

Safe, Robust and Adaptive Model-based Learning for Agile Robots: Autonomous Racing

Thesis by
Dvij Kalaria
CMU-RI-TR-24-53

In Partial Fulfillment of the Requirements for the
Degree of *Masters of Science in Robotics*



The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:

Prof. John M. Dolan, *chair*

Prof. Guanya Shi

Prof. Qin Lin

Simin Liu

Defended 3 August 2024

© **Thesis Committee:**

Prof. John M. Dolan, *chair*

Prof. Guanya Shi

Prof. Qin Lin

Simin Liu

Dvij Kalaria

CMU-RI-TR-24-53

ORCID: 0009-0003-3610-9921

All rights reserved

ACKNOWLEDGEMENTS

My experience during my master’s program at CMU has been unparalleled. I am forever grateful for this wonderful chapter in my life that CMU has given me. I consider myself extremely fortunate to have had the support of incredible mentors, friends, and family throughout my master’s program and different stages of my life that has led me to it.

First and foremost, I would like to express my deep and heartfelt gratitude to my advisor, Prof. John Dolan. His unwavering and unconditional support, profound wisdom, and constant encouragement have helped me thrive over these past two years. I have immense respect for his passion and commitment to research, which has been a tremendous source of inspiration for me. I deeply admire his qualities: how organized, punctual, humble, and supportive he is to everyone. I look up to him as a role model and would consider myself lucky to inculcate even a small fraction of those qualities. I am highly grateful for the opportunity to work and learn from him for over three years now. During this time, he has helped me grow not only as a researcher but also as a person and an orator. His valuable constructive feedback on my presentation skills and the opportunities he has provided for me to present as a teaching assistant have been instrumental in my development.

I would also like to express my gratitude to Prof. Qin Lin, who has co-advised me on most of my thesis projects. His insights and attention to detail have inspired me to strive for excellence in everything I undertook during my master’s program. His remarkable ability to be highly ambitious and to choose impactful yet practically feasible problems has forever altered my approach to research and life. His wisdom and support in helping me master formal paper writing skills is something I am forever indebted to. He has been a constant source of support, not only as a research advisor but also as a friendly mentor, consoling and encouraging me to never give up after conference rejections.

Additionally, I would like to extend my gratitude to Prof. Guanya for his support throughout my ongoing project. His valuable insights and brainstorming sessions during my last semester have been very impactful in shaping my future research direction. I would also like to thank Rachel Burcin and the entire RISS cohort, who were instrumental in nurturing my research instincts before I joined the master’s program.

I would like to thank all the current and past DRIVE lab members during my tenure in alphabetical order: Adam Villafior, Chen Fu, Emanuel Samir, Haoru Xue, Juan Alvarez-Padilla, Kai Yun, Shivesh Khaitan, Simin Liu, Srujan Deolasee, Tony Tao, Wenli Xiao, Yiwei Lyu and Yogita Choudhary. I gained so much knowledge from each one of them, sharing both

joyful memories and profound research discussions. Special thanks to Haoru and Wenli, with whom I spent the most time and collaborated on research projects. I thoroughly enjoyed the lab dinners, late-night experiments, and recent lawn tennis sessions, which I hope will continue in the future during my PhD. A heartfelt thanks to Juan, who has been a close friend over the last two years. He has been one of the most cheerful labmates I have had. Halloween and birthday parties organized by him, my futile attempts to learn salsa with him, our long rants, and our research discussions are memories I will cherish forever. Sadly, despite many attempts, we could not collaborate on research, but I hope we will have the opportunity to do so during my PhD.

Further, I would like to thank all my AIRacingTech teammates, collaborators, and friends. AIRacingTech is a collaborative team between UC San Diego, UC Berkeley, Hawaii University, and CMU, participates in the Indy Autonomous Challenge. Special thanks to Mr. Gary Passon, Dr. Allen Y. Yang, Prof. S. Shankar Sastry, Prof. Jack Silberman, and Prof. John Dolan, who have been great team principals and faculty leads who have fostered the growth of AIRacingTech. Under their guidance, I have had the wonderful opportunity to deploy my research on a full-scale Indy racing car. Special thanks to Wade Kogline and Robert Webber, who have ensured that we are always ready to go out for practice and race. I consider myself very lucky to have worked with some of the most brilliant minds in the team. I would like to thank Siddharth Saha, Haoru Xue, C.K. Wolfe, Eric Berndt, Chris Lai, Wayne Lai, Raymond Song, Dominic Nightangle, Tianlun Zhang, Adith Sundram, and "Cake" Deng, whom I have had the opportunity to closely work with so far.

I would also like to acknowledge the invaluable presence of my friends outside the lab, without whom this journey would have been impossible. In alphabetical order, I want to recognize Aman Mehra, Anish Madan, Atharva Naik, Bharath Raj, Jay Karkhade, Mansi Saraswata, Nikhil Keetha, Poorvi Hebbar, Pushkal Katara, Prachi Garg, Pranay Gupta, Rutav Shah, Sarthak Bhagat, Shagun Uppal, Shreya Sharma, Sriram Narayanan, Srujan Deolasee, Tanvi Dev, and Yash Butala. I will cherish many moments, including impromptu trips, coffee chats, our daily tradition of playing table tennis in the RoboLounge, morning hikes, festival celebrations, random outings, and some meaningful, while some pointless, long conversations. Special mention to Shreya, who has always been kind enough to lend an ear, lift my spirits in times of need, and encourage me with constructive feedback to strive for a better version of myself. Thanks to Mansi, who has been kind and patient enough to listen to all my rants during our impromptu meal plans. Special thanks also to Rutav, who has been my close friend since undergrad, and despite living 1500 miles apart, we have maintained a strong bond through our spontaneous adventure trips.

Last but not least, I want to express my profound gratitude to my parents, whose persistent courage propelled me 8,000 miles to pursue my studies. They are the cornerstone of my journey, and I attribute all my achievements to them. They supported me through tough times and instilled in me unshakeable confidence, acting as my rock during the stormiest periods. I am forever indebted to their unconditional love and staunch belief in me, which they maintained while making countless sacrifices themselves. I highly uphold the values they taught me, including the importance of education. Special thanks to my mom, who has been my prime source of inspiration in pursuing research as a career. I greatly admire her dedication to my education, taking time out of her busy career to tutor me and ensure I understood the fundamentals of science during my middle and high school years. I would also like to extend my gratitude to my uncle, aunt, brother, maternal grandfather, late grandparents, and close relatives who have always supported me during my growing years.

