be used within gradient-based methods. We then provide several illustrative examples to demonstrate the versatility of stlcg.

**Chapter 6**: We focus on the problem of synthesizing a controller such that the resulting trajectory satisfies a prescribed STL specification. We leverage stlcg to synthesize a neural-network trajectory-dependent controller and illustrate how temporal logic and expert demonstrations can improve the data efficiency of a data-driven control synthesis process.

**Chapter 7**: We conclude with a summary of the contributions presented in this dissertation and outline directions for future work. We also highlight broader important research questions related to the topics discussed in this dissertation.
Part I

Safety Assurance for Human-Robot Interactions
Human behavior is inconsistent across populations, settings, and even different instants, with all other factors equal—addressing this inherent uncertainty is one of the fundamental challenges in human-robot interactions (HRI). Even when a human’s broader intent is known, there are often multiple distinct courses of action (i.e., multimodal) they may pursue to accomplish their goals. For example, a driver signaling a lane change may throttle up aggressively to pass in front of a blocking car, or brake to allow the adjacent driver to pass first. To an observer, the choice of mode may seem to have a random component, yet also depend on the evolution of the human’s surroundings, e.g., whether the adjacent driver begins to yield.

Taking into account the full breadth of possibilities in how a human may respond to a robot’s actions is a key component of enabling anticipatory and proactive robot interaction policies. With the goal of creating robots that interact intelligently with human counterparts, observing data from human-human interactions provides valuable insight into predicting interaction dynamics [6, 7, 8, 9]. In particular, a robot may reason about human actions, and corresponding likelihoods, based on how it has seen humans behave in similar settings.

Accordingly, an increasingly active research area is in human behavior prediction (also referred to as trajectory forecasting), especially in the autonomous driving field where safe yet proactive decision-making is critical for many driving interactions, such as merging into dense traffic or navigating an unprotected left turn. In Chapter 2 we propose a multimodal deep generative model to predict future human behaviors and use the model in constructing proactive robot action policies that account for complex interaction dynamics. We demonstrate the efficacy of our approach on a traffic-weaving scenario which possesses many characteristics that exemplify interesting yet challenging qualities of HRI.

Despite steady progress made in the field of human behavior prediction both along the axis of data collection and model complexity, ensuring safety remains a core challenge due to many reasons, including: (i) Even with accurate probabilistic prediction models, planning with complex prediction models is non-trivial and it is difficult to provide strong probabilistic safety guarantees without making simplifying assumptions (e.g., assuming Gaussian uncertainty). (ii) The data used to train a prediction model may be limited or biased and therefore not adequate in capturing the full spectrum of possible human behaviors when deployed in the real world. (iii) Planners typically do not operate at a high enough frequency to respond to split-second threats, such as a distracted driver suddenly swerving into the adjacent lane. (iv) Safety competes with other planning objectives, such as maintaining speed or minimizing time, and often a planner utilizes simplified models and/or simplifying assumptions that do not translate accurately to real-world settings.

With that said, there is still a lot of ongoing progress towards designing robust planning algorithms, but ultimately due to the aforementioned reasons, there will inevitably be situations where a robot interaction planner may be “surprised” and therefore result in a potentially dangerous situation.
In Chapter 3, we devise a reachability-based safety controller that principally infuses strict notions of safety into a low-level tracking controller to safeguard against “surprises” that a planner may encounter. We leverage Hamilton-Jacobi (HJ) reachability, a formal verification technique, to define notions of safe and unsafe sets and propose a safety controller that will only step in whenever the system is unsafe and control a robot to take minimally-interventional evasive controls only to the extent necessary to stay safe but without unduly impacting planning performance. We couple our reachability-based safety controller with the traffic-weaving interaction planner discussed in Chapter 2 and test the combined planning and control stack on a full-scale steer-by-wire vehicle. We show that our proposed minimally-interventional safety controller enables a robot to achieve safe yet still high-performing behavior. Finally, in Chapter 4 we extend the ideas presented in Chapter 3 to multi-agent settings and investigate how we can elevate HJ reachability, a typically low-level control technique, to the planning level to equip a planner and controller with a consistent notion of safety.
Chapter 2

Multimodal Probabilistic Model-based Planning for Human-Robot Interactions

A core challenge in robot planning for HRI is in capturing the uncertainty from human behaviors. As previously discussed, even when a human’s broader intent is known, there are often multiple distinct courses of action they may pursue to accomplish their goals. Accordingly, the objective of this chapter is to devise a data-driven framework for HRI that leverages learned multimodal human action distributions in constructing robot action policies.

2.1 Introduction

Methods for robot decision-making under uncertainty may be classified as model-free, whereby human action possibilities and associated likelihoods are implicitly encoded in a robot control policy learned from trial experience, or model-based, whereby a probabilistic understanding of the interaction dynamics is used as the basis for policy construction. In this work, we take a model-based approach to pairwise human-robot interaction, seeking to explicitly characterize a possibly multimodal distribution over human actions at each time step conditioned on interaction history as well as future robot action choices. By decoupling action/reaction prediction from policy construction, we aim to achieve a degree of transparency in a planner’s decision-making process that is typically unavailable in model-free approaches. Conditioning on history allows a robot to reason about hidden factors like experience, mood, or engagement level that may influence the distribution, and conditioning on the future takes into account response dynamics.
2.1. INTRODUCTION

Figure 2.1: A traffic weaving scenario whereby two cars initially side-by-side must swap lanes in a short amount of time and distance, emulating a highway on/off-ramp.

We develop our work around a traffic weaving case study (Figure 2.1) for which we adapt methods from deep neural network-based language and path prediction [11, 12, 13, 14] to learn a generative model of human driver behavior. The traffic weaving scenario is representative of the many challenges central to HRI, in particular, the behaviors of each driver are tightly coupled and there is uncertainty in how the other driver may act in response to the actions of the other. We validate our model as the basis for a limited-lookahead autonomous driver action policy, applied in a receding horizon fashion, the behavior of which we explore with human-in-the-loop testing.

We highlight five key considerations that motivate our modeling and policy construction framework:

1. **Response dynamics** — For interactive situations, the modeling framework must be capable of predicting human behavior in response to (i.e., conditioned on) candidate robot actions.

2. **Time scale** — We target decision-making scenarios with characteristic action and response time scales on the order of \( \sim 1 \) second. This is in contrast to higher-level reasoning over a set of multi-second action sequences or action-generating policies (e.g., whether a driver intends to turn or continue straight at an intersection [15], or intends to initiate a highway lane change [16]). Nor do we attempt to emulate lower-level reactive controllers (e.g., emergency collision avoidance systems [17]) that must operate on the order of milliseconds.

3. **Multimodality** — As previously stressed, the uncertainty in human actions on this time scale may be multimodal, corresponding to varied optimal robot action plans.

4. **History dependence** — We desire a prediction model that is history-dependent, capable of inferring latent features of human behavior.

5. **Model-based control** — Although our prediction model is trained “end-to-end” from state observations to human action distributions, this work decouples dynamics learning from policy construction to aid interpretability and enable flexibility with respect to robot goals.

\(^1\)Those works were considered state-of-the-art in 2017–2018 during the time the work in Chapter 2 was developed. There has been tremendous progress made in deep generative modeling for human behavior prediction since then.

\(^2\)Considerations of lower-level controls for safe robot decision-making can be found in Chapter 3.
2.2 Related Work

A major challenge in learning generative probabilistic models for HRI is accounting for the rapid growth in problem size due to the time-series nature of interaction; state-of-the-art approaches rely on some form of dimensionality reduction to make the problem tractable. One option is to model humans as optimal planners and represent their motivations at each time step as a state/action-dependent cost (equivalently, negative reward) function. Minimizing this function, e.g., by following its gradients to select next actions, may be thought of as a computational proxy for human decision-making processes.

This cost function has previously been expressed as a linear combination of potential functions in a driving context [16, 18]; Inverse Reinforcement Learning (IRL) is a generalization of this idea whereby a parameterized family of cost functions is fit to a dataset of human state-action trajectories [19, 20, 21, 22]. Typically, the cost function is represented as a linear combination of possibly nonlinear features, \( c(x, u) = \theta^T \phi(x, u) \), and the weight parameters \( \theta \) are fit to minimize a measure of error between the actions that optimize \( c \) and the true human actions [19, 20].

Maximum entropy IRL interprets this cost function as probability distribution over human actions, with \( p(u) \propto \exp(-c(x, u)) \), and in this case the weights \( \theta \) are fit according to maximum likelihood [6, 21, 22]. This framework is employed in the context of interactive driving in [7] to construct robot policies that avoid being overly defensive by leveraging expected human responses to robot actions. In that work, however, the probabilistic interpretation of \( c \) is used only in fitting the human model, not in robot policy construction, where it is assumed that the human selects best responses to robot actions in a Stackelberg game formulation. This analysis yields a unified and tractable framework for prediction and policy construction, but fundamentally represents a unimodal assumption on interaction outcome; we note that this style of reasoning has proven dangerous in the case that critical outcomes go uncaptured [23]. Regarding multimodal probabilistic dynamics, it is true that with sufficiently complex and numerous features the cost function \( \theta^T \phi(x, u) \) may approximate any log-probability distribution over \( u \), conditioned on state \( x \), arbitrary well. Without some form of state augmentation, however, this formulation is Markovian and incapable of conditioning on interaction history when reasoning about the future.

An additional requirement of cost/motivation-based human modeling methods for use in interactive scenarios is a means to reason about a human’s reasoning process. Game theory has been applied to combine human action/reaction inference and robot policy construction [24, 25, 26]. Common choices are Stackelberg formulations whereby human and robot alternate “turns,” hierarchical reasoning or “level-k” approaches whereby agents recursively reason about others reasoning about themselves down to a bounded base case, and equilibria assumption, taking \( k \to \infty \). In this work we approach the human modeling problem phenomenologically, attempting to directly learn the action probability distribution that might arise from such a game formulation.
Computationally tractable human action modeling has also been achieved by grouping actions over multiple time steps and reasoning over a discrete set of template action sequences or action-generating policies [15, 16, 27, 28]. These methods often tailor means for modeling dependencies between observations, predictions, and latent variables specific to their application that may be used to incorporate information about interaction history. In the context of mutual adaptation for more efficient task execution, [28] consider multimodal outcomes and maintain a latent state representing a human’s inclination to adapt. The approach of [15] for synthesizing autonomous driving policies is very similar in spirit to our work; their prediction model is capable of capturing multimodal human behavior at every time step, dependent on state history and robot future, which they use to score and select from a set of candidate policies for an autonomous vehicle. However, they restrict their treatment of time to changepoint-delineated segments within which the human action distribution takes the form of a Gaussian Mixture Model (GMM). The mean trajectories of these Gaussian components (modes) are predetermined by the choice of a finite set of high-level driving behaviors, and mixture weight inference takes place over the current time segment.

Inspired by the recent groundbreaking success of deep neural networks in modeling distributions over language sequences [12, 29] and geometric paths [13, 14], we seek instead to learn a generative model for human behavior that does not decouple time segmentation from probabilistic mode inference and is capable of learning the equivalent of arbitrary mode policies from data. Instead of directly fitting a neural network function approximator to log probability, akin to extending maximum entropy IRL with deep-learned features, we use a Conditional Variational Autoencoder (CVAE) [11] setup to learn an efficiently sampleable mixture model with terms that represent different driving behaviors over a prediction time horizon. We use a Recurrent Neural Network (RNN) to iteratively condition each time step’s action prediction on the preceding time steps. This opens up the possibility for another level of multimodality within the prediction horizon (e.g., uncertainty in exactly when a human will initiate a predicted braking maneuver), that would otherwise require too many mixture components. An alternate interpretation of why this second level of multimodality is required is to address the case that a mode changepoint, in the language of [15], lies within the fixed prediction horizon. We would also like to briefly mention [9] as a recent nonparametric learning method that compares human driver trajectories to a form of nearest neighbors from an interaction dataset to predict future behavior, however, the authors of [9] note that this comparison procedure is difficult to scale at run time for online policy construction.

2.3 Problem Formulation

In this section, we will describe the problem formulation for interaction dynamics, robot goal, and the traffic weaving scenario which will be a running example through this chapter and the next chapter as well.
2.3.1 Interaction Dynamics

We refer to an ego agent, that is, an agent that we want to design a planner for, as a “robot”, and a human agent which the robot is to interact with as a “human”. Let the deterministic, time-invariant, discrete-time state space dynamics of a human and robot be given by

\begin{align}
\text{Human dynamics: } x_{HR}^{t+1} &= f_{H}(x_{HR}^t, u_{HR}^t), \\
\text{Robot dynamics: } x_{R}^{t+1} &= f_{R}(x_{R}^t, u_{R}^t) \tag{2.1}
\end{align}

where \( x_{HR}^t \in \mathbb{R}^{N_{HR}}, x_{R}^t \in \mathbb{R}^{N_{R}} \) denote the states of the human and robot and \( u_{HR}^t \in \mathbb{R}^{M_{HR}}, u_{R}^t \in \mathbb{R}^{M_{R}} \) their chosen control actions at time \( t \in \mathbb{N}_{\geq 0} \). Let \( x_{HR}^t = (x_{HR}^t, x_{R}^t) \) and \( u_{HR}^t = (u_{HR}^t, u_{R}^t) \) denote the joint state and control of the two interaction agents. We consider interactions that end when the joint state first reaches a terminal set \( \mathcal{T} \subset \mathbb{R}^{N_{HR}+N_{R}} \) and let \( T \) denote the final time step, \( x_{HR}^T \in \mathcal{T} \).

We assume that at each time step \( t < T \), the human’s next action \( u_{HR}^{t+1} \) is drawn from a distribution conditioned on the joint interaction history \((x_{HR}^0, u_{HR}^0)\) and the robot’s next action \( u_{R}^{t+1} \), that is,

\begin{align}
U_{HR}^{t+1} &\sim P(x_{HR}^0, u_{HR}^0, u_{R}^{t+1}) \tag{2.2}
\end{align}

is a random variable (capitalized to distinguish from a drawn value \( u_{HR}^{t+1} \)). We suppose additionally that \( U_{HR}^{t+1} \) is distributed according to a probability density function (pdf) which we write as \( p(u_{HR}^{t+1} \mid x_{HR}^0, u_{HR}^0, u_{R}^{t+1}) \). In this work we assume full observability of all past states and actions by both agents. Note that by iteratively propagating (2.1) and sampling (2.2), a robot may reason about the random variable \( U_{HR}^{t+1:t+N} \), denoting a human’s response sequence to robot actions \( u_{R}^{t+1:t+N} \) over a horizon of length \( N \).

2.3.2 Robot Goal

We aim to design a limited-lookahead action policy \( u_{R}^{t+1:t+N} = \pi_{R}(x_{HR}^t, u_{HR}^t) \) for the robot that minimizes expected cost over a fixed horizon of length \( N \). We consider a running cost \( J(x_{HR}^t, u_{R}^t, x_{HR}^{t+1}) \) which takes the form of a terminal cost \( J(x_{HR}^T, u_{R}^T, x_{HR}^{T+1}) = J_f(x_{HR}^T) \) should the interaction end before the horizon is reached (with \( J(x_{HR}^t, u_{R}^t, x_{HR}^{t+1}) = 0 \) for \( t \geq T \)). That is, we seek \( \pi_{R}(x_{HR}^t, u_{HR}^t) \) as an approximate solution to the following minimization problem:

\begin{align}
\arg\min_{u_{R}^{t+1:t+N} \in \mathbb{R}^{M_{R}}^N} \mathbb{E} \left[ \sum_{i=1}^{N} \gamma^i J(x_{HR}^{t+i}, u_{R}^{t+i}, x_{HR}^{t+i+1}) \right] \tag{2.3}
\end{align}

where \( \gamma \in [0, 1] \) is a discount factor. In practice we take \( N = 15 \) (with time interval 0.1s) and iteratively solve (2.3), executing only the first action in an Model Predictive Control (MPC) fashion.

\footnote{We note that if \( U_{HR}^{t+1} \) has a discrete component, e.g., a zero acceleration action with positive probability mass, we may add a small amount of white noise to observed values \( u_{HR}^{t+1} \) when fitting distributions for \( U_{HR}^{t+1} \) to preserve this assumption.}
Note that at time $t$ we are solving for the action to take at time step $t+1$, as owing to nonzero computation times we regard the robot action $u^t_R$ to already be in progress, having been computed at the previous time step.

### 2.3.3 Traffic Weaving Scenario

Although we have developed our approach generally for pairwise human-robot interactions where the human may be regarded as acting stochastically, we focus in particular on a traffic weaving scenario as depicted in Figure 2.1. We learn a human action model and compute a robot policy with the assumption that both agents are signaling an intent to swap lanes. Let $(s, \tau)$ be the coordinate system for the two-lane highway where $s$ denotes longitudinal distance along the length of the highway (with 0 at the cutoff point and negative values before) and $\tau$ denotes lateral position between the lanes (with 0 at the left-most extent of the left lane and negative values to the right). We consider the center of mass of the human-controlled car (“human”) to obey double-integrator dynamics ($u_H = [\ddot{s}_H, \ddot{\tau}_H]$, $x_H = [s_H, \tau_H, \dot{s}_H, \dot{\tau}_H]$ in continuous form). These dynamics may be transformed to various simple car models and map closely to body-frame longitudinal acceleration and steering angle at highway velocities. It is for this latter consideration that we choose a second-order system model, as we believe it most straightforward to fit a generative pdf to inputs on the same order of the true human inputs (throttle and steering command), even if the cost $(2.3)$ is a function only of human position or velocity. We use a similar model for the robot-controlled car (“robot”), but with a triple-integrator in lateral position ($u_R = [\ddot{s}_R, \ddot{\tau}_R], x_R = [s_R, \tau_R, \dot{s}_R, \dot{\tau}_R, \ddot{\tau}_R]$ in continuous form) to ensure a continuous steering command. The robot receives a large penalty for colliding with the human and is motivated to switch lanes before entering the terminal set $T = \{x_{HR} = (x_H, x_R) | s_R \geq 0\}$ by a running cost proportional to lateral distance to target lane $|\tau_R - \tau^\text{target}_R|$ multiplied by an increasing measure of urgency as $s_R$ approaches 0, as well as a reward term that encourages joint states $x_{HR}$ where the two cars are moving apart longitudinally. Further details of the cost expression are discussed in Section 2.6.3.

A few comments are in order. First, we note that the distributional form $(2.2)$ admits the possibility that the human has knowledge of the robot’s next action before selecting his/her own. This Stackelberg assumption has been employed previously in similar learning contexts [7], but we stress that in the present work this assumption has no bearing on policy construction, where we only sample from $P$. Conditioning on extraneous information should have a negligible effect on learning if $u^t_{H}$ and $u^t_{R}$ are truly independent. Second, this formulation is intended to capture factors that may be observed in the prior interaction history $x^t_{HR}$, e.g., population differences in driving style [22], as well as the response dynamics of the interaction, e.g., game-theoretic behaviors that have previously been modeled explicitly [25] [26]. Admittedly, approaching this modeling problem phenomenologically, as opposed to devising a more first-principles approach, requires significantly more data to fit. However, with large-scale industrial data collection operations already in place,
the ability to utilize large datasets in studying relatively common interaction scenarios, such as the traffic weaving example studied in this work, is becoming increasingly common. Third, we note that the robot cost objective sketched above is admittedly rather ad hoc; this is a consequence of requiring the robot to make decisions over a fixed time horizon. We argue that it is difficult to trust interactive prediction models over long horizons $N \gg 0$ in any case, necessitating some sort of long-term cost heuristic to inform short-term actions. Furthermore, our model-based framework can accommodate other cost objectives should they better suit a system designer’s preferences, though is not clear what a quantitative measure of quality should be for this traffic weaving scenario. Changing lanes smoothly and courteously, as a human would, does not imply, e.g., that the robot should plan for minimum time or safest possible behavior.

2.4 Multimodal Human Behavior Prediction Model

We model the conditional human action distribution in (2.2) as a neural network CVAE with recurrent subcomponents to manage the time-series nature of the interaction. First, we will describe a general CVAE [11] and apply it specifically to the context of human behavior prediction.

2.4.1 Conditional Variational Autoencoder

![Figure 2.2: Probabilistic graphical model of a conditional variational autoencoder.](image)

Given a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$, the goal of conditional generative modeling is to fit a model of the conditional probability distribution $p(y | x)$, which may be used for downstream applications such as inference (i.e., calculating the likelihood of observing a particular sample $y$ given $x$), or to generate new samples $y$ given $x$. In this work, we consider parametric models, whereby we consider $p(y | x)$ within a family of distributions defined by a fixed set of parameters, which we fit to the dataset to maximize the likelihood of the observed data. Due to their expressivity, neural networks are often used to represent complex and high-dimensional distributions.

A CVAE is a latent conditional generative model. The goal is still to approximate $p(y | x)$, but before outputting $p(y | x)$, the model first projects the inputs into a lower-dimensional space,
2.4. MULTIMODAL HUMAN BEHAVIOR PREDICTION MODEL

called the *latent space*, which acts as a bottleneck to encourage the model to uncover salient features with the intended purposes of improving performance, and potentially aiding in interpretability. Figure 2.2 illustrates the graphical structure of a CVAE. An encoder, parameterized by \( \theta \), takes the input \( x \) and produces a distribution \( p_0(z \mid x) \) where \( z \) is a latent variable that can be continuous or discrete \([30, 31]\). A decoder, parameterized by \( \phi \), uses \( x \) and samples from \( p_0(z \mid x) \) to produce \( p_\phi(y \mid x, z) \). In practice, the encoder and decoder are neural networks. The latent variable \( z \) is then marginalized out to obtain \( p(y \mid x) \),

\[
p(y \mid x) = \sum_z p_\phi(y \mid x, z)p_0(z \mid x). \tag{2.4}
\]

To efficiently perform the marginalization in (2.4), we desire values of \( z \) that are likely to have produced \( y \), otherwise \( p_0(z \mid x) \approx 0 \) and will contribute almost nothing to \( p(y \mid x) \). To this end, we perform importance sampling by instead sampling from \( q(z \mid x, y) \), a proposal distribution, which will help us select values of \( z \) that are likely to have produced \( y \). Since we are free to choose \( q(z \mid x, y) \), we parameterize it (often as a neural network) with \( \varphi \), denoted by \( q_\varphi(z \mid x, y) \). We can rewrite (2.4) by multiplying and dividing by the proposal distribution, and using the definition of expectation,

\[
p(y \mid x) = \sum_z \frac{p_\phi(y \mid x, z)p_0(z \mid x)}{q_\varphi(z \mid x, y)}q_\varphi(z \mid x, y) = \mathbb{E}_{q_\varphi(z \mid x, y)} \left[ \frac{p_\phi(y \mid x, z)p_0(z \mid x)}{q_\varphi(z \mid x, y)} \right].
\]

The goal is to fit parameters \( \phi, \theta \), and \( \varphi \) that maximize the log-likelihood of \( p(y \mid x) \) over the dataset \( D \). By taking the log of both sides, using Jensen’s inequality, and rearranging the terms, the *evidence lower-bound* (ELBO) is derived,

\[
\log p(y \mid x) \geq \mathbb{E}_{q_\varphi(z \mid x, y)} \left[ \log p_\phi(y \mid x, z) \right] - D_{\text{KL}}[q_\varphi(z \mid x, y) \parallel p_0(z \mid x)]
\]

where \( D_{\text{KL}}(q||p) = \mathbb{E}_{q(z)} \left[ \log \frac{q(z)}{p(z)} \right] \) is the Kullback-Liebler divergence. The ELBO is a lower bound on \( \log p(y \mid x) \), the quantity that we are trying to maximize, but which is often intractable to compute directly by (2.4). Instead, we maximize the ELBO as a proxy. By using the reparameterization trick \([30, 31, 32]\), the ELBO is tractable to compute and can be optimized via stochastic gradient descent. The loss for a single training example \((x, y)\) is,

\[
\mathcal{L}(x, y) = -\mathbb{E}_{q_\varphi(z \mid x, y)} \left[ \log p_\phi(y \mid x, z) \right] + D_{\text{KL}}[q_\varphi(z \mid x, y) \parallel p_0(z \mid x)].
\]

\(^4\)For this work, we focus on a discrete latent space but note the following equations still apply by replacing the summation with an integral.

\(^5\)We note that if the size of the discrete latent space is small, we can tractably compute the summation in (2.4) exactly.
During training, we minimize the Monte Carlo estimate of the expected loss over the training set.

### 2.4.2 Interaction-aware Human Behavior Prediction

With particular choices for \( x \) and \( y \) in a CVAE described in Section 2.4.1, we can build a human behavior prediction model that predicts future human action sequences conditioned on both interaction history and candidate future robot action sequences thereby taking into account the coupling between human and robot actions, i.e., interaction dynamics. Figure 2.3 illustrates how the multimodal prediction over future human action sequences vary, both in the shape of the trajectory and the likelihood of that trajectory mode occurring, as the candidate future robot action sequence varies. Note that we can turn action sequences into trajectories by integrating the human and robot dynamics described by (2.1).

Considering a fixed prediction time step \( t \), let \( x = (x_{0:t}^{HR}, u_{0:t}^{HR}, u_{t+1:t+N}^{H}) \) be the conditioning variable (joint interaction history + candidate robot future) and \( y = u_{t+1:t+N}^{H} \) be the prediction output (human future). We choose a discrete latent space such that \( z \) is a categorical random variable with distribution factored over \( N_z \) independent elements, \( p(z = (z_1, \ldots, z_{N_z})) = \prod_{i=1}^{N_z} p(z_i) \) where each element has \( K_z \) categories, resulting in a latent space dimension of size \( K_z^{N_z} \). Thus \( p(y|x) = \sum_{z} p(y|x, z)p(z|x) \) may be thought of as a mixture model with components corresponding to each discrete instantiation of \( z \). The motivating idea in choosing a discrete latent space is that each \( z \) value will manifest as different human response modes, e.g., accelerating/decelerating or driving straight/turning over the next \( N \) time steps. Each colored (orange, blue, and green) trajectory “tube” in Figure 2.3 represents different human response modes. Modeling this multimodal human response behavior is a core focus of this work; the benefits of using a discrete latent space to learn multimodal distributions have been previously studied in [30, 31, 33].

The encoder and decoder are made up of Long-Short Term Memory (LSTM) networks, a type of RNN tailored towards efficiently processing time-series inputs. As shown in Figure 2.4, the
2.4. MULTIMODAL HUMAN BEHAVIOR PREDICTION MODEL

Figure 2.4: A CVAE-based multimodal human behavior prediction model with LSTM networks for the encoder and decoder. The dashed lines represent components that are used only during training, while components with solid lines are used in both training and testing.

The encoder \( p_\theta(z \mid x) \) is made up of two LSTM networks, one for the interaction history \( x_{HR}^{0: t}, u_{HR}^{0: t} \), and another for candidate future robot actions \( u_{R}^{t+1: t+N} \). The final outputs of each LSTM are concatenated to produce \( h_x \), a hidden state, which is then passed through a linear projection for which the output is used to characterize the categorical distribution \( p_\theta(z \mid x) \). The parameters of the proposal distribution \( q_\phi(z \mid x, y) \) are computed similarly; the final output of the future human trajectory LSTM encoder is concatenated with \( h_x \) to produce \( h_y \), which is then passed through a linear projection whose output will characterize \( q_\phi(z \mid x, y) \), a categorical distribution. Given the parameters of \( p_\theta(z \mid x) \) and \( q_\phi(z \mid x, y) \), we can then sample from those distributions to obtain \( z \) (the dice icon in Figure 2.4 indicates sampling) which is used to initialize the LSTM decoder.

The discrete latent variable \( z \) has the responsibility of representing high-level behavior modes, while a second level of multimodality within each such high-level behavior is facilitated by an autoregressive RNN sequence decoder. The RNN maintains a hidden state to allow for drawing each future human action conditioned on the actions drawn at previous future times:

\[
p(y \mid x, z) = \prod_{i=1}^{N} p(y^{(i)} \mid x, z, y^{(1:i-1)})
\]  

(2.5)

where we use the notation \( y^{(i)} = u_{HR}^{t+i} \), the \( i \)-th future human action after the current time step \( t \). A GMM output layer of each RNN cell with \( M_{GMM} \) components in human action space provides the basis for learning arbitrary distributions for each \( p(y^{(i)} \mid x, z, y^{(1:i-1)}) \). This combined structure is designed to account for variances in human trajectories for the same latent behavior \( z \). For example, an action sequence of braking at the fourth time step instead of at the third time step would likely
need to belong in a different component of, e.g., a non-recurrent Gaussian Mixture Model, causing a combinatorial explosion in the required number of distinct $z$ values.

A note on implementation; in addition to typical model training techniques (e.g., recurrent dropout regularization [34] and hyperparameter annealing), we employ the method introduced in [35] to prevent prediction errors from severely cascading into the future. During train time in the decoder, with a rate of 10% we use the predicted value $\hat{u}_t^H$ as the input into the next cell, otherwise, we use the true value $u_t^H$ from the training data. That is, instead of learning individual terms of (2.5), we occasionally learn them jointly. Similar to [14] we augment the human action inputs for the autoregressive decoder RNN with an additional context vector $c^t$ composed of the robot action at that time step $u_t^R$, the latent variable $z$ and the output from the encoder $h_x$. We do this to more explicitly mimic the form of the expression $p(y^{(i)}|x, z, y^{(1:i-1)})$.

2.5 Robot Policy Construction

We use the CVAE prediction model described in Section 2.4 to select the best candidate robot action sequence to take. We apply an exhaustive approach to optimizing problem (2.3) over possible robot action sequences where at each time step the robot is allowed to take one of a finite set of actions (in particular, we group the $N = 15$ future time steps into five 3-step windows over which the robot takes 1 of 8 possible actions, see Section 2.6.3 for details). We stress that we are only discretizing the robot action space for policy computation efficiency; the human prediction model is still computed over a continuous action space. While Monte Carlo Tree Search (MCTS) methods have seen successful application in similar problems with continuous state/action spaces [36], due to the massively parallel GPU implementation of modern neural network frameworks [37] we find it is more expedient to simply evaluate the expected cost of taking all action sequences (or a significant fraction of them) for sufficiently short horizons $N$. We take a two-step approach, approximately evaluating all action sequences with a low number of samples (human response futures) each, and then reevaluating the most promising action sequences with a much larger number of samples to pick the best one (see Section 2.6.3). We note that gradient-based methods could prove useful for further action sequence refinement, but for scenarios characterized by multimodal outcomes, some form of broader search must be applied lest optimization end in a local minimum.

2.6 Traffic-weaving Case-study

The human-human traffic weaving dataset and source code for all results in this section, including all network architecture details and hyperparameters, are available at [https://github.com/StanfordASL/TrafficWeavingCVAE](https://github.com/StanfordASL/TrafficWeavingCVAE). All simulation and computation were done on a computer running Ubuntu 16.04 equipped with a 3.6GHz octocore AMD Ryzen 1800X CPU and an NVIDIA
2.6. TRAFFIC-WEAVING CASE-STUDY

2.6.1 Data Collection

State and action trajectories of two humans navigating a traffic weaving scenario were collected using a driving simulator [1] shown in Figure 2.5 (left). 1105 human-human driving interaction trials were recorded over 19 different pairs of people (16 unique individuals, all graduate students with driving experience in the United States). The drivers were instructed to swap lanes with each other (without verbal communication) within 135 meters of straight road. Each human trajectory in each trial exhibits interaction behavior to be learned, effectively doubling the data set. Furthermore, since we are conditioning on interaction history and potential future, cumulatively speaking we have roughly 35 histories per trial: each trial is approximately five seconds long, equating to $T \approx 50$ with 0.1s time steps, and taking the prediction horizon of $N = 15$ into account. Hence in total, our dataset contains roughly 77,000 $x = (x_{HR}^{0:t}, u_{HR}^{0:t}, u_{HR}^{t+1:t+N})$, to $y = u_{HR}^{t+1:t+N}$ exemplars. We note that owing to the interactive nature of this scenario we elected to collect our own dataset using the simulator rather than use existing real-world data (e.g., [38, 39, 40]). Fitting the parameters of our model requires a high volume of traffic weaving interaction exemplars which these open datasets do not encompass, but which we believe a targeted industrial effort might easily procure. Each scenario begins with initial conditions (IC) drawn randomly as follows: car 1 (left or right lane) starts with speed $v_1 = 29$m/s (65mph) and car 2 at a speed difference of $\Delta v = v_1 - v_2 \in \{0, \pm 2\}$ m/s ($\pm 4.5$ mph). The faster car starts a distance $|\Delta v| t_{co}$ behind the other car where $t_{co} \in \{1, 2, 3\}$ represents a “crossover time” when the cars would be side-by-side if neither accelerates or decelerates. The ICs were designed to make ambiguous which car should pass in front of the other, to encourage multimodality in action sequences and outcomes that may occur. This ambiguity is indicated by

![Image of driving simulation setup](image_url)
Relative speed $\Delta v = v_{\text{left}} - v_{\text{right}}$

<table>
<thead>
<tr>
<th>$t_{\text{co}}$</th>
<th>-2m/s</th>
<th>0m/s</th>
<th>2m/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>1s</td>
<td>7.2%</td>
<td>85.7%</td>
<td></td>
</tr>
<tr>
<td>2s</td>
<td>13.1%</td>
<td>76.2%</td>
<td></td>
</tr>
<tr>
<td>3s</td>
<td>23.6%</td>
<td>66.7%</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: Fraction of human-human trials where the car starting in left lane passes in front.