I conclude this section with a popular Sanskrit phrase I grew up with: "Vasudhaiva Kutumbakam" which translates to "The world is one family" and I pledge to dedicate my research career to making it a better place for everyone.

ABSTRACT

In recent years there has been a rapid development in agile robots capable of operating at their limits in dynamic environments. Autonomous racing and recent developments in it also spurred by competitions such as the Indy Autonomous Challenge, A2RL, and F1Tenth have shown how modern autonomous control algorithms are capable of operating racecars at their handling limits and also possibly capable of beating humans in the near future. What becomes challenging for such agile robots is their need to adapt rapidly to changing environmental conditions like surface change due to tire degradation, weather change, tire temperature changes in racing; and also the need to maintain a high control frequency necessary to execute such demanding tasks. The robots also need to stay safe while executing such tasks. In this work we show how we can leverage safe learning for agile robots operating at their limits, which safely enables them to learn and exploit their limits. We take autonomous racing as a case study, but will formulate an approach which can be applied generally to any agile control system.

PUBLISHED CONTENT AND CONTRIBUTIONS

- [Kal+24] Dvij Kalaria et al. “WROOM: An Autonomous Driving Approach for Off-Road Navigation”. In: *ArXiv* abs/2404.08855 (2024). URL: <https://api.semanticscholar.org/CorpusID:269149459>.
- [KLD23a] Dvij Kalaria, Qin Lin, and John M. Dolan. “Adaptive Planning and Control with Time-Varying Tire Models for Autonomous Racing Using Extreme Learning Machine”. In: *ArXiv* abs/2303.08235 (2023). URL: <https://api.semanticscholar.org/CorpusID:257532643>.
- [KLD23b] Dvij Kalaria, Qin Lin, and John M. Dolan. “Towards Optimal Head-to-head Autonomous Racing with Curriculum Reinforcement Learning”. In: *ArXiv* abs/2308.13491 (2023). URL: <https://api.semanticscholar.org/CorpusID:261214486>.
- [KLD23c] Dvij Kalaria, Qin Lin, and John M. Dolan. “Towards Safety Assured End-to-End Vision-Based Control for Autonomous Racing”. In: *ArXiv* abs/2303.02267 (2023). URL: <https://api.semanticscholar.org/CorpusID:257364884>.
- [KLD22a] Dvij Kalaria, Qin Lin, and John M Dolan. “Delay-aware robust control for safe autonomous driving”. In: *2022 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2022, pp. 1565–1571.
- [KLD22b] Dvij Kalaria, Qin Lin, and John M. Dolan. “Delay-Aware Robust Control for Safe Autonomous Driving and Racing”. In: *IEEE Transactions on Intelligent Transportation Systems* 25 (2022), pp. 7140–7150. URL: <https://api.semanticscholar.org/CorpusID:260499046>.
- [Mun+22] Emanuel Munoz et al. “Online Adaptive Compensation for Model Uncertainty Using Extreme Learning Machine-based Control Barrier Functions”. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 10959–10966. DOI: 10.1109/IROS47612.2022.9981680.
- [Kal+21] Dvij Kalaria et al. “Local NMPC on Global Optimised Path for Autonomous Racing”. In: *ArXiv* abs/2109.07105 (2021).

TABLE OF CONTENTS

Acknowledgements	iii
Abstract	vi
Published Content and Contributions	vii
Table of Contents	vii
List of Illustrations	xii
List of Tables	xvii
Chapter I: Introduction	1
Chapter II: Preliminaries	3
2.1 Racing Line	3
2.2 Vehicle models	4
Kinematic model	4
Dynamic model	5
Extended kinematic model	5
2.3 Control barrier functions	6
Chapter III: Delay-aware robust control	7
3.1 Introduction	7
3.2 Related work	10
3.3 Methodology	11
Notation	11
System dynamics	11
Robust Tube MPC	12
Delay-aware robust tube MPC	13
Control limits ($\mathcal{U}$)	15
State CBF Constraints ($\mathcal{X}$)	15
Disturbance-invariant set ($\mathcal{Z}$)	15
Stability	16
Feasibility	16
INFLUENCE for computation time delay	17
Controller Plan A	18
Controller Plan B	18
Lane constraints	19
Collision avoidance constraints	20
3.4 Experimental Results	21
Controller plan A	21
Static obstacle avoidance (non-racing)	21
Racing scenario	22
Sudden change scenario (racing)	22
Overtaking in a turn scenario (racing)	24
Controller plan B	24

Path following in racing scenario	24
Overtaking at turn in racing scenario	26
Hardware experiments in an RC car	27
Experiment setup	27
Results	28
3.5 Conclusion	28
Chapter IV: Online model-based adaptive learning controller for autonomous racing .	30
4.1 Introduction	30
4.2 Preliminaries	33
Extreme Learning Machine	33
4.3 Methodology	34
Learning based control	34
Data collection	34
Training and online adaptation	35
Adaptive speed planning	35
4.4 Experimental Results	37
Numeric simulator	37
Constant friction decay	38
Sudden friction change	38
Carla	40
Hardware experiments in an RC car	41
Experiment setup	41
Results	41
4.5 Conclusion	42
Chapter V: Safe reinforcement learning	44
5.1 Introduction	44
5.2 Preliminaries	46
Disturbance Observer (DOB)	46
Model Residual learning	47
5.3 Methodology	48
Residual model and DOB-based Control Barrier Function (RES-DOB-CBF)	48
Safe-RL policy training with RES-DOB-CBFs	48
5.4 Experimental Results	48
Point robot experiments	49
Car robot experiments	52
RC car results	54
5.5 Case study: Head-to-head autonomous racing	54
High-level planner	55
State discretization	56
State transition model	56
High-level optimization	57
Low-level controller	57
Reward design	57
Network architecture	58
Training environment	58

Vehicle model transition	58
Safety CBF	59
Simulation Results	61
Training results	61
Comparison with baselines	62
5.6 Case study: Offroad driving	64
Simulator design	64
Reinforcement Learning (RL)	67
Safety CBF	68
Reward Design	68
Policy distillation	69
Results	69
5.7 Conclusion	71
Chapter VI: Decoupled safety and optimality for end-to-end autonomous racing with automated-imitation learning	72
6.1 Introduction	72
6.2 Imitation learning for autonomous racing	74
Problem definition	74
DAgger	75
Network architecture	76
Expert controller	76
6.3 Safety controller	77
Control Barrier Functions (CBFs)	77
Probabilistic Control Barrier Functions	79
Accounting for uncertainty in DNN predictions	80
6.4 Training cycle	81
6.5 Experimental results	82
Carla simulator	82
Track	82
Vehicle model and Hyperparameter settings	83
Experiments	83
RC car	83
Track	84
Localization for training	85
Vehicle model	85
Experiments	87
Visualizing feature maps	87
6.6 Conclusion	87
Chapter VII: Towards foundation model for fully learnable model-based control for autonomous racing	89
7.1 Previous works	89
7.2 Methodology	90
Meta learning	90
Model architecture	90
Uncertainty estimation	91

Model Predictive Path Integral (MPPI) Control	92
Uncertainty-aware MPPI	93
Online adaptation	93
7.3 Results	94
Numeric sim	94
Unity sim	95
Hardware platform	101
7.4 Conclusion	102
Chapter VIII: Conclusion and Future Work	103
Bibliography	105

LIST OF ILLUSTRATIONS

<i>Number</i>		<i>Page</i>
1.1	Venn diagram describing what each chapter aims to solve	2
2.1	Example (a) raceline and (b) minimum curvature line on a track meant for 1/31 th scale racecar	4
3.1	An illustration comparing a conventional control and a delay-aware control. The conventional method (case A) always uses the current state to compute the future control action. Due to a delay, the computed action will not be instantaneously executed (see the shifted steering wheel symbols from black to blue). Our controller is aware of the delay, thus in the earliest time step (see the leftmost calculated command in case B), a steering command has been calculated. The movements in case A (orange) are from the late steering command sequence shown in the first row of blue steering wheel symbols. Clearly, the movements in case B (blue) are safer.	8
3.2	Overview of two control plans. Plan A incorporates the actuator's dynamics in the MPC. In Plan B, since we cannot change the black-box controller, we consider its output as the nominal control. An external optimization considers the nominal control as a reference and refines it based on the actuator's dynamics. In both plans, the type II and type III delays are estimated to determine the shifting time of the initial state. Control barrier functions are used in the two plans as safety constraints.	9
3.3	Our delay-aware robust MPC with dual cycles	14
3.4	Controller Plan A for delay-aware robust tube MPC.	19
3.5	Controller Plan B, delay compensation for an LE controller.	20
3.6	Experiment 3.4, blue: predicted computation time using INFLUENCE without variance; green: predicted computation time using INFLUENCE with variance; orange: actual computation time including INFLUENCE's computation cost; red: actual computation time without INFLUENCE's computation cost. ROS time at the x axis is the timestamp in the ROS operation system.	21
3.7	Performance comparison: red boxes (without delay compensation) show that the vehicle ran outside the road boundary while the vehicle controlled with delay compensation (blue boxes) always stays inside the boundary. The vehicle moves counterclockwise.	23

3.8	Robustness test with additional computation time from 0.02 s to 0.15 s. . . .	23
3.9	Comparison in Experiment 3.4. The opponent vehicle starts braking at point A. The ego vehicle crashes into the opponent vehicle at point B without delay compensation.	24
3.10	Comparison in Experiment 3.4. The ego vehicle crashes at point A in the case in which delay compensation is not considered.	25
3.11	Performance comparison: red boxes (w/o delay compensation) show that the vehicle ran outside the road while the vehicle controlled with delay compensation (blue) always stays inside the boundary. The vehicle moves clockwise. . .	25
3.12	Ego vehicle speed for the case when delay compensation is not considered. . .	26
3.13	Ego vehicle speed for the case when delay compensation is considered.	26
3.14	Comparison in Experiment 3.4. The ego vehicle crashes at point A without any delay compensation.	27
3.15	RC car setup	27
3.16	Performance comparison for 5 runs each: red lines (without delay compensation) show that the vehicle ran outside the road boundary while the vehicle controlled with delay compensation (blue lines) always stays inside the boundary. The vehicle moves from bottom to top.	28
3.17	Experiment 3.4, predicted computation time (blue): predicted value using INFLUENCE without adding variance; computation time taken (green): predicted value using INFLUENCE with variance; actual computation time (orange): ground truth delay.	29
4.1	Overview of our approach. Our approach consists of an online learning component marked in red with fits the residual between actual model and the nominal e-kinematic model and estimates the friction coefficient. The estimated friction coefficient is used to select the raceline that is used by the controller with the updated vehicle model to track.	32
4.2	Architecture of a single layer ELM.	33
4.3	Training cycle. In the forward pass, given C_{lf} , C_d , and two ELMs, we calculate forces, then a residual error can be obtained to further update the model in the backward pass.	36
4.4	Comparison for w/o and w/ learning on numeric sim	39
4.5	Comparison with other approaches on numeric sim	39
4.6	Online run comparison with model compensation & adaptive speed planning vs. if none are used (sudden friction change experiment)	40
4.7	Online run with model compensation in Carla	40