Figure 2.6: Predictions (50 samples) from three different models of human future longitudinal acceleration actions conditioned on the available interaction history (solid lines) and future robot actions (dashed blue line). The dashed red line represents the true future action sequence taken by the human. The line color of the prediction corresponds to different $z$ values, if applicable. Eval NLL denotes negative log-likelihood prediction loss over a fixed validation set of interaction trials (lower is better). Not Pictured: basic LSTM prediction model (no multimodality, with $N_z = K_z = M_{GMM} = 1$, Eval NLL = -9.70).

Table 2.1 whereas $t_{\text{co}}$ increases, the car moving faster but starting further behind, which usually passes in front of the other car, becomes less likely to cut in front. Reckless and irresponsible “video game” driving was discouraged by having a speedometer displayed on-screen with engine hum sound feedback to reflect current speed and a high-pitched alert sound when speed exceeded 38m/s (85mph).

2.6.2 Generative Model for Human Driver Actions

We implemented the neural network CVAE architecture discussed in Section 2.4.2 in Tensorflow and fit human driver action distributions over a 1.5s time horizon conditioned on all available history up to prediction time, as well as the next 1.5s of the other driver’s trajectory (i.e., a robot’s candidate plan). Figure 2.6 illustrates why we chose to design these action distributions as mixtures indexed by a discrete latent variable $z$, where the component distributions are a combination of recurrent hidden state propagation and GMM sampling. All three models capture the same broad prediction: eventually, the human will cease accelerating. Comparing the left plot to the middle plot, the CVAE addition of a latent $z$ manifests as modes predicting cessation on a few different time scales; this unsupervised clustering aids interpretability and slightly improves the performance metric of validation set negative log-likelihood. Comparing the right plot, of essentially a mixture
of basic LSTM models, to the rest, we see that neglecting multimodality on the time step to time step scale prevents the prediction of sharp behavior (e.g., a human driver’s foot quickly lifted off of the throttle).

2.6.3 Robot Policy Construction

We consider a discrete set of robot candidate futures over the 1.5s prediction horizon. We target a replanning rate of 0.3s and break the prediction horizon up into five 0.3s fixed action windows within which the robot may choose one of four longitudinal actions, \( \dot{s}_R \in \{0, 4, -3, -6\} \text{m/s}^2 \), and one of two lateral actions, moving towards either the left lane or the right lane. Specifically, the robot control \( \dot{\tau}_R \) is selected as the first 0.3s of the optimal control for the two point boundary value problem steering from \((\tau_0, \dot{\tau}_0, \ddot{\tau}_0)\), at the start of the window, to \((\tau_{\text{target}}, 0, 0)\), at some free final time \( t_f \) after the start of the window, with cost objective \( \int_{t_0}^{t_f} 1 + \frac{\dot{\tau}_R^2}{1000} \, dt \). In total the robot has 8 possible actions per time window; given that actions in the first window are assumed fixed from the previous planning iteration this results in \( 8^4 = 4096 \) possible action sequences the robot may select from. These 4096 sequences are visualized in \((s, \tau)\) coordinates for a few initial robot states with different headings in Figure 2.7.

![Figure 2.7: 4096 candidate robot action sequences (dependent on current robot state and action in progress) scored each planning loop. Lane boundaries are marked as dotted horizontal lines; the visible branching corresponds to the robot’s decision to target a particular lane during each time window.](image)

The robot’s running cost consists of four terms \( J(x_{HR}^{t+i}, u_R^{t+i}, x_{HR}^{t+i+1}) = J_c + J_a + J_l + J_d \) corresponding to collision avoidance, control effort, lane change incentive, and longitudinal disambiguation incentive defined as:

\[
J_c = 1000 \cdot 1_{\{(|\Delta s|<8\land|\Delta \tau|<2)\}} \cdot (9.25 - \sqrt{\Delta s^2 + \Delta \tau^2})
\]

\[
J_a = \dot{s}_R^2
\]

\[
J_l = -500 \cdot \min(1.5 + s_R/150, 1) \cdot |\tau_R - \tau_{\text{goal}}|
\]

\[
J_d = -100 \cdot \min(\max(\Delta s \Delta \dot{s}, 0), 1)
\]
where we have omitted a time index \((t + i)\) from all state/action quantities, and \(\Delta s = s_R - s_H\), \(\Delta \tau = \tau_R - \tau_H\), \(\Delta \dot{s} = \dot{s}_R - \dot{s}_H\). Briefly, \(J_c\) is a radial penalty past a near-collision threshold, \(J_a\) is a quadratic control cost on longitudinal acceleration (lateral motion is free), \(J_l\) is proportional to distance from target lane centerline \((\tau_{goal})\) with an urgency weight that increases as the robot nears the end of the road at \(s = 0\), and \(J_d\) incentivizes reaching states where \(\Delta s\) is the same sign as \(\Delta \dot{s}\), i.e., the two cars are moving apart from each other longitudinally. The constants in these cost terms were tuned by hand to achieve a qualitatively “neutral” behavior in our human trials; alternative choices such as fitting these robot policy parameters to imitate the driving style of a human expert would also work within our model-based approach. We use a discount factor \(\gamma = 0.9\).

We draw 16 samples of human responses to each of the 4096 candidate robot action sequences and average their respective costs to approximate the expectation in problem (2.3). We select the top 32 sequences by this metric for further analysis, scoring 1024 sampled human trajectories each to gain a confident estimate (up to the model fidelity) of the true cost of pursuing that robot action sequence. This two-stage sampling and scoring process for our prediction model takes \(\sim 0.25s\) parallelized on a single GTX 1080 Ti, simulating nearly 100 000 human responses in total. The robot action sequence with the lowest expected cost from the second stage is selected for enactment over the next action window. In particular, the second action window of the sequence is propagated next (as the first was already being propagated as the policy computation was running) and becomes the fixed first action window of the next search iteration.

### 2.6.4 Human-in-the-loop trials

This robot policy was integrated with the simulator, Figure 2.5, enabling real-time human-in-the-loop validation of the robot action sequences it selects. As in the human-human input dataset, both parties intend to switch lanes. Though, as noted in Section 2.3.2, the desired behavior is qualitative and near-impossible to reliably quantify (especially human-in-the-loop), we highlight here some interesting emergent behaviors in robot reasoning.

Figure 2.8 illustrates an example of the robot’s decision-making at a single time step early in the interaction. The robot is aware of multiple possible actions the human might take, and even aware of how to elicit specific interaction modes, but chooses to wait in accordance with the sequence that minimizes its expected cost (2.3).

Figure 2.9 illustrates two examples where both the human and robot tried to be proactive in cutting in front of the other. When the human does not apply control actions early on, the robot nudges towards the lane divider expecting the human to yield, showcased in the left figures. The trial in the right figures starts much the same, but the human’s continued acceleration causes the robot to change its mind and brake to let the human pass. We note that the robot’s cost function contains no explicit collaboration term, meaning it is essentially fending for itself while reasoning about relative likelihoods from its human action model. This is not unlike many drivers on the
2.7. SUMMARY

Figure 2.8: Example of considerations the robot makes when choosing the best candidate action sequence. Left: the best-scoring sequence is the robot first waiting then planning to eventually accelerate since it has time to wait and see if the human will accelerate or slow down. Center: a candidate action sequence showing a braking maneuver that predicts the human likely to accelerate (to pass in front) in response. Right: a candidate robot future action sequence showing a hard acceleration maneuver which predicts the human likely to decelerate (to let the robot pass) in response. However, these sequences (center and right) incur a higher control cost penalty $J_s$ than waiting (left).

road today, but we note that our framework accommodates adjusting the robot cost but keeping the human model the same—friendlier behavior may be achieved through adding or changing cost terms.

2.7 Summary

We have presented a robot policy construction framework for HRI that takes as input (i) a dataset of human-human interaction trials to learn an explicit, sampleable representation of human response behavior, and (ii) a cost function defined over a planning horizon, so that the desired robot behavior within the interaction model may be achieved through exhaustive action sequence evaluation applied in a receding horizon fashion. This framework makes no assumptions on human motivations nor does it rely on reasoning methods or features designed specifically for the traffic weaving scenario (other than the robot cost objective); it learns relative likelihoods of future human actions and responses at each time step from the raw state and action dataset. As such the robot is essentially blind to what it has not seen in the data—this framework is designed for probabilistic reasoning over relatively short time horizons in nominal operating conditions, but an important next step is to integrate it with lower-level emergency collision avoidance routines and higher-level inference algorithms, e.g., what if the human is not explicit in signaling its intent to change lanes?

In the next chapter, we move from human-in-the-loop experiments in a simulator to a full-scale experimental platform and develop a lower-level emergency safety controller that steps in when a robot interaction planner, such as the one presented in this chapter, is “surprised”, i.e., is faced in an unexpected situation.
Figure 2.9: Interaction action sequences and $(s, \tau)$-trajectories of the robot (blue) and a human driver (red) who tries to cut in front. Top Left: robot chooses to accelerate, causing the human to brake to change lanes near the end of the road. Top Right: both the human and robot accelerate at first, but the human is more persistent and cuts off the robot. Bottom: Lateral and longitudinal positions of the human and robot car, showcasing successful autonomous human-in-the-loop traffic-weaving maneuvers.
Chapter 3

Infusing Reachability-based Safety Assurance within Planning Frameworks

In the previous chapter, we developed a multimodal probabilistic model-based interaction planner that was informed by a data-driven human behavior prediction model. While the data-driven element of the planner provides the planner with capabilities to reason about complex and uncertain interactions, it is also limited by what the data does not encompass. In this chapter, we investigate how probabilistic interaction planners, such as the one studied in Chapter 2, would benefit from a low-level safety controller that would step in whenever the robot is faced with an unsafe situation.

3.1 Introduction

Decision-making and control for mobile robots are typically stratified into levels. A high-level planner, informed by representative yet simplified dynamics of a robot and its environment, might be responsible for selecting an optimal, yet coarse trajectory plan, which is then implemented through a low-level controller that respects more accurate models of the robot’s dynamics and control constraints. While additional components may be required to flesh out a robot’s full control stack from model to motor commands, selecting the right “division of responsibilities” is fundamental to system design.

One consideration that defies clear classification, however, is how to ensure a mobile robot’s safety when operating in close proximity with a rapidly evolving and stochastic environment. Safety is a function of uncertainty in both the robot’s dynamics and those of its surroundings; high-level
CHAPTER 3. REACHABILITY-BASED SAFETY ASSURANCE

planners typically do not replan sufficiently rapidly to ensure split-second reactivity to threats, yet
low-level controllers are typically too short-sighted to ensure safety beyond their local horizon.

Human-robot interactions are an unavoidable aspect of many modern robotic applications and
ensuring safety for these interactions is critical, especially in applications such as autonomous driving
where collisions may lead to life-threatening injury. However, ensuring safety within the planning
and control framework can be very challenging due to the uncertainty in how humans may behave.
To quantify this uncertainty, robots often rely on generative models of human behavior to inform
their planning algorithms [7, 41], thereby enabling more efficient and communicative interactions.
In general, under nominal operating conditions that reflect the modeling assumptions, these model-
based probabilistic planners can offer high performance (e.g., minimizing time and control effort
for the robot). However, these planners alone are typically insufficient for ensuring absolute safety
because (i) they depend on probabilistic models of human behavior and thus safety is not enforced
deterministically, (ii) dangerous but low-likelihood events may not be adequately captured in the
human behavior prediction model, and (iii) reasoning about these probabilistic behavior models is
typically too computationally expensive for the planners to react in real time when humans strongly
defy expectations and/or diverge from modeling assumptions.

In this work, we implement a control stack for a full-scale autonomous car (the “robot”) engaging
in close proximity interactions with a human-controlled vehicle (the “human”). Our control stack
aims to stay true to planned trajectories from a high-level planner since freedom of motion is essential
for the planner to carry out the driving task while conveying future intent to the other vehicle. At the
same time, we allow the robot to deviate from the desired trajectory to the point that is necessary to
maintain safety. Our primary tool for designing a controller that does not needlessly impinge upon
the planner’s choices is Hamilton-Jacobi (HJ) backward reachability. We provide a brief overview of
the reachability analysis literature relevant to our work in Section 3.4.

3.2 Related Work

For high-level planners, safety is often incentivized, but not strictly enforced as a hard constraint. For
example, safety is often part of the objective function when selecting optimal plans, or represented
via artificial potential fields [18, 7, 41]. Although these approaches are designed to account for
interactive scenarios in which another sentient agent is a key environmental consideration, they
contain competing objectives (i.e., collision avoidance vs. goal-oriented performance), often do not
plan at a sufficiently high rate to account for rapidly-changing environments, and ultimately provides
no theoretical framework for ensuring safety for both the human and the robot.

Another common approach to finding collision-free plans is to use forward reachability. The
idea is to fix a time horizon and compute the set of states where the other agents could possibly
be in the future and plan trajectories for the robot that avoid this set [42, 43, 44]. Although this
gives a stronger sense of safety, this is only practical for short time horizons otherwise it will lead to overly conservative robot behaviors or even planning problem infeasibility as the set to avoid grows. Consequently, forward reachability has limited usage in interactive scenarios where freedom of motion (e.g., nudging) is necessary to convey intent. Additionally, forward reachability-based methods, as well as methods that enforce safety as an objective, are often subjected to model simplification for computational tractability reasons. Such modeling simplifications can lead to overly conservative or imprecise results which may impede performance.

Safety can also be introduced at the control level which often obeys higher fidelity dynamics models than the planner. A common approach to ensuring safe low-level controls is to use reactive collision avoidance techniques—the robot is normally allowed to apply any control, but switches to an avoidance controller when near safety violation. Examples of this general approach include HJ backward reachability-based controllers which have been applied in such a switched fashion [45, 46, 47] and have proven to be effective for avoiding other interacting agents and static obstacles. HJ reachability has been studied extensively and applied successfully in a variety of safety-critical interactive settings [48, 49, 50, 51, 52, 53] due to its flexibility with respect to system dynamics, and its optimal (i.e., non-overly-conservative) avoidance maneuvers stemming from its equivalence to an exhaustive search over joint system dynamics. Other approaches include using a precomputed emergency maneuver library [54]. A key drawback to switching controllers, however, is that the performance goals considered by the high-level planner are completely ignored when the reactive controller steps in. For some cases, this may be acceptable and necessary, but in general, it is desirable to ensure safety without unduly impacting planner performance where possible. Instead of switching to a different controller, [17, 55, and 56] adapt the low-level online optimal controller to incorporate constraints that avoid static obstacles. The approach taken in [55] aims to be minimally interventional, however, they only consider cases with disturbance uncertainty rather than environmental uncertainty stemming from the system interacting with other human agents. These approaches are able to strike the optimal balance between tracking performance and safety (i.e., optimizing performance subject to safety as a hard constraint), but these approaches as presented are effective for avoiding static obstacles only.

Inspired by the non-conservative nature and optimality of HJ reachability, and the performance of online optimal control for trajectory tracking, this work aims to infuse reachability-based safety assurance into the low-level controller such that when the robot is near safety violation, the robot can simultaneously maintain safety and follow the high-level plans without unduly impacting performance. To the best of our knowledge, there has not been any work that explicitly addresses the integration of reachability-based safety controllers as a component within a robot’s control stack, i.e., with safety as a constraint upon a primary planning objective.
3.3 Contributions

The core contributions discussed in this chapter are twofold. First, we propose a method for formally incorporating reachability-based safety within an existing optimization-based control framework. The main insight that enables our approach is the recognition that, near safety violation, the set of safety-preserving controls often contains more than just the optimal avoidance control. Instead of directly applying this optimal avoidance control when prompted by reachability considerations, as in a switching control approach, we quantify the set of safety-preserving controls and pass it to the broader control framework as a constraint. Our intent is to enable minimal intervention against the direction of a higher-level planner when evasive action is required. Second, we evaluate the benefits, performance, and trade-offs of this safe control methodology in the context of a probabilistic planning framework for the traffic weaving scenario studied at a high level in Chapter 2 wherein two cars, initially side-by-side, must swap lanes in a limited amount of time and distance. Experiments with a full-scale steer-by-wire vehicle reveal that our combined control stack achieves better safety than applying a tracking controller alone to the planner output, and smoother operation (with similar safety) compared with a switching control scheme; in our discussion, we provide a roadmap towards improving the level of safety assurance in the face of practical considerations such as unmodeled dynamics, as well as towards generalizations of the basic traffic weaving scenario.

3.4 Hamilton-Jacobi Backward Reachability Analysis

Given a dynamics model governing a robotic system incorporating control and disturbance inputs, reachability analysis is the study of the set of states that the system can reach from its initial conditions (forward reachability) or the set of states it should be to reach a target set (backward
reachability). It is often used for formal verification as it can give guarantees on whether or not the evolution of the system will be safe, i.e., whether the reachable set includes undesirable outcomes. Reachability analysis can be divided into two main paradigms: (i) forward reachability and (ii) backward reachability.

### 3.4.1 Forward Reachability Analysis

The *Forward Reachable Set (FRS)* is the set of states that the system could potentially be in after some time horizon $t$. This is computed by propagating the dynamics combined with all feasible control sequences and disturbances forward in time. When considering the interaction between two agents, for example, a human and a robot, the forward reachable set is computed for the human and the robot plans to avoid this set to ensure collision-free trajectories (see Figure 3.1a). This open-loop mentality, however, leads to an overly conservative robot outlook. That is, while considering controls in the present, the robot does not incorporate the possibility that its future observations of where the human goes might influence how much it needs to take avoidance controls; in short, the robot’s plan to avoid the FRS does not incorporate closed-loop feedback. To reduce the over-conservative nature of forward reachability, the time horizon $t$ for which the FRS is computed over is typically kept small and is recomputed frequently. This approach has been found to be effective in finding collision-free trajectories [57], but it is difficult to extend to interactive scenarios where there are more uncertainties in the rapidly changing environment. Aside from the overly conservative nature of FRS, the key drawback with using forward reachability is that safety (i.e., avoiding the FRS) hinges on the planner’s capabilities and operating frequency. Even if computing the FRS is instantaneous, the planner may still be unable to react to split-second threats.

### 3.4.2 Backward Reachability Analysis

Let the target set represent a set of undesirable states (e.g., collision states of the system). The *Backward Reachable Set (BRS)* is the set of states that could result in the system being in the target set, assuming worst-case disturbances, after some time horizon $t$. Specifically, the BRS represents the set of states from which there does not exist a controller that can prevent the robot dynamics from being driven into the target set under worst-case disturbances within a time horizon $t$. As such, to rule out such an eventuality, the BRS is treated as the “avoid set”. Critical differences between the FRS and BRS are that (i) the BRS is computed backward in time, and (ii) the BRS is computed assuming closed-loop reactions to worst-case disturbances. Illustrated in Figure 3.1b for the case of relative dynamics between human- and robot-controlled vehicles, computation of the BRS takes into account the fact that the robot can react to the human at any time and in any state configuration (if the human was to swerve into the robot, the robot can swerve too to avoid a collision). This leads to the BRS being less overly conservative than the FRS, but still conservative in the sense that the BRS is calculated assuming the worst-case (bounded) disturbance will occur. In practice, the
results of the BRS computation are cached via a look-up table, and at run-time, the optimal robot policy can be computed via a near-instant lookup of the reachability cache. As such, we can always compute optimal controls for the robot at any state configuration and regardless of the high-level planner used.

We elect to use backward reachability because:

1. The less overly conservative nature of BRS (compared to FRS) stemming from the closed-loop computation ensures that safe controls will be used only when necessary and not unduly impact planner performance.

2. Safety is defined intrinsically in the BRS computation whereas safety using FRS-based approaches depends on whether the robot’s planned trajectory intersects the FRS.

3. BRS provides a computational handle on safe controls which can be evaluated at high operating frequencies to react to split-second threats.

Moreover, we will compute the BRS using HJ reachability analysis, a particular approach to computing reachable sets. There are many existing approaches\cite{58,59,60,61,62}, but there is always a trade-off between modeling assumptions, scalability, and representation fidelity (i.e., whether the method computes over- or under-approximations of the reachable set). Compared to alternatives approaches, HJ reachability is the most computationally expensive, but it is able to compute the BRS exactly\footnote{With precision dependent on parameters of the numerical solver, e.g., discretization choices in mesh size/time step.} for any general nonlinear dynamics with control and disturbance inputs because it essentially uses a brute force computation via dynamic programming. As a result, the BRS solution to the HJ reachability problem represents constructive proof of the existence of a safety-preserving control policy (i.e., a safety certificate) for states outside the BRS. Despite the apparent computational drawbacks, we note that the BRS may be computed offline, and requires only a near-instant lookup during runtime, allowing it to be used in controllers or planners that run at a very high operational frequency.

### 3.4.3 Hamilton-Jacobi Reachability Analysis

We briefly review relevant HJ backward reachability definitions for the remainder of this section; see\cite{46} for a more in-depth treatment. HJ reachability casts the reachability problem as an optimal control problem and the reachable set can be computed by solving the Hamilton-Jacobi-Isaacs (HJI) partial differential equation (PDE).

The general HJ reachability formulation is as follows. Let the system dynamics be given by $\dot{x} = f(x, u, d)$ where $x \in \mathbb{R}^n$ is the state, $u \in \mathcal{U} \subset \mathbb{R}^m$ is the control, and $d \in \mathcal{D} \subset \mathbb{R}^p$ is the disturbance. For example, $x$ could be the state of a robot, $u$ be the robot’s controls, $d$ be...
environmental disturbances such as wind, and $f$ be the dynamics of the robot. The system dynamics $f: \mathbb{R}^n \times U \times D \to \mathbb{R}^n$ are assumed to be uniformly continuous, bounded, and Lipschitz continuous in $x$ for a fixed $u$ and $d$. Let $T \subseteq \mathbb{R}^n$ be the target set that the system wants to avoid at the end of a time horizon $|t|$ (note that $t < 0$ when propagating backward in time). For collision avoidance, $T$ typically represents the set of states that are in collision with an obstacle.

For brevity, the following description of HJ backward reachability will use notation relevant to the rest of the chapter, that is, we are interested in collision avoidance for human-robot interactions. Despite the notation, the following theory is still applicable to general systems $\dot{x} = f(x, u, d)$ outside the domain of human-robot interactions.

In the context of human-robot interactions, $f(x, u, d)$ describes the relative dynamics between the human and the robot where the human is treated as a “disturbance”. More concretely, let $(x_R, u_R)$ represent the robot state and control, $(x_H, u_H)$ represent the human state and control, and $x_{rel}$ be the relative state between the human and the robot. Thus the relative dynamics of the robot and human are given by $\dot{x}_{rel} = f_{rel}(x_{rel}, u_R, u_H)$, and $T$ represents the set of relative states corresponding to when the human and robot are in collision. The formal definition of the BRS, denoted by $\bar{A}(t)$, for the human-robot relative system is

$$\bar{A}(t) := \{ \bar{x}_{rel} \in \mathbb{R}^n \mid \exists u_H(\cdot), \forall u_R(\cdot), x_{rel}(t) = \bar{x}_{rel} \land \dot{x}_{rel} = f_{rel}(x_{rel}, u_R, u_H) \land x_{rel}(0) \in T\}. \quad (3.1)$$

The BRS $\bar{A}$ represents the set of states such that if the human followed an adversarial policy $u_H(\cdot)$ and the robot execute any policy $u_R(\cdot)$, including the optimal collision-avoiding policy, then the relative state trajectory $x_{rel}(\cdot)$ will be inside $T$ in exactly $|t|$ seconds. A natural variant on the BRS is the backward reachable tube (BRT), which is defined similarly but instead considers the case when the relative state trajectory $x_{rel}(\cdot)$ will be inside $T$ within $|t|$ seconds. The formal definition is as follows:

$$A(t) := \{ \bar{x}_{rel} \in \mathbb{R}^n \mid \exists u_H(\cdot), \forall u_R(\cdot), \exists s \in [t, 0], x_{rel}(t) = \bar{x}_{rel} \land \dot{x}_{rel} = f_{rel}(x_{rel}, u_R, u_H) \land x_{rel}(s) \in T\}. \quad (3.2)$$

Since we are concerned with a collision occurring at any time, not just at the end of the time horizon, we will be using the BRT from here on, and will also refer to the BRT as the “avoid set”. Assuming optimal (i.e., adversarial) human controls, and optimal robot controls, $A(t)$ can be computed by solving the HJI variational inequality,

$$\frac{\partial V(t, x_{rel}(t))}{\partial t} + \min \left\{ 0, \max_{u_R \in U_R} \min_{u_H \in U_H} \nabla V(t, x_{rel}(t))^T f_{rel}(x_{rel}(t), u_R, u_H) \right\} = 0 \quad (3.3)$$

where the solution, called the value function $V(t, x_{rel})$, gives the BRT as its zero sub-level set,

$$A(t) = \{ x_{rel} : V(t, x_{rel}) \leq 0 \}. \quad (3.4)$$
The target set \( T \) is the set of relative states that we want the relative system to avoid. Typically, for collision avoidance problems, \( T \) represents the set of relative states where the human and robot are in collision. Using the same zero sub-level set representation as the value function, the target set can be represented similarly,

\[
T = \{ x_{rel} : \ell(x_{rel}) \leq 0 \}.
\]

The function \( \ell \) is used as a boundary condition for (3.3) and often the signed distance function is used due to its simple geometrical interpretation. The signed distance function outputs negative values when the two agents are in collision, positive values when the two agents are apart, and the magnitude corresponds to the amount of penetration/separation between the two agents. Assuming \( \ell \) is computed using the signed distance function, \( \ell(x_{rel}) = -0.1 \) indicates that \( x_{rel} \) corresponds to a relative state where the two agents are in collision with a penetration distance of 0.1 meters. Similarly, \( \ell(x_{rel}) = 0.1 \) corresponds to a relative state where the two agents are 0.1 meters apart.

Accordingly, the value function, the solution to (3.3), has the following interpretation: \( V(t, x_{rel}) \) represents the lowest signed distance that the relative state trajectory will end up in over \(|t|\) seconds if the human followed an adversarial policy and the robot followed an optimal policy to evade the human. For instance, \( V(t, x_{rel}) = -0.1 \) means that if the human followed an adversarial policy and the robot did its best to avoid a collision, then 0.1 meters of penetration between the two agents will inevitably occur. In short, given that \( \ell \) is defined as separation distance, the value function can be viewed as a measure of how far away from a collision (i.e., safety measure) the future is (over \(|t|\) seconds) under the worst-case controls by the human agent. A simple illustrative example that visualizes what the target set and BRT is given in Example 1.

**Example 1.** Consider two agents, agent A and agent B, each obeying the following dynamics with \( v_A = v_B = 5 \text{m/s} \):

\[
\begin{bmatrix}
\dot{p}_{x,A} \\
\dot{p}_{y,A} \\
\dot{\psi}_A
\end{bmatrix} =
\begin{bmatrix}
v_A \cos(\psi_A) \\
v_A \sin(\psi_A) \\
\omega_A
\end{bmatrix},
\begin{bmatrix}
\dot{p}_{x,B} \\
\dot{p}_{y,B} \\
\dot{\psi}_B
\end{bmatrix} =
\begin{bmatrix}
v_B \cos(\psi_B) \\
v_B \sin(\psi_B) \\
\omega_B
\end{bmatrix}.
\]

The relative dynamics which transforms agent B into agent A’s reference frame can be written as follows:

\[
\begin{bmatrix}
\dot{p}_{x,rel} \\
\dot{p}_{y,rel} \\
\dot{\psi}_{rel}
\end{bmatrix} =
\begin{bmatrix}
\sin(\psi_A) & \cos(\psi_A) \\
\cos(\psi_A) & -\sin(\psi_A)
\end{bmatrix}
\begin{bmatrix}
p_{x,B} - p_{x,A} \\
p_{y,B} - p_{y,A}
\end{bmatrix}, \Rightarrow
\begin{bmatrix}
\dot{p}_{x,rel} \\
\dot{p}_{y,rel} \\
\dot{\psi}_{rel}
\end{bmatrix} =
\begin{bmatrix}
-v_A + v_B \cos(\psi_{rel}) + p_{y,rel}\omega_A \\
v_B \sin(\psi_{rel}) - p_{x,rel}\omega_B \\
\omega_B - \omega_A
\end{bmatrix}.
\]

Given the relative dynamics between agent A and agent B, we interpret agent A’s controls as the “control input” and agent B’s controls as the “disturbance input”. As such, the HJ reachability formulation computes the BRT and optimal control for agent A to guard against all possible, including
worst-case/collision-seeking, controls that agent B can take. Let us consider an unsafe state to be states where the distance between agent A and agent B is less than 5 meters. Illustrations of slices of the target set and the BRT computed 2.8 seconds backward in time are given in Figure 3.2. With $\psi_{rel} = 144^\circ$, it is more dangerous for agent B to be situated in front of agent A since this corresponds to agent B moving almost head-on towards agent A.

![Slice of the target set](image1.png)

(a) A slice of the target set $T$ computed using signed distance given the relative dynamics given in Example 1. The zero sub-level set corresponds to the states where the distance between the two agents is less than 5 meters.

![Slice of the BRT](image2.png)

(b) A slice of the BRT ($\psi_{rel} = 144^\circ$) with a time horizon $t = -2.8$ seconds.

Figure 3.2: Slices of the target set and BRT of a simple example system described in Example 1.

For the case of the vehicle-vehicle interactions investigated in this work, when the control and maximum velocity capabilities of the human car are no greater than those of the robot car, one can take the limit $t \to -\infty$ and obtain the infinite time horizon BRT $A_\infty$ with corresponding value function $V_\infty(x_{rel})$.

Intuitively, this prescribed parity in control authority ensures that if the human and robot start sufficiently far apart, then the human will never be able to “catch” the robot. This holds even if the human car may have transient maneuverability advantages over the robot as we assume later in this work. That is, we expect that the BRT will not encompass the entire state space as $t \to -\infty$, and in practice, we compute the BRT over a sufficiently large finite time horizon to the point where it appears that the BRT has converged. We recognize that we make a strong assumption on the human and robot’s control authorities in enabling this computation. In reality, the human and robot car may have very different control authorities (e.g., different engines resulting in different acceleration and velocity capabilities) and the infinite BRT may not be bounded. In such cases, practitioners may compute the BRT over a horizon suitable for the interaction where guarantees afforded by HJ reachability only hold over that time horizon.

---

2For ease of notation going forward we will often write $V := V_\infty$. 
3.4.4 Optimal HJI Collision Avoidance Control

After solving for the value function (and its gradients) by solving the HJI variational inequality\(^3\) we can compute the HJI optimal robot collision avoidance control,

\[
u^*_R = \arg \max_{u_R} \min_{u_H} \nabla V(x_{rel})^T f_{rel}(x_{rel}, u_R, u_H)
\]  \hspace{1cm} (3.5)

which offers the greatest increase in \(V(x_{rel})\) assuming optimal (worst-case) actions by the human. For general nonlinear systems, computing the optimal collision avoidance control in (3.5) may be nontrivial as there could be multiple local maxima. For control/disturbance affine systems, however, the solutions to the optimal control/disturbance are bang-bang.

Recall from (3.2) that when the system is outside of the avoid set \(A(t)\) (i.e., \(\{x_{rel} \mid V(t, x_{rel}) > 0\}\)), there exists a robot policy (e.g., (3.5)) that keeps the robot safe over a time horizon \(|t|\) regardless of any (including adversarial) policy taken by the human. Previous applications of HJI for safety-critical tasks switch to the optimal control when near the boundary of the BRT, i.e., when safety is nearly violated \([45, 47]\). This reflects the goal of HJ reachability-based safety which is to ensure that the system always stays outside of the avoid set \(A(t)\), that is, the value function should always stay positive.

However, in an interactive scenario where, for example, we may want to let a robot planner convey intent by nudging towards the human car to an extent that is safe, we prefer a less extreme control strategy. In the next section, we introduce the notion of a safety-preserving control set and describe in detail how to infuse reachability-based safety assurance within a multi-tiered control framework that consists of different planning objectives.

3.4.5 Safety-Preserving HJI Control

The optimal HJI control is just one control the robot could possibly take to increase the value function. Instead, we can consider the set of robot controls that ensures that the value function is nondecreasing over time and therefore is not becoming any less safe than it already is. That is, we define the set of safety-preserving controls,

\[
U^*_R(x_{rel}) = \{u_R : \min_{u_H} \nabla V(x_{rel})^T f_{rel}(x_{rel}, u_R, u_H) \geq 0\},
\]  \hspace{1cm} (3.6)

which represent the set of robot controls that ensure the value function is nondecreasing under the worst-case human controls. We note that since we are using the infinite-horizon BRT, \(\frac{\partial V_\infty(x_{rel})}{\partial t} = 0\). Like the optimal control (3.5), this safety-preserving control set can be computed online. Note that if the system is affine in robot controls, \(U^*_R\) represents a half-plane in the robot’s control space.

\(^3\)There exists toolboxes that solves the HJI PDE, such as \([63, 64, 65]\).
### 3.5 Minimally-Interventional Safety Controller

In this section, we propose using a safety-preserving HJI controller instead of switching to the optimal HJI controller (3.5), and describe how to incorporate it within an existing control stack—a high-level planner feeding desired trajectories to a low-level tracking controller—to enable safe human-robot interactions that minimally impinges on the high-level planning performance objective. The control stack architecture is illustrated in Figure 3.3. The proposed control stack applies to general human-robot interactions (e.g., an autonomous car interacting with a human-driven car, an autonomous manipulator arm working alongside a human, wheeled mobile robots navigating a crowded sidewalk). However, in this work, we focus on the traffic-weaving scenario previously studied in Chapter 2 wherein two cars initially side-by-side must swap lanes in a limited amount of time and distance, because it is a representative interactive scenario that encapsulates many challenging characteristics inherent to human-robot interaction. Successful and smooth traffic-weaves rely on action anticipation, intent prediction, and proactive behavior from each vehicle, and ensuring safety is critical because collisions may lead to life-threatening injury.
3.5.1 Interaction Planner

Our proposed control framework is agnostic to the interaction planner used. The only assumption for the planner is that it outputs desired trajectories (which presumably reflect high-performance goal-oriented nominal behavior) for the robot car to track. In general, high-level planners often optimize objectives that weigh safety considerations (e.g., distance between cars) relative to other concerns (e.g., control effort), and typical to human-robot interactive scenarios, they may reason anticipatively with respect to a probabilistic interaction dynamics model. That is, although the planner is encouraged to select safer plans, safety is not enforced as a deterministic constraint at the planning level.