4.8	RC car experiment on rounded rectangle trajectory	42
4.9	Adaptation to new surface	42
5.1	Proposed safe RL framework via using RES-DOB-CBF. At each time step t , the RL agent action u_t^{RL} is filtered by RES-DOB-CBF based safety to avoid any unsafe actions	46
5.2	Comparisons for Goal1 task on Point robot	49
5.3	Comparisons for Goal2 task on Point robot	49
5.4	Comparisons for Button1 task on Point robot	49
5.5	Comparisons for Button2 task on Point robot	50
5.6	Comparisons for Push1 task on Point robot	50
5.7	Comparisons for Push2 task on Point robot	50
5.8	Reference frame of the robot for obstacle avoidance. The blue circle is the robot, and the orange circle is the moving obstacle.	51
5.9	Comparisons for Goal1 task on Car robot	52
5.10	Comparisons for Goal2 task on Car robot	52
5.11	Comparisons for Button1 task on Car robot	53
5.12	Comparisons for Button2 task on Car robot	53
5.13	Comparisons for Push1 task on Car robot	53
5.14	Snapshot of twin RC car simulator in unity	54
5.15	Trajectories followed for different approaches for RC car experiment	55
5.16	Converting a continuous state to a discrete state	56
5.17	Tire curve variation for curriculum learning	59
5.18	A snapshot of the racing simulator on the left where two racecars compete against each other to win the race. The blue vehicle is controlled by our algorithm, while the red vehicle's controller is based on combining Monte Carlo tree search and linear quadratic regulator. The green line is the racing line, moving along which guarantees best lap times. The right image shows the top-down view of the track map on which the races are conducted.	61
5.19	(a) Training rewards (b) No of wins across training steps. Ours : With using both model curricula and RES-DOB-CBF filter during learning	62
5.20	The race win statistics. For example, the first bar is for the comparison between ours with other approaches. We won all 20 races (see the value from 49 to 68 in the purple block) against the end-to-end approach.	63

5.21	Overview of the proposed approach, WROOM : The end-to-end RL agent collects depth camera and IMU measurements from the environment and outputs steering, throttle, and braking commands. The reward function evaluates not only the agent’s progress in the environment but also the smoothness and safety of its maneuvers, as reported by the control barrier function (CBF). Imitation learning is utilized to kick-start the agent, followed by Proximal Policy Optimization (PPO) to refine its policy, and finally, policy distillation is employed for real-world deployment.	65
5.22	Scandots (in purple) as privileged information to the expert controller.	66
5.23	Real-world deployment using an RC car.	66
5.24	Stills from the proposed simulation environment, OffTerSim	67
5.25	Real-world deployment of our proposed approach, WROOM	70
5.26	Training reward plots highlighting the significance of each component of the proposed approach, WROOM	71
6.1	Control prediction model architecture	76
6.2	Vehicle state representation	80
6.3	Training and testing in Carla. M_{com} : end-to-end control network; M_{st} : end-to-end state network	81
6.4	Racing line	83
6.5	Trajectories for all iterations (Carla, Racing line)	85
6.6	RC car setup	85
6.7	Racing line	86
6.8	Trajectories for all iterations (RC car, Racing line)	87
6.9	GradCAM projections on two example images	88
7.1	Model architecture	91
7.2	Trajectory followed for an instance	96
7.3	Lateral errors	97
7.4	Predicted ω	97
7.5	Circuit track	98
7.6	Steer bias	98
7.7	Decaying friction	99
7.8	Go-kart	99
7.9	Mud	100
7.10	Mars	100
7.11	Grass	100
7.12	Banked-oval track	101

7.13	Trajectory followed by RC car with vicon	102
------	--	-----

LIST OF TABLES

<i>Number</i>		<i>Page</i>
3.1	Comparison in lap times with and without delay compensation included.	22
3.2	Comparison in lap times and safety: w/ and w/o delay compensation	26
4.1	Our motivation and comparison with others	31
4.2	Track 1 (ETHZ track [LDM15])/Track 2 (ETHZMobil track [Jai+20]). Comparison in statistics for numeric simulator. Violation time is the total time the vehicle was outside the track limits.	39
4.3	Statistics for w/ and w/o model adaptation	41
5.1	Average commute times for RC car experiment	54
5.2	Quantitative ablation of individual components of WROOM using various metrics.	69
6.1	Statistics for all iterations (Carla simulator). *: Didn't complete the track . .	84
6.2	Statistics for all iterations (RC car). *: Didn't complete the track	86
7.1	Numerical results for numeric sim	95
7.2	[Lateral error (in m) / Lap time (in s)] Results table for Unity sim experiments	101

Chapter 1

INTRODUCTION

In recent years there has been a rapid development in agile robots capable of operating at their limits in dynamic environments. Autonomous racing and recent developments in it also spurred by competitions such as the Indy Autonomous Challenge, A2RL, and F1Tenth have shown how modern autonomous control algorithms are capable of operating racecars at their handling limits and also possibly capable of beating humans in the near future. What becomes challenging for such agile robots is their need to adapt rapidly to changing environmental conditions like surface change due to tire degradation, weather change, tire temperature changes in racing; and also the need to maintain a high control frequency necessary to execute such demanding tasks. The robots also need to stay safe while executing such tasks. In this work we show how we can leverage safe learning for agile robots operating at their limits, which safely enables them to learn and exploit their limits. We take autonomous racing as a case study, but will formulate an approach which can be applied generally to any agile control system.

We will first describe how we outline our works towards this common goal of achieving highly agile control algorithms to operate robots at their limits. High speed autonomous racing consists of two main objectives: 1) to formulate precise control at the operational limits; and 2) to enforce safety constraints which may require long-term planning. The second objective of enforcing safety constraints is often interlinked with the primary objective of achieving optimal behavior. This is because at the handling limits, even a little error can break the control limits, leading to the car spinning out or making it impossible to respect constraints, leading to a crash, which ultimately leads to its losing more time and affecting its optimality. Hence in autonomous racing, it is necessary to find a sweet spot between safety and agility. To address this combined objective, we are faced with various practical challenges, such as: 1) delays from computation time and physical system leading to inaccurate controls; 2) sensor noise; 3) imperfect models; 4) a dynamic environment due to changing friction surface and weather conditions. In the current work, we aim to tackle all of these challenges, which are common to a wide range of agile systems, such as drones, quadrupeds in wild environments, robots playing sports like table tennis, etc.

We first describe some preliminaries associated with autonomous racing in chapter 2 which will be used in all subsequent chapters. Venn diagram in Figure 1.1 best describes where

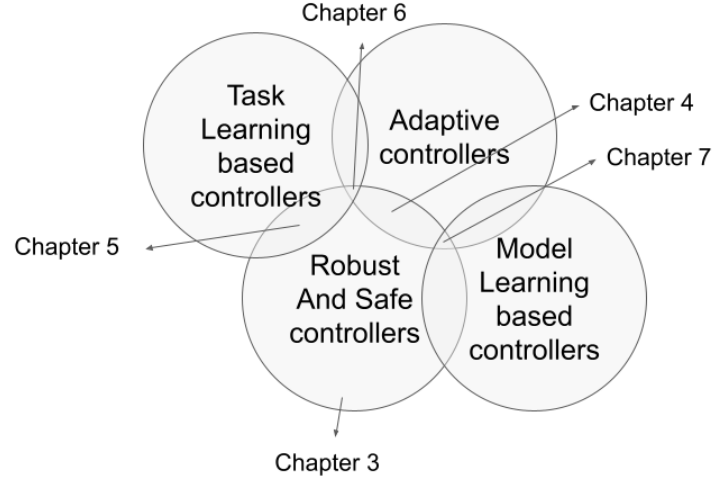


Figure 1.1: Venn diagram describing what each chapter aims to solve

the contribution of this work falls and where each chapter stands. In chapter 3, we address the issue of delays classically and modify our robust tube MPC with a delay compensation scheme which can also account for bounded model errors and sensor noises. That works well if the vehicle model parameters are accurate and do not change over time. However, for racing tire degradation is a very common phenomenon; the tires lose traction over time and thus the friction parameters actively change during the race. To account for the dynamic changes, we need to employ an adaptive solution, which we further describe in chapter 4. In chapter 5, we also discuss how we can consider separate solutions for environmental disturbances and model errors for safe model-free reinforcement learning. We show how this can improve solutions for practical deployments of reinforcement learning in autonomous head-to-head racing and offroad driving scenarios. In chapter 6 we show how decoupling safety and optimality for automated imitation learning helps significantly in terms of sample efficiency. Finally, in chapter 7, we demonstrate that a completely end-to-end model can be pre-trained via meta-learning to learn a robust initial model that can be fine-tuned quickly on any environment with zero knowledge of vehicle model and thus can be deployed on any vehicle across a wide range of types and can be quickly adapted to any environment changes.

Chapter 2

PRELIMINARIES

2.1 Racing Line

Race drivers follow a racing line for specific maneuvers. This line can be used as a reference path by the motion planner to assign time-optimal trajectories while avoiding collision. The racing line is minimum-time or minimum-curvature. They are similar, but the minimum-curvature path additionally allows the highest cornering speeds given the maximum legitimate lateral acceleration [Hei+20].

There are many proposed solutions to finding the optimal racing line, including nonlinear optimization [RB20; Hei+20], genetic algorithm-based search [Ves15] and Bayesian optimization [JM20]. However, for our work, we calculate the minimum-curvature optimal line, which is close to the optimal racing line as proposed by [Hei+20]. The race track is represented by a sequence of tuples (x_i, y_i, w_i) , $i \in \{0, \dots, N-1\}$, where (x_i, y_i) denotes the coordinate of the center location and w_i denotes the lane width at the i -th point. The output racing line consists of a tuple of seven variables: coordinates x and y , longitudinal displacement s , longitudinal velocity v_x , acceleration a_x , heading angle ψ , and curvature κ . It is obtained by minimizing the following cost:

$$\begin{aligned} \min_{\eta_1 \dots \eta_N} \quad & \sum_{n=0}^{N-1} \kappa_i^2(n) \\ \text{s.t.} \quad & \eta_i \in \left[-\frac{w_i}{2} + \frac{w_{veh}}{2}, \frac{w_i}{2} - \frac{w_{veh}}{2} \right] \end{aligned} \tag{2.1}$$

where the vehicle width is w_{veh} , and η_i is the lateral displacement with respect to the reference center line.

To create a velocity profile, we need to consider the vehicle's constraints on both longitudinal and lateral acceleration [Hei+20]. Our approach involves generating a library of velocity profiles, each tailored to specific lateral acceleration limits determined by the friction coefficients for the front (μ_f) and rear (μ_r) tires, as well as the vehicle's mass (m) and the gravitational constant (g). In particular, we produce a set of velocity profiles covering a range of maximum lateral forces corresponding to the friction μ_{eff} within the interval $[\mu_{min}, \mu_{max}]$. This library allows us to retrieve a velocity profile that matches a given value of μ . Interpolation is necessary when we encounter a friction value that falls within the valid range but is not explicitly present in the library.

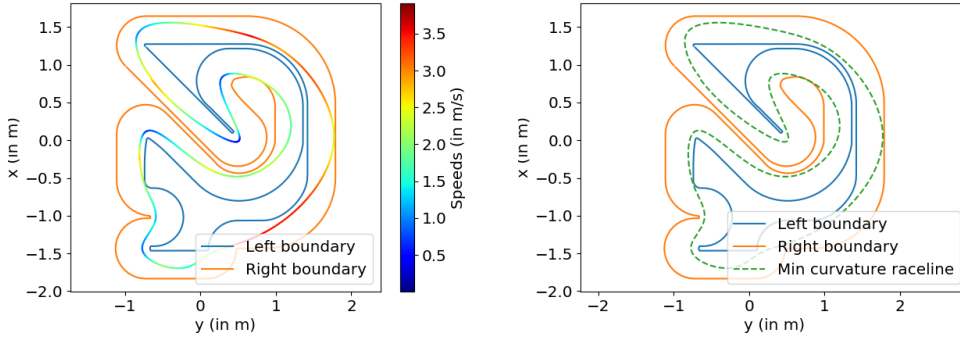


Figure 2.1: Example (a) raceline and (b) minimum curvature line on a track meant for 1/31th scale racecar

An example of a racing line calculated on racetrack meant for 1/31th size racecar [FL21] is shown in Figure 2.1. As can be seen the racing line usually close to the minimum curvature path as shown in Figure 2.1. This is to get as much high speed as possible. At certain regions it also stays closer to the geometrically minimum distance path to travel less geometric distance and get around the track as fast as possible.