In this work, we use the traffic-weaving interaction planner studied in Chapter 2. It uses a predictive model of future human behavior to select desired trajectories for the robot car to follow, updated at \( \sim 3\text{Hz} \). We slightly modify this work by using a hindsight optimization policy \([66]\) instead of the limited-lookahead action policy to encourage more information-seeking actions from the robot.

3.5.2 Tracking Controller

Given a desired trajectory from the interaction planner, the tracking controller computes optimal controls to track the desired trajectory. We assume that the outputs of the low-level tracking controller are directly usable by the robot’s actuators (e.g., steering and longitudinal force commands for a vehicle, torque commands for each joint on a manipulator arm). As such, the tracking controller typically uses a more accurate dynamics model and operates at a much higher frequency (\( \sim 100\text{Hz} \)) than the interaction planner, often at the expense of environmental awareness. To ensure safety with respect to a dynamic obstacle (i.e., a human-driven vehicle) we incorporate additional constraints computed from HJ reachability analysis into the tracking optimization problem. These constraints are designed to ensure that at each control step the robot car does not enter an unsafe set of relative states that may lead to inevitable collision.

In this work, we are concerned with vehicle trajectory tracking. We adapt the real-time MPC tracking controller from \([56]\) by modifying it to include an additional invariant set constraint derived from HJ reachability theory. This MPC tracking controller, operating at 100Hz, computes optimal controls to track a desired trajectory by solving an optimization problem at each iteration. The optimization problem is based on a single-track vehicle model (also known as the bicycle model) and incorporates friction and stability control constraints (in addition to control and state constraints) while minimizing a combination of tracking error and control derivatives. A more in-depth treatment of this combined MPC and HJ reachability controller is given in Section 3.7.
3.5. MINIMALLY-INTERVENTIONAL SAFETY CONTROLLER

3.5.3 Safety-Preserving HJI Control Constraint

Rather than switching to the optimal avoidance controller defined in (3.5) when nearing safety violation (i.e., $V(x_{rel})$ is nearing zero), we propose adding containment in the set of safety-preserving controls, defined in (3.6), as an additional constraint to the low-level MPC tracking controller. Online we employ a safety buffer $\epsilon > 0$ so that when the condition $V(x_{rel}) \leq \epsilon$ holds, indicating that the robot is nearing safety violation, we add the constraint $u_R \in \mathcal{U}^*_R(x_{rel})$ to the list of tracking controller constraints.

By adding this additional safety-preserving constraint when near safety violation, the MPC tracking controller selects control actions that prevent the robot from further violating the safety threshold while simultaneously optimizing for tracking performance and obeying additional constraints. We argue that this results in a \textit{minimally interventional safety controller}—the MPC tracking controller will only minimally deviate from the desired trajectory to the extent necessary to maintain safety for the robot car. In contrast, \textit{switching} to the optimal HJ collision avoidance control is considered extremely invasive as it overrides the planner’s output and therefore ignoring any high-level performance goals that the robot strives to achieve. For instance, the hard switch to the optimal collision avoidance control may result in an autonomous vehicle slamming on the brakes in the middle of a highway which would be extremely uncomfortable for passengers whereas a minimally interventional control, as we will show, may choose to lightly brake and swerve instead.

For the traffic-weaving scenario investigated in this work, the MPC problem for vehicle trajectory tracking \cite{56} is formulated as a quadratic program (QP) (to enable fast solve time amenable to a 100Hz operating frequency) and hence requires the constraints to be linear. As such, we instead apply the constraint $u_R \in \tilde{\mathcal{U}}^*_R(x_{rel})$ where

$$\tilde{\mathcal{U}}^*_R(x_{rel}) = \{ u_R : M^T_{HJI} u_R + b_{HJI} \geq 0 \}$$

(3.7)

is a linearized approximation of $\mathcal{U}^*_R(x_{rel})$. Specifically, for the current relative state $\tilde{x}_{rel}$, current robot control $\tilde{u}_R$, and optimal, i.e., worst-case, human action defined analogously to (3.5) $u^*_H$, the terms in the linearization are

$$M_{HJI} = \frac{\partial}{\partial u_R} \left( \nabla V^T f_{rel}(x_{rel}, u_R, u_H) \right) \bigg|_{(\tilde{x}_{rel}, u^*_R, \tilde{u}_R)}, \quad b_{HJI} = \nabla V^T f_{rel}(\tilde{x}_{rel}, \tilde{u}_R, u^*_H) - M^T_{HJI} \tilde{u}_R.$$

In general, $\mathcal{U}^*_R(x_{rel})$ may not be a half-space, leading to the linear approximation $\tilde{\mathcal{U}}^*_R(x_{rel})$ including controls outside of $\mathcal{U}^*_R(x_{rel})$. However, since we bound the change in the control inputs across each time step, feasible controls remain close to the linearization point where the approximation error is small.
CHAPTER 3. REACHABILITY-BASED SAFETY ASSURANCE

Figure 3.4: Schematic of the single track model (bicycle model) tracking a path. This is the dynamics model used to describe the robot vehicle.

3.6 Dynamics: Pairwise Vehicle-Vehicle Interaction

In this section, we detail the vehicle dynamics model used to model the robot and human car, the relative dynamics model between the human and the robot necessary for computing the BRT, and the tracking dynamics used for the MPC tracking controller. We use a six-state nonlinear single track model to describe the robot car’s dynamics and assume the human car obeys a four-state dynamically extended nonlinear unicycle model with longitudinal acceleration and yaw rate as control inputs. As a result, the relative dynamics model has seven states. This represents a compromise between model fidelity and the number of state dimensions in the relative dynamics; increasing the former reduces the amount of model mismatch with the real system while reducing the latter is essential since solving the HJI PDE suffers greatly from the curse of dimensionality. Additionally, we build in an extra layer of conservatism by modeling the human agent to be more agile than it truly is in real life. The computation becomes notoriously expensive past five or more state dimensions without compromising grid discretization or employing some decoupling strategy.

3.6.1 Robot Vehicle Dynamics

The robot car (denoted by subscript R) will be modeled using the single-track vehicle model illustrated in Figure 3.4. Let \( (p_{x,R}, p_{y,R}) \) be the position of the robot car’s center of mass defined in an inertial reference frame and \( \psi_R \) be the yaw angle (heading) of the robot car relative to the horizontal axis. \( U_{x,R} \) and \( U_{y,R} \) are the velocities in the robot car’s body frame, and \( r_R \) is the yaw rate. The state for the single track model is \( x_R = [p_{x,R} \ p_{y,R} \ \psi_R \ U_{x,R} \ U_{y,R} \ r_R]^T \). The control input \( u_R = [\delta \ F_x]^T \) consists of the steering command \( \delta \) and longitudinal tire force \( F_x \) which is distributed between the front and rear tires \( F_x = F_{x,f} + F_{x,r} \) via a fixed mapping. Assuming a quadratic model of longitudinal drag force \( (F_{x,drag} = -C_{d_0} - C_{d_1} U_{x,R} - C_{d_2} U_{x,R}^2) \) and for vehicle
mass \((m)\) and moment of inertia \((I_{zz})\), and the distances from the center of mass to the front and rear axles \((d_l, d_r)\), the equations of motion for the robot car are

\[
\begin{align*}
\dot{x}_R &= \begin{bmatrix}
\dot{p}_{x,R} \\
\dot{p}_{y,R} \\
\dot{\psi}_R \\
\dot{U}_{x,R} \\
\dot{U}_{y,R} \\
\dot{r}_R
\end{bmatrix} = \begin{bmatrix}
U_{x,R} \cos \psi_R - U_{y,R} \sin \psi_R \\
U_{x,R} \sin \psi_R + U_{y,R} \cos \psi_R \\
\frac{1}{m} (F_{x,f} \cos \delta - F_{y,f} \sin \delta + F_{x,r} + F_{x,\text{drag}}) + r_R U_{y,R} \\
\frac{1}{m} (F_{y,f} \cos \delta + F_{y,r} + F_{x,f} \sin \delta) - r_R U_{x,R} \\
\frac{1}{I_{zz}} (d_l F_{y,f} \cos \delta + d_l F_{x,f} \sin \delta - d_r F_{y,r})
\end{bmatrix}
\end{align*}
\]  

(3.8)

The controls are assumed to be limited by the steering system, friction limits, and power capacity of the vehicle. Using the brush coupled tire model by [67], the lateral tire force \(F_{y,i}\) at the front and rear tires is a function of slip angle \((\alpha_l, \alpha_r)\), tire cornering stiffness \((C_{\alpha_l}, C_{\alpha_r})\), longitudinal tire forces \((F_{x,f}, F_{x,r})\), coefficient of friction \((\mu)\), and normal tire forces \((F_{z,f}, F_{z,r})\). As such, the lateral tire force for either the front or rear tires (denoted by \(i \in \{f, r\}\)) is

\[
F_{y,i} = \begin{cases}
0 & \text{if } F_{x,i} > \mu F_{z,i} \\
-C_{\alpha_i} \tan \alpha_i (1 - \gamma + \frac{1}{2} \gamma^2) & \text{if } \gamma < 1 \\
-F_{y,\text{max}} \tan \alpha_i & \text{if } \gamma \geq 1
\end{cases}
\]

(3.9)

where \(\gamma = \frac{C_{\alpha_i} \tan \alpha_i}{3F_{y,\text{max}}}\) and \(F_{y,\text{max}} = \sqrt{\mu^2 F_{z,i}^2 - F_{x,i}^2}\). The slip angles and normal forces (accounting for weight transfer due to \(F_{z,i}\)) for the front and rear tires can be computed using the following equations,

\[
\begin{align*}
\alpha_l &= \tan^{-1} \left( \frac{U_{y,R} + d_r r_R}{U_{x,R}} \right) - \delta, \\
\alpha_r &= \tan^{-1} \left( \frac{U_{y,R} - d_r r_R}{U_{x,R}} \right), \\
F_{z,f} &= mg d_t - h \tilde{F}_x, \\
F_{z,r} &= mg d_t + h \tilde{F}_x,
\end{align*}
\]

where \(h\) is the distance from the center of mass to the ground, \(L = d_l + d_r\), and \(\tilde{F}_x = F_{x,f} \cos \delta - F_{y,f} \sin \delta + F_{x,r}\) is the total longitudinal force in the vehicle’s body frame.

### 3.6.2 Human Vehicle Dynamics

The human car (denoted by subscript \(H\)) will be modeled using the dynamically extended unicycle model illustrated in Figure 3.5. Let \((p_{x,H}, p_{y,H})\) be the position of the center of the human car’s rear axle defined in an inertial reference frame and \(\psi_H\) be the yaw angle (heading) of the human car relative to the horizontal axis. The velocity of the human car in the vehicle frame is \(v_H\). The state for the dynamically extended unicycle model is \(x_H = \begin{bmatrix} p_{x,H} & p_{y,H} & \psi_H & v_H \end{bmatrix}^T\). The control
CHAPTER 3. REACHABILITY-BASED SAFETY ASSURANCE

Figure 3.5: Schematic of the dynamically-extended unicycle model. This is the dynamics model used to describe the human vehicle.

Input $u_H = [\omega \ a]^T$ consists of the yaw rate $\omega$ and longitudinal acceleration $a$. The control limits of the human car are chosen such that the robot and human car share the same power, steering, and friction limits. The equations of motion for the human car are

$$
\dot{x}_H = \begin{bmatrix}
\dot{p}_{x,H} \\
\dot{p}_{y,H} \\
\dot{\psi}_H \\
\dot{v}_H
\end{bmatrix} = \begin{bmatrix}
v_H \cos \psi_H \\
v_H \sin \psi_H \\
\omega \\
a
\end{bmatrix}.
$$

(3.10)

Due to its simpler dynamics representation, the human car has a transient advantage in control authority over the robot car (it may change its path curvature discontinuously, while the robot may not), but by equating the steady-state control limits we ensure that the infinite time horizon BRT computation converges.

3.6.3 Relative Human-Robot Dynamics

The relative state (denoted by subscript rel) between the robot car and human car is defined with respect to a coordinate system centered on and aligned with the robot car’s vehicle frame. We define the relative position $(p_{x,rel}, p_{y,rel})$ as

$$
\begin{bmatrix}
p_{x,rel} \\
p_{y,rel}
\end{bmatrix} = \begin{bmatrix}
\cos \psi_R & \sin \psi_R \\
-\sin \psi_R & \cos \psi_R
\end{bmatrix} \begin{bmatrix}
p_{x,H} - p_{x,R} \\
p_{y,H} - p_{y,R}
\end{bmatrix},
$$

and the relative heading $\psi_{rel}$ as $\psi_{rel} = \psi_H - \psi_R$.

Since the velocity states are defined with respect to the vehicle frame, we cannot define analogous relative velocity states and must include the individual velocity states of each vehicle. As such, the relative state for the human-robot vehicle system is $x_{rel} = [p_{x,rel} \ p_{y,rel} \ \psi_{rel} \ U_{x,R} \ U_{y,R} \ v_H \ r_R]^T$. In the language of HJ reachability, the disturbance input of the system is the human car’s control $d = u_H = [\omega \ a]^T$ and the control input is the robot car’s control $u = u_R = [\delta \ F_x]^T$. Combining (3.8) and (3.10), the equations of motion for the relative system are
3.7. The MPC+HJI Trajectory Tracking Controller

In this section, we detail the optimization problem used in the MPC+HJI tracking controller, describe how this problem can be adapted to accommodate collision avoidance for static obstacles, and provide numerical details about the BRT computation used in this formulation. Central to an MPC controller is an optimization problem; at each time step, the controller solves an optimization problem to find an optimal control sequence, passes the first control input to the actuator, and then repeats this process. To be amenable to real-time applications, the optimization problem requires

\[ \dot{x}_{\text{rel}} = \begin{bmatrix} \dot{p}_{x,\text{rel}} \\ \dot{q}_{y,\text{rel}} \\ \dot{U}_{x,R} \\ \dot{U}_{y,R} \\ \dot{v}_{H} \\ \dot{r}_{R} \end{bmatrix} = \begin{bmatrix} \psi_{H} \cos \psi_{\text{rel}} - U_{x,R} + p_{y,\text{rel}} r_{R} \\ \psi_{H} \sin \psi_{\text{rel}} - U_{y,R} - p_{x,\text{rel}} r_{R} \\ \frac{1}{m} (F_{x,t} \cos \delta - F_{y,t} \sin \delta + F_{x,r} + F_{x,\text{drag}}) + r_{R} U_{y,R} \\ \frac{1}{m} (F_{y,t} \cos \delta + F_{y,r} + F_{x,t} \sin \delta) - r_{R} U_{x,R} \\ \frac{1}{I_{zz}} (d_{f} F_{y,t} \cos \delta + d_{f} F_{x,t} \sin \delta - d_{r} F_{y,r}) \\ \frac{1}{I_{zz}} (d_{f} F_{y,t} \cos \delta + d_{f} F_{x,t} \sin \delta - d_{r} F_{y,r}) - \kappa(s) \end{bmatrix}. \] (3.11)

### 3.6.4 MPC Tracking Dynamics

The tracking MPC controller relies on an error dynamics model. Define a path (see Figure 3.4) through space where \( s \) is the distance along the path and at any distance \( s \), we know the path heading \( \psi(s) \), the curvature \( \kappa(s) \), and a coordinate system \( (\hat{s}, \hat{\epsilon}) \) tangential and normal to the path respectively. Given the position and heading of the robot car, we can project the car to the closest point on the path. Let the lateral error \( e \) be the lateral distance along direction \( \hat{\epsilon} \) and \( \Delta \psi = \psi_{R} - \psi_{\text{path}} \) be the robot car’s heading relative to the path computed from this closest point. Using this projection and the robot car’s dynamics from (3.8), we can compute the error dynamics relative to the desired path. Using the same notation defined previously in (3.8), the state for the robot car tracking a desired path is \( \dot{x} = [s \quad U_{x,R} \quad U_{y,R} \quad r_{R} \quad \Delta \psi \quad e]^{T} \) and the controls \( u = u_{R} = [\delta \quad F_{x}]^{T} \). The tracking dynamics are

\[ \dot{x} = \begin{bmatrix} \dot{s} \\ \dot{U}_{x,R} \\ \dot{U}_{y,R} \\ \dot{r}_{R} \\ \dot{\Delta \psi} \\ \dot{\epsilon} \end{bmatrix} = \begin{bmatrix} U_{x,R} \cos \Delta \psi - U_{y,R} \sin \Delta \psi \\ \frac{1}{m} (F_{x,t} \cos \delta - F_{y,t} \sin \delta + F_{x,r} + F_{x,\text{drag}}) + r_{R} U_{y,R} \\ \frac{1}{m} (F_{y,t} \cos \delta + F_{y,r} + F_{x,t} \sin \delta) - r_{R} U_{x,R} \\ \frac{1}{I_{zz}} (d_{f} F_{y,t} \cos \delta + d_{f} F_{x,t} \sin \delta - d_{r} F_{y,r}) \\ \frac{1}{I_{zz}} (d_{f} F_{y,t} \cos \delta + d_{f} F_{x,t} \sin \delta - d_{r} F_{y,r}) - \kappa(s) \end{bmatrix}. \] (3.12)

### 3.7 The MPC+HJI Trajectory Tracking Controller

In this section, we detail the optimization problem used in the MPC+HJI tracking controller, describe how this problem can be adapted to accommodate collision avoidance for static obstacles, and provide numerical details about the BRT computation used in this formulation. Central to an MPC controller is an optimization problem; at each time step, the controller solves an optimization problem to find an optimal control sequence, passes the first control input to the actuator, and then repeats this process. To be amenable to real-time applications, the optimization problem requires
a fast solve time (~ 0.01s). In this work, the MPC tracking problem is formulated as a convex optimization problem, namely a QP, enabling the use of efficient solvers [68, 69] which are capable of solving the QP within the tight operating frequency.

### 3.7.1 Optimization Problem

Both the trajectory tracking objective and safety-preserving control constraint rely on optimizing robot steering and longitudinal force inputs simultaneously. Let

\[
q^{(k)} = \begin{bmatrix}
\Delta s^{(k)} & U_x^{(k)} & U_y^{(k)} & r^{(k)} & \Delta \psi^{(k)} & e^{(k)}
\end{bmatrix}^T
\]

be the state of the robot car with respect to a nominal trajectory at discrete time step \( k \). \( \Delta s^{(k)} \), \( e^{(k)} \) and \( \Delta \psi^{(k)} \) denote longitudinal, lateral, and heading error; \( U_x^{(k)}, U_y^{(k)} \), and \( r^{(k)} \) are body-frame longitudinal and lateral velocity, and yaw rate respectively as defined in Figure 3.4. Let

\[
u^{(k)} = \begin{bmatrix}
\delta^{(k)} & F_x^{(k)}
\end{bmatrix}^T
\]

be the controls at step \( k \) and let

\[
A^{(k)} q^{(k)} + B^{(k)} u^{(k)} + B^{(k)}_w u^{(k+1)} + c^{(k)} = q^{(k+1)}
\]

denote linearized first-order-hold dynamics of (3.12). We adopt the varying time steps method (\( N_{\text{short}} \) time steps of size \( \Delta t_{\text{short}} \) and \( N_{\text{long}} \) time steps of size \( \Delta t_{\text{long}} \)) and stable handling envelope constraint from [56] (expressed as \( H^{(k)} \) and \( G^{(k)} \) in the problem formulation below). To ensure the existence of a feasible solution, we use slack variables \( \sigma_{\beta}^{(k)} \), \( \sigma_{r}^{(k)} \), and \( \sigma_{\text{HJI}}^{(k)} \) on the stability and HJI constraints. The HJI reachability constraint \( M_{\text{HJI}} u^{(k)} + b_{\text{HJI}} \geq -\sigma_{\text{HJI}} \) is activated only when \( V(x_{\text{rel}}) \leq \epsilon \). Although HJI theory suggests that applying this constraint on the next action alone is sufficient, we apply it over the next \( N_{\text{HJI}} = 3 \) timesteps (30ms lookahead) to account for the approximations inherent in our QP formulation. The MPC tracking problem is a quadratic program of the form.
3.7. THE MPC+HJI TRAJECTORY TRACKING CONTROLLER

$$\begin{align*}
\min_{q,u,\sigma,\sigma_{\text{HJI}}} & \sum_{k=1}^{T} \left( (\Delta s^{(k)})^T Q \Delta s^{(k)} + (\Delta \psi^{(k)})^T Q \Delta \psi^{(k)} + (\epsilon^{(k)})^T Q \epsilon^{(k)} + \right. \\
& \left. \left( \frac{\Delta \delta^{(k)}}{\Delta t^{(k)}} \right)^T R \Delta \delta \left( \frac{\Delta \delta^{(k)}}{\Delta t^{(k)}} \right) + \left( \frac{\Delta F_x^{(k)}}{\Delta t^{(k)}} \right)^T R \Delta F_x \left( \frac{\Delta F_x^{(k)}}{\Delta t^{(k)}} \right) + \\
& W_{\beta} \sigma_{\beta}^{(k)} + W_{r} \sigma_{r}^{(k)} + W_{\text{HJI}} \sigma_{\text{HJI}}^{(k)} \right) \Delta t^{(k)} \\
\text{subject to} & \quad q^{(1)} = q^{(\text{curr})}, \quad u^{(1)} = u^{(\text{curr})}, \\
& \quad \delta^{(k+1)} - \delta^{(k)} = \Delta \delta^{(k)}, \\
& \quad \dot{\delta}_{\text{min}} \Delta t^{(k)} \leq \Delta \delta^{(k)} \leq \dot{\delta}_{\text{max}} \Delta t^{(k)}, \\
& \quad \delta_{\text{min}} \leq \delta^{(k)} \leq \delta_{\text{max}}, \\
& \quad F_x^{(k+1)} - F_x^{(k)} = \Delta F_x^{(k)}, \\
& \quad F_{x,\text{min}} \leq F_x^{(k)} \leq F_{x,\text{max}}, \\
& \quad U_{x,\text{min}} \leq U_x^{(k)} \leq U_{x,\text{max}}, \\
& \quad \sigma_{\beta}^{(k)} \geq 0, \quad \sigma_{r}^{(k)} \geq 0, \\
& \quad A^{(k)} q^{(k)} + B_{-}^{(k)} u^{(k)} + B_{+}^{(k)} u^{(k+1)} + c^{(k)} = q^{(k+1)}, \\
& \quad H^{(k)} \begin{bmatrix} U_{y}^{(k)} \\ U_{H}^{(k)} \end{bmatrix} - G^{(k)} \leq \begin{bmatrix} \sigma_{\beta}^{(k)} \\ \sigma_{r}^{(k)} \end{bmatrix}, \quad \text{for } k = 1, \ldots, N_{\text{short}} + N_{\text{long}}, \\
& \quad \sigma_{\text{HJI}}^{(j)} \geq 0 \quad (\text{if } V(x_{\text{rel}}) \leq \epsilon), \\
& \quad M_{\text{HJI}} u^{(j)} + b_{\text{HJI}} \geq -\sigma_{\text{HJI}}^{(j)} \quad (\text{if } V(x_{\text{rel}}) \leq \epsilon), \quad \text{for } j = 1, \ldots, N_{\text{HJI}}.
\end{align*}$$

The objective strives to minimize a combination of tracking error (longitudinal, lateral, and angular), control rates (steering and longitudinal tire forces), and magnitude of the slack variables. The constraints ensure (i) continuity with the current and next state and control, (ii) the change in controls across each time step is bounded, (iii) the positivity of slack variables, (iv) the (linearized) dynamics are satisfied, (v) the vehicle stability constraints are satisfied, and (vi) the HJI safety-preserving half-plane control constraint is satisfied when $V(x_{\text{rel}}) \leq \epsilon$.

The time-discretization and linearizations (dynamics and constraints) we apply amount to an approximate variant of sequential quadratic program (SQP). In particular, we solve one QP at each MPC step rather than the usual iteration until convergence. Since the tracking problems are so similar from one MPC step to the next, we find that this approach yields sufficient performance for our purposes. We interpolate along each solution trajectory to compute the linearization nodes for the QP at the next MPC step.
We use the \texttt{ForwardDiff.jl} automatic differentiation package implemented in the Julia programming language \cite{70} to linearize the trajectory tracking dynamics as well as the HJI relative dynamics for the safety-preserving constraint. We call the Operator Splitting Quadratic Program solver \cite{69} through the \texttt{Parametron.jl} modeling framework \cite{71}; this combination of software enables us to solve the following MPC optimization problem at 100Hz. The MPC code, including optimization parameters and vehicle parameters (also included in Appendix A.1 and A.2), can be found here: \url{https://github.com/StanfordASL/Pigeon.jl}

### 3.7.2 Adding Static Obstacles

The current formulation does not prevent the robot from driving very far from its nominal trajectory (e.g., completely off the road) to avoid the human. In more realistic road settings, there could be environmental constraints such as a concrete road boundary. In the same way that collision avoidance with a human-controlled vehicle is formulated as an additional constraint to the robot’s MPC problem, we can add another safety constraint to account for a static wall to prevent the robot from swerving completely off the road. We consider two approaches for deriving this constraint: the first based on an additional HJI computation and giving rise to a similar control constraint applied over the first $N_{\text{HJI}}$ timesteps, and the second treating the wall as a static obstacle with associated state constraints that apply over the whole duration of the MPC trajectory optimization.

For the first approach, we can compute a value function $V_{\text{WALL}}(x_R)$ describing the interaction between the robot and the wall by using\cite{3.8} without the $p_{x,R}$ state (we only care about the distance from the wall and not how far along the wall the robot is). Note that since the wall is static, there is no disturbance input. The robot-wall HJI control constraint $M_{\text{WALL}}u_R + b_{\text{WALL}} \geq 0$ becomes active when $V_{\text{WALL}}(x_R) \leq \epsilon$. This means that it is possible for both HJI-safety constraints to be active simultaneously. We note that theoretically, we could consider the robot, human, and wall simultaneously by computing the BRT for the joint system. However, naively increasing the state size without any decomposition would be computationally undesirable even offline. Thus we treat the human-robot and robot-wall systems separately but will discuss later the impact of this design choice.

Alternatively, we can account for a static wall by adding lateral error bound constraints into the MPC tracking problem—this is the approach taken in \cite{56}. This involves always adding left and right lateral error constraints ($e_{\text{min}}^{(k)} \leq e^{(k)} \leq e_{\text{max}}^{(k)}$) at each node point along the MPC trajectory such that the lateral deviation from the desired trajectory does not exceed the lateral distance to the wall. Specifically, we add the following constraints to the QP in \cite{3.13}:

$$e^{(k)} - e_{\text{min}}^{(k)} \geq -\sigma_e, \quad e_{\text{max}}^{(k)} - e^{(k)} \geq -\sigma_e \quad \text{for} \quad k = 1, ..., N$$

as well as a slack penalty, $W_e \sigma_e$ to the cost function. With a large weighting on the slack penalty,
3.7. THE MPC+HJI TRAJECTORY TRACKING CONTROLLER

Figure 3.6: X1: a steer-by-wire experimental vehicle platform. The 1/10-scale RC car has a mast that is visible to LiDARs.

this approach ensures no collision with the wall over the MPC time horizon only (in contrast to HJI which is over an infinite time horizon) and in general provides higher tracking performance because the MPC controller can optimize tracking states and controls over the entire tracking trajectory. We investigate both these approaches and provide more discussion in Section 3.9.

3.7.3 Reachability Computation

We use the BEACLS toolkit [64] implemented in C++ to compute the BRT. Since HJ reachability suffers from the curse of dimensionality, this is, to the best of our knowledge, the first attempt to use HJ reachability to compute the BRT for a seven-state relative system, especially with such high modeling fidelity. We, however, do sacrifice on grid size and use a relatively coarse grid compared to other literature standards. We linearly interpolate between grid points to evaluate the value function and its gradient at any given state (human-robot relative state or robot-wall state). We use a grid size of $13 \times 13 \times 9 \times 9 \times 9 \times 9$ for our 7D system uniformly spaced over $(p_{x,\text{rel}}, p_{y,\text{rel}}, \psi_{\text{rel}}, U_{x,\text{R}}, U_{y,\text{R}}, v_{H}, r_{R}) \in [-15, 15] \times [-5, 5] \times [-\pi/2, \pi/2] \times [1, 12] \times [-2, 2] \times [1, 12] \times [-1, 1]$; computing the BRT with this system and discretization takes approximately 70 hours on a 3.0GHz octocore AMD Ryzen 1700 CPU.

Computing the BRT requires computing the optimal control and disturbance defined in Equation (3.5). For the relative dynamics model, the optimal disturbance (i.e., human controls) is a bang-bang solution since the disturbance is affine. Due to the highly nonlinear nature of the dynamics, we use a uniform grid search across $\delta$ and $F_x$ to calculate optimal controls for the robot.

For the robot-wall system, we compute the optimal control actions in the same fashion. We use a grid size of $21 \times 9 \times 9 \times 9$ uniformly spaced over $(p_{y,\text{R}}, \psi_{\text{R}}, U_{x,\text{R}}, U_{y,\text{R}}, r_{\text{R}}) \in [-3, 7] \times [-\pi/2, \pi/2] \times [1, 12] \times [-2, 2] \times [-1, 1]$; computing the BRT with this system and discretization takes approximately 40 minutes.
3.8 Traffic-Weaving Experiments

In this section, we describe our experimental set-up, and the different tiers of experimentation including experiments with a virtual human-driven vehicle, experiments with and without a static wall, comparison with a baseline approach, and experiments with a human-controlled 1/10-scale RC car.

3.8.1 Experimental Set-up

X1 is a versatile steer-by-wire, drive-by-wire, and brake-by-wire experimental vehicle developed by the Stanford Dynamic Design Lab (see Figure 3.6). It is equipped with three LiDARs (one 32-beam and two 16-beam), a differential GPS/INS which provides 100Hz pose estimates accurate to within a few centimeters as well as high fidelity velocity, acceleration, and yaw rate estimates. To control X1, desired steering ($\delta$) and longitudinal tire force ($F_{x,f}, F_{x,r}$) commands are sent to the dSpace MicroAutoBox (MAB) which handles all sensor inputs except LiDAR (handled by the onboard PC) and implements all low-level actuator controllers. Similarly, state information about the vehicle is obtained from the MAB. We use the Robot Operating System (ROS) to communicate with the MAB which handles sensing and control at the hardware level; the planning/control stack described in this work is running onboard X1 on a consumer desktop PC running Ubuntu 16.04 equipped with a quadcore Intel Core i7-6700K CPU and an NVIDIA GeForce GTX 1080 GPU. X1 parameters used in the equations of motion (3.8) are listed in Appendix A.1.

We also perform experiments using a LiDAR-visible 1/10-scale RC car (Figure 3.6 right) as the human-driven car to investigate the robustness of our proposed control stack with perception uncertainty.

The code used to run the experiments can be found here: https://github.com/StanfordASL/safe_traffic_weaving.

3.8.2 Experimental Results

To evaluate our proposed control stack—a synthesis of a high-level probabilistic interaction planner with the MPC+HJI tracking controller—we perform full-scale human-in-the-loop traffic-weaving trials with X1 taking on the role of the robot car. To ramp up towards testing with two full-scale vehicles in the near future, we investigate two types of human car: (i) a virtual human-driven car, and (ii) a 1/10-scale LiDAR-visible human-driven RC car. We scale the highway traffic-weaving scenario from a mean speed of $\sim$28m/s) down to a mean speed of $\sim$8m/s by shortening the track (reducing longitudinal velocity by a constant) and scaling time by a factor of 4/3 (with the effect of scaling speeds by 3/4 and accelerations by 9/16). The parameter values in the MPC tracking problem are listed in Appendix A.2.
3.8. TRAFFIC-WEAVING EXPERIMENTS

We investigate and evaluate the effectiveness of our control stack by allowing the human car to act carelessly (i.e., swerving blindly towards the robot car) during the experiments. In this section, we compare our proposed controller (MPC+HJI) against a tracking-only MPC controller (MPC) and a controller that switches to the HJI optimal avoidance controller when near safety violation (switching). In Section 3.8.3 we compare the two static obstacle avoidance approaches developed in Section 3.7.2 to a scenario with a virtual wall constraining the robot’s trajectory. We investigate applying an extension of the same state-constraint-based static obstacle avoidance strategy to the problem of collision with a dynamic obstacle (the human) in Section 3.8.4 and discuss its shortcomings. Finally, in Section 3.8.5 we present experiments with the human car physically embodied by an RC car.