2.2 Vehicle models

The most widely used vehicle models in the context of autonomous driving and racing are the kinematic and dynamic bicycle models. The kinematic model uses x , y coordinates, ϕ heading angle in the global inertial frame, velocity v , and acceleration a . The dynamic model has states x , y , and ϕ in the global frame; longitudinal velocity v_x , lateral velocity v_y , and yaw angular velocity ω in the vehicle's body frame; and steering angle δ . For the kinematic model, the control variables are acceleration a and steering δ , and throttle d and change in steering $\Delta\delta$ define the action space.

Kinematic model

This model is defined exclusively by the geometric parameters of the vehicle that can be physically measured, without taking into consideration any contact surfaces. It ignores the effect of lateral tire slip, which is crucial at high speeds. Hence, it is suitable for lower speeds, but at cornering speeds, it is inaccurate due to considerable lateral tire slip.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\phi} \\ \dot{v} \\ \dot{\delta} \\ \beta \end{bmatrix} = \begin{bmatrix} v \cos(\phi + \beta) \\ v \sin(\phi + \beta) \\ \frac{v \sin(\beta)}{l_r} \\ a \\ \frac{\Delta \delta}{\Delta t} \\ \tan^{-1}\left(\frac{l_f}{l_f + l_r} \tan \delta\right) \end{bmatrix} \quad (2.2)$$

Dynamic model

The dynamic model has states p_x, p_y , and ϕ as position and orientation in the global frame; longitudinal velocity v_x , lateral velocity v_y , and yaw angular velocity ω in the vehicle's body frame; and steering angle δ . For the dynamic model, $F_{r,x}$ is the longitudinal force on the rear tire in the tire frame assuming a rear-driven vehicle, $F_{f,y}$ and $F_{r,y}$ are the forces on the front and rear tires, respectively, and α_f and α_r are the corresponding slip angles. r and p are the roll and pitch respectively of the vehicle. We denote the mass of the vehicle m , the moment of inertia in the vertical direction about the center of mass of the vehicle I_z , the length of the vehicle from the COM (center of mass) to the front wheel l_f , and the length from the COM to the rear wheel l_r . $B_{f/r}$, $C_{f/r}$, $D_{f/r}$ are the Pacejka tire model parameters specific to the tire and track surface. C_{m1} and C_{m2} are longitudinal parameters that dictate the longitudinal force generated from the vehicle given the throttle command.

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{\phi} \\ \dot{v}_x \\ \dot{v}_y \\ \dot{\omega} \\ \dot{\delta} \end{bmatrix} = \begin{bmatrix} v_x \cos(\phi) - v_y \sin(\phi) \\ v_x \sin(\phi) + v_y \cos(\phi) \\ \omega \\ \frac{1}{m}(F_{r,x} - F_{f,y} \sin(\delta) + mv_y \omega - mg \sin(p)) \\ \frac{1}{m}(F_{r,y} + F_{f,y} \cos(\delta) - mv_x \omega + mg \sin(r)) \\ \frac{1}{I_z}(F_{f,y} l_f \cos(\delta) - F_{r,y} l_r) \\ \frac{\Delta \delta}{\Delta t} \end{bmatrix} \quad (2.3)$$

where $F_{r,x} = (C_{m1} - C_{m2}v_x)d - C_{lf} - C_d v_x^2$, $F_{f,y} = D_f \sin(C_f \tan^{-1}(B_f \alpha_f))$, $\alpha_f = \delta - \tan^{-1}\left(\frac{\omega l_f + v_y}{v_x}\right)$, and $F_{r,y} = D_r \sin(C_r \tan^{-1}(B_r \alpha_r))$, $\alpha_r = \tan^{-1}\left(\frac{\omega l_r - v_y}{v_x}\right)$.

Extended kinematic model

The extended kinematic model (**E-kinematic model**) [Jai+20] extends the commonly used kinematic bicycle model from 4-DOF to 7-DOF to make the dimension consistent with the dynamic model, since v_x , v_y and w are not in the kinematic model. Also, slip angles α_f and

α_r are assumed to be 0 in the kinematic model. It is derived just by having $\dot{\alpha}_f = 0$ and $\dot{\alpha}_r = 0$. For longitudinal forces, the effect of aerodynamic forces is ignored.

$$\begin{bmatrix} \dot{p}_x \\ \dot{p}_y \\ \dot{\phi} \\ \dot{v}_x \\ \dot{v}_y \\ \dot{\omega} \\ \dot{\delta} \end{bmatrix} = \begin{bmatrix} v_x \cos(\phi) - v_y \sin(\phi) \\ v_x \sin(\phi) + v_y \cos(\phi) \\ \omega \\ \frac{F_{r,x}}{m} \\ \frac{l_f}{l_f + l_r} (\dot{\delta} v_x + v_x \dot{\delta}) \\ \frac{1}{l_f + l_r} (\dot{\delta} v_x + v_x \dot{\delta}) \\ \frac{\Delta \delta}{\Delta t} \end{bmatrix} \quad (2.4)$$

where $F_{r,x} = (C_{m1} - C_{m2}v_x)d$. This model extends the dimension of the kinematic model to obtain the same number of states as the dynamic model. In our work, the E-kinematic model will serve as a **nominal model**. The discrepancy between the dynamic model and the nominal model will be learned and compensated by our learning component.

2.3 Control barrier functions

Control Barrier Functions (CBF) is a popular model-based tool to enforce constraints which we will be using in subsequent chapters. Hence, we give a standard introduction to CBFs. We define a continuous and differentiable safety function $h(\mathbf{x}) : \mathcal{X} \rightarrow \mathbb{R}$. The *superlevel set* $\mathcal{C} \in \mathbb{R}^n$ can be named as a safe set. Let the set $\mathcal{C}$ obey $\mathcal{C} = \{\mathbf{x} \in \mathcal{X} : h(\mathbf{x}) \geq 0\}$

The safety function $h(\mathbf{x})$ is called a CBF of a control affine system $\dot{\mathbf{x}} = f(\mathbf{x}) + g(\mathbf{x})\mathbf{u}$, if there exists a $\gamma > 0$ such that

$$\sup_{\mathbf{u} \in \mathcal{U}} [L_f h(\mathbf{x}) + L_g h(\mathbf{x})\mathbf{u} + \gamma(h(\mathbf{x}))] \geq 0 \quad (2.5)$$

for all $\mathbf{x} \in \mathcal{X}$. $L_f h(\mathbf{x})$ and $L_g h(\mathbf{x})$ are Lie derivatives. $\gamma(h(\mathbf{x}))$ is particularly chosen as a special class $\mathcal{K}$ function $\gamma h(\mathbf{x})$. The solution $\mathbf{u}$ assures that the set $\mathcal{C}$ is a forward invariant, i.e., if the solutions starting at any $\mathbf{x}(0) \in \mathcal{C}$ satisfy $\mathbf{x}(t) \in \mathcal{C}$ for $\forall t \geq 0$. Recently, CBFs have been used in conjunction with MPC [ZZS21; WH04]. Having a barrier function-based constraint in MPC not only ensures strict satisfaction of safety constraints but also facilitates a smooth trajectory.

Chapter 3

DELAY-AWARE ROBUST CONTROL

As mentioned in Chapter 1, we will first formulate a controller for autonomous racing in this chapter with known model parameters with some error and is robust to bounded environment noise. We will formally prove how this controller can be proven to be safe under mild assumptions. One more important aspect that is usually ignored in robot controller designs but is critical for dynamic problems like autonomous racing is system and computation delays. Delays endanger the safety of autonomous systems functioning in the rapidly changing environments of autonomous driving and high-speed racing. Unfortunately, the consideration of delays is often overlooked during controller design or learning-enabled controller training phases prior to deployment in the physical world. This chapter systematically and comprehensively addresses both the computation delay arising from nonlinear optimization for control and other inevitable delays caused by actuators. First, we propose a new filtering approach with no prior knowledge of dynamics and disturbance distribution to adaptively estimate the time-variant computation delay. Second, we model actuation dynamics for steering delay. Third, all the constrained optimization is realized in a robust tube model predictive controller. In terms of application merits, our approach is a novel design for a standalone delay-aware controller; in addition, our approach can also serve as the existing controller's delay compensator. Video¹ and code² are available.

3.1 Introduction

Autonomous vehicles operate in highly complex environments that demand extensive computational resources. Despite efforts to enhance algorithm efficiency, the unavoidable requirement for computation time remains. Given the risks associated with even minor delays in rapidly changing conditions such as highways and autonomous racing, it is incorrect to assume that the optimal control is instantaneously generated by an optimization program and executed on the vehicle.

Fig. 3.1 illustrates the motivation of our work in a collision avoidance scenario. The sequential movements in orange color result from a traditional controller without considering any delay, which could be hazardous in a highly dynamic environment (e.g., high-speed scenario). However, our delay-aware controller takes a safe action earlier (e.g., earlier steering

¹https://youtu.be/nURL_HTW_Mo

²<https://github.com/dvij542/Delay-aware-Robust-Tube-MPC>

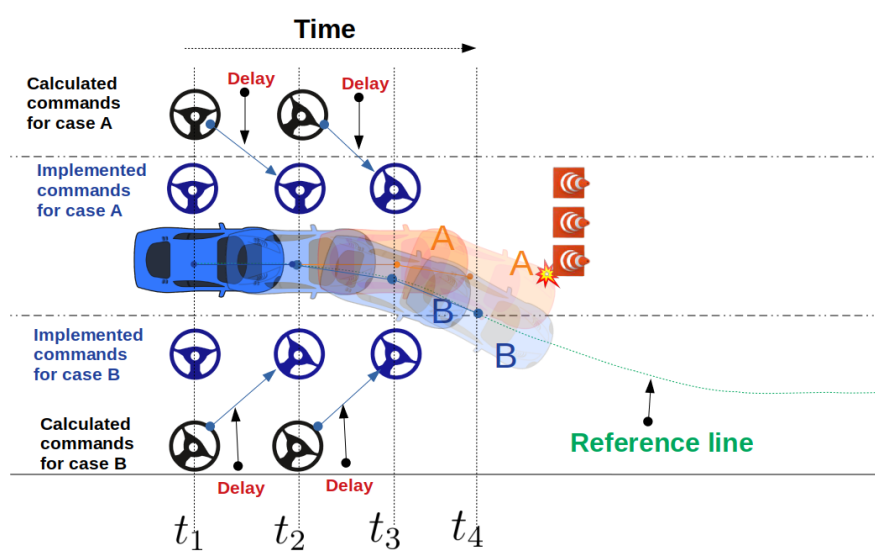


Figure 3.1: An illustration comparing a conventional control and a delay-aware control. The conventional method (case A) always uses the current state to compute the future control action. Due to a delay, the computed action will not be instantaneously executed (see the shifted steering wheel symbols from black to blue). Our controller is aware of the delay, thus in the earliest time step (see the leftmost calculated command in case B), a steering command has been calculated. The movements in case A (orange) are from the late steering command sequence shown in the first row of blue steering wheel symbols. Clearly, the movements in case B (blue) are safer.

for an evasive maneuver with the blue color). Our approach deals with three types of delay in a unified manner: 1) actuator dynamics delay; 2) control action processing delay; and 3) computation time delay. First, we augment the commonly used vehicle dynamic modeling by including steering action delay. Second, we propose a novel adaptive Kalman filter variant: INFLUENCE (adaptIve kalmaN FiLter with Unknown process model and Noise Covariance) to probabilistically and safely estimate the control action processing delay and computation time delay caused by online optimization. INFLUENCE simultaneously identifies an unknown process model and noise covariances. Third, we extend robust tube model predictive control (MPC) theory by adding a new delay-aware feature.