**Virtual Human-Driven Vehicle**

To ensure a completely safe experimental environment, our first tier of experimentation uses a joystick-controlled virtual vehicle for the human car and allows the robot control stack to have perfect observation of the human car state. Experimental trials of the probabilistic planning framework using (i) our proposed approach (MPC+HJI), and (ii) switching to the optimal HJI controller (switching) are shown in Figures 3.7 and 3.8 along with a simulated comparison between the other control
CHAPTER 3. REACHABILITY-BASED SAFETY ASSURANCE

Figure 3.8: Controller comparison on experimental data where the robot car uses the switching controller. Left: Simulations of using the MPC, MPC+HJI, and switching controllers when $V(x_{rel}(t))$ first drops below $\epsilon = 0.05$ are shown and are compared against the executed trajectory (from experiments) and the desired trajectory (from the interaction planner). Right: The corresponding evolutions of $V(x_{rel}(t))$. Bottom: Illustration of the traffic-weaving interaction. The human-controlled car (blue car) carelessly drives into the path of the robot car (red car), causing the robot car to react by swerving. The transparency of the rectangles corresponds to the speed of the car (higher transparency corresponds to higher speed).

schemes. In figures illustrate a time instance where $V(x_{rel}(t))$ first drops below $\epsilon = 0.05$. For comparative purposes, each controller was simulated with the nominal trajectory (at that time instance) held fixed, but in reality, the nominal trajectory in these experiments was updated at $\sim 3$ Hz. As expected, we see that when a safety violation occurs, the MPC+HJI controller represents a middle ground between the tracking-only MPC which does not react to the human car’s intrusion, and the switching controller which arguably overreacts with a large excursion outside the lane boundaries. Evidently, our proposed controller tries to be minimally interventional—the robot car swerves/brakes but only to an extent that is necessary.

Looking at the value function, we see that, as expected, the MPC+HJI controller aims to keep the value positive, but does not necessarily strive to increase it, while the switching controller aims to increase the value as much as possible. The MPC (tracking only) controller fails to increase the value at all when safety violation occurs. All controllers, however, experience a period where the value is negative, even the two HJI-based controllers which theoretically guarantee that the value should never be below $\epsilon$. We believe this is due to model mismatch; we will discuss this point in more depth in Section 3.9.

Additionally, in some trials when the robot car was traveling faster, the activation of the HJI constraint resulted in the robot car performing a large but smooth swerve that traversed completely outside the lane boundaries. We address this limitation by adding a wall constraint into the MPC.
formulation discussed in Section 3.8.3

Safety and Efficiency Trade-off

![Diagram showing trade-off between total safety and average efficiency, and between worst-case safety and worst-case efficiency.]

(a) Trade-off between total safety and average efficiency. (b) Trade-off between worst-case safety and worst-case efficiency.

Figure 3.9: Safety and efficiency trade-offs for different control strategies.

Our MPC+HJI controller considers the set of safety-preserving controls while optimizing its tracking performance when it is near safety violation. In contrast, the HJI switching controller uses the optimal avoidance control policy and as a result, neglects to track the desired trajectory that was selected by the planner for interaction performance. As such, there is a trade-off between safety, defined with respect to the value function, and efficiency.

For use as a comparison metric, we define the notion of total safety of an interaction of time length $T$ as

$$S_{\text{total}} := \int_0^T 1[V(x_{\text{rel}}(t)) \leq 0] V(x_{\text{rel}}(t)) \, dt$$

which is the integral of the value function when it is negative. Total safety considers not only the magnitude of the safety violation but also the duration of the violation. We can also define the worst-case safety as

$$S_{\text{worst}} := \min_{t \in [0,T]} V(x_{\text{rel}}(t))$$

which does not consider the duration of a safety violation, but rather the worst-case safety violation with respect to the value function over the interaction.

The efficiency of an interaction is more difficult to quantify. In this analysis, we define efficiency in terms of acceleration as this is a proxy for control effort and passenger comfort. Specifically, we normalize the magnitude of acceleration with the earth’s gravitational acceleration $g = 9.81 \text{ms}^{-2}$ and then subtract it from one so that larger values correspond to better efficiency. Mathematically,
we define average efficiency as

\[ E_{\text{avg}} := 1 - \frac{1}{T} \int_0^T \frac{1}{g} \sqrt{a_x(t)^2 + a_y(t)^2} \, dt \]

where \( a_x(t) \) and \( a_y(t) \) are the \( x \) and \( y \) acceleration of the robot car; larger \( E_{\text{avg}} \) values correspond to better efficiency. We note that due to friction limits of a (typical) vehicle, the magnitude of the acceleration should not exceed one. We also define the worst-case efficiency as

\[ E_{\text{worst}} := \max_{t \in [0, T]} \left( 1 - \frac{1}{g} \sqrt{a_x(t)^2 + a_y(t)^2} \right) \]

which considers the largest acceleration experienced during the interaction.

Given these quantities, we can compare the safety and efficiency trade-offs between the MPC, MPC+HJI, and switching controllers. Multiple experimental trials using X1 were carried out using each controller, and Figure 3.9a compares the average/total metrics while Figure 3.9b compares the worst-case metrics. We see that in both cases, the MPC+HJI controller provides a good balance between safety and efficiency; the safety almost equaling that of the optimal switching controller, and efficiency almost equaling that of the MPC controller. The switching controller provides the highest level of safety (with respect to the value function) but experiences lower efficiency since it often results in heavy braking and sharp swerving. The MPC controller provides lower safety scores but with larger variations as the resulting safety score is scenario dependent rather than controller dependent.

### 3.8.3 Adding Static Obstacles

![Comparison of control sequences from using MPC lateral error state constraints, robot-wall HJI control constraint, and no constraints to avoid a static road boundary.](image)

Figure 3.10: Comparison of control sequences from using MPC lateral error state constraints, robot-wall HJI control constraint, and no constraints to avoid a static road boundary. The plot begins when the robot car is planning autonomously.

To avoid a static (virtual) road boundary wall when the robot car is swerving off the road to avoid a collision with the human car, we investigate two methods, (i) persistently adding lateral error...
Figure 3.11: Trajectory comparison between using lateral error constraints (top), a robot-wall HJI control constraint (middle), and using no wall constraints (bottom). The human car is in the left (top) lane and is constant across all three trials, and the robot car is in the right (bottom) lane. The transparency of the car corresponds to the speed of the car (higher transparency corresponds to higher speed).

3.8. TRAFFIC-WEAVING EXPERIMENTS

state constraints into the MPC problem and (ii) adding an additional HJI-based control constraint for a robot-wall relative system into the MPC problem when $V_{\text{WALL}}(x_R) \leq \epsilon$. Figure 3.10 compares the control sequences and Figure 3.11 compares the trajectory of both approaches. Here, the human car is commanded to stay in the lane and is initialized inside the BRT so that when the robot starts planning autonomously, the robot car will react immediately and swerve out of the way—we investigate the nature of the swerving. We do note that the human car is able to make the robot car swerve even though it is staying in its lane. This is due to the wide, pear-shape BRT when the two cars are parallel and at similar speeds. We discuss this in more detail in Section 3.9.

Around 20 meters from the start of the roadway in Figure 3.11 the robot car begins to plan autonomously. Before then, the cars are following a straight line in order to speed up from $\approx 1m/s$ to the desired interaction speed. To provide the cleanest comparison, all results in this subsection are derived from simulation. Starting in the right lane, we see in all cases that when safety is violated the robot car swerves to the right ($\delta < 0$). When using the lateral error constraints (Figure 3.11 top), the robot car essentially has a “look-ahead” capability because it is able to optimize its steering commands over the MPC tracking horizon, essentially distributing the responsibility of avoiding the wall across the entire tracking MPC horizon. As a result, the robot car can successfully and smoothly
steer back onto the road and avoid the wall.

When using the robot-wall HJI control constraint (Figure 3.11 center), the robot car does not preemptively swerve back onto the road and instead only reacts to the wall when $V_{\text{WALL}}(x_R) \leq \epsilon$ as designed. The robot car performs a hard brake to the point of almost stopping and then begins to eventually command a maximum steering angle (18 deg). The sharp and abrupt behavior stems from the HJI formulation assuming the robot can and will take extreme actions, including cases when using the safety-preserving control set. As a result, the responsibility for evasive action is triggered at the very latest possible instance and compressed into a control constraint over a single (in practice 3) MPC timestep. This is in contrast to the former approach of always having MPC lateral error state constraints over the entire MPC horizon. The MPC+HJI controller is effective in avoiding dynamic obstacles, but for static obstacles, HJI is not suitable because it is unnecessary to reason about the dynamics of something that is static. With no wall constraints (Figure 3.11 bottom), the robot car swerves completely off the road, and in order to quickly get to the left lane before the end of the road (inscribed as an objective in the high-level planner), it overshoots to the other side of the road and also drives beyond the road boundary on the other side.

### 3.8.4 Baseline Comparison

Motivated by the success of applying state constraints in the previous section, we compare our proposed framework (MPC+HJI for safe human-robot interaction) with a baseline approach of avoiding collision with the human car using lateral error constraints. To define safety state constraints throughout the robot trajectory, we must however make some assumptions on the human’s future trajectory. For this baseline, at each MPC iteration, we assume that the human car will continue moving at its current heading and with its current velocity. The lateral error constraint at each MPC time step is computed between the robot’s desired trajectory and the human’s projected trajectory—this is analogous to applying the approach taken in [56] but with (deterministic) dynamic obstacles. A simulation of our proposed approach and the baseline approach is shown in Figure 3.12.

Notably, in this simulation, the human car defies the baseline’s linear-extrapolation-based prediction model and curves into the other lane towards the robot car.

We see in Figure 3.12a that when using the MPC+HJI controller (with lateral error state constraints for wall avoidance), we are able to successfully avoid a collision with the human car and the wall, and maintain relative states outside the BRT. The baseline approach demonstrated in Figure 3.12b, on the other hand, collides with the human because its consideration of only one possible future (i.e., human trajectory) at each MPC step leads the robot into a region of inevitable collision.

---

4 Since the dynamic bicycle model is ill-defined for low speeds (thus explaining the oscillations in $F_x$), the experiment (when using the robot-wall HJI control constraint) is essentially over around $t = 9$. In general, the MPC problem can switch to the kinematic model at low speeds [22], which is well defined in that speed region.

5 Alternatively, one could use a sample or a maximum a posteriori estimate from the interaction planner’s prediction model.
(defined against all, including worst-case, human controls). In particular, we see that as the human trajectory increases its curvature towards the middle of the lane change, the lateral error bounds (based on an assumption of constant velocity) are not stringent enough to maintain robot safety. The utility of the MPC+HJI approach is that it distills safety considerations with respect to all possible realizations of the human trajectory into a single control constraint. Moreover, this constraint is not overly conservative (to improve the state-constraint-based approach one might imagine defining lateral error constraints to avoid the entire forward reachable set of the human), because it incorporates the concept of closed-loop feedback into its BRT computation.

3.8.5 1/10-Scale Human-Driven Vehicle

To begin investigating the effects of perception uncertainty on our safety assurance framework, we use three LiDARs onboard X1 to track a human-driven RC car, and implement a Kalman filter for human car state estimation (position, velocity, and acceleration). Even with imperfect observations, we show some successful preliminary results (an example is shown in Figure 3.13) at mean speeds of 4m/s, close to the limits of the RC car + LiDAR-visible mast in crosswinds at the test track. We observe similar behavior as in the virtual human car experiments, including the fact that the value function dips briefly below zero before the MPC+HJI controller is able to arrest its fall; we discuss this behavior in the next section.

3.9 Discussion

Beyond the qualitative and quantitative confirmation of our design goals, our experimental results reveal three main insights. 

**Takeaway # 1:** The reachability cache is underly-conservative with respect to robot car dynamics and overly-conservative with respect to human car dynamics.

In all cases—hardware experiments as well as simulation results—the HJI value function \( V \) dips below zero, indicating that neither the HJI+MPC nor even the optimal avoidance switching controller is capable of guaranteeing safety in the strictest sense. The root of this apparent paradox is in the computation of the reachability cache used by both controllers as the basis of their safety assurance. Though the 7-state relative dynamics model \( \text{(3.11)} \) subsumes a single-track vehicle model that has proven successful in predicting the evolution of highly dynamic vehicle maneuvers \( [17, 59] \), the way it is employed in computing the value function \( V \) omits relevant components of the dynamics. In particular, when computing the optimal avoidance control \( \text{(3.5)} \) as part of solving the HJI PDE, we assume total freedom over the choice of robot steering angle \( \delta \) and longitudinal force command \( F_x \), up to maximum control limits. This does not account for, e.g., limits on the steering slew rate (traversing \( [\delta_{\text{min}}, \delta_{\text{max}}] \) takes approximately 2 seconds), and thus the value function is computed under the assumption that the robot can brake/swerve far faster than it actually can.
We note that simply tuning the safety buffer $\epsilon$ is insufficient to account for these unmodeled dynamics. In Figure 3.7 we see that $V$ may drop from approximately 0.5 (the value when the two cars traveling at 8m/s start side-by-side in lanes) to -0.3 in the span of a few tenths of a second. Selecting $\epsilon > 0.5$ might give enough time for the steering to catch up, but such a selection would prevent the robot car from accomplishing the traffic weaving task even under nominal conditions, i.e., when the human car is equally concerned about collision avoidance. Even for $\epsilon = 0.05$, we see that the human car can cause the robot to swerve by just staying in its lane but with a small offset towards the robot’s lane (see Figure 3.11). This is because the safety controller would push the robot car outside of its lane from the outset to maintain the buffer. This behavior follows from wide level sets associated with the transient control authority asymmetry (recall that in the HJI relative dynamics the human car may adjust its trajectory curvature discontinuously), assumed as a conservative safety measure as well as a way to keep the relative state dimension manageable.

The simplest remedy for both of these issues is to increase the fidelity of the relative dynamics model by incorporating additional integrator states $\dot{\delta}$ for the robot and $\dot{\omega}$ for the human. Naively increasing the state dimension to 8 or 9, however, might not be computationally feasible (even offline) without devising more efficient HJI solution techniques or choosing an extremely coarse discretization over the additional states. By literature standards we already use a relatively coarse discretization grid for solving the HJI PDE; associated numerical inaccuracies may be another source of the observed safety mismatch and alternate grid choices are a possible subject of future investigation. We believe that simulation, accounting for slew rates, could be a good tool to prototype such efforts, noting that as it stands we have relatively good agreement between simulation and the experimental platform in our testing.

**Takeaway # 2:** Interpretability of the value function $V$ should be a key consideration in future work.

In this work the terminal value function $V(0, x_{rel})$ is specified as the separation/penetration distance between the bounding boxes of the two vehicles, a purely geometric quantity dependent only on $p_{x,rel}$, $p_{y,rel}$, and $\psi_{rel}$. Recalling that $V := V_\infty$ represents the worst-case eventual outcome of a differential game assuming optimal actions from both robot and human, we may interpret the above results through the lens of worst-case outcomes, i.e., a value of -0.3 may be thought of as 30cm of collision penetration assuming optimal collision seeking/avoidance from human/robot. When extending this work to cases with environmental obstacles (e.g., concrete highway boundaries that preclude large deviations from the lane), or multi-agent settings where the robot must account for the uncertainty in multiple other parties’ actions, for many common scenarios it may be the case that guaranteeing absolute safety is impossible. Instead of avoiding a BRT of states that might lead to collision, we should instead treat the value function inside the BRT as a cost. In particular, we should specify more contextually relevant values of $V(0, x_{rel})$ for states in collision, e.g., negative kinetic energy or another notion of collision severity as a function of the velocity states $U_{x,R}$, $U_{y,R}$,
$v_H$, and $r_R$ in addition to the relative pose. This would lead to a controller that prefers, in the worst case, collisions at lower speeds, or perhaps “glancing blows” where the velocities of the two cars are similar in magnitude and direction.

**Takeaway # 3: Static obstacles should be accounted for using MPC tracking state constraints and not HJI control constraints.**

It is natural to use HJI when reasoning about potential dangers from a dynamic, unpredictable agent, but for static obstacles, there is no uncertainty in how the obstacle may act. That is, when the other agent is deterministic, there is no notion of optimizing robot safety policies over all worst-case actions by the other agent. Indeed, in this case, it is possible to confidently extend the safety constraint along the entire MPC horizon, as demonstrated in Section 3.8.3. This is as opposed to deriving a safety constraint to be applied at the first step (on the present control action; the HJI+MPC approach), as must be done with a nondeterministic agent with unknown future trajectory. Spreading the collision avoidance constraint over the problem horizon allows for the MPC optimization to more smoothly accomplish its objectives (i.e., control smoothness and tracking performance) while maintaining safety. We note, however, that naive extensions of this approach to nondeterministic agents (recall Section 3.8.4) do not similarly provide strong safety assurances.

### 3.10 Summary

We have investigated a control scheme for providing real-time safety assurance to underpin the guidance of a probabilistic planner for human-robot vehicle-vehicle interactions. By essentially projecting the planner’s desired trajectory into the set of safety-preserving controls whenever safety is threatened, we preserve more of the planner’s intent than would be achieved by adopting the optimal control with respect to separation distance. Our experiments show that with our proposed minimally interventional safety controller, we accomplish the high-level objective (traffic weaving) despite the human car swerving directly onto the path of the robot car, and accomplish this relatively smoothly compared to using a switching controller that results in the robot car swerving more violently off the road. Further, we investigate the addition of a road boundary wall to our formulation to prevent the robot car from swerving completely out of the lane which could be dangerous in realistic road settings and compare against a baseline approach using state constraints to avoid dynamic obstacles to show that our framework is better designed to keep the robot car safe even in the face of worst-case human car behaviors.

We note that this work represents only a promising first step towards the integration of reachability-based safety guarantees into a probabilistic planning framework. Future work includes investigating this framework for cases where the human and robot have very different dynamics, such as a pedestrian or cyclist interacting with a car, or a human interacting with a robotic manipulator, and for
interactions involving multiple (more than two) agents. We may also consider adapting our approach
to ensure high planning performance while guaranteeing the satisfaction of constraints other than
safety, e.g., task requirements specified by temporal logic constraints as considered in [73]. In the
context of human-robot vehicle interactions, we have already discussed the concrete modifications to
this controller we believe are necessary to improve the practical impact of our theoretical guarantees;
further study should also consider improving the fitting of the planning objective at the controller
level. That is, instead of performing a naive projection, i.e., the one that minimizes trajectory
tracking error, it is likely that a more nuanced selection informed by the planner’s prediction model
would represent a better “backup choice” in the case that safety is threatened. We recognize that
ultimately, guaranteeing absolute safety on a crowded roadway may not be realistic, but we believe
that in such situations value functions derived from reachability may provide a useful metric for
near-instantly evaluating the future implications of a present action choice.
3.10. SUMMARY

(a) Simulation trial of using MPC+HJI for human car collision avoidance (including lateral error constraints for wall collision avoidance).

(b) Simulation trial of using lateral error constraints for both human car and wall collision avoidance.

Figure 3.12: Comparison between the MPC+HJI controller (top) and a baseline approach not based on any reachability analysis (bottom) for the task of dynamic obstacle avoidance. The trajectory of the human car starting in the left (top) lane is visualized in gray; the trajectory of the robot car starting in the right (bottom) lane is visualized in color. In each close-up view, “error bars” accompanying each snapshot of the robot illustrate the first \( k = 1 \) lateral state constraint of the MPC problem at that step. To help correspond the positions of the cars over the trials, each pair of matching-shape markers represents a common time instant.
Figure 3.13: Time-lapse of pairwise vehicle interaction: X1 with 1/10-scale human-driven RC car. The RC car (green trajectory) nudges into X1 (red trajectory), which swerves gently to avoid. Inset: The value function over time of the X1-human driven RC car experiment.
Chapter 4

Extensions to Safe Multi-agent Interactions

In Chapter 3, we discussed how to incorporate reachability-based safety into a robot autonomy stack. In particular, we demonstrated how our proposed safety controller can achieve minimal intervention, thus enabling a high-level planner to seamlessly continue optimizing for planning performance subject to safety considerations. However, our investigation in Chapter 3 was limited to pairwise interaction scenarios where it is significantly less challenging than multi-agent crowded environments. For multi-agent scenarios, the environment is more crowded and therefore more challenging to ensure that a safety controller, if used, will be feasible. For instance, if an ego agent is boxed-in by multiple other agents (see Figure 4.1), then a HJ reachability safety controller will be infeasible since all the other agents could potentially swerve into the boxed-in ego agent. In this chapter, we extend the work from Chapter 3 to account for multi-agent settings by additionally including reachability-based considerations at the planning level. Specifically, we investigate the elevation of HJ reachability, a typically low-level control technique, to the planning level and therefore creating a shared sentiment of safety throughout the autonomy stack.

4.1 Introduction

As previously described in Chapter 3, decision-making and control for robots are typically stratified into levels, with each having different purposes. A high-level planner, informed by representative yet simplified dynamics of a robot and its environment, is designed to be far-sighted and selects plans that optimize performance metrics (e.g., minimize time to goal, control effort, and perception uncertainty). While a low-level controller, running at a much higher frequency than the planner, tends to be more short-sighted and respects more accurate models of the robot’s dynamics and
control constraints in order to implement controls necessary to follow the desired plan. However, when it comes to ensuring safety for the robot, the divide between these components can lead to a diffusion of responsibilities—the planner and controller may each devise their own safety protocols, or even assume the other bears the full responsibility, but when combined together, they may not necessarily complement each other in achieving the shared goal of ensuring safety for the system. For multi-agent settings, the consequences of the divide can become more pronounced as there are typically more constraints that are imposed on the system.

Chapter 3 investigated using low-level safety controllers to intervene whenever a robot planner leads the robot to an unsafe state. However, there was a subtle underlying assumption that any applied safety intervention did not drastically oppose the planner’s planning objective. For example, the approach developed in Chapter 3 would not be effective if the robot planning objective was to aggressively tailgate another vehicle because HJ reachability would cause the robot to perform a hard braking maneuver therefore directly opposing the planner’s objective. As such, sole reliance on low-level safety controllers to provide safety assurance for the entire autonomy stack may not be sufficient especially if the planner unknowingly leads the robot into a situation where the low-level safety controller is infeasible or starkly opposes the high-level planning objective.

In this work, we design a safe complementary planning and control stack that provides safety assurance for a robot without unduly impacting planner performance. We leverage HJ reachability theory to provide a consistent notion of safety throughout a planning and control stack. In particular, a planner and safety controller can maintain their unique purposes but share the same sentiment when it comes to understanding safety. Further, in a multi-agent setting, a robot may be in conflict as to which agent to avoid if there is a danger of colliding with multiple agents. We use HJ reachability theory to optimally select minimally-interventional safe controls that reason about collision avoidance with all agents threatening a robot’s safety. We demonstrate the efficacy of the proposed framework in a multi-agent highway setting where an autonomous car must move quickly and safely through a densely populated highway. The case study shows that by having a planner and controller share a common interpretation of safety, the overall interaction yields a good balance between safety and performance compared to reactive safety controllers.
4.2 RELATED WORK

In Chapter 3, we described works that incorporated safety considerations either in the planner or controller. While the proposed planning and control techniques may have their unique merits, they, however, are designed in isolation and therefore do not consider the interaction between the planner and controller. There have been a few works that consider the planning and control problem jointly \cite{74,75} in the context of autonomous driving (the application domain that we are focusing on in this dissertation). By considering the planning and control problems jointly, the planned trajectories can exploit the full dynamic capabilities of the system and operate at the handling limits of the vehicle which may be necessary when avoiding collision in highly dangerous situations and achieving a high level of planning performance, such as minimizing time around a race track \cite{75}. Essentially, a large nonlinear optimization problem is formulated that solves for a collision-free trajectory while constrained to a high-fidelity dynamics model that is typical of a low-level controller. The core challenge to these types of approaches is computational since the nonlinear optimization problems are generally challenging, let alone be used in a high-frequency planning loop. Despite the successful demonstration of \cite{74,75} on full-scale vehicles, the scope of their problem has so far been limited to settings with static obstacles only. Extensions to account for moving and stochastic obstacles are non-trivial and add additional complexity to an already computationally challenging problem.

In terms of safe multi-agent settings, safety guarantees are made by making assumptions on other agent’s policies, or it is assumed that we have control over multiple agents \cite{76,77}. However, for settings that we are interested in this dissertation, a robot must interact with human agents or human counterparts and therefore does not have access to the human’s policies. As such, we focus on designing a planning and control stack where the planner and controller remain modular but have a shared interpretation of safety to avoid running into conflict as to what the robot should do when faced in a potentially unsafe situation.

The key research goal of this work is to bring the planning and control modules closer together in order to provide stronger safety assurances in the context of multi-agent interactions. Solely relying on a safety controller may be insufficient because an optimal collision avoidance control may not always exist when near safety violation, or it starkly opposes the high-level planning objective, especially in the case of multi-agent scenarios. This necessitates the need to jointly design a planner and safety controller—the planner should be cognizant of what type of situations the safety controller can succeed in, while the safety controller should be aware of the planner’s objectives such that it does not unduly impact overall performance.
4.3 Multi-Agent HJ Reachability Controller

In this section, we present an optimization-based approach to synthesize a HJ reachability-based safe control strategy for multi-agent settings. Like in Chapter 3, we follow a typical decision-making and control paradigm of utilizing a high-level planner to compute a coarse trajectory and a low-level controller to track it. Focusing on the controller in this section, we assume that given a desired plan produced by the high-level planner, there is a desired $u_0$ which corresponds to optimally, with respect to some performance metric, tracking the desired plan. The goal is to apply similar ideas from Chapter 3 but for multi-agent settings.

Unfortunately, we cannot trivially perform the HJ reachability computation for multi-agent systems because HJ reachability suffers from the curse of dimensionality. Naively extending the dimensionality of the system to account for multiple agents causes the HJ reachability computation to become intractable very quickly. To address this curse, and the fact that the number of agents that a robot may be interacting with may vary over time, we consider decomposing the joint system into multiple pairwise systems for each possible human-robot pair, as shown in Figure 4.2. Then given the set of human-robot pairwise systems that are near safety violation, we can then consider the intersection of the safety-preserving control set (refer to Section 3.4.5) of each system.

When describing the multi-agent HJ reachability controller, we use the same notation introduced in Chapter 4. Let $V$ be the HJ value function computed using the relative dynamics for a human-robot system. Let $x_{\text{rel}}^{(1)}, \ldots, x_{\text{rel}}^{(J)}$ be the relative states for $J$ human-robot pairs. Let $\mathcal{J} = \{j \mid V(x_{\text{rel}}^{(j)}) \leq \epsilon, \epsilon > 0, j = 1, ..., J\}$ represent the indices where that human-robot system is near safety violation.
Then the multi-agent safety-preserving control set is,

$$U^*_{\text{multi}}(x_{\text{rel}}^j) := \bigcap_{j \in \mathcal{J}} U^*_{\text{R}}(x_{\text{rel}}^j). \quad (4.1)$$

Unfortunately, depending on the scenario, there is no guarantee that $U^*_{\text{multi}}(x_{\text{rel}}^j)$ is non-empty, let alone that the desired control input $u_0$ is inside $U^*_{\text{multi}}(x_{\text{rel}}^j)$. To address these issues, we propose designing (i) an optimization problem that considers both a tracking error objective and safety constraints for multiple agents, and (ii) a high-level planner that is cognizant of the HJ reachability considerations in the low-level controller. More details on the HJ reachability-aware planner is presented in Section 4.4.

Assuming that $u_0$ is given, there is still the problem of selecting a control that respects both what the planner wants to do, and the safety considerations derived via HJ reachability analysis. To this end, we propose the following optimization problem that optimizes for a $u_R$ that is as close to $u_0$ as possible while achieving the least amount of violation as possible with respect to (4.1). For a value HJ value function $V$ computed for a human-robot system, cost matrix $R$ that penalizes the tracking error, slack variables $\eta_j$, robot controls $u_R$, human controls $u_H$, and robot control bounds $\underline{u}$ and $\overline{u}$, the following optimization problem synthesizes minimally interventional controls for multi-agent interactions,

$$u_R^* = \arg\min_{u_R, \eta_1, \ldots, \eta_J} (u_R - u_0)^TR(u_R - u_0) + \gamma \max_{j \in \mathcal{J}} \eta_j$$

s.t. $\min_{u_H} \nabla V(x_{\text{rel}}^j)^T f_{\text{rel}}(x_{\text{rel}}^j, u_R, u_H) \geq -\eta_j$

$$\eta_j \geq 0, \ \forall j \in \mathcal{J}$$

$$\underline{u} \leq u_R \leq \overline{u}. \quad (4.2)$$

If the relative dynamics $f_{\text{rel}}$ are affine in robot and human controls, which is the case for many car-like dynamics (e.g., dynamically-extended simple car), then (4.2) is convex, and therefore can be solved efficiently. In the case where $U^*_{\text{multi}}(x_{\text{rel}}^j) \neq \emptyset$, the positivity of the slack variables in (4.2) incentivizes the solution to satisfy the HJ safety-preserving control constraint, but no further incentive is given as to how far into the HJ safety-preserving set the solution is. Where the solution lies inside the HJ safety-preserving set (if non-empty) is determined by where the desired control is. Illustrations of when $U^*_{\text{multi}}(x_{\text{rel}}^j) \neq \emptyset$ are shown by the green crosses in Figures 4.3a and 4.3b. In the case where $U^*_{\text{multi}}(x_{\text{rel}}^j) = \emptyset$ (see Figure 4.3c), (4.2) strives to minimize a weighted combination of the degree of violation of being outside each $U^*_{\text{multi}}(x_{\text{rel}}^j), i \in \mathcal{J}$ and distance to the desired control.

Since we are considering multiple agents, choosing the optimal HJ reachability controller described in (3.5) is ill-defined since there are possibly multiple agents that are near safety violation. Instead, we can derive HJ switching controls for multi-agent interactions by removing the tracking
error objective and the positivity constraint of the slack variables in (4.2),

\begin{align}
    u^*_R &= \arg\min_{u_R, \eta_1, \ldots, \eta_J} \max_{j \in J} \eta_j \\
    &\text{s.t.} \min_{u_H} \nabla V(x^{(j)}_{rel})^T f_{rel}(x^{(j)}_{rel}, u_R, u_H) \geq -\eta_j \\
    &\forall j \in J \\
    &u \leq u_R \leq \bar{u}.
\end{align}

(4.3)

Intuitively, removing the tracking error cost and positivity of slack variables constraint incentivizes the solution to be as far into the interior of the \(U^*_{\text{multi}}(x^{J}_{rel})\) as possible while ignoring what the desired control is. Illustrated in Figure 4.3, the optimal switching control, denoted by a red cross, is equidistant from the boundary of \(U^*_{\text{multi}}(x^{J}_{rel})\).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure4_3.png}
\caption{Illustration of optimized control using (4.2) and (4.3) under different settings for \(U^*_{\text{multi}}(x^{J}_{rel})\).}
\end{figure}

Naturally, it is desirable that \(U^*_{\text{multi}}(x^{J}_{rel}) \neq \emptyset\), and that \(u_0\) is close to \(U^*_{\text{multi}}(x^{J}_{rel})\). In the next section, we describe how we can encourage the planner to choose plans to achieve this desirable setup.

### 4.4 Reachability-Based Reward Augmentation

In this section, we describe a reward augmentation for a multi-agent interaction planner that takes into account the HJ reachability considerations of the low-level controller. Specifically, via the proposed augmentation, the planner is encouraged to choose plans that will lead the robot to avoid regions where \(U^*_{\text{multi}}(x^{J}_{rel}) = \emptyset\), such as regions where the robot can be boxed-in by other agents. Essentially, the planner is encouraged to guide the robot to regions where, if the safety constraints were activated, the low-level controller will be the most useful.

We note that our proposed reward augmentation does not provide any guarantees on whether or not \(U^*_{\text{multi}}(x^{J}_{rel})\) is non-empty. Our proposed reward-augmentation, however, encourages the robot...
to select plans that are favorable by the low-level HJ controller. In Section 4.5, we investigate a multi-agent highway setting and show that with our proposed HJ reachability planning and control stack, the robot achieves the best balance between safety and planning performance.

We consider a multi-agent setting where a robot is operating in an environment with multiple agents not controlled by the robot (e.g., humans). Let $z_R^{(t)} \in Z_R \subset \mathbb{R}^{m_R}$ and $a_R^{(t)} \in A_R \subset \mathbb{R}^{m_A}$ be the robot planner state and action at time step $t$ respectively. Let $z_O^{(t)} \in Z_O \subset \mathbb{R}^{m_O}$ and $a_O^{(t)} \in U_O \subset \mathbb{R}^{m_O}$ be the state and control of $J$ other agents in the environment at time step $t$.

Further, let the time-invariant and discrete-time state space dynamics for a robot and other agents in the environment be given respectively by, $z_R^{(t+1)} = f_R(z_R^{(t)}, a_R^{(t)})$, $z_O^{(t+1)} = f_O(z_O^{(t)}, a_O^{(t)})$.

The goal of a robot planner is to find a sequence of robot actions $a_R^{(t:t+N)} = \pi(z_R^{(t)}, z_O^{(t)})$ that maximizes an expected reward $R(z_R^{(t)}, a_R^{(t)}, z_O^{(t)})$ over a fixed horizon of length $N$. That is, we want to find a solution to the following maximization problem

$$
\pi^*(z_R^{(t)}, z_O^{(t)}) = \arg \max_{a_R^{(t:t+N)}} E \left[ \sum_{i=0}^{N} \gamma^i R(z_R^{(t+i)}, a_R^{(t+i)}, z_O^{(t+i)}) \right] \tag{4.4}
$$

where $\gamma \in [0, 1]$ is a discount factor. Note that for interactive scenarios, (4.4) is generally difficult to solve due to stochasticity and coupling in the dynamics between the robot and other agents, and that system may not be Markovian (i.e., depend on interaction history). We assume that such planners typically operate at around $\leq 10$ Hz. As such, they are not always able to account for split-second threats, necessitating the need for a safety controller to intervene, such as the one described in Section 4.3.

The reward in (4.4) reflects desired goals, such as reaching a goal state, maintaining speed, or reducing time. We propose adding a term encompassing the HJI value function between the robot and all other agents into a planner’s reward function,

$$
R_{\text{total}}(z_R, a_R, z_O, x_{rel}) = \gamma_R R(z_R, a_R, z_O) + (1 - \gamma_R) R_{\text{HJI}}(x_{rel}), \tag{4.5}
$$

where $x_{rel}$ is the relative state between the robot and all other agents, and $\gamma_R$ is a constant that controls the weighting between the nominal planning reward term $R$ and the HJ reward term $R_{\text{HJI}}$. We note that the states describing $x_{rel}$ are often different than what $z_R$ describes. As a consequence, this reward augmentation scheme will result in a state augmentation as well. The impact of the state augmentation on the planning complexity will depend on the planner algorithm used.

The designer can choose how to define $R_{\text{HJI}}(x_{rel})$ depending on the application. For example, a reasonable choice could be $R_{\text{HJI}}(x_{rel}) = V(\tau, x_{rel}^{(i)})$. The designer can also adjust $\gamma_R$ to tune the robot’s preference between (nominal) reward-seeking or safety-seeking behavior.

If the target set $T$ when computing the HJ value function is the set of collision states (or a proxy for safety), the HJ value function represents an upper bound on how safe the future will be.
under worst-case disturbances. With all else equal, a planner with an additional HJI term in its reward function will select plans that will avoid states of possible inevitable collision, such as being boxed in by other agents. We make little assumptions on the structure of the planner—the planner could reason about the interaction dynamics jointly unlike our pairwise HJ reachability formulation. The only assumption we make is that the planner strives to maximize a reward function, and can accommodate the addition of the proposed HJI term. For instance, this could be applied to a search-based planner described in Chapter 2.

4.5 Case Study: Highway Driving

In this section, we investigate a multi-agent highway driving case study where the goal is to design an autonomous car (the robot) to drive through traffic as fast as possible while staying safe. The goals of weaving as fast as possible through traffic and staying safe are inherently in conflict with each other. We strive to show that with our provided planning and control stack, the robot achieves a good balance between safety and performance—it is still able to accomplish a high, albeit decreased, performance score while maintaining a minimum level of acceptable safety.

4.5.1 Experimental Set-Up

The highway environments are conducted in a simulation environment (see Figure 4.4) developed by [2]. A robot car is tasked to drive through traffic as fast as possible while avoiding collision. The other cars on the road interact with each other using the Intelligent Driver Model (IDM) [78, 79] and minimizing-overall-braking-induced-by-lane-change (MOBIL) model [80] for longitudinal and lateral control, respectively. These are common, and well-studied traffic flow models. Our results, however, do not depend critically on this modeling choice. The high-level planner runs at 1 Hz while the low-level controller runs at 50 Hz.

4.5.2 Robot Planner

We now describe the details of the planning algorithm used in this case study.
4.5. CASE STUDY: HIGHWAY DRIVING

High-level Planner

We formulate the highway environment as a Markov decision process (MDP) and apply the optimistic planning (OP) algorithm proposed in [81]. OP is a tree search algorithm where each branch represents a possible future and the tree is explored optimistically. We omit implementation details about the planner since our proposed approach is agnostic to the type of planner used. Instead, we highlight some key features regarding our planner that is representative of the class of planners that our approach applies to. First, our planner relies on a model of the environment, but in general, this model is only an approximation of the true system. The existence of model mismatch is prevalent in many model-based control problems, including those for stochastic environments. In this case study, the robot does not have access to the true modeling parameters of the other cars which are drawn from a normal distribution but instead assumes the mean of the distribution. Second, our planner is reward-based and safety is promoted via the objective function. As such, there will be times where the robot will end up in an unsafe state, necessitating the use of a safety controller.

Planner Dynamics and Reward Function

The robot planner state is \( z_R = (s_p, \ell_p, v_p) \) where \( s_p \) represents the longitudinal distance along a lane, \( \ell_p \) is the lane index, and \( v_p \) is the velocity of the robot. The state representing the \( J \) other cars \( z_O \) is also of this structure, \( z_O = (s_p^{(1)}, \ell_p^{(1)}, v_p^{(1)}), \ldots, (s_p^{(J)}, \ell_p^{(J)}, v_p^{(J)}) \). The action space of the planner is \( \mathcal{U} = \{ \text{increase } v_p \text{ by } 1ms^{-1}, \text{ decrease } v_p \text{ by } 1ms^{-1}, \text{ left lane change, right lane change, idle} \} \). Given this state and action representation, we design the following reward function (without the HJI term),

\[
R(z_R, a_R, z_O) = r_{\text{speed}}(z_R) + r_{\text{crash}}(z_R, z_O) + r_{\text{lane}}(z_R)
\]

\[
r_{\text{speed}}(z_R) = \gamma_1 \frac{v_p - \underline{v}}{\bar{v} - \underline{v}}, \quad r_{\text{lane}}(z_R) = \gamma_2 \frac{\ell_p - \ell}{\bar{\ell} - \ell},
\]

\[
r_{\text{crash}}(z_R, z_O) = -\gamma_3 1[z_R \text{ in collision}],
\]

where \( \underline{v} \) and \( \bar{v} \) corresponds to the speed limits, \( \ell \) and \( \bar{\ell} \) corresponds to the right-most and left-most lanes on the highway. We select \( \gamma_1 = 0.4, \gamma_2 = 1.0, \gamma_3 = 1.0, \underline{v} = 15ms^{-1}, \bar{v} = 30ms^{-1} \) for our experiment. This reward function is designed to encourage the robot to stay in the left most lane, maintain high speed, and avoid collision states. In order to test the effectiveness of our proposed safety controller, we encourage the robot to drive dangerously and weave through dense traffic at a high speed.

4.5.3 Robot Controller

In this section, we describe the low-level control dynamics, the computation of the desired control input \( u_0 \), and the HJ value function computation.
Low-level Controller

We use the dynamically extended simple car model for the low-level dynamics of the robot car and all other cars. Let the low-level controller state be $x = (p_x, p_y, \theta, v)$ and control input be $u = (\delta, a)$ (for ease of notation, we drop the subscript $r$ and $o$ denoting robot and the other cars). The equations of motion for the dynamically extended simple car model is,

$$
\begin{bmatrix}
\dot{p}_x \\
\dot{p}_y \\
\dot{\theta} \\
\dot{v}
\end{bmatrix} =
\begin{bmatrix}
v \cos \theta \\
v \sin \theta \\
v \tan \delta \\
a
\end{bmatrix}
$$

(4.7)

where $L$ is the length of the car, $p_x$ and $p_y$ are the longitudinal and lateral positions of the car in a fixed inertial reference frame respectively, $v$ is the signed speed of the car, and $\delta$ and $a$ are the steering and acceleration commands, respectively.

For the robot car, a closed-loop feedback controller is deployed to track the desired planner trajectory. In the case of tracking a sequence of waypoint, let $z_R = (s_p, \ell_p, v_p)$ be the desired planner state. Let $x_R = (p_{x,R}, p_{y,R}, \theta_R, v_R)$ be the low-level controller state for the robot and $\Delta \ell$ be the signed lateral distance between the robot and the center line of $\ell_p$ (left of the centerline is positive). The closed-loop feedback controller \[2\] that computes low-level controls to track the desired planner state $u_R = (\delta, a)$ is,

$$
\delta_R = \arctan\left(-\frac{LK_\theta}{v_R} \left[ \theta_R + \arcsin\left( \frac{K_1 \Delta \ell}{v_R} \right) \right]\right),
\quad a_R = K_2 (v_p - v_R)
$$

(4.8)

where $K_\theta$, $K_1$, and $K_2$ are control gains to be chosen. In our experiments, $K_\theta = 5.0$, $K_1 = 2.0$, and $K_2 = 1.67$.

HJI Relative Dynamics

To compute the HJI value function for a robot-agent system, we need relative dynamics of the pairwise system. To balance between modeling fidelity and dimension of the system (recall that HJ reachability suffers from the curse of dimensionality), we use the dynamically extended unicycle model for the robot, and a simplified unicycle model for the other car. For ease of notation, we drop the superscript denoting the $j$th pair, but the following equations are referring to a particular
robot-agent pair. The dynamics for the robot and other car used for HJI value computations are,

\[
\begin{bmatrix}
\dot{p}_x, R \\
\dot{p}_y, R \\
\dot{\theta}_R \\
\dot{v}_R \\
\end{bmatrix}
= \begin{bmatrix}
v_R \cos \theta_R \\
v_R \sin \theta_R \\
\omega_R \\
a_R \\
\end{bmatrix},
\quad
\begin{bmatrix}
\dot{p}_x, O \\
\dot{p}_y, O \\
\dot{\theta}_O \\
\dot{v}_O \\
\end{bmatrix}
= \begin{bmatrix}
v_O \cos \theta_O \\
v_O \sin \theta_O \\
a_O \\
\end{bmatrix},
\]

(4.9)

where \( u_{R, HJI} = (\omega_R, a_R) \) and \( u_{O, HJI} = (\theta_O, a_O) \) are robot and other car controls, respectively. We assume the control limits: \( \omega_R \leq \omega_R \leq \bar{\omega}_R, a_R \leq a_R \leq \bar{a}_R, \) and \( \dot{\theta}_O \leq \theta_O \leq \bar{\theta}_O. \) We select dynamics slightly different to (4.7) to ensure tractability of (4.1) (see Section 4.3), and simpler dynamics when modeling the other cars to (i) provide some conservatism to our model by assuming the other cars are more agile than the robot car, and (ii) prevent the relative state from becoming too large since the value function computation suffers from the curse of dimensionality. We define the relative coordinate frame to be aligned with the inertial frame and take the difference in position coordinates, \((p_x, rel, p_y, rel) = (p_x, R - p_x, O, p_y, R - p_y, O)\). Let the relative state between the robot and the \( j \)th other agent be (still dropping the superscript \( j \) for ease of notation) \( x_{rel} = (p_x, rel, p_y, rel, \theta_R, v_R, v_O) \). The relative dynamics become,

\[
\begin{align*}
\dot{p}_x, rel &= v_R \cos \theta_R - v_O \cos \theta_O \\
\dot{p}_y, rel &= v_R \sin \theta_R - v_O \sin \theta_O \\
\dot{\theta}_R &= \omega_R \\
\dot{v}_R &= a_R \\
\dot{v}_O &= a_O \\
\end{align*}
\]

(4.10)

**HJI Value Function**

To compute the value function which represents the set of states the robot wants to avoid, rather than using the signed distance function to define the target set which is typically used in HJ reachability literature, we instead use the definition of Responsibility-Sensitive Safety (RSS) introduced in [82]. We consider unsafe states to be a function of both relative position and velocity. For brevity, we refer the reader to [82] for the mathematical formulation but provide a brief description here. Safety is decoupled into longitudinal and lateral components. The longitudinal stopping, \( d_{long} \), is defined as the distance between a front and rear car if the front car applies maximum braking, and the rear car accelerates maximally over a response time before applying maximum braking. Analogously, \( d_{lat} \), is the minimum lateral distance. A neighboring car is considered unsafe if both the longitudinal and lateral distances between the robot car and that car are less than \( d_{long} \) and \( d_{lat} \), respectively. Given the velocities corresponding to a particular \( x_{rel} \), we define \( V(0, x_{rel}) \) as follows:

\[
V(0, x_{rel}) = \max(\vert p_{x, rel} \vert - d_{long}, 4(\vert p_{y, rel} \vert - d_{lat})^3). \]

(4.11)
and compute the HJ value function for a time horizon of 3 seconds.

**HJI Safety Controller**

To compute the safety-preserving control set, let

\[
\nabla V(t, x_{rel}) = \left( \frac{\partial V}{\partial p_{x,rel}}, \frac{\partial V}{\partial p_{y,rel}}, \frac{\partial V}{\partial \theta_R}, \frac{\partial V}{\partial v_R}, \frac{\partial V}{\partial v_O} \right).
\]

For ease of notation, we temporarily drop the superscript \(j\), but the above notation is referring to a particular robot-agent pair. Then, we can write,

\[
\min_{u_O \in U_O} \nabla V(t, x_{rel})^T f_{rel}(x_{rel}, u_R, u_O) = c_1 + c_2 + \frac{\partial V}{\partial \theta_R} \omega_R + \frac{\partial V}{v_R} a_R,
\]

where \(c_0 = c_1 + c_2\) represents components not dependent on \(u_R\) and \(a_R\). From (4.2) and using (4.12), the minimally interventional multi-agent safety-preserving control is the solution to the optimization problem (now bringing back the superscript notation to indicate different robot-agent pairs),

\[
\min_{\omega_R, a_R} \lambda_1 (\omega_R - \omega_{des})^2 + \lambda_2 (a_R - a_{des})^2 + \lambda_3 \max_{j \in J} \eta_j
\]

\[\text{s.t. } \frac{\partial V}{\partial \theta_R} (j) \omega_R + \frac{\partial V}{v_R} (j) a_R \geq -c_0(j) - \eta_j \quad \forall j \in J \]

\[\eta_j \geq 0 \quad \forall j \in J \]

\[\omega_R \leq \omega_R \leq \omega_R \]

\[a_R \leq a_R \leq a_R,\]

where \(\lambda_i > 0\) are weights (we use \(\lambda_1 = \sigma_R^{-2}, \lambda_2 = \sigma_R^{-2}, \lambda_3 = 1\)), \(\eta_j\) are slack variables, and \(\omega_{des}\) and \(a_{des}\) are the desired controls from (4.8) (to map the HJI safety control \(\omega\) to \(\delta\), we use \(\delta = \tan^{-1} \frac{\omega}{v}\)). The safety controller will choose controls that balance satisfying, or violating, each safety constraint equally while being near the previous steering input. Alternatively, the robot can use a switching strategy by letting \(\lambda_3 = 0\), \(\omega_{des} = \omega_{prev}\) (to discourage discontinuous steering inputs), and removing the \(\eta_j \geq 0\) constraint. By our choice of HJI dynamics (4.9), the optimization problem (4.13) is convex and we solve it using CVXPY [83].
4.5.4 Results

To evaluate the benefits of our proposed planning and control stack, we perform an ablation study and compare our approach to the RSS policy proposed in [82]. For the high-level planner, we investigate the following configurations,

- **OP**: An OP planner only, i.e., $\gamma_R = 1$ in (4.5).
- **HJOP**: An OP planner with a HJI reward term, i.e., $\gamma_R = 0.9$ in (4.5).

With a fixed planner (HJOP or OP), we investigate different low-level safety control strategies used either in a switching (SW), or minimally interventional (MI) scheme (see Section 4.3 and 4.5.3 for the formulation):

- **None**: No safety controller is used.
- **RSS**: The RSS proper response policy proposed in [82] provides the minimum longitudinal and lateral acceleration necessary to maintain safety. We use the RSS proper response set instead of the HJI constraint in (4.13).
- **SPC**: Our proposed multi-agent HJI safety-preserving controller (SPC) in (4.13).

To compare each approach, we use the following metrics,

- **Time-to-Collision (TTC)**: The estimated time before collision assuming both vehicles continue at constant speed [84]. Lower values indicate a more dangerous situation, while higher values are better with diminishing returns.
- **Brake Threat Number (BTN) and Steer Threat Number (STN)**: The required longitudinal and lateral acceleration for collision avoidance as defined in [85] divided by maximum available longitudinal/lateral acceleration. Lower values indicate safer situations.
- **Mean velocity**: The mean speed of the robot car over all sampled points. Higher values indicate better performance based on the planner’s reward function.
- **Mean acceleration magnitude**: The mean magnitude of acceleration of the robot car over all sample points. Lower values indicate more efficient and smoother driving.
- **Intervention percentage**: Percentage of samples where the safety controller stepped in. Lower values imply a safer planner.

We performed 20 episodes of 30-second highway simulation with 50 Hz data collection for each planner-controller configuration. Each episode consists of 100 other vehicles that the robot needs to weave through as shown in Figure 4.4. This setup provides a sufficiently dense and challenging environment to evaluate the effectiveness of each planner-controller configuration.
Chapter 4. Extensions to Safe Multi-Agent Interactions

Planner Controller Scheme | TTC ≥ 3 percentile | TTC 10\textsuperscript{th} percentile | BTN ≤ 1 percentile | BTN 90\textsuperscript{th} percentile | STN ≤ 1 percentile | STN 90\textsuperscript{th} percentile
---|---|---|---|---|---|---
1 HJOP SPC SW | 1.000 | 9.927 | 1.000 | 0.079 | 1.000 | 0.016
2 HJOP RSS SW | 0.996 | 17.125 | 0.998 | 0.030 | 0.994 | 0.006
3 **HJOP SPC MI** | 0.999 | 9.607 | 0.995 | 0.109 | 0.994 | 0.017
4 HJOP RSS MI | 0.996 | 13.914 | 0.998 | 0.028 | 0.995 | 0.008
5 HJOP None | 0.956 | 7.009 | 0.975 | 0.213 | 0.982 | 0.034
6 OP SPC SW | 0.996 | 7.045 | 0.977 | 0.152 | 0.960 | 0.084
7 OP RSS SW | 0.998 | 13.148 | 0.999 | 0.030 | 0.998 | 0.009
8 OP SPC MI | 0.999 | 8.417 | 0.988 | 0.091 | 0.984 | 0.040
9 OP RSS MI | 0.997 | 13.971 | 1.000 | 0.024 | 0.998 | 0.008
10 OP None | 0.502 | 0.785 | 0.714 | 5.616 | 0.782 | 3.189

Table 4.1: Safety and control statistics for different planner and safety controller configurations. The values in the TTC ≥ 3, BTN ≤ 1, and STN ≤ 1 column represent the fraction of samples that satisfy the inequality. Our proposed method is in bold (Row 3).

| Planner | Controller | Scheme | Mean $v_R$ (m/s) | Mean $|a_R|$ (m/s\textsuperscript{2}) | Interventions % |
|---|---|---|---|---|---|
| Row | ↑ / ↓: higher/lower is better | ↑ | ↓ | - |
| 1 | HJOP | SPC | SW | 21.878 | 1.273 | 9.5 |
| 2 | HJOP | RSS | SW | 20.343 | 4.066 | 55.0 |
| 3 **HJOP** | SPC | MI | 22.000 | 1.154 | 18.1 |
| 4 | HJOP | RSS | MI | 20.301 | 2.788 | 57.5 |
| 5 | HJOP None | — | 22.554 | 0.416 | 0.0 |
| 6 | OP | SPC | SW | 21.144 | 5.940 | 51.7 |
| 7 | OP | RSS | SW | 19.571 | 5.398 | 71.2 |
| 8 | OP | SPC | MI | 21.039 | 3.470 | 63.2 |
| 9 | OP | RSS | MI | 21.074 | 2.596 | 67.2 |
| 10 | OP None | — | 28.141 | 0.386 | 0.0 |

Table 4.2: Performance statistics for different planner and safety controller configurations. Our proposed method is in bold (Row 3).

Statistics of the simulations are listed in Table 4.1 and 4.2 while the trade-off between safety, measured by the 1\textsuperscript{st} percentile TTC, and performance, measured by mean speed, is shown in Figure 4.5. We select TTC = 3 as the boundary between unsafe and safe to reflect the popular “three-second rule” while driving, but note that this threshold could be different.

We highlight four key takeaways from these results.

**Takeaway # 1:** Consideration of only geometry-based safety is insufficient in achieving high safety metrics in highly dynamic environments. Looking at Table 4.1 there is a stark difference in the TTC values in Rows 5 and 10, a planner with and without a HJ reward term and without any safety controllers in place. This difference indicates that by incorporating a “dynamics-aware” safety metric, namely the HJ value function, into the planning objective, we immediately see a large difference in safety performance. This is because the planner only has a finite planning horizon and...
therefore the collision-avoidance term in the nominal reward function only penalizes the robot if a collision occurs, but not for being in states where collision is inevitable, e.g., traveling at a relatively higher speed closely behind another agent.

**Takeaway # 2**: Neglecting closed-loop interactions when reasoning about safety can result in overly-conservative behaviors. In Table 4.1 comparing rows 1, 3, 6, 8 which correspond to control configurations using HJ reachability with rows 2, 4, 7, 9 which correspond to control configuration using RSS, we see that the latter results in significantly higher TTC $10^{th}$ percentile values. This result indicates that RSS, although producing safe behaviors since almost all samples are TTC $\geq 3$, performs perhaps too conservatively and defensively since larger TTC values have diminishing returns. The main difference between how safety is defined via HJ reachability and via RSS, is that the former reasons about closed-loop reactions between agents while RSS does not. Specifically, RSS makes an open-loop assumption about how agents will perform—everyone performs a hard brake prefixed by some reaction time. Therefore RSS will rule out regions as unsafe when in actuality those regions can be considered safe if the ego were able to perform a maneuverer different from the predetermined hard brake (e.g., braking and swerving).

**Takeaway # 3**: Aligning notions of safety in the planner and controller results in more efficient behaviors. In Table 4.2 we see that when the planner and controller both use HJ reachability-based safety (rows 1 and 3), the mean acceleration and amount of interventions are significantly lower than when the planner and controller use different notions of safety (rows 2, 4, 6, 7, 8, 9). By providing conceptual consistency regarding safety between the planner and controller such that the planner will avoid regions where the controller also considers unsafe, we immediately observe smoother and more efficient behaviors.

**Takeaway # 4**: Using a minimally interventional safety control strategy results in slightly more efficient and higher-performing behaviors. In Table 4.2 we see that using a minimally interventional control strategy (rows 3, 4, 8, 9) in general results in slightly higher mean speeds and notably lower mean accelerations.

In addition to the statistics presented in Table 4.1 and 4.2 we plot the safety-performance trade-off of each planning-control regime and this is illustrated in Figure 4.5. Safety is represented as the $1^{st}$ percentile TCC while performance is measured in terms of mean speed. The top right corner corresponds to the region with the highest safety and performance values. The OP+None configuration (row 10) clearly results in the robot traveling at high speeds but the most unsafe. Although the RSS controllers with HJOP provide higher safety metrics, the diminishing return nature of TTC indicates that our proposed method HJOP+SPC (green) is in the ideal region—it is above the TTC = 3 line and furthest to the right.
CHAPTER 4. EXTENSIONS TO SAFE MULTI-AGENT INTERACTIONS

Figure 4.5: The trade-off between safety (1st percentile TTC) and performance (mean speed) computed from samples taken over ten-second intervals across each episode. Three standard deviation ellipses are shown.

4.6 Summary

By sharing the same interpretation of what it means to be safe, the planning and control modules complement each other providing safe yet performant behaviors. Our proposed approach leverages HJ reachability theory and, as demonstrated with a multi-agent highway driving scenario, equips a robot with the foresight to avoid regions of possible inevitable collision, and the ability to minimally deviate from the desired trajectory to the extent necessary to avoid collision with multiple agents.

As mentioned earlier in this section, we address the curse of dimensionality of HJ reachability when dealing with multiple agents by considering a set of independent pairwise interactions between the robot and each agent. Such a decomposition ignores interactions between other agents and therefore may lead to a sub-optimal solution because the pairwise decomposition is unable to capture the possibility of multiple other agents colluding to cause a collision with the robot.

We propose three future research directions; the first would be to extend the idea of using a HJI value function as part of the objective function beyond planning algorithms, such as for trajectory optimization, or as a feature in inverse reinforcement learning. The second would entail exploring different ways to define the target set (i.e., $V(0, x)$) when computing the value function, and understanding how it affects the safety-performance trade-off. The third is to design smarter priority assignment, such as through chance constraints, when safety is nearly violated by multiple agents.
Part II

Logical Reasoning in Robot Learning
So far, we have been studying how to provide safety assurances for a robot’s planning and control stack via HJ reachability analysis. In particular, we have developed a minimally interventional safe planning and control strategy that leads a robot to take evasive control without unnecessarily compromising on planning performance whenever the robot is near safety violation. We also showed that especially for multi-agent scenarios, it is important for both a planner and controller to share the same sentiment when it comes to safety otherwise a robot will find itself in situations where the existence of a safe control strategy is not possible.

Now, we will circle back to the first challenge we addressed in this dissertation, which is, how do we model interactions? Specifically, how do we model interactions in a meaningful and structured manner that can enhance the planning and control pipeline. The approach we started with in Chapter 2 was a purely phenomenological approach whereby we directly learned a distribution over future human actions from data. However, in many human-robot interactions such as in autonomous driving or social navigation, there is logical structure in how interactions play out. For instance, there are road rules that dictate how drivers on the road should operate. As such, leveraging such structure can improve prediction performance and also provide a prior, or inductive bias, for what a model should be producing.

To this end, the second part of this dissertation is focused on developing a computational paradigm, namely stlcg, that enables the encoding of logical specifications within gradient-based methods, a class of methods that is ubiquitous in robot decision-making and control, e.g., trajectory optimization, neural network training, and control synthesis. Specifically, we focus on using Signal Temporal Logic (STL), a formal logic language used to express spatio-temporal properties of signals, and leverage its quantitative semantics which describes how robust a signal is with respect to an STL specification. The robustness value of a signal with respect to an STL specification is a measure of how much the signal satisfies or violates the STL specification, and it is precisely this notion of robustness that enables us to differentiate (i.e., take gradients) through temporal logic specifications. We also provide examples of how stlcg can be applied in a number of applications, and in Chapter 6 we take a deeper dive into how stlcg can be used to synthesis a neural network controller capable of producing trajectories that satisfy STL specifications. While neural network models have led to tremendous progress in the field of artificial intelligence, controls, and robotics, the modeling capabilities of neural networks is mainly dictated by the richness of the dataset used to train a neural network. In low-data regimes, e.g., settings where it is difficult to collect data, designing robust neural network models remains a challenge. In Chapter 6 we investigate how STL can be used in low-data regimes to provide the necessary structure to train neural network controllers to perform a desired task.
Chapter 5

Backpropagation Through
STL Specifications

In this chapter, we introduce STL and describe a technique, stlcg, that transcribes STL quantitative semantics into computation graphs. stlcg provides a platform which enables the incorporation of logical specifications into robotics problems that benefit from gradient-based solutions. With this representation, and by leveraging modern automatic differentiation tools, we can efficiently backpropagate through STL robustness formulas and hence enable a natural and easy-to-use integration with many gradient-based approaches used in robotics. We demonstrate, through examples stemming from various robotics applications, that stlcg is versatile, computationally efficient, and capable of incorporating spatio-temporal information into the problem formulation.

5.1 Introduction

There are rules or spatio-temporal constraints that govern how a system should operate. Depending on the application, a system could refer to a robotic system (e.g., autonomous car or drone), or components within an autonomy stack (e.g., controller, monitor, or predictor). Such rules or constraints can be explicitly known based on the problem definition or stem from human expert knowledge. For instance, road rules largely dictate how drivers on the road should behave, or search-and-rescue protocols prescribe how a drone should survey multiple regions of a forest. Knowledge of such rules is not only critical for successful deployment but is also a useful form of inductive bias when designing such a system.

A common approach to translating rules or spatio-temporal constraints expressed as natural language into a mathematical representation is to use a logic-based formal language, a mathematical language that consists of logical operators and a grammar describing how to systematically combine
logical operators to build more complex logical expressions. While the choice of a logic language can be tailored towards the application, one of the most common logic languages used in robotics is Linear Temporal Logic (LTL) \[86, 87, 88, 89, 90\], a temporal logic language that can describe and specify temporal properties of a sequence of discrete states (e.g., the traffic light will always eventually turn green). For example, \[91\] uses LTL to synthesize a “fire-fighting” robot planner that patrols a “burning” house to rescue humans and remove hazardous items. Part of the popularity of LTL is rooted in the fact that many planning problems with LTL constraints, if formulated in a certain way, can be cast into an automaton for which there are well-studied and tractable methods to synthesize a planner satisfying the LTL specifications \[87\]. However, the discrete nature of LTL makes its usage limited to high-level planners and faces scalability issues when considering high-dimensional, continuous, or nonlinear systems. Importantly, LTL is incompatible with gradient-based methods since LTL operates on discrete states.

Recently, STL \[92\] was developed and, in contrast to LTL, STL is a temporal logic that is specified over dense-time real-valued signals, such as state trajectories produced from a continuous dynamical system. Furthermore, STL is equipped with quantitative semantics which describe the robustness of a signal for an STL specification, i.e., a continuous real-value that measures the degree of satisfaction or violation. By leveraging the quantitative semantics of STL, it is possible to use STL within a range of gradient-based methods. Naturally, STL is becoming increasingly popular in many fields including machine learning and deep learning \[93, 94\], reinforcement learning \[95\], and planning and control \[96, 97, 98, 99\]. While STL is attractive in many ways, there are still algorithmic and computational challenges when taking into account STL considerations. Unlike LTL, STL is not equipped with such automaton theory and thus the incorporation of STL specifications into the problem formulation may require completely new algorithmic approaches or added computational complexity.

The goal of this work is to develop a framework that enables STL to be easily incorporated in a range of applications—we view STL as a useful mathematical language and tool to provide existing techniques with added logical structure or logic-based inductive bias, but at the same time, it should not cause significant changes to the algorithmic and computational underpinnings when solving the problem. We are motivated by the wide applicability, vast adoption, computational efficiency, ease-of-use, and accessibility of modern automatic differentiation software packages that are prevalent in many popular programming languages. Additionally, many of these automatic differentiation tools are utilized by popular deep learning libraries such as PyTorch \[100\] which are used widely throughout the learning, robotics, and controls communities and beyond. To this end, we propose \texttt{stlcg}, a technique that translates any STL robustness formula into a computation graph. As a result, by leveraging readily available automatic differentiation tools, we can efficiently backpropagate through STL robustness formulas and therefore make STL amenable to gradient-based computations. Furthermore, by specifically utilizing deep learning libraries that have in-built
automatic differentiation functionalities, we are essentially casting STL and neural networks in the same computational language and thereby bridging the gap between spatio-temporal logic and deep learning.

5.2 Signal Temporal Logic (STL)

In this section, we provide the definitions and syntax for STL and describe the underlying graphical structure of STL formulas which will be used to construct a computation graph for computing STL robustness.

5.2.1 Signals

STL formulas are interpreted over signals, which we define formally as follows.

**Definition 5.2.1 (Signal).** A signal \( s_{t_0} = (x_0, t_0), (x_1, t_1), \ldots, (x_T, t_T) \) is an ordered \((t_{i-1} < t_i)\) finite sequence of states \( x_i \in \mathbb{R}^n \) and their associated times \( t_i \in \mathbb{R} \). For ease of notation, \( s \) (i.e., dropping the subscript) denotes the entire signal.

A signal represents real-valued, discrete-time outputs (i.e., continuous-time outputs sampled at finite time intervals) from any system of interest, such as a sequence of robot states, the temperature of a building, or the speed of a vehicle. In this work, we assume that the signal is sampled at uniform time steps \( \Delta t \). Further, we define a subsignal.

**Definition 5.2.2 (Subsignal).** Given a signal \( s_{t_0} \), a subsignal \( s_{t_i} \) is also a signal where \( i \geq 0, t_i \geq t_0, \) and \( s_{t_i} = (x_i, t_i), (x_{i+1}, t_{i+1}), \ldots, (x_T, t_T) \).

5.2.2 Syntax and Semantics

STL formulas are defined recursively according to the following grammar [92, 101].

\[ \phi ::= \top \mid \mu c \mid \neg \phi \mid \phi \land \psi \mid \phi U_{[a,b]} \psi. \]  \hspace{1cm} (5.1)

Like in many other STL (and LTL) literature, the STL grammar presented in (5.1) is written in Backus-Naur form, a context-free grammar commonly used by developers of programming languages to specify the syntax rules of a language. The STL grammar describes a set of rules that outline the syntax of the language and ways to construct a “string” (i.e., an STL formula). An STL formula \( \phi \) is generated by selecting an expression from the list of expressions on the right separated by the pipe symbol ( \( | \) ) in a recursive fashion. Now, we describe each element of the grammar:

- \( \top \) means true and we can set an STL formula as \( \phi = \top \)
• \( \mu_c \) is a predicate of the form \( \mu(x) > c \) where \( c \in \mathbb{R} \) and \( \mu : \mathbb{R}^n \to \mathbb{R} \) is a differentiable function that maps a state \( x \in \mathbb{R}^n \) to a scalar value. An STL formula can be defined as \( \phi = \mu_c \).

• \( \neg \phi \) denotes applying the negation operation \( \text{Not} \) onto an STL formula \( \phi \)

• \( \phi \land \psi \) denotes combining \( \phi \) with another STL formula \( \psi \) via the \( \text{And} \) operation \( \land \)

• \( \phi U_{[a,b]} \psi \) denotes combining \( \phi \) with another STL formula \( \psi \) via the \( \text{Until} \) temporal operation \( U_{[a,b]} \).

One can generate an arbitrarily complex STL formula by recursively applying STL operations. See Example 2 for a concrete example. Additionally, other commonly used logical connectives (\( \lor \) (\( \text{Or} \)) and \( \Rightarrow \) (\( \text{Implies} \))), and temporal operators (\( \Diamond \) (\( \text{Eventually} \)), and \( \Box \) (\( \text{Always} \))) can be derived using the grammar presented in (5.1),

\[
\begin{align*}
\text{(Or)} & \quad \phi \lor \psi = \neg (\neg \phi \land \neg \psi) \\
\text{(Implies)} & \quad \phi \Rightarrow \psi = \neg \phi \lor \psi \\
\text{(Eventually)} & \quad \Diamond_{[a,b]} \phi = \top U_{[a,b]} \phi \\
\text{(Always)} & \quad \Box_{[a,b]} \phi = \neg \Diamond_{[a,b]} (\neg \phi).
\end{align*}
\]

Example 2. Let \( x \in \mathbb{R}^n \), \( \phi_1 = \mu_1(x) < c_1 \) and \( \phi_2 = \mu_2(x) \geq c_2 \). Then, for a given signal \( s_t \), the formula \( \phi_3 = \phi_1 \land \phi_2 \) is true if both \( \phi_1 \) and \( \phi_2 \) are true. Similarly, \( \phi_4 = \Box_{[a,b]} \phi_3 \) is true if \( \phi_3 \) is true over the entire interval \([t+a, t+b]\).

We use the notation \( s_t \models \phi \) to denote that a signal \( s_t \) satisfies an STL formula \( \phi \) according to the formal semantics below. Informally, \( s_t \models \phi U_{[a,b]} \psi \) if there is a time \( t' \in [t+a, t+b] \) such that \( \phi \) holds for all time before \( t' \) and \( \psi \) holds at time \( t' \). Also, \( s_t \models \Diamond_{[a,b]} \phi \) if at some time \( t \in [t+a, t+b] \), \( \phi \) holds at least once, and \( s_t \models \Box_{[a,b]} \phi \) if \( \phi \) holds for all \( t \in [t+a, t+b] \). Formally, the \( \text{Boolean} \) semantics of a formula (i.e., formulas to determine if the formula is true or false) with respect to a

\[1\] Equality and the other inequality relations can be derived from the STL grammar in (5.1), i.e., \( \mu(x) < c \Leftrightarrow -\mu(x) > -c \), and \( \mu(x) = c \Leftrightarrow \neg (\mu(x) < c) \land \neg (\mu(x) > c) \).
5.2. SIGNAL TEMPORAL LOGIC (STL)

A signal $s_t$ are defined as follows.

\[
\begin{align*}
    s_t &\models \mu_c &\iff& & \mu(x_t) > c \\
    s_t &\models \neg \phi &\iff& & \neg(s_t \models \phi) \\
    s_t &\models \phi \land \psi &\iff& & (s_t \models \phi) \land (s_t \models \psi) \\
    s_t &\models \phi \lor \psi &\iff& & (s_t \models \phi) \lor (s_t \models \psi) \\
    s_t &\models \phi \Rightarrow \psi &\iff& & \neg(s_t \models \phi) \lor (s_t \models \psi) \\
    s_t &\models \diamond_{[a,b]} \phi &\iff& & \exists t' \in [t+a, t+b] \text{ s.t. } s_{t'} \models \phi \\
    s_t &\models \Box_{[a,b]} \phi &\iff& & \forall t' \in [t+a, t+b] \text{ s.t. } s_{t'} \models \phi \\
    s_t &\models \phi \mathcal{U}_{[a,b]} \psi &\iff& & \exists t' \in [t+a, t+b] \text{ s.t. } (s_{t'} \models \psi) \land (s_t \models \Box_{[0,t']} \phi)
\end{align*}
\]

STL admits a notion of robustness, that is, it has quantitative semantics that calculates the degree of satisfaction or violation of a signal given a formula. Positive robustness values indicate satisfaction, while negative robustness values indicate violation. Like Boolean semantics, the quantitative semantics of a formula with respect to a signal $s_t$ is defined as follows,

\[
\begin{align*}
    \rho(s_t, T) &= \rho_{\max} \quad \text{where } \rho_{\max} > 0 \\
    \rho(s_t, \mu_c) &= \mu(x_t) - c \\
    \rho(s_t, \neg \phi) &= -\rho(s_t, \phi) \\
    \rho(s_t, \phi \land \psi) &= \min(\rho(s_t, \phi), \rho(s_t, \psi)) \\
    \rho(s_t, \phi \lor \psi) &= \max(\rho(s_t, \phi), \rho(s_t, \psi)) \\
    \rho(s_t, \phi \Rightarrow \psi) &= \max(-\rho(s_t, \phi), \rho(s_t, \psi)) \\
    \rho(s_t, \diamond_{[a,b]} \phi) &= \max_{t' \in [t+a, t+b]} \rho(s_{t'}, \phi) \\
    \rho(s_t, \Box_{[a,b]} \phi) &= \min_{t' \in [t+a, t+b]} \rho(s_{t'}, \phi) \\
    \rho(s_t, \phi \mathcal{U}_{[a,b]} \psi) &= \max_{t' \in [t+a, t+b]} \min\{\rho(s_{t'}, \psi), \min_{\tau \in [0,t']} \rho(s_{\tau}, \phi)\}.
\end{align*}
\]

Further, we define the robustness trace as a sequence of robustness values of every subsignal $s_t$, of signal $s_{t_0}$, $t_i \geq t_0$.

**Definition 5.2.3** (Robustness trace). Given a signal $s_{t_0}$ and an STL formula $\phi$, the robustness trace $\tau(s_{t_0}, \phi)$ is a finite sequence of robustness values of $\phi$ for each subsignal of $s_{t_0}$. Specifically, $\tau(s_{t_0}, \phi) = \rho(s_{t_0}, \phi), \rho(s_{t_1}, \phi), \ldots, \rho(s_{t_{t-1}}, \phi)$ where $t_{i-1} < t_i$. 

5.2.3 Graphical Structure of STL Formulas

Recall that STL formulas are defined recursively according to the grammar in (5.1). We can represent the recursive structure of an STL formula with a parse tree \( T \) by identifying the ordering in which the STL operators of the STL formula were applied.

**Definition 5.2.4** (Subformula). Given an STL formula \( \phi \), a subformula of \( \phi \) is a formula to which the outer-most (i.e., last) STL operator is applied to. The operators \( \land, \lor, \Rightarrow \), and \( \mathcal{U} \) will have two corresponding subformulas, while predicates and True are not defined recursively and thus have no subformulas.

**Definition 5.2.5** (STL Parse Tree). An STL parse tree \( T \) represents the syntactic structure of an STL formula according to the STL grammar (defined in (5.1)). Each node represents an STL operation and it is connected to its corresponding subformula(s).

For example, the subformula of \( \Box \phi \) is \( \phi \), and the subformulas of \( \phi \land \psi \) are \( \phi \) and \( \psi \). Given an STL formula, the root node of its corresponding parse tree represents the outermost operator for the formula. This node is connected to the outermost operator of its subformula(s), and so forth. Applying this recursion, each node of the parse tree represents each operation that makes up the STL formula; the outermost operator is the root node and the innermost operators are the leaf nodes (i.e., the leaf nodes represent predicates or the terminal True operator.).

An example of a parse tree \( T_\theta \) for a formula \( \theta = \Diamond \Box (\phi \land \psi) \) is illustrated in Figure 5.1a where \( \phi \) and \( \psi \) are assumed to be predicates. By flipping the direction of all the edges in \( T_\theta \), we obtain a directed acyclic graph \( G_\theta \), shown in Figure 5.1b. The graph \( G_\theta \) defines the structure of the computation graph.
Figure 5.2: Computation graph of the mathematical operation $z = x + y$. (to be detailed in the next section). At a high level, signals are passed through the root nodes, $G_\phi$ and $G_\psi$, to produce $\tau(s, \phi)$ and $\tau(s, \psi)$, robustness traces of $\phi$ and $\psi$. The robustness traces are then passed through the next node of the graph, $G_\land$, to produce the robustness trace $\tau(s, \phi \land \psi)$, and so forth until the root node is reached and the final output will be the robustness trace of $\theta$, $\tau(s, \Box (\phi \land \psi))$.

5.3 stlcg

We first describe how to represent each STL robustness formula described in (5.2) as a computation graph, and then show how to combine them to form the overall computation graph that computes the robustness trace of any given STL formula. Further, we provide smooth approximations to the max and min operations to help make the gradients smoother when backpropagating through the graph, and introduce a new robustness formula which addresses some limitations of using the max and min functions. The resulting computational framework, stlcg, is implemented using PyTorch [100] and the code can be found at https://github.com/StanfordASL/stlcg. Further, this toolbox includes a graph visualizer, illustrated in Figure 5.1c, to show the graphical representation of the STL formula and how it depends on the inputs and parameters from the predicates.

5.3.1 Computation Graphs

A computation graph is a directed graph where the nodes correspond to operations or variables. Variable values are passed through the nodes where the operation is applied, and the outputs feed into the next connected node. For example, the computation graph of the mathematical operation $z = w(x + y)$ is illustrated in Figure 5.2 where $x$, $y$, and $w$ are inputs into the graph, $+$ and $\times$ are mathematical operations, and $z$ is the output. When the operations are differentiable, we can leverage the chain rule and therefore backpropagate through the computation graph to obtain derivatives of the output with respect to any variable that the output depends on.
In doing so, we can compute the robustness value for every subsignal of the input signal. The computation graph structure, similar to the recurrent structure of recurrent neural networks (RNN), is less so. To accomplish this, we apply dynamic programming by using a recurrent robustness trace, which is straightforward, computing the robustness trace value. Computation graph construction needs to be treated differently than the non-temporal operators. The robustness formulas for the non-temporal operators are relatively straightforward to implement as computation graphs because they involve simple mathematical operations, such as subtraction, max, and min, and do not involve looping over time. The computation graph for a predicate (\(\mu_c\)), negation (\(\neg\)), implies (\(\Rightarrow\)), and (\(\land\)), and or (\(\lor\)) are illustrated in Figure 5.3. The input \(x\) (and \(y\)) is either the signal of interest (for predicates) or the robustness trace(s) of a subformula(s) (for non-predicate operators). While the output \(z\) is the robustness trace after applying the corresponding STL robustness formula. For example, when computing the robustness trace of \(\phi \land \psi\) for a signal \(s\), the inputs for \(G\land\) are \(\tau(s, \phi)\) and \(\tau(s, \psi)\), and the output is \(\tau(s, \phi \land \psi)\).

5.3.2 Computation Graphs of STL Operators

First, we consider the computation graph of each STL robustness formula given in Section 5.2.2 individually.

Non-temporal STL Operators

The robustness formulas for the non-temporal operators are relatively straightforward to implement as computation graphs because they involve simple mathematical operations, such as subtraction, max, and min, and do not involve looping over time. The computation graph for a predicate (\(\mu_c\)), negation (\(\neg\)), implies (\(\Rightarrow\)), and (\(\land\)), and or (\(\lor\)) are illustrated in Figure 5.3. The input \(x\) (and \(y\)) is either the signal of interest (for predicates) or the robustness trace(s) of a subformula(s) (for non-predicate operators). While the output \(z\) is the robustness trace after applying the corresponding STL robustness formula. For example, when computing the robustness trace of \(\phi \land \psi\) for a signal \(s\), the inputs for \(G\land\) are \(\tau(s, \phi)\) and \(\tau(s, \psi)\), and the output is \(\tau(s, \phi \land \psi)\).

Temporal STL Operators

Due to the temporal nature of the eventually \(\Diamond_{[a,b]}\), always \(\Box_{[a,b]}\), and until \(U_{[a,b]}\) operators, the computation graph construction needs to be treated differently than the non-temporal operators. While computing the robustness value of any of the temporal operators given an input signal or robustness trace is straightforward, computing the robustness trace in a computationally efficient way is less so. To accomplish this, we apply dynamic programming by using a recurrent computation graph structure, similar to the recurrent structure of recurrent neural networks (RNN). In doing so, we can compute the robustness value for every subsignal of the input signal. The
5.3. **STLCG**

Figure 5.4: A schematic of an unrolled recurrent computation graph for the ◇ (eventually) or □ (always) temporal operators which differs only in the operation (max or min) used in the recurrent cells, depicted as orange circles. The dashed lines indicate the decomposition/reconstruction of the input/output signal via the time dimension.

The complexity of this approach is linear in the length of the signal for ◇ and □, and at most quadratic for the $\mathcal{U}$ operator.

**Eventually and Always Operators**

We consider the Eventually operator but noting that similar constructions apply for the Always operator. Let $\psi = \Diamond_{[a,b]} \phi$; the goal is to construct the computation graph $\mathcal{G}_{\Diamond_{[a,b]}}$ which takes as input $\tau(s, \phi)$ (the robustness trace of the subformula) and outputs $\tau(s, \Diamond_{[a,b]} \phi)$. For ease of notation, we denote $\rho(s_t, \phi) = \rho_t^{\phi}$ as the robustness value of $\phi$ for subsignal $s_t$. Figure 5.4 illustrates the unrolled graphical structure of $\mathcal{G}_{\Diamond_{[a,b]}}$. The $i$-th recurrent (orange) node takes in a hidden state $h_i$ and an input state $\rho_{T-i}^{\phi}$, and produces an output state $o_{T-i}$ (note the time indices are reversed in order to perform dynamic programming). By collecting all the $o_i$’s, the outputs of the computation graph form the (backward) robustness trace of $\psi$, i.e., $\tau(s, \psi)$. The output robustness trace is treated as the input to another computation graph representing the next STL operator dictated by $\mathcal{G}$.

Before describing the mechanics of the computation graph $\mathcal{G}_{\Diamond_{[a,b]}}$, we first define $M_N \in \mathbb{R}^{N \times N}$ to be a square matrix with ones in the upper off-diagonal and $B_N \in \mathbb{R}^N$ to be a vector of zeros with a one in the last entry. If $x \in \mathbb{R}^N$ and $u \in \mathbb{R}$, then the operation $M_N x + B_N u$ removes the first element of $x$, shifts all the entries up one index and replaces the last entry with $u$. We distinguish four variants of $\mathcal{G}_{\Diamond_{[a,b]}}$ which depends on the interval $[a, b]$ attached to the temporal operator.

**Case 1: $[0, \infty)$**. Let the initial hidden state be $h_0 = \rho_T^{\phi}$. The hidden state is designed to keep
track of the largest value the graph has processed has so far. The output state for the first step is
\( o_T = \max(h_0, \rho_T^0) \), and the next hidden state is defined to be \( h_1 = o_T \). Hence the general update step becomes,

\[
h_{i+1} = o_{T-i}, \quad o_{T-i} = \max(h_i, \rho_T^0).
\]

By construction, the last step in this dynamic programming step is,

\[
o_0 = \max(h_T, \rho_T^0)
o_0 = \max(\max(h_{T-1}, \rho_T^0), \rho_0^0)
\]

\[
\vdots
\]

\[
o_0 = \max(h_0, \rho_T^0, \rho_{T-1}^0, \ldots, \rho_0^0)
o_0 = \rho(s_{t_0}, \hat{\phi}) = \rho(s_{t_0}, \psi).
\]

The output states of \( G_{\hat{\psi}([0, \infty], \phi)}, o_0, o_1, \ldots, o_T \), form \( \tau(s, \hat{\psi}([0, \infty], \phi)) \), the robustness trace of \( \psi = \hat{\psi}([0, \infty], \phi) \), and the last output state \( o_0 \) is \( \rho(s, \hat{\psi}([0, \infty], \phi)) \).

**Case 2:** \([0, b], b < \infty\). Let the initial hidden state be \( h_0 = (h_{0_1}, h_{0_2}, \ldots, h_{0_{N-1}}) \), where \( h_{0_i} = \rho_T^0, \forall i = 1, \ldots, N - 1 \)\(^2\) and \( N \) is the number of time steps contained in the interval \([0, b]\). The hidden state is designed to keep track of inputs in the interval \([t_i, t_i + b]\). The output state for the first step is \( o_T = \max(h_0, \rho_T^0) \) (the max operation is over the elements in \( h_0 \) and \( \rho_T^0 \)), and the next hidden state is defined to be \( h_1 = M_{N-1} h_0 + B_{N-1} \rho_T^0 \). Hence the general update step becomes,

\[
h_{i+1} = M_{N-1} h_i + B_{N-1} \rho_T^0,
o_{T-i} = \max(h_i, \rho_T^0).
\]

By construction, \( o_0 \) corresponds to the definition of \( \rho(s_{t_0}, \hat{\psi}([0, b], \phi)), b < \infty \).

**Case 3:** \([a, \infty), a > 0\). Let the hidden state be a tuple \( h_i = (c_i, d_i) \). Intuitively, \( c_i \) keeps track of the maximum value over future time steps starting from time step \( t_i \), and \( d_i \) keeps track of all the values in the time interval \([t_i, t_i + a]\). Specifically, \( d_i = \rho_{t_i+a}^0 \). Let the initial hidden state be \( h_0 = (\rho_T^0, d_0) \) where \( d_0 = (d_{0_1}, d_{0_2}, \ldots, d_{0_{N-1}}) \), \( d_{0_i} = \rho_T^0, \forall i = 1, \ldots, N - 1 \) and \( N \) is the number of time steps encompassed in the interval \([0, a]\). The output for the first step is \( o_T = \max(c_0, d_{0_i}) \), and the next hidden state is defined to be \( h_1 = (o_T, M_{N-1} d_0 + B_{N-1} \rho_T^0) \). Following this pattern, the general update step becomes,

\[
h_{i+1} = (o_{T-i}, M_{N-1} d_i + B_{N-1} \rho_T^0),\o_{T-i} = \max(c_i, d_{i_i}).
\]

\(^2\)This corresponds to padding the input trace with the last value of the robustness trace. A different value could be chosen instead. This applies to Cases 3–4 as well.
Table 5.1: Hidden and output states of $\diamond [s > 0]$ for a signal $s = 1, 1, 2, 3, 1$ and different intervals $I$. The signal is fed into the computation graph backward.

<table>
<thead>
<tr>
<th>$\diamond [0, \infty) (s &gt; 0)$</th>
<th>$h_0$</th>
<th>$o_1$</th>
<th>$h_1$</th>
<th>$o_2$</th>
<th>$h_2$</th>
<th>$o_2$</th>
<th>$h_3$</th>
<th>$o_1$</th>
<th>$h_4$</th>
<th>$o_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\diamond [0, 1) (s &gt; 0)$</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$\diamond [0, 2) (s &gt; 0)$</td>
<td>(1, 1)</td>
<td>1</td>
<td>(1, 1)</td>
<td>3</td>
<td>(1, 3)</td>
<td>3</td>
<td>(3, 2)</td>
<td>3</td>
<td>(3, 2)</td>
<td>3</td>
</tr>
<tr>
<td>$\diamond [2, \infty) (s &gt; 0)$</td>
<td>(1, 1)</td>
<td>1</td>
<td>(1, 1)</td>
<td>1</td>
<td>(1, 1)</td>
<td>1</td>
<td>(1, 2)</td>
<td>3</td>
<td>(3, 2)</td>
<td>3</td>
</tr>
<tr>
<td>$\diamond [1, 2) (s &gt; 0)$</td>
<td>(1, 1)</td>
<td>1</td>
<td>(1, 1)</td>
<td>1</td>
<td>(1, 3)</td>
<td>3</td>
<td>(3, 2)</td>
<td>3</td>
<td>(3, 2)</td>
<td>3</td>
</tr>
</tbody>
</table>

By construction, $o_0$ corresponds to the definition of $\rho(s_{t_0}, \diamond [a, \infty) \phi)$, $a > 0$.

**Case 4:** $[a, b]$, $a > 0$, $b < \infty$. Let the initial hidden state be $h_0 = (h_{0_1}, h_{0_2}, \ldots, h_{0_{N-1}})$, where $h_{0_i} = \varphi_i^\phi$, $\forall i = 1, \ldots, N - 1$ and $N$ is the number of time steps encompassed in the interval $[0, b]$. The hidden state is designed to keep track of all the values between $(t_i, t_i + b]$. Let $M$ be the number of time steps encompassed in the $[a, b]$ interval. The output for the first step is $o_T = \max(h_{0_1}, h_{0_2}, \ldots, h_{0_M})$. The next hidden state is defined to be $h_1 = M_{N-1}h_0 + B_{N-1}\varphi_T^\phi$. Hence the general update step becomes,

$$h_{t+1} = M_{N-1}h_t + B_{N-1}\varphi_T^\phi,$$

$$o_{T-i} = \max(h_{i_1}, h_{i_2}, \ldots, h_{i_M})$$

By construction, $o_0$ corresponds to the definition of $\rho(s_{t_0}, \diamond [a, b) \phi)$, $a > 0$, $b < \infty$.

The computation graph for the Always operation is the same but instead uses the min operation instead of max. The time complexity for computing $\tau(s, \diamond [a, b) \phi)$ (and also $\tau(s, \square [a, b) \phi)$) is $O(T)$ since it requires only one pass through the signal. See Example 3 for a concrete example.

**Example 3.** Let the values of a signal be $s = 1, 1, 2, 3, 1$. Then the hidden and output states for $\diamond_I$ with different intervals $I$ are given in Table 5.1.

### Until Operator

Recall that the robustness formula for the Until operator is,

$$\rho(s_t, \phi U_{[a, b]} \psi) = \max_{t' \in [t + a, t + b]} \min_{t' \in [t + a, t + b]} \rho(s_{t'}, \psi), \min_{\tau \in [0, t']} \rho(s_{\tau}, \phi).$$

Using the definition of the Always robustness formula, the Until robustness formula can be rewritten as,

$$\rho(s_t, \phi U_{[a, b]} \psi) = \max_{t' \in [t + a, t + b]} \min_{t' \in [t + a, t + b]} \rho(s_{t'}, \psi), \rho(s_{t_0}, \square \phi)$$

(5.3)

where $s_{t_0}'$ denotes a signal consisting of values from $t = t_0$ to $t = t'$. Before we describe the computation needed to compute the robustness formula for different instantiations of $[a, b]$, we first
define the following notation. Let $\overleftarrow{s_{t_0}}$ denote the reversed signal of $s_{t_0}$. We also want to note that the inputs into $\mathcal{G}_{\phi \mathcal{U}[a,b]}$ are $\tau(s_{t_0}, \phi)$ and $\tau(s_{t_0}, \psi)$.

**Case 1:** $[0, \infty)$. For each time step $t_i$, we can easily compute $\tau(s_{t_i}, \square \phi)$ (using the methodology outlined in Section 5.3.2). Therefore, we can compute $\min \{\rho(s_{t_i}, \psi), \rho(s_{t_i}, \square \phi)\}$ for all subsignals of $s_{t_0}$. After looping through all $t_i$’s, we can perform the outer maximization in (5.3) and therefore produce $\tau(s_{t_0}, \phi \mathcal{U}_{[0,\infty)} \psi)$. Since we looped through all the values of $s_{t_0}$ once and computed the robustness trace of $\square \phi$ within each iteration, the time complexity is $O(T^2)$ where $T$ is the length of the signal.

**Case 2:** $[a, b]$, $0 \leq a < b < \infty$. First, loop through each time $t_i$. Each iteration of this loop will produce $\rho(s_{t_i}, \phi \mathcal{U}[a,b] \psi)$, the robustness value for subsignal $s_{t_i}$. For a given $t_i$, we can compute $\rho(s_{t_i}, \square \phi)$ for all $t' \in [t_i + a, t_i + b]$ by computing the robustness trace of $\square \phi$ over $\overleftarrow{s_{t_i}^{t_i+b}}$ and then taking the last $N$ values of robustness trace where $N$ is the number of time steps inside $[a,b]$. As such, we can compute $\rho(s_{t_i}, \phi \mathcal{U}[a,b] \psi) = \max_{t' \in [t_i + a, t_i + b]} \min \{\rho(s_{t'}, \psi), \rho(s_{t_i}, \square \phi)\}$. Since we loop through all time steps in the signal, and compute the Always robustness trace over $\overleftarrow{s_{t_i}^{t_i+b}}$, the time complexity is $O(TM)$ where $T$ is the length of the signal, and $M$ is the number of time steps contained in $[0, b]$.

**Case 3:** $[a, \infty)$, $0 \leq a < \infty$. This is the same as Case 2 but instead we compute $\rho(s_{t_i}, \square \phi)$ for all $t' \in [t_i + a, \infty)$ (practically speaking, we just consider up until the end of the signal). The time complexity in this case is $O(T^2)$.

We have just described how we can compute the robustness trace of the Until operation by leveraging the computation graph construction of the Always operator and a combination of for loops, and max and min operations, all of which can be expressed using computation graphs.

### 5.3.3 Calculating Robustness with Computation Graphs

Since we can now construct the computation graph corresponding to each STL operator of a given STL formula, we can now construct the computation graph $\mathcal{G}$ by stacking together computation graphs according to the parse tree $\mathcal{T}$. The overall computation graph takes a signal $s$ as its input, and outputs the robustness trace as its output. The total time complexity of computing the robustness trace for a given STL formula is at most $O(|\mathcal{T}| T^2)$ where $|\mathcal{T}|$ is the number of nodes in $\mathcal{T}$, and $T$ is the length of the input signal.

By construction, the robustness trace generated by the computation graph matches exactly the true robustness trace of the formula, and thus $\texttt{stlcg}$ is correct by construction (see Theorem 5.3.1).

**Theorem 5.3.1** (Correctness of $\texttt{stlcg}$). For any STL formula $\phi$ and any signal $s$, let $\mathcal{G}_{\phi}$ be the computation graph produced by $\texttt{stlcg}$. Then passing a signal $s$ through $\mathcal{G}_{\phi}$ produces the robustness trace $\tau(s, \phi)$. 

5.3.4 Smooth Approximations to stlcg

Due to the recursion over max and min operations, there is the potential in practice for the gradients to be numerically challenging to work with. To mitigate this, we can leverage smooth approximations to the max and min functions. Let $x \in \mathbb{R}^n$ and $w \in \mathbb{R}_{\geq 0}$, then the max and min approximations are

$$\tilde{\max}(x; w) = \frac{\sum_{i} x_i \exp(wx_i)}{\sum_{j} \exp(wx_j)}, \quad \tilde{\min}(x; w) = \frac{\sum_{i} x_i \exp(-wx_i)}{\sum_{j} \exp(-wx_j)},$$

where $x_i$ represents the $i$-th element of $x$, and $w \geq 0$ operates as a scaling parameter. The approximation approaches the true solution when $w \to \infty$ while $w = 0$ results in the mean value of the entries of $x$. The logsumexp max/min approximation can also be used instead,

$$\tilde{\max}(x; w) = \frac{1}{w} \log \sum_{i=1}^{n} \exp(wx_i), \quad \tilde{\min}(x; w) = -\frac{1}{w} \log \sum_{i=1}^{n} \exp(-wx_i)$$

where the approximation approaches the true maximum/minimum value when $w \to \infty$ where $w > 0$. The benefit over the logsumexp approximation is that the approximation error can be bounded,

$$\max(x) < \frac{1}{w} \log \sum_{i=1}^{n} \exp(wx_i) \leq \max(x) + \frac{\log(n)}{w}$$

In practice, $w$ can be annealed over gradient-descent iterations.

Further, max and min are pointwise functions. As a result, the robustness of an STL formula can be highly sensitive to one single point in the signal, potentially providing an inadequate robustness metric especially if the signal is noisy [104], or causing the gradients to accumulate to a single point. We propose using an integral-based STL robustness formula $I^M_{[a,b]}$, a variation on the Always robustness formula, and it is defined as follows. For a given weighting function $M(t)$,

$$\rho(s_t, I^M_{[a,b]} \phi) = \sum_{\tau=t+a}^{t+b} M(\tau) \rho(s_\tau, \phi).$$

The integral robustness formula considers the weighted sum of the input signal over an interval $[a, b]$. In contrast to the Always robustness formula which uses min, a pointwise function, the integral operator can produce a smoother robustness trace and this behavior is demonstrated in Section 5.4.1.
5.4 Applications of stlcg

We demonstrate the versatility and computational advantages of using stlcg by investigating a number of case studies in this section. In these examples, we show (i) how our approach can be used to incorporate logical requirements into motion planning problems (Section 5.4.1), (ii) the computational efficiency of stlcg achieved via parallelization and batching (Section 5.4.2), and (iii) how we can use stlcg to translate human-domain knowledge into a form that can be integrated with deep neural networks (Section 5.4.3). Code can be found at https://github.com/StanfordASL/stlcg.

5.4.1 Motion Planning With STL Constraints

Recently, there has been a lot of interest in motion planning with STL constraints (e.g., [96, 99, 105, 106] and see Section 6.2 for more description); the problem of finding a sequence of states and controls that drives a robot from an initial state to a final state while obeying a set of constraints which includes STL specifications. For example, in a surveying task, or an autonomous car navigating through a stop intersection, the robot, while striving to reach a goal state/region, may be required to first enter another region for a finite duration while avoiding an obstacle before moving towards its final destination. Rather than reformulating the problem as a MILP to account for STL constraints as done in [96], we consider a simpler approach of treating the STL constraints as soft constraints by penalizing constraint violation in the objective function. We augment the loss function with a robustness term, thus enabling a higher degree of customization in how we like the STL specification to be incorporated into the problem. For instance, depending on the STL specification of interest, different subformulas can be weighted differently, or have a different penalty function applied.

Consider the following motion planning problem illustrated in Figure 5.5: a robot must compute a sequence of states $X = x_{1:N}$ and controls $U = u_{1:N-1}$ that takes it from the yellow circle (position $= (-1, -1)$) to the yellow star (position $= (1, 1)$) while satisfying an STL constraint $\phi$. Assume a robot with state $x \in \mathbb{R}^2$ and control $u \in \mathbb{R}^2$ follows single integrator dynamics $\dot{x} = u$ and has a control constraint $\|u\|_2 \leq u_{\text{max}}$. We can also express the control constraint as an STL formula: $\theta = \square \|u\|_2 \leq u_{\text{max}}$.

This motion planning problem can be cast as an unconstrained optimization problem,

$$\min_{X, U} \|Ez - D\|_2^2 + \gamma_1 J_m(\rho(X, \phi)) + \gamma_2 J_m(\rho(U, \theta))$$

where $z = (X, U)$ is the concatenated vector of states and controls. Since the dynamics are linear, we can express the dynamics, and start and end point constraints across all time steps with a single linear equation $Ez = D$. The other function, $J_m$, represents the cost on the STL robustness value with margin $m$. We use $J_m(x) = \text{ReLU}(-(x - m))$ where $\text{ReLU}(x) = \max(0, x)$ is the rectified linear
5.4. APPLICATIONS OF STLCG

Figure 5.5: Motion planning with STL specifications solved using stlcg.

To showcase a variety of STL constraints, we consider four different STL specifications,

\[
\begin{align*}
\phi_1 &= \Diamond \Box [0, 5] \text{inside } B2 \land \Diamond \Box [0, 5] \text{inside } B3 \land \Box \neg \text{inside C} \\
\phi_2 &= \Diamond \Box [0, 5] \text{inside } B1 \land \Box \neg \text{inside } B3 \land \Box \neg \text{inside C} \\
\psi_1 &= \Diamond \mathcal{J}_{[0, 5]} \text{inside } B2 \land \Diamond \mathcal{J}_{[0, 5]} \text{inside } B3 \land \Box \neg \text{inside C} \\
\psi_2 &= \Diamond \mathcal{J}_{[0, 5]} \text{inside } B1 \land \neg \mathcal{J}_{[0, 5]} \text{inside } B3 \land \Box \neg \text{inside C},
\end{align*}
\]

where B1, B2, B3, and C are the regions illustrated in Figure 5.5. \(\phi_1\) translates to: the robot needs to eventually stay inside the red box (B2) and green box (B3) for five time steps each, and never enter the blue circular region (C). Similarly, \(\phi_2\) translates to: the robot eventually needs to stay inside the orange box (B1) for five time steps, and never enter the green box (B3) nor the blue circular region (C). \(\psi_1\) and \(\psi_2\) are similar to \(\phi_1\) and \(\phi_2\) except that the integral robustness formula (described in Section 5.3.4) is used instead of the Always operator.

We initialize the solution with a straight line connecting the start and end points, noting that the initialization violates the STL constraint. We then perform standard gradient descent (constant step size of 0.05) with \(\gamma_1 = \gamma_2 = 0.3\), \(m = 0.05\), and \(\Delta t^{-1} = \frac{1}{0.1}\); the solutions are illustrated in Figure 5.5. As anticipated, using the integral robustness formula (Figure 5.5b) results in a smoother trajectory and smoother control signal than using the Always robustness formula (see Figures 5.5a and 5.6). We also note that the control constraint \((u_{\text{max}} = 0.8)\) was satisfied for all cases.
CHAPTER 5. BACKPROPAGATION THROUGH STL SPECIFICATIONS

Figure 5.6: Control effort along the trajectory for all cases with $\theta = \square \|u\|_2 \leq 0.8$.

Figure 5.7: Computation time of different methods used to solve a pSTL problem. This was computed using a 3.0GHz octocore AMD Ryzen 1700 CPU and a Titan X (Pascal) GPU.

5.4.2 Parametric STL for Behavioral Clustering

In this example, we consider parametric STL (pSTL) problems \[93, 107\], and demonstrate that since modern automatic differentiation software such as PyTorch can be used to implement stlcg, we have the ability to parallelize the computation and leverage GPU hardware. The pSTL problem is a form of logic-based parameter estimation for time series data. It involves first constructing an STL template formula where the predicate parameter values and/or time intervals are unknown. The goal is to find parameter values that best fit a given signal. As such, the pSTL process can be viewed as a form of feature extraction for time-series data. Logic-based clustering can be performed by clustering on the extracted pSTL parameter values. For example, pSTL has been used to cluster human driving behaviors in the context of autonomous driving applications \[93\].
5.4. APPLICATIONS OF STLCG

The experimental setup for our example is as follows. Given a dataset of $N$ step responses from randomized second-order systems, we use pSTL to help cluster different types of responses, e.g., under-damped, critically damped, or over-damped. Based on domain knowledge of second order step responses, we design the following pSTL formulas,

$\phi_1 = ☐_{[50,100]} |s - 1| < \epsilon_1$  \hspace{1cm} \text{(final value)}

$\phi_2 = ☐ s < \epsilon_2$  \hspace{1cm} \text{(peak value)}

For each signal $s^{(j)}$, $j = 1, \ldots, N$, we want to find $\epsilon_{1j}$ and $\epsilon_{2j}$ ($\epsilon_1$ and $\epsilon_2$ for the $j$th signal) that provides the best fit, i.e., robustness equals zero. Using stlcg, we can batch our computation since each signal is decoupled from one another, and hence solve for all $\epsilon_{ij}$ simultaneously. We use gradient descent to solve the optimization problem for each pSTL formula $\phi_i$, $i = 1, 2$, with the following loss function, $\min_{\epsilon_{ij}} \sum_{j=1}^{N} \text{ReLU}(-\rho(s^{(j)}, \phi_i))$. In contrast, [93] proposes a binary search approach to solve monotonic pSTL formulas, formulas where the robustness value increases monotonically as the parameters increases (or decreases). However, this method is naturally a sequential process. As $\phi_1$ and $\phi_2$ are monotonic pSTL formulas, we can compare the average computation time taken to find a solution to all $\epsilon_{ij}$'s using stlcg with the binary search approach in [93]. Both approaches converged to the same solution, and the resulting computation times are illustrated in Figure 5.7.

The computation time for the binary search method, as expected, increases linearly as the batch size increases, while the computation time using stlcg increases at a much lower rate but requires some initial overhead to set up the computation graph. Further, since our stlcg toolbox is implemented in PyTorch, we can very easily make use of the GPU to accelerate the optimization process. Using GPU parallelization provides near-constant time computation for sufficiently large problem sizes.

For cases where the solution has multiple local minima (e.g., non-monotonic pSTL formulas), we can additionally batch the input with samples from the parameter space, and anneal $\beta$, the scaling parameter for the max and min approximation over each iteration. The samples will converge to a local minimum, and, with sufficiently many samples and an adequate annealing schedule, we can (hopefully) find the global minimum. However, we note that we are currently not able to optimize time parameters that define an interval as we cannot backpropagate through those parameters. Future work will investigate how to address this, potentially leveraging ideas from forget gates in long short-term memory (LSTM) networks [108].

5.4.3 Robustness-Aware Neural Networks

We demonstrate via a number of illustrative examples how stlcg can be used to make learning-based models (e.g., deep neural networks) more robust and reflective of desired behaviors stemming

---

3We can even account for variable signal length by padding the inputs and keeping track of the signal lengths.

4The $\epsilon_{ij}$'s are initialized to zero, which gives negative robustness values and hence results in a non-zero gradient.
Figure 5.8: A simple supervised learning problem without (left) and with (right) STL regularization. The output is required to satisfy $\phi = \Box_{[1,2.75]}(s > 0.48 \land s < 0.52)$ despite the training data being noisy and violating $\phi$.

Figure 5.9: Architecture of a neural network enhanced with STL regularization in its loss function.

from domain expertise.

**Model Fitting With Known Structure**

In this example, we show how stlcg can be used to regularize a neural network model to prevent overfitting to the noise in the data. Consider the problem of using a neural network to model temporal data such as the noisy signal in the orange dashed line in Figure 5.8. Suppose that based on domain knowledge of the problem, we know that the data must satisfy the STL formula $\phi = \Box_{[1,2.75]}(s > 0.48 \land s < 0.52)$. Due to the noise in the data, $\phi$ is violated. Figure 5.8 (left) illustrates the output of a two layer neural network trained only with a mean-square-error loss. Evidently, the model produces outputs that violate $\phi$. To mitigate this, we regularize the learning processing by augmenting the loss function with a term that penalizes negative robustness values. Specifically, the loss function becomes

$$L = L_0 + \gamma L_{\text{STL}}$$

(5.4)

where $L_0$ is the original loss function (e.g., reconstruction loss), and, in this example, $L_{\text{STL}} = \text{ReLU}(-\rho(s, \phi))$ is the robustness loss evaluated over the outputs computed over all time steps, $s = [f_\theta(t_0), ..., f_\theta(t_N)]$, and $\gamma = 2.0$. A schematic representing how, in general, a neural network can be regularized using stlcg is illustrated in Figure 5.9.
5.4. APPLICATIONS OF STLcG

Figure 5.10: Schematic of a rolled-out RNN network illustrating how to leverage STLcG to improve long-term predictions with access to only short-term data.

As shown in Figure 5.8 (right), the model with robustness regularization is able to obey $\phi$ more robustly. Note that regularizing the loss function with an STL loss does not guarantee that the output will satisfy $\phi$ but rather the resulting model is encouraged to violate $\phi$ as little as possible.

**Sequence-to-Sequence Prediction**

In this example, we demonstrate how STLcG can be used to influence long-term behaviors of sequence-to-sequence prediction problems to reflect domain knowledge despite only having access to short-term data. Sequence-to-sequence prediction models are often used in robot decision-making, such as in the context of model-based control where a robot may predict future trajectories of other agents in the environment given past trajectories, and use these predictions for decision-making and control (see Chapter 2). Often contextual knowledge of the environment is not explicitly labeled in the data, for instance, cars always drive on the right side of the road, or drivers tend to keep a minimum distance from other cars. STLcG provides a natural way, in terms of language and computation, to incorporate contextual knowledge into the neural network training process, thereby infusing desirable behaviors into the model which, as demonstrated through this example, can improve long-term prediction performance.

We generate a dataset of signals $s$ by adding noise to the tanh function with random scaling and offset. Illustrated in Figure 5.10, we design an LSTM encoder that takes the first 10 time steps as inputs (i.e., trajectory history). The encoder is connected with an LSTM decoder network that predicts the next 10 time steps (i.e., future trajectory). Note, the time steps are of size $\Delta t = 0.1$. Suppose, based on contextual knowledge, we know a priori that the signal will eventually, beyond the next ten time steps captured in the data, be in the interval $[0.4, 0.6]$. Specifically, the signal will satisfy $\phi = \Diamond \Box_{[0,1]} (s > 0.4 \land s < 0.6)$. We can leverage knowledge of $\phi$ by rolling out the RNN model over an extended time horizon beyond what is provided in the training data, and apply the robustness loss on the rolled-out signal $s^{(i)}$ corresponding to the input $x^{(i)}$. We use (5.4) as the
CHAPTER 5. BACKPROPAGATION THROUGH STL SPECIFICATIONS

Figure 5.11: Comparison of a sequence-to-sequence prediction model without (left) and with (right) robustness regularization. With robustness regularization, the model achieves desired long-term behavior $\phi = \lozenge \square_{[0,1]} (s > 0.4 \land s < 0.6)$ despite having access to only short-term behaviors during training.

Figure 5.12: Architecture of a latent space model leveraging stlcg for added structure.

loss function where $L_0$ is the mean square error reconstruction loss over the first ten predicted time steps, and $L_{STL} = \sum_i \text{ReLU}(-\rho(s^{(i)}, \phi))$ is the total amount of violations over the extended roll-out. With $\gamma = 0.1$, Figure 5.11 illustrates that with STL robustness regularization, the RNN model can successfully predict the next ten time steps and also achieve the desired long-term behavior despite being only trained on short-horizon data.

We envision that with a larger and more complex sequence-to-sequence model, such as the one investigated in Chapter 2, stlcg can be used to make the training process more data-efficient, rule-aware, and improve prediction performance over longer time horizons.

Latent Space Structure

In this example, we demonstrate how stlcg can be used to enforce logic-based interpretable structure into deep latent space models and thereby improve model performance. As previously introduced in Chapter 2, deep latent space models are a type of neural network models that introduce a bottleneck in the architecture to force the model to learn salient (i.e., latent) features from the input in order to (potentially) improve modeling performance and provide a degree of interpretability. Specifically, an encoder network transforms the inputs into latent variables, which have a lower dimension than the inputs, and then a decoder network transforms the latent variables into desired outputs. An
5.4. APPLICATIONS OF STLCG

Figure 5.13: A VAE example leveraging stlcg to provide more structure into the latent space representation. Left: Training data used to train a VAE with a discrete latent space. Middle: Generated outputs from a VAE model leveraging stlcg. Right: Generated outputs from a standard VAE without leveraging stlcg.

Illustration of an encoder-decoder architecture is shown in Figure 5.12 in blue.

Variational autoencoders (VAEs) [32] are a type of probabilistic deep latent space models used in many domains such as image generation, and trajectory prediction [109]. Given a dataset, a VAE strives to learn a distribution such that samples drawn from the distribution will be similar to what is represented in the dataset (e.g., given a dataset of human faces, a VAE can generate new similarly-looking images of human faces). For brevity, we omit describing details of VAEs but refer the reader to [30, 110] for an overview on VAEs. However, the VAE latent representation is learned in an unsupervised manner whereby any notions of interpretability are not explicitly enforced, nor is there any promise that the resulting latent space will adequately encompass human-interpretable features. Developing techniques to add human-understandable interpretability to the latent space representation is currently an active research area [111].

In this example, we demonstrate that we can use stlcg to project expert knowledge about the dataset into the latent space construction, and therefore aid in the interpretability of the latent space representation. Consider the following dataset illustrated in Figure 5.13 (left). The dataset is made up of (noisy) trajectories; Gaussian noise was added onto the trajectories shown in solid color. Notably, the data is multimodal as there are three trajectory archetypes contained in the data, each denoted by different colors. The goal is to learn a VAE model that can generate similar-looking trajectories. However, training a VAE for multimodal data is generally challenging (e.g., can suffer from mode collapse).

We consider a VAE model with a discrete latent space with dimension $N_z = 3$ because, based on our “expert knowledge”, there are three different types of trajectory modes. The latent variable $z \in \{0, 1\}^3$ is a one-hot vector with length three, i.e., $z \in \{[1, 0, 0]^T, [0, 1, 0]^T, [0, 0, 1]^T\}$. Additionally, we use LSTM networks for the encoder and decoder networks due to the sequential nature of the data. Based on our “expert knowledge”, the data satisfies the following STL formula,

$$
\phi_c = \Box_{[7.5, 9.75]} [(x > c - \epsilon) \land (s < c + \epsilon)],
$$
where \( c \in \{0.7, 0.35, 0\} \) and \( \epsilon = 0.025 \). This STL formula describes the convergence of a signal to a value \( c \) within some tolerance \( \epsilon \). Let \( C = [0.7, 0.35, 0]^T \) be a column vector corresponding to the possible parameter values of \( \phi_c \) (based on expert knowledge). Then the dot product \( z^T C \) selects the entry of \( C \) corresponding to 1’s position in \( z \). As such, if \( s \) is the output trajectory from the VAE mode, and \( z \) is the corresponding latent vector, then we can construct an STL loss, \( L_{\text{STL}} = \text{ReLU}(\rho(s, \phi_z^T C)) \). This STL loss enforces each latent vector instantiation to correspond to different STL parameter values and therefore ensures that the resulting output satisfies the associated STL formula. The overall loss function used to train a VAE model with STL information is,

\[
\mathcal{L} = \mathcal{L}_{\text{VAE}} + \gamma \mathcal{L}_{\text{STL}},
\]

where \( \mathcal{L}_{\text{VAE}} \) is the standard loss function used when training a typical VAE model.

Figure 5.13 (middle) illustrates the trajectories generated from a VAE trained with an additional STL loss (\( \gamma = 4 \)), while Figure 5.13 (right) showcases trajectories generated with the same model (i.e., same neural network architecture and size) trained without the STL loss (\( \gamma = 0 \)). Note, the hyperparameters used during training (e.g., seed, learning rate, number of iterations, batch size, etc.) were the same across the two models. We can see that by leveraging expert knowledge about the data and encoding that knowledge via STL formulas, we are able to generate outputs that correctly cover the three distinct trajectory modes. With more training iterations, it could be possible, though not guaranteed, that the vanilla VAE model would have been able to capture the three distinct trajectory modes. Nonetheless, this example demonstrates that \texttt{stlcg} could potentially be used to accelerate training processes, and enable models to more rapidly converge to a desirable level of performance.

5.5 Summary

In this chapter, we showed that STL is an attractive language to encode spatio-temporal specifications, either as constraints or a form of inductive bias, for a diverse range of problems studied in the field of robotics. In particular, we proposed a technique, \texttt{stlcg}, which transcribes STL robustness formulas as computation graphs, and therefore enabling the incorporation of STL specifications in a range of problems that rely on gradients. We demonstrated through several illustrative examples that we can infuse logical structure in a diverse range of robotics problems such as motion planning, behavior clustering, and deep neural networks for model fitting, intent prediction, and generative modeling.

We highlight several directions for future work that extend the theory and applications of \texttt{stlcg} as presented in this work. The first aims to extend the theory to enable optimization over parameters defining time intervals over which STL temporal operators hold and to also extend the language to express properties that cannot be expressed by the standard STL language. The second involves
investigating how \texttt{stlcg} can help verify and improve the robustness of learning-based components in safety-critical settings governed by spatio-temporal rules, such as in autonomous driving and urban air-mobility contexts. The third is to explore more ways to connect supervised structure induced by logic with unsupervised structure learning present in latent spaces models. This connection may help provide interpretability via the lens of temporal logic for neural networks that are typically difficult to analyze.
Chapter 6

Control Synthesis With STL Specifications

In Chapter 5 we developed stlcg to provide a computational language to incorporate temporal logic specifications into gradient-based techniques. In this chapter, we take a deep dive into the problem of control synthesis with STL specifications whereby the goal is to design a computationally efficient closed-loop controller for a dynamical system such that it will satisfy a desired STL specification. Specifically, we develop a technique to synthesize a closed-loop neural network controller by using an adversarial training scheme offline, and then adapting the controller online to improve robustness against disturbances. Key to this endeavor, we introduce an element of imitation learning to regularize the control synthesis process and foster better policy exploration. Our experiments demonstrate that the imitation-based regularization results in higher-performing robot behaviors than solely optimizing over the STL objective only. We demonstrate the efficacy of our approach with an illustrative case study and show that our proposed method outperforms a state-of-the-art shooting method in both robustness performance and computation time.

6.1 Introduction

In Chapter 5 we introduced stlcg, a technique that translates STL robustness formulas into computation graphs. stlcg leverages modern auto-differentiation tools used widely in the field of deep learning, e.g., Pytorch [100], and hence bridges the gap between temporal logic and deep learning as they are now share the same computational backbone. This bridge provides a way to combine temporal logic with data-driven approaches and hence diversifying ways in which temporal logic can be used with deep neural networks.

In this work, we develop a learning-based control synthesis technique that leverages structure
6.1. INTRODUCTION

Figure 6.1: We use STL and expert demonstrations to synthesize a trajectory-feedback controller satisfying a desired spatio-temporal specification. The controller consists of a LSTM network and an environment encoding which are optimized offline via an adversarial training strategy. Online, an adaptation scheme refines the controller to robustify against disturbances.

stemming from different modalities (i.e., heterogeneous structure). Specifically, we propose a neural network control synthesis technique that utilizes temporal logic and expert demonstrations to (i) instill robot behaviors that satisfy desired high-level specifications while attuned to human intuition, and (ii) improve exploration of the search space during the control synthesis process. At the same time, we are cognizant of the difficulties in deriving theoretical guarantees for neural network controllers [112]. Thus key to this endeavor, we additionally propose an online adaptation scheme that updates the controller to robustify against disturbances and limitations in the pre-computed control policy. A schematic of the proposed approach is visualized in Figure 6.1. Our method can be interpreted as a middle ground between receding horizon open loop and closed-loop control—we synthesize a closed-loop neural network controller offline and then refine it online to adapt to changes in the environment and correct for any undesirable behaviors.

The contributions of this chapter are fourfold: (i) We develop a semi-supervised trajectory-feedback controller designed to satisfy a desired STL specification and demonstrate the benefit of using (few) expert demonstrations to help guide the control synthesis process. A trajectory-feedback, as opposed to state-feedback, controller is crucial since satisfaction of an STL specification is history-dependent. (ii) We generalize our trajectory-feedback controller to new but similarly structured environments to prevent the need to re-synthesize a new STL controller whenever the environment changes. Environment generalization is achieved by conditioning the control parameters on environment parameters. (iii) We combine an offline iterative adversarial training algorithm with an online adaptation scheme to improve robustness against disturbances. We show empirically that even when the neural network controller (trained offline) produces trajectories that violate the STL specification, the online adaptation step produces satisfying trajectories. (iv) We demonstrate our controller on a relatively complex STL specification and show that it outperforms a state-of-the-art shooting method in terms of STL robustness and computation time.
6.2 Related Work

We will provide an overview of state-of-the-art methods in temporal logic control synthesis, and learning-based approaches that use temporal logic.

6.2.1 Temporal Logic Synthesis

Temporal logic provides a formalism to convert specifications in natural language into a concise mathematical representation. In particular, LTL, a popular temporal logic language has received a lot of attention over the years; a finite abstraction model of a dynamical system and LTL specifications can be converted into an automaton for which there are well-studied methods to synthesize correct-by-construction closed-loop controllers [113] satisfying the LTL specification. However, the synthesis procedure is doubly exponential [114] and therefore smaller fragments of the LTL language are used instead. Additionally, if the environment changes, then a new controller needs to be re-synthesized. Unfortunately, an analogous automaton-based approach for STL does not exist, and constructing closed-loop STL controllers remain a challenging problem.

Instead of synthesizing a closed-loop controller, an open-loop trajectory satisfying LTL specifications can be constructed by solving a Mixed Integer Linear Program (MILP) and executed in a receding horizon fashion [115, 116]. The encoding of LTL specifications as integer constraints has been extended to STL too [96, 117, 118, 119]. While receding horizon control can adapt to changes in the environment, these MILPs are NP-hard and do not scale well with problem size, both along the axis of specification complexity and trajectory length. As such, these MILP-based approaches become impractical for general nonlinear systems that need to perform complex tasks over longer time horizons. Instead, [105] utilizes smooth approximations to STL robustness formulas and proposes a sequential quadratic programming (SQP) to solve an optimal control problem maximizing robustness. While SQP can account for nonlinear dynamics and relatively more complex STL specifications, the solve time is still intractable for real-time applications (roughly 30 minutes). Additionally, the solution may converge to an undesirable local minimum. To address the high computation times, [106] proposes a hierarchical approach specific to reach-avoid multi-quadrotor missions—the algorithm optimizes over sparse way-points which are used to generate a higher-resolution continuous-time minimum jerk trajectory. While hardware experiments seem promising, the computation solve time was still a bottleneck especially when scaling up to more agents and increasing the time horizon.

In summary, computational tractability remains a core challenge as we consider longer horizon problems with more complex STL specifications. In the next section, we discuss recent learning-based techniques that are more computationally tractable for synthesizing closed-loop STL controllers.
6.3. PROBLEM FORMULATION

6.2.2 Temporal Logic in Learning-Based Approaches

Recently, there has been a growing interest in using STL as a form of inductive bias within a variety of learning-based approaches, such as in deep neural networks \[120, 121\], reinforcement learning \[122, 123\], and learning from demonstrations \[124, 95\]. Given an STL specification, these approaches augment the loss (or reward) with an STL robustness term that bias the model towards producing outputs that have higher STL robustness values. Deep neural networks provide a computationally tractable way to synthesize controllers, especially for highly complex, potentially learned, dynamical systems. However, it is difficult to verify that the resulting neural network will satisfy the desired specification for all possible inputs. In general, neural network verification is a challenging yet growing area of research \[112\].

A common approach to bolster the performance of a neural network controller is to leverage adversarial training examples; an adversarial search procures falsifying examples which are then used to retrain the model (e.g., \[124, 125, 98\]). The process is repeated until no more falsifying examples can be found or a stopping criteria is met. While this generally improves the performance of a model, it does not necessarily provide any formal guarantees on performance.

Works that are most similar to the work presented in this chapter are \[98\] and \[99\]. In \[98\], a state-feedback neural network controller was synthesized via an iterative adversarial training scheme whereby the training objective aimed at maximizing robustness only. However, the state-feedback element of the controller prevents exploiting knowledge of past spatio-temporal behaviors. In general, satisfaction of an STL specification depends on the entire trajectory, and not just a single state. Additionally, the method is tailored towards a fixed environment, thus requiring a new controller to be synthesized if the environment changes. To address these limitations, \[99\] synthesizes a recurrent neural network (RNN) controller to account for past spatio-temporal behaviors, and then online, the controller is complemented with a control barrier function \[126\] to avoid previously unseen obstacles in the environment. However, the approach (i) focuses on reach-avoid STL specifications only, and (ii) assumes that the training data is generated by directly solving an STL-constrained trajectory optimization problem. As discussed previously, solving an STL-constrained trajectory optimization problem is nontrivial even if performed offline.

In this work, we develop a semi-supervised learning-based STL control synthesis method such that the resulting controller can be deployed in a variety of environments without the need to retrain, and is robust to disturbances applied to the system. Compared to similar works, we demonstrate that our method is applicable to more complex STL specifications, and is more data efficient.

6.3 Problem Formulation

Let \( z_t \in \mathcal{X} \subset \mathbb{R}^n \), \( u_t \in \mathcal{U} \subset \mathbb{R}^m \), \( d_t \in \mathcal{D} \subset \mathbb{R}^{m_d} \) be the state, control, and disturbance of a system at time \( t \). Additionally, let \( \tau_p = z_{0:p} \) denote a state trajectory starting at \( t = 0 \) up until \( t = p \). Let
$X_0 \subseteq \mathcal{X}$ be the set of states the system can start in. Further, let the time-invariant, discrete-time state space dynamics for a system be $z_{t+1} = f(z_t, u_t, d_t)$, and $\mathcal{E}$ denote the set of environment parameters (e.g., image of the environment).

Let $\xi_{\pi, \mathcal{P}, \tau, \mathcal{D}} = z_{0:T}$ denote a trajectory where $z_{0:p} = \tau_p$, and $z_{p+1:T}$ is produced by following a control policy $u_t = \pi(\cdot)$, and is subject to a disturbance $d_t \sim \mathcal{P}$ at each time step. For ease of notation, when $p = 0$, we write $\xi_{z_{0}, \mathcal{P}, \tau, T} = z_{0:T}$. Let $\phi$ represent an STL specification that we desire a system to satisfy. Then the problem we seek to solve is:

For a time horizon $T$, find $\pi(\cdot)$ such that with $d_t \sim \mathcal{P}$, $\forall e \in \mathcal{E}$, and $\forall z_0 \in X_0$, $\xi_{z_{0}, \mathcal{P}, \tau, T} \models \phi$.

In words, we want to find a control policy $\pi$ such that under disturbance inputs $d_t \sim \mathcal{P}$ for all $t = 0, ..., T$, for all possible environments in $\mathcal{E}$, and initial states in $X_0$, all trajectories $\xi_{z_{0}, \mathcal{P}, \tau, T}$ will satisfy $\phi$.

### 6.4 STL Control Synthesis

We propose a control synthesis framework that (i) leverages (few) expert demonstrations to aid in policy exploration, (ii) uses an adversarial training scheme to improve the performance of the closed-loop policy, and (iii) employs an online adaptation step for robustification against disturbances.

#### 6.4.1 Overview

Our method represents a middle ground between receding-horizon open-loop control and closed-loop control. In the offline computation, we construct a data-driven closed-loop trajectory-feedback control policy. However, due to limitations in neural network verification, the policy may result in violating trajectories. To address this limitation, an online computation will update $\pi$ whenever a falsifying trajectory is expected to occur. Our combined closed-loop controller with an online adaptation step reduces the computation burden compared to receding horizon methods that solve an open-loop problem at each time step. We describe the architecture of the closed-loop neural network controller in Section 6.4.2 then detail the control synthesis procedure in Section 6.4.3. Finally, we outline the online adaptation set in Section 6.4.4.

#### 6.4.2 Trajectory-Feedback Controller Architecture

In this work, we use a Long Short Term Memory (LSTM) network, a specific type of RNN, to enable the construction of a trajectory-feedback controller. Consideration of past spatio-temporal behaviors is a critical when reasoning about the spatio-temporal properties of an entire signal. To generalize the LSTM controller to different environments, we use the environment to initialize the LSTM initial hidden state. Since $\mathcal{E}$ represents the set of parameters describing the environment, we use $e \in \mathcal{E}$ to construct an LSTM initial hidden state. In this work, we assume that $\mathcal{E}$ is a set
of images describing the layout of the environment, and therefore we use a Convolutional Neural Network (CNN) \cite{CNN127} to summarize $e \in \mathcal{E}$. However, different transformations can be used depending on the environment representation (e.g., range measurements, occupancy grid, set of numbers). A schematic of the neural control architecture is illustrated in Figure 6.2. Since the policy $\pi$ depends on state but the arrows are not shown to reduce visual clutter. The LSTM is unrolled for visualization purposes.

### LSTM Component

LSTMs take as inputs time-series data, and produce another time-series. Using the past trajectory $\tau_t = z_{0:t}$ that a system has already traversed, an LSTM cell can summarize, with a hidden state $h_t$, past spatio-temporal behaviors without the need for state augmentation.

Let $o_t, h_{t+1} = g_{\text{LSTM}}^{n_h}(z_t, h_t)$ denote the input-output relationship described by an LSTM cell with hidden state size $n_h$ ($o_t, h_t \in \mathbb{R}^{n_h}$) \footnote{For brevity, we omit the details of the internal operations of the LSTM cell. See \cite{LSTM104, LSTM108} for details on the architecture of the LSTM cell.}. For a time step $t$, a state $z_t$ and hidden state $h_t$ is fed into $g_{\text{LSTM}}^{n_h} : \mathbb{R}^n \times \mathbb{R}^{n_h} \rightarrow \mathbb{R}^n \times \mathbb{R}^{n_h}$ to produce an output state $o_t \in \mathbb{R}^{n_h}$, and the next hidden state $h_{t+1}$. We will discuss how to initialize $h_0$ in Section 6.4.2.

When unrolling the LSTM with the input trajectory $\tau_p$, we simply feed in $z_t$ and $h_t$ at each time step up to $t = p$ to obtain $h_{p+1}$ and $o_p$. Then for $t \geq p$, the output state, $o_t$, is passed through a multi-layer perceptron (MLP), denoted by $g_{\text{MLP}}^{n_h \rightarrow m}$ which projects the output state $o_t \in \mathbb{R}^{n_h}$ to controls $u_t \in \mathbb{R}^m$. To ensure that the control inputs satisfy control constraints $u \in [\underline{u}, \overline{u}]$, we take $u'_t = g_{\text{MLP}}^{n_h \rightarrow m}(o_t)$ and apply the following transformation,

$$u_t = \frac{\overline{u} - u}{2} \tanh u'_t + \frac{\overline{u} + u}{2}.$$
Given the newly computed $u_t$, the current state $z_t$, and disturbance $d_t \sim \mathcal{P}_D$, we can compute the next state $z_{t+1} = f(z_t, u_t, d_t)$. By incorporating the dynamics in the unrolling of the LSTM, we ensure that the resulting trajectory is dynamically feasible. The next state $z_{t+1}$ and next hidden state $h_{t+1}$ are then passed through the LSTM cell again to compute the next output state and control and so forth. We continue unrolling the LSTM cell up to $t = T$, a predetermined time horizon. The overall trajectory (joining the past trajectory $\tau_p$ and propagated trajectory $z_{p+1;T}$) is denoted by $\xi_{\tau_p, T}^{\pi, d}$. Next, we discuss how to compute $h_0$, the initial hidden state to the LSTM network.

**CNN Component**

To generalize the controller to new environments, we condition the LSTM hidden state with a summary vector representing the environment. In this work, we summarize an image of the environment using CNNs, though a different summary could be used depending on the environment representation.

Given an STL specification $\phi$, there are components that describe how the system should be interacting with the environment. For instance, could be regions in the environment that the system should reach or avoid. As such, we can decompose each “region type” (i.e., regions to reach or avoid) to different image channels. For example, consider

$$\phi = (\Diamond \Box [0, 8] \phi_{\text{cov}} \cup \Diamond \phi_{\text{goal}}) \land \Box \neg \phi_{\text{obs}},$$

$$\phi_{\text{cov}} = \text{inside coverage set},$$

$$\phi_{\text{goal}} = \text{inside goal set},$$

$$\phi_{\text{obs}} = \text{inside obstacle set},$$

which translates to “eventually stay inside the coverage set for 8 time steps before eventually reaching the goal set, and always avoid the obstacle set.” There are elements in the environment that correspond to the coverage, goal, and obstacle set (and initial set). Thus each image channel of $e$ is associated to the different regions characterizing the environment. We then use a CNN to encode $e$ into a summary vector $c_E$ and then use $c_E$ to initialize the hidden state of the LSTM. Let $g_{\text{CNN}}^{e \rightarrow n_c}$ be a CNN network encoding the environment image $e$ into a hidden state $c_E \in \mathbb{R}^{n_c}$, and $g_{\text{MLP}}^{n_c \rightarrow n_h}$ be an MLP projecting $c_E$ to $h_0$. Then, the initial hidden state for the LSTM can be computed by,

$$h_0 = g_{\text{MLP}}^{n_c \rightarrow n_h}(g_{\text{CNN}}^{e \rightarrow n_c}(e)).$$

Although the structure of $e$ is dependent on the STL specification of interest, we can still generalize across new and unseen environments for which $\phi$ is still valid. For example, if the obstacles have

\footnote{Two MLPs are actually needed since the hidden state of LSTMs is a tuple of two vectors, each of size $n_h$.}
moved to a new location, we simply use the corresponding \( e \) to compute a new \( h_0 \) and avoid retraining the controller from scratch.

### 6.4.3 Learning the Control Parameters

Given the neural network architecture described in Section 6.4.2, the goal is to learn \( \theta \), a vector of neural network parameters from the CNN, LSTM, and MLP networks, such that given any \( e \in \mathcal{E} \) and any \( z_0 \in \mathcal{X}_0, \xi^\pi_{z_0,T} |\phi \). There are two key aspects to our training scheme: (i) We leverage expert demonstrations to introduce an imitation regularization loss to help guide the training to a better local optimum, compared to the case when optimizing only with STL robustness. For this reason, we refer to our approach as a semi-supervised approach. (ii) We take on an iterative sampling-based approach that iterates between a training step to optimize \( \theta \) via gradient descent, and an adversarial step which searches for initial states and environments where the controller produces a violating trajectory. The next training step updates the model to correct for those violating samples. The pseudocode for the training process is outlined in Algorithm 1, and the details are provided next.

We note, however, there are no theoretical guarantees that Algorithm 1 will converge and that no adversarial samples exists. We address this limitation in Section 6.4.4 by proposing an online adaptation step.

**Algorithm 1: STL control synthesis**

Result: \( \pi_\theta \)

Initialization: \( S = \{ (z_0,i,e_i) \}_{i=1,...,N} \) where \((z_0,i,e_i) \sim \text{Uniform}[\mathcal{X}_0 \times \mathcal{E}] \); 

\( k \leftarrow 0; \)

Model training: Optimize \( \theta \) using \( N_{\text{full}} \) training iterations on (6.2) over \( S \); 

while \( k < K \) do

Adversarial search: Find \( S_{\text{adv}} = \{ (z_0,j,e_j) \mid \xi^\pi_{z_0,j,T} \not|\phi \}_{i=1,...,N_{\text{adv}}} \) with \( \theta \) fixed (e.g., acceptance-rejection sampling); 

Update initial states: Re-sample from \( \text{Uniform}[\mathcal{X}_0 \times \mathcal{E}] \) to get \( S_0 = \{ (z_0,i,e_i) \}_{i=1,...,N} \); 

\( S \leftarrow S_0 \cup S_{\text{adv}}; \)

Model training: Optimize \( \theta \) using \( N_{\text{mini}} \) training iterations on (6.2) over \( S \); 

\( k \leftarrow k + 1; \)

end

**Training Step**

Let \( S = \{ (z_0,i,e_i) \}_{i=1,...,N} \) represent samples from \( \mathcal{X}_0 \times \mathcal{E} \) (the samples can be sampled uniformly), and let \( \Xi^{\exp} = \{ \xi^\exp_{z_0,i,T}; e_i^{\exp} \}_{i=1,...,N_{\exp}} \) represent trajectories corresponding to expert demonstrations that satisfy \( \phi \), the STL specification that we aim to design a controller for. We make an assumption that we have access to expert demonstrations, such as from real-world operations or from simulation (see Figure 6.1). Approximate solutions from direct numerical optimization could be used
but may require additional human supervision for refinement. Our approach is semi-supervised—the
demonstrations are used to regularize the training loss and therefore we do not require a significant
amount of demonstrations, and favorably so if data collection is expensive. In contrast, fully super-
vised learning approaches require many demonstrations to train the neural network policy (e.g., [99]
uses 500 demonstrations).

We apply stochastic gradient descent on $\theta$ to minimize the following loss objective,

$$
L_{\text{train}}(\theta; S, \Xi^{\exp}) = L_{\text{STL}}(\theta; S) + \gamma L_{\text{limit}}(\theta; \Xi^{\exp}),
$$

where $\text{LeakyReLU}(x) = \max(0, 0.01x, x)$ and $\Delta(\xi, \xi; e) = \text{MSE}(z_a, z_b) + \gamma_{\text{imit}} \text{MSE}(u_a, u_b)$ is a
weighted sum of the mean-square-error in state and controls between trajectories $\xi_a$ and $\xi_b$ un-
der environment $e$. Note that (6.3) differs from maximizing robustness as done in [98, 105, 124].

The LeakyReLU function focuses on minimizing the amount of violation and less on increasing
the amount of satisfaction. The purpose of (6.4) is to help regularize the training process; since
(6.3) is highly nonlinear and non-convex especially for more complex STL specifications; (6.4) helps
guide the exploration towards regions where the controller produces trajectories consistent with how
humans would behave and therefore avoid superfluous or unreasonable trajectories.

There are multiple hyperparameters that characterize the training process, namely $\gamma_{\text{imit}}$: the
weight between the state and control MSE when computing reconstruction loss, $\gamma$: the weight on
$L_{\text{imit}}$ in (6.2), $p_{\text{LSTM}}$: the probability of using the state computed from the previous LSTM cell
instead of the ground truth value when reconstructing a human-demonstrated trajectory (this is to
help mitigate cascading errors in the final model), and $\beta_{\text{STL}}$ which is the smoothing parameter the
max and min approximations used in STL robustness formulas. Some of these parameters, especially
$p_{\text{LSTM}}$ and $\beta_{\text{STL}}$ can be set on an annealing schedule to increase over each iteration.

**Adversarial Search**

The training step is optimized using samples of initial states and environments. As such, there could
still exist samples where $\rho_{\phi}(\xi^{\exp}, PD) < 0$. The goal of the adversarial search is to find a set of initial
states and environments $S^{\text{adv}} = \{(z_{0,i}^{\text{adv}}, e_{i})\}_{i=1,...,N_{\text{adv}}}$ such that the resulting trajectories violate
$\phi$. A number of methods can be used to search for adversarial samples, such as batched (projected)
gradient descent on $z_0$ to minimize robustness, cross-entropy method, or simulated annealing [128].

Since we strive to find any samples that produce a violating trajectory, we opt for a simpler approach
of acceptance-rejection sampling whereby we uniformly sample from $X_0 \times E$ and reject any samples
that produces satisfying trajectories. We continue sampling until \( N_{\text{adv}} \) adversarial samples are found (or some other termination criterion).

### 6.4.4 Deploying the Controller Online

Unfortunately, there are no guarantees that \( \pi_\theta \) will produce satisfying trajectories for all initial states and environments, especially under the presence of disturbances to the system. There is a lot of effort towards neural network verification [112], though verifying RNNs, such as LSTMs, remain challenging due to the temporal aspect to the problem, and constraints to the recurrent cell architecture.

To address this limitation, we propose updating \( \pi_\theta \) online if we find that \( \pi_\theta \) is expected to produce a violating trajectory. Specifically, we perform a few gradient descent steps on \( \theta \) to increase the robustness value if \( \mathbb{E}_{d \sim P_D}[\rho(\xi_{\pi_\theta,T}, \phi)] < 0 \). To compute \( \mathbb{E}_{d \sim P_D}[\rho(\xi_{\pi_\theta,T}, \phi)] \), we perform a Monte Carlo estimate by simulating multiple futures with disturbances sampled from \( d_t \in P_D \), and compute robustness over each entire trajectory \( \xi_{\pi_\theta,T} \). Instead of performing gradient descent on all of the parameters \( \theta \), we perform gradient descent only on the “last layer”, i.e., the parameters of \( g_{\text{MLP}}^{n_h \to m} \). If desired, a different risk metric could be used instead. The online adaptation procedure is formalized in Algorithm 2.

### Algorithm 2: Deployment of \( \pi_\theta \) in an online setting.

**Result:** \( z_{0:T} \): a trajectory satisfying \( \phi \).

**Initialization:** STL specification \( \phi \), initial state \( z_0 \), environment \( e \), time horizon \( T \), neural network weights \( \theta_0 \) produced by Algorithm 1, step size \( \eta \), and maximum number of gradient steps \( N_{\text{gd}} \);

\( \theta \leftarrow [\tilde{\theta}, \hat{\theta}] \) where \( \tilde{\theta} \) are the neural network parameters of \( g_{\text{MLP}}^{n_h \to m} \) to be potentially updated, and \( \hat{\theta} \) are the remaining neural network parameters which will be fixed;

\( \tau_0 \leftarrow z_0; \)

**for** \( t = 0 : T \) **do**

**if** \( \mathbb{E}[\rho(\xi_{\pi_\theta,T}, \phi)] < 0 \) **then**

\( j \leftarrow 0; \)

**while** \( (\mathbb{E}[\rho(\xi_{\pi_\theta,T}, \phi)] < 0) \land (j < N_{\text{gd}}) \) **do**

\( \hat{\theta} \leftarrow \hat{\theta} + \eta \nabla_{\theta} \mathbb{E}_{d \sim P_D}[\rho(\xi_{\pi_\theta,T}, \phi)]; \)

\( \theta \leftarrow [\hat{\theta}, \hat{\theta}_0]; \)

\( j \leftarrow j + 1; \)

**end**

\( u_t = \pi_\theta(\tau_t; e); \)

\( d_t \sim P_D; \)

\( z_{t+1} = f(z_t, u_t, d_t); \)

\( \tau_t \leftarrow z_{t+1}; \)

**end**
6.5 Experiments

We investigate the offline and online performance of our proposed controller applied to a nonlinear system.

6.5.1 Case-Study Set-Up

We investigate a car-like robot where the discrete-time dynamics are given by applying zero-order hold on controls and disturbance for the following kinematic bicycle model with time step $\Delta t = 0.5$ seconds,

\[
\begin{align*}
\dot{x} &= V \cos(\psi + \beta), \quad \dot{y} = V \sin(\psi + \beta), \quad \dot{\psi} = \frac{V}{l_R} \sin(\beta), \\
\dot{V} &= a + d_a, \quad \tan(\beta) = \frac{l_R \tan(\delta + d_\delta)}{l_R + l_F}.
\end{align*}
\]

The speed of the vehicle is bounded, $V \in [0, 5]$, the controls are bounded with $a \in [-3, 3]$ and $\delta \in [-0.344, 0.344]$, and the distance from the center of mass to the front and rear axle are $l_F = 0.5$, and $l_R = 0.7$. There is a disturbance $d = [d_\delta, d_a]$ applied onto the control inputs with $d \sim \mathcal{N}([0, 0], [0.05, 0.02])$. We set $T = 55$.

The environment is characterized by an initial state, coverage, obstacle, and goal region, and the image $e$ represents a top-down view of the environment (see Figure 6.3). The position of the
coverage (white circle) and obstacle (red circle) region vary—the $x$-position of the coverage region ($x_{cov}$) varies with a fixed $y$-position, while the $x$-position of the obstacle is always half way between the coverage and goal region. We constrain the environment in this way to ensure the problem remains feasible for a fixed time horizon and avoids instances where it is trivial to avoid the obstacle region. The regions are assumed to be circles to make predicates easy to compute, but in general can be more complex as long as we can backpropagate through $\mu(z_t)$. We consider the following STL specification,

$$
\phi = (\Diamond[0,8] \phi_{cov} \bigcup \Diamond \phi_{goal}) \land \square \neg \phi_{obs}
$$

$$
\phi_{cov} = \text{inside coverage region} \land V_t < 2.0
$$

$$
\phi_{goal} = \square (\text{inside goal region} \land V_t < 0.5)
$$

$$
\phi_{obs} = \text{inside obstacle region}.
$$

(6.5)

In words: $\phi$ requires the robot to first slow down to less than 2m/s in the coverage region for $8\Delta t$ seconds before moving to the goal region and staying inside with velocity less than 0.5m/s. Simultaneously, the robot should always avoid the obstacle region. We highlight that (6.5) is more complex than the reach-avoid specifications studied in related works [99, 98] because (i) (6.5) consists of a bounded time interval indicating the minimum duration to stay inside a coverage region, (ii) there are restrictions on the velocity of the robot, and (iii) there are three nested temporal operators whereas others have at most two.

We provide 32 expert demonstrations which were collected in simulation with a human using an XBox controller to control the robot (see Figure 6.1). The demonstrations satisfy $\phi$. The simulation environment was implemented using the Robot Operating System (ROS) and visualized in RViz. We used PyTorch [100] to implement our neural network controller, and stlcg [120], an STL robustness toolbox leveraging PyTorch’s autodifferentiation framework, for the STL robustness calculations. The code accompanying this work can be found at https://github.com/StanfordASL/stlcg_ctrl_syn.

More details about the neural network parameters and hyperparameters used for the training process can be found in Appendix B.1.

### 6.5.2 Analysis and Discussion

We first discuss the our offline training procedure, and then the performance of our proposed online adaptive method including comparisons to a baseline approach.

**Offline Training (Without Online Adaptation)**

Figure 6.4 illustrates the closed-loop trajectories (without the online adaptation step) produced by $\pi_\theta$ trained with different values of $\gamma$, the weighting on the imitation loss. When $\gamma = 0$, corresponding
to the case where we optimize over STL robustness only, the controller performs worse—the model converges to a local optimum which produces trajectories that only pass to the right of the obstacle (i.e., one homotopy class) and consequently, is unsuccessful in reaching the coverage region when the region is further to the left (see left plot in Figure 6.4). When $\gamma > 0$, the model is able to mirror the expert trajectory and encapsulate the two homotopy classes (i.e., passing to the left and right of the obstacle), and therefore perform better over all environments. This behavior indicates that even though the end goal is to satisfy $\phi$, optimizing purely on STL robustness is not the most effective as it can lead the model into an undesirable local optimum. Instead, expert demonstrations can guide the policy exploration to a better local optimum, especially one that mirrors naturalistic human behavior.

From a quantitative perspective, the distribution of robustness values when using models trained with different $\gamma$ values is presented in Figure 6.5 (left). The mean and median is the highest when
\[ \gamma = 1.4. \] Figure 6.5 (right) illustrates the STL robustness distribution corresponding to controllers trained with different numbers of adversarial training iterations \((K)\) and with \(\gamma = 1.4\). We see that using more adversarial steps help shift the distribution towards higher robustness. Moving forward, we will use a model trained with \(\gamma = 1.4\) and \(K = 5\) in the following results.

**Online performance**

We compare both qualitatively and quantitatively the performance of our proposed trajectory-feedback STL controller (with and without online adaptation) against a baseline shooting method similar to the approach proposed in [105].

**Baseline:** Consider an optimal control problem,

\[
\begin{align*}
    u^*_{t:T-1} &= \max_{u_{t:T-1}} \phi(\xi_{\tau_{t},T}) \\
    \text{s.t. } &u \leq u_t \leq \bar{u}, \\
    &z_{t+1} = f(z_t, u_t, 0), \quad \forall t = t, \ldots, T - 1,
\end{align*}
\]  

(6.6)

solved in a receding horizon fashion. Note the zero disturbance in the dynamics. We used a projected limited-memory BFGS (L-BFGS) [129] gradient descent optimizer to solve (6.6). We used PyTorch’s built-in L-BFGS optimizer with step size 0.05 and clipped \(u_t\) to make sure the control constraints were satisfied. Similar to Algorithm 2, the optimization is only performed if the planned trajectory results in negative robustness (due to disturbances to the system). This shooting method is designed to mimic the SQP approach proposed in [105]. However, the Hessian computation was very expensive to compute in PyTorch (~10 seconds), and thus we used L-BFGS, a quasi-newton method instead. Due to the highly nonlinear nature of (6.6), we use the control sequence generated by propagating \(\pi_\theta\) from the initial state to warm-start (6.6) at the first time step. Then we used the solution from the previous time step as an initial guess for the next time step. We set \(N_{gd} = 3\), the maximum number of gradient steps allowed at each time step.

**Open-loop:** Given the initial state and environment, we compute the control sequence resulting from propagating the state over the time horizon using \(\pi_\theta\). Then we execute the control sequence in an open-loop fashion.

**TF:** At each time step, the past trajectory is passed into \(\pi_\theta\) to compute the next control input. No online gradient steps will be used in this approach.

**TF*–\(N_{gd}\):** The TF approach with online adaptation. This represents the core approach proposed in this work. We consider two cases, \(N_{gd} = 1\) (TF*–1) and \(N_{gd} = 3\) (TF*–3). We use the default Adam optimizer [130] in PyTorch.

We simulated (with noise) each of the control strategies, and Figure 6.6 showcases the trajectories deployed from each of the aforementioned methods for a particular environment. To better distinguish the characteristics of the different methods, we chose a challenging environment with \(x_{\text{cov}} = 11\).
Figure 6.6: Trajectories generated from different online control strategies. Our proposed method (TF*) satisfies the STL specification and is robust against disturbances. The corresponding robustness values are (higher is better): Open-loop: -1.08, Baseline: 0.27, TF: -0.29, TF*-1: 0.13, TF*-3: 0.42. which is outside of the distribution used to generate training data, $x_{cov} \sim U[1, 10]$. The open-loop and TF approaches resulted in negative robustness, while the Baseline and TF* approaches, all of which are able to adapt to the new environment and disturbances, resulted in positive robustness values with TF*-3 achieving the highest value. These behaviors highlight the importance of the online adaptation step complementing the learned closed-loop controller.

We ran 100 trials with random initial states and environments (consistent with the training distribution) and compared the STL robustness values and computations (see Figure 6.7). We highlight four takeaways from these results: (i) Rolling out the trajectory over the entire time horizon and evaluating the robustness value takes roughly 100ms. The computation time may be reduced with more tailored software. (ii) Both TF*-1 and TF*-3 produces 100% success rate in
producing satisfying trajectories with TF*-3 having a slightly higher mean robustness. However, in terms of computation times, there were a few instances (less than 0.5%) where the computation time was greater than 300ms, corresponding to times where gradient steps were needed. Those instances often occurred at the first few time steps, therefore the adaptation could potentially be performed offline to help keep the online computation times amenable to real-time applications. (iii) The Baseline has a 93% success rate, but about 10% of the time steps result in a computation of roughly 1000ms or more, therefore rendering the Baseline approach impractical for real-time applications unless significant improvements to the solve time are made. (iv) As expected, the Open-loop and TF approaches have very low computation times. The success rates are 48% and 91% respectively. As such, the TF and Baseline methods perform similarly in terms of robustness performance, but TF is more desirable due to its lower computation time.

6.5.3 Autonomous Driving Example

We synthesized a new controller for an autonomous driving setting and with a new STL specification,

\[
\psi = \phi_{\text{obs}} \land \phi_{\text{slow}} \land \phi_{\text{goal}},
\]

\[
\phi_{\text{obs}} = \Box (\text{avoid road boundary} \land \text{avoid obstacles}),
\]

\[
\phi_{\text{slow}} = \Box (\phi_{\text{near}} \rightarrow V < 0.55), \quad \phi_{\text{near}} = \text{dist. to obs.} < 1.2,
\]

\[
\phi_{\text{goal}} = \Diamond \Box_{[0,2]}(\text{in goal} \land V > 1.0).
\]

The specification \(\psi\) requires a car to stay on the road and avoid obstacles, slow down when near obstacles, and speed up for at least 2 time steps once it is past the obstacles. The position of the obstacles can vary. This particular setup is inspired by situations where an autonomous car is approaching a construction zone and will need to slow down and weave through some traffic cones.
and then speed up once past the construction zone.

To synthesize a controller satisfying $\psi$, we followed the same procedure described in this chapter and used eight human demonstrations. Figure 6.8 illustrates a trajectory computed with $N_{gd} = 3$; the autonomous car slows down as it weaves past the circular obstacles (the rectangles are more densely packed) and then speeds up once it is in the goal region past the obstacles.

### 6.6 Summary

We have presented a semi-supervised approach for synthesizing a trajectory-feedback controller designed to satisfy a desired STL specification. Specifically, by leveraging expert demonstrations as a form of regularization, we were able to improve the policy search and converge to a more desirable local optimum. To account for disturbances and lack of guarantees in neural network controllers, the controller was supplemented with an online adaptation scheme. We showed through an illustrative case study that the combined offline and online schemes achieve better performance and computation times compared to a state-of-the-art shooting approach. Future directions include (i) learning a value function to avoid long roll-outs and therefore reduce the computation time, (ii) using more complex dynamics (e.g., expressed as neural networks) and images such as a map or camera images attached to the robot, and (iii) extending the controller to account for multi-agent settings where the STL specifications become even more complex.
Chapter 7

Conclusion

In this chapter, we summarize the contributions of this dissertation and outline future research directions sparked as a consequence of the work presented here, and also touching on broader research questions in the field of safe and trustworthy robot autonomy.

7.1 Dissertation Summary

In this dissertation, we have developed modeling techniques, algorithms, and tools for understanding uncertainty in human behaviors, safety assurance within probabilistic planning frameworks, and providing structure into algorithms that underpin the robot autonomy stack. We have been motivated by the challenge of using deep learning for safety-critical applications. On the one hand, deep learning has opened up new and powerful modeling paradigms to perceive, understand, and predict a rapidly evolving and stochastic environment, such as those with humans present. On the other hand, it is difficult to validate the performance and robustness of deep neural networks, understand what a deep neural network has “learned” from data, and have downstream modules trust the outputs. Accordingly, we investigated frameworks to integrate techniques from formal methods within a learning-enabled robot planning and control stack to provide a layer of safety assurance and interpretability. We leverage the strengths of deep neural networks in places where their benefits are most valued while leveraging strong theoretical backing from formal methods to guide how the neural networks should be utilized.

In Part III of this dissertation, we saw that deep generative models are extremely powerful and suitable for learning human prediction models. However, since these models are learned directly from data and therefore are only approximations of the true model, and we are dealing with highly complex distributions where we do not fully understand the tails of, it is difficult for a robot to make safe and robust plans that account for all possible future actions and reactions of other agents.
in the environment. As such, we utilize results from Hamilton-Jacobi reachability theory, a formal verification technique, to inform a low-level controller that is downstream of the learning-based robot planner when the outputs of a learning-enabled model-based planner should not be trusted and how to respond accordingly.

In Part II of this dissertation, we revisited the challenge of better capturing the underlying structure in how humans behave by noting that humans do not behave in a completely random fashion, but rather, are guided by rules or intents that inform their motion. For instance, there are road rules that govern how drivers on the road should behave. While such rules or intents are encoded in data and can be “learned”, they are often treated implicitly and any “interpretable behaviors” are viewed as an emergent result. In our work, we aimed to make rules and intents explicit. We developed stlcg, a computational technique that expresses spatio-temporal specifications in the same language as computation graphs, and demonstrated how stlcg can be used to seamlessly incorporated spatio-temporal structure in a variety of modeling, planning, and analysis techniques. We investigated the problem of control synthesis and showed that if we leverage the spatio-temporal structure of the problem, we are still able to construct a neural network control even in a very low data regime.

7.2 Future Research Directions

This dissertation aimed to address specific elements of broader and more challenging research questions under the banner of “safe and robust robot autonomy”. In particular, we approached the challenge by leveraging techniques from the field of formal methods. In this section, we outline directions for future work and contextualize them with broader research questions.

**Interpretable Deep Generative Models**: As introduced in Chapter 2, deep generative models currently dominate state-of-the-art methods in human behavior prediction—they are populating the top spots of leaderboards in autonomous driving behavior prediction challenges. Despite their success, there is still the challenge of ensuring that the predictions are cognizant of road rules and reasonable driving behaviors, such as slowing down to let another vehicle pass in front. While such behaviors may be emergent, it is not explicitly accounted for during the training process. The motivation behind better understanding the underlying structure of the prediction is that it may be used to streamline the downstream planning process and pave way for more efficient and intelligent decision-making algorithms. Additionally, improving the connection between prediction and planning is another interesting research direction in itself. Future work entails the integration of stlcg within behavior prediction neural network architectures and loss functions such that the resulting behaviors are guided by well-understood spatio-temporal specifications that stem from domain expertise. For instance, a direction would be to explore how stlcg could be used to augment latent spaces of deep generative models to provide more structure and interpretability into the prediction.
7.2. FUTURE RESEARCH DIRECTIONS

Typically, latent spaces are designed to learn any salient features of data in an unsupervised fashion and any meaningful or interpretable elements of the latent space are analyzed after the training process and unknown before training.

**Safety Throughout the Autonomy Stack**: In Chapters 3 and 4, we investigated how to provide safety assurance within a probabilistic planning framework. Specifically, we leveraged Hamilton-Jacobi reachability theory which provides theoretical guarantees regarding regions of inevitable collision under worst-case policies by other interacting agents. While we showed the efficacy of our work in human-in-the-loop experiments on a full-scale vehicle, there were some drawbacks with our approach regarding the conservatism of the unsafe set. An interesting research avenue is investigating how the definition of safety, typically defined via the signed distance between two geometric shapes, could be refined to capture more nuanced notions of safety, such as capturing elements of collision severity. Furthermore, much of this exploration could be data-driven—we could learn from real human driving data when and how evasive maneuvers are taken and use that data to inform the shape of the unsafe set. However, this leads to another interesting research question of how to collect such datasets or filter out interesting safety-critical scenarios because many readily available driving datasets contain predominantly average driving behaviors where safety-critical events are extremely rare or essentially non-existent. In particular, how does one measure the degree of “safety-criticality” and “interactiveness”? Developing a principled mathematical framework describing the safety-criticality and interactiveness of a scenario is also an interesting research direction. Complementary to these research directions is understanding the role of safety throughout the autonomy stack. Chapter 4 provided a preliminary study in how shared notions of safety in the planner and controller can provide more efficient behaviors. Further exploration is needed to better understand the implications of such systems integration, and possibly derive some theoretical backing to support such design choices and reason principally the benefits of one design choice over another.

**Human-in-the-Loop Experimental Validation**: A core focus of this dissertation is on safety-critical human-robot interactions. As such, human-in-the-loop experiments are necessary in order to validate an autonomy stack designed for such an application. However, performing full-scale immersive physical experiments may be infeasible, due to costs, accessibility to resources, and safety considerations. Simulation may be a suitable alternative but many simulation software and models do not adequately capture naturalistic human decision-making that may be critical to the experiment, nor does the user-interface experience provide sufficient realism and therefore humans will behave differently. Developing algorithms and computational techniques to simulate realistic human behaviors for simulation is in itself an interesting research topic. Assuming that adequately immersive human-in-the-loop experiments were possible, an important and currently under-investigated research question is in developing validation metrics for planning and control in safety-critical scenarios. This is related to questions related to developing measures of safety-criticality and interactiveness discussed in the previous future direction. These are challenging questions because a
successful interaction is one where nothing seemingly happens— the objective was reached and no collision occurred. How do we distinguish interactions where danger was imminent but was evaded because of the proposed autonomy stack with those interactions where danger was never imminent in the first place?

7.3 Final Remarks

As the field of robot autonomy moves towards a regime where many elements of the autonomy stack are data-driven and rely on deep neural networks, it is important to keep in mind not only where and when data and deep learning are the most beneficial, and also when and where they could potentially threaten the integrity of the system. Safety is of utmost importance and therefore needs to be principled in its definition and well-understood in how it is applied throughout the autonomy stack. In addition to safety, it is also important to understand where prior and domain knowledge of the application can aid in the design of components of a robot autonomy stack. While we are in an era where it is possible to collect large datasets and use deep learning to “automatically” extract relevant information from data, we should also directly leverage domain knowledge as a form of inductive bias. A motivating reason why it is important to continually ground notions of safety and leverage domain knowledge despite the tremendous growth and usage of deep neural networks and data-driven techniques is that if a robot were to be involved in a collision, or violated rules, it will still be possible to explain why a robot did what it did, and therefore be a data point on which improvements can be made upon.
Appendices
Appendix A

Tracking Controller Parameters

A.1 Vehicle Parameters

See Table A.1 for X1 parameter values relevant to (3.8).

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$g$</td>
<td>Standard Earth gravity</td>
<td>9.80665 ms$^{-2}$</td>
</tr>
<tr>
<td>$m$</td>
<td>Total mass</td>
<td>1964 kg</td>
</tr>
<tr>
<td>$I_{zz}$</td>
<td>Yaw moment of inertia</td>
<td>2900 kgm$^2$</td>
</tr>
<tr>
<td>$h$</td>
<td>Vertical height of CG</td>
<td>0.47 m</td>
</tr>
<tr>
<td>$d_f$</td>
<td>Distance from CG to front axle</td>
<td>1.4978 m</td>
</tr>
<tr>
<td>$d_r$</td>
<td>Distance from CG to rear axle</td>
<td>1.3722 m</td>
</tr>
<tr>
<td>$C_{\alpha_f}$</td>
<td>Front cornering stiffness</td>
<td>150 kN/rad</td>
</tr>
<tr>
<td>$C_{\alpha_r}$</td>
<td>Rear cornering stiffness</td>
<td>220 kN/rad</td>
</tr>
<tr>
<td>$C_{d_0}$</td>
<td>Drag coefficient constant</td>
<td>241 N</td>
</tr>
<tr>
<td>$C_{d_1}$</td>
<td>Drag coefficient linear</td>
<td>25.1 Nm$^{-1}$s</td>
</tr>
<tr>
<td>$C_{d_2}$</td>
<td>Drag coefficient quadratic</td>
<td>0.0 Nm$^{-2}$s$^2$</td>
</tr>
<tr>
<td>$F_{x,f}$</td>
<td>Front wheel drive fraction ($F_x \geq 0$)</td>
<td>0</td>
</tr>
<tr>
<td>$F_{x,r}$</td>
<td>Rear wheel drive fraction ($F_x \geq 0$)</td>
<td>1</td>
</tr>
<tr>
<td>$F_{x,f}$</td>
<td>Front wheel brake fraction ($F_x &lt; 0$)</td>
<td>0.6</td>
</tr>
<tr>
<td>$F_{x,r}$</td>
<td>Rear wheel brake fraction ($F_x &lt; 0$)</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Table A.1: X1 parameters relevant to defining the equations of motion in (3.8).

A.2 MPC Parameters

See Table A.2 for parameter values of the MPC tracking optimization problem detailed in (3.13).
### A.2. MPC PARAMETERS

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Q_{\Delta s}$</td>
<td>Quadratic cost on longitudinal error</td>
<td>$1.0\text{m}^{-2}\text{s}^{-1}$</td>
</tr>
<tr>
<td>$Q_{\Delta \psi}$</td>
<td>Quadratic cost on heading error</td>
<td>$1.0\text{s}^{-1}$</td>
</tr>
<tr>
<td>$Q_e$</td>
<td>Quadratic cost on lateral error</td>
<td>$1.0\text{m}^{-2}\text{s}^{-1}$</td>
</tr>
<tr>
<td>$R_{\Delta \delta}$</td>
<td>Quadratic cost on change in steering angle</td>
<td>$0.1\text{s}$</td>
</tr>
<tr>
<td>$R_{\Delta F_x}$</td>
<td>Quadratic cost on change in longitudinal tire force</td>
<td>$0.5\text{N}^{-2}\text{s}$</td>
</tr>
<tr>
<td>$W_{\beta}$</td>
<td>Linear cost on sideslip stability slack variable</td>
<td>$\frac{900}{\pi}\text{s}^{-1}$</td>
</tr>
<tr>
<td>$W_r$</td>
<td>Linear cost on yaw rate stability slack variable</td>
<td>$50.0$</td>
</tr>
<tr>
<td>$W_{\text{HJI}}$</td>
<td>Linear cost on HJI slack variable</td>
<td>$500.0\text{m}^{-1}$</td>
</tr>
<tr>
<td>$W_e$</td>
<td>Linear cost on lateral bound slack variable</td>
<td>$500.0\text{m}^{-1}\text{s}^{-1}$</td>
</tr>
<tr>
<td>$\delta_{\min}$</td>
<td>Minimum steering angle</td>
<td>$-18 \times \frac{\pi}{180}$</td>
</tr>
<tr>
<td>$\delta_{\max}$</td>
<td>Maximum steering angle</td>
<td>$18 \times \frac{\pi}{180}$</td>
</tr>
<tr>
<td>$\dot{\delta}_{\min}$</td>
<td>Minimum steering rate</td>
<td>$-0.344 \text{ s}^{-1}$</td>
</tr>
<tr>
<td>$\dot{\delta}_{\max}$</td>
<td>Maximum steering rate</td>
<td>$0.344 \text{ s}^{-1}$</td>
</tr>
<tr>
<td>$F_{x,\min}$</td>
<td>Minimum longitudinal tire force</td>
<td>$-16794\text{N}$</td>
</tr>
<tr>
<td>$F_{x,\max}$</td>
<td>Maximum longitudinal tire force</td>
<td>$\min(5600, \frac{75000}{11})\text{N}$</td>
</tr>
<tr>
<td>$U_{x,\min}$</td>
<td>Minimum longitudinal velocity</td>
<td>$1.0\text{ms}^{-1}$</td>
</tr>
<tr>
<td>$U_{x,\max}$</td>
<td>Maximum longitudinal velocity</td>
<td>$15\text{ms}^{-1}$</td>
</tr>
<tr>
<td>$N_{\text{long}}$</td>
<td>Number of long MPC time steps</td>
<td>$10$</td>
</tr>
<tr>
<td>$N_{\text{short}}$</td>
<td>Number of short MPC time steps</td>
<td>$5$</td>
</tr>
<tr>
<td>$N_{\text{HJI}}$</td>
<td>Number of time steps with HJI constraint applied</td>
<td>$3$</td>
</tr>
<tr>
<td>$\Delta t_{\text{long}}$</td>
<td>Length of long MPC time step</td>
<td>$0.2\text{s}$</td>
</tr>
<tr>
<td>$\Delta t_{\text{short}}$</td>
<td>Length of short MPC time step</td>
<td>$0.01\text{s}$</td>
</tr>
<tr>
<td>$\epsilon$</td>
<td>Size of HJI value function buffer</td>
<td>$0.05\text{m}$</td>
</tr>
</tbody>
</table>

Table A.2: Parameter values relevant for the MPC tracking optimization in (3.13).
Appendix B

STL Control Synthesis

B.1 Neural Network Parameters

Here, we provide details of the neural network architecture used to synthesize an STL controller described in Chapter 6. Note that the image $e$ is of size (number of channels) 480 x 480. In our experiments, there were four channels. The state dimension of our system is $n = 4$ and with control input of size $m = 2$. The size of the hidden state for the LSTM cell is $n_h = 128$.

- Summarizing $e$ into a hidden state ($g^{[e] \rightarrow n_c}_{\text{CNN}}$): Our CNN has 2 layers. The first convolution layer applies four 8x8 filters with a stride of 4 and 4 padded zeros per spatial dimension. This is followed by batch normalization, the ReLU nonlinear activation function, and 4x4 max-pooling. The second convolution layer then applies a single 8x8 filter with a stride of 4 and 4 padded zeros per spatial dimension to yield the final CNN output $c_E$ with $n_c = 64$.

- Initializing LSTM hidden state ($g^{n_c \rightarrow n_h}_{\text{MLP}}$): Two identically-sized three layer MLPs are used, each to project $c_E$ into each component of the LSTM hidden state which is a 2-tuple of a $n_h$ dimensional vector. Each MLP is of the form: a linear layer of size $(64 + n, n_h)$ with bias, followed by a tanh() activation, a linear layer $(n_h, n_h)$ with bias, another tanh() activation, and then another linear layer $(n_h, n_h)$.

- LSTM cell ($g^n_{\text{LSTM}}$): An LSTM cell with hidden state size of $n_h$.

- Projecting hidden state to control ($g^{n_h \rightarrow m}_{\text{MLP}}$): A single linear layer of size $(n_h, m)$ with bias.

We detail the hyperparameters used in the offline training process below.

- We use the Adam optimizer with PyTorch’s default parameters and a weight decay = 0.05.

- $N_{\text{full}} = 200$: The number of training epochs used to train the model before applying adversarial training iterations.
B.1. NEURAL NETWORK PARAMETERS

- \( N_{\text{adv}} = 128 \): The number of adversarial samples to search for.
- \( N_{\text{mini}} = 20 \): The number of training epochs used to train the model during the adversarial training iterations.
- \( N = 128 \): Number of samples in the training set. We used a mini-batch size of 8 during our stochastic gradient descent steps.
- \( \gamma_{\text{imit}} = 0.5 \): The weight on control reconstruction when computing \( L_{\text{imit}} \).
- We use a sigmoidal function \( \sigma_{\text{anneal}}(i, l, u, b, c) = l + (u - l) \frac{\exp\left(\frac{i-b}{c}\right)}{1 + \exp\left(\frac{i-b}{c}\right)} \) to characterize the annealing schedule various hyperparameters. Let \( i \) denote the current training epoch, and \( N \) denote the number of iterations for that training round (either \( N_{\text{full}} = 200 \) or \( N_{\text{mini}} = 20 \)). Let \( l \) and \( u \) denote the smallest and largest value that the annealed hyperparameter can take.

For the full training phase where \( N = N_{\text{full}}, b = \frac{8N}{1000} \) and \( c = 6 \):
- \( p_{\text{LSTM}}(i) = \sigma_{\text{anneal}}(i, 0.1, 1.0, b, c) \)
- \( \gamma(i) = \sigma_{\text{anneal}}(i, 0.1, \gamma, b, c) \)
- \( \beta_{\text{STL}}(i) = \sigma_{\text{anneal}}(i, 0.1, 50, b, c) \)

For the training phase during the adversarial training iterations where \( N = N_{\text{mini}}, b = \frac{8N}{1000} \) and \( c = 6 \):
- \( p_{\text{LSTM}}(i) = \sigma_{\text{anneal}}(i, 0.8, 1.0, b, c) \)
- \( \gamma(i) = \frac{3}{2} \gamma \)
- \( \beta_{\text{STL}}(i) = \sigma_{\text{anneal}}(i, 20, 50, b, c) \)
Bibliography


130

duty vehicle platooning: A game theoretical approach,” *IFAC World Congress*, vol. 44, no. 1,

K. Margellos and J. Lygeros, “Hamilton-Jacobi formulation for reach-avoid differential games,”

safety and goal satisfaction of unmanned aerial platoons on air highways,” *AIAA Journal of


safe navigation in partially known environments,” in *Proc. IEEE Conf. on Robotics and
Automation*, 2015.

A. Gray, Y. Gao, J. K. Hedrick, and F. Borrelli, “Robust predictive control for semi-
autonomous vehicles with an uncertain driver model,” in *IEEE Intelligent Vehicles Symposium*,
2013.

M. Brown, J. Funke, S. Erlien, and J. C. Gerdes, “Safe driving envelopes for path tracking in

M. Althoff and J. Dolan, “Online verification of automated road vehicles using reachability


A. Kurzhanski and P. Varaiya, “Ellipsoidal techniques for reachability analysis: Internal ap-

G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard,

M. Althoff and B. H. Krogh, “Reachability analysis of nonlinear differential-algebraic systems,”

A. Majumdar, R. Vasudevan, M. M. Tobenkin, and R. Tedrake, “Convex optimization of
nonlinear feedback controllers via occupation measures,” *Int. Journal of Robotics Research*,
vol. 33, no. 9, pp. 1209–1230, 2014.
[63] M. Chen and contributors. helperOC Library. Available at https://github.com/HJReachability/helperOC.