This study introduces two control plans, Plan A (Fig. 3.2a) and Plan B (Fig. 3.2b). Both plans address the compensation of computation and actuator processing delays by adjusting the initial system state. The key distinction between the two plans lies in the presence or absence of an existing primary controller in the system. In Plan A, we develop a standalone controller based on robust tube MPC, incorporating actuator steering as an extended state. Plan B, on the other hand, is motivated by scenarios where a primary controller, such as a legacy system or a neural network controller in a learning-enabled system, cannot be

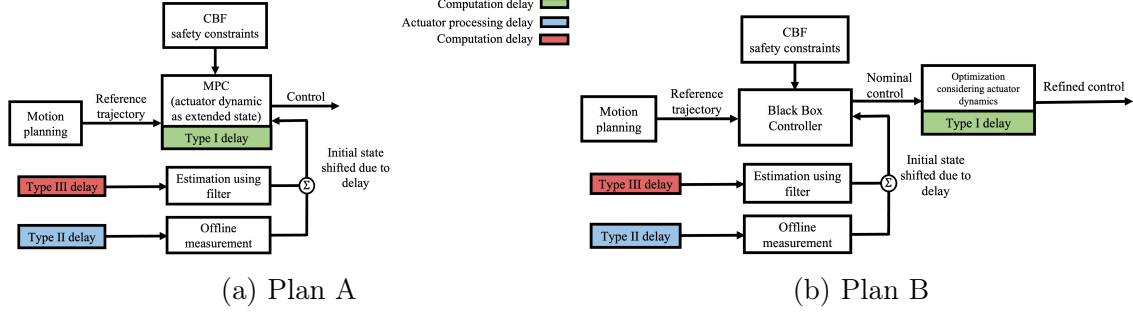


Figure 3.2: Overview of two control plans. Plan A incorporates the actuator’s dynamics in the MPC. In Plan B, since we cannot change the black-box controller, we consider its output as the nominal control. An external optimization considers the nominal control as a reference and refines it based on the actuator’s dynamics. In both plans, the type II and type III delays are estimated to determine the shifting time of the initial state. Control barrier functions are used in the two plans as safety constraints.

modified internally. As depicted in Fig. 3.2b, the original control, which disregards delay, is refined and regulated through an external optimization process that considers actuator dynamic delay. Plan B aligns with the theme of safe artificial intelligence (AI). It is common to make naive delay-free assumptions during the training of the learning-enabled control policy in a simulation environment. However, this simulation-to-real gap poses challenges for the reliable deployment of safety-critical autonomous systems, such as self-driving cars. We validate our framework in on-road autonomous driving and autonomous racing scenarios.

We make the following contributions in this chapter:

1. We extend the conventional robust tube MPC with the capacity to deal with various types of practical delays.
2. Unlike many existing filters, INFLUENCE does not require prior knowledge of model dynamics and noise distribution. In addition, its probabilistic and adaptive estimation mitigates overconservatism.
3. We propose a safety guard component to compensate for delays of a learning-enabled (LE) controller that does not consider delays. In addition, in our control optimization framework, the delay compensator is combined with barrier functions (CBFs) for collision avoidance.

The rest of this chapter is organized into the following sections. A review of some relevant related works is presented in Section 3.2. The methodology is described in Section 3.3.

The experiments and their results are presented in Section 3.4. Section 3.5 contains the conclusions and lays foundation for follow-up works in subsequent chapters.

3.2 Related work

General delay compensation control: [Cor+11] considers a discrete control problem: it proposes to shift one discrete step ahead for the initial state in MPC, for the particularly simple one-step delay problem. [STL13] uses a cache mechanism for buffering previously computed control actions. However, their approaches only works for static scenarios with fixed horizon length, which leads to a less responsive controller. The first significant improvement in our work is that our filtering approach actively estimates a time-variant local upper bound of the computation delay.

Actuation delay compensation control: To deal with delays in actuator dynamics, [Nah+19] proposes to augment the original state space model with an extra first-order ordinary differential equation (ODE) for the delay behavior. The particular application in their work is braking control. For control action processing delay, which is usually time-invariant, [LL18] proposes to compensate for such a delay by transitioning the initial state in a preview controller. [KI11] further extends the idea and considers actuator saturation. The major disadvantage of the preview controller is that it is not flexible enough to simultaneously consider multiple constraints such as dynamics, state, and control limits. Note that our work deals with delays and all constraints of state and control in a unified robust MPC optimization. [XJ22] considers a fixed communication delay in their connected cruise control problem with other vehicles driven by humans. This communication delay can be handled in a similar way to actuation delay compensation in our design.

Safe control: [AKJ21] proposes an approach for designing safety-critical controllers involving CBFs for nonlinear systems with time-varying input delay. It combines a state predictor that compensates for the time-varying input delay with a CBF-based feedback law for the nominal system, which is delay-free. We combines CBF with MPC to account for delays and uncertainties in the system and the environment. Several works consider robustness against uncertainty using uncertainty-aware MPC [Lim+10; Gao+14; Lan+04; MNM22]. The robust MPC in [Bao+23] provides probabilistic safety guarantees. [ZZS21] proposes using CBF as constraints, which would make the MPC more potentially feasible and allow smooth obstacle avoidance. We extend the design in our proposed controller to include delay compensation and robustness to uncertainties. There are also some works combining machine learning techniques with safe control, such as iterative learning control [Li+22] and reinforcement learning [MNM22; HL23]. We propose a controller design which is robust

to both the system and environment uncertainties/noises as well as the system delays. A disturbance observer can be used to estimate the uncertainties for a robust MPC [Zho+23]. It would be interesting to extend our approach with this feature in the future, but in this chapter, we assume the disturbance has a known bound.

3.3 Methodology

Notation

A polytope is the convex hull of finite vertices in $\mathbb{R}^d$. The Minkowski sum of two polytopes, A and B , is another polytope in d -dimensional space defined as $A \oplus B := \{a + b | a \in A, b \in B\}$. The Pontryagin difference between two polytopes, A and B , is another polytope in d -dimensional space defined as $A \ominus B := \{x | x + b \in A, \forall b \in B\}$.

System dynamics

A dynamic bicycle model is used in frenet frame in this chapter which is different from what we described in Section 2.2

$$\begin{aligned}
 \begin{bmatrix} \dot{e}_1 \\ \ddot{e}_1 \\ \dot{e}_2 \\ \ddot{e}_2 \\ \dot{v}_x \\ \dot{s} \\ \dot{\delta}_a \end{bmatrix} &= \begin{bmatrix} \dot{e}_1 \\ -\frac{2C_f+2C_r}{mV_x}\dot{e}_1 + \frac{2C_f+2C_r}{m}e_2 - \frac{-2C_fl_f+2C_rl_r}{mV_x}\dot{e}_2 \\ \dot{e}_2 \\ -\frac{2C_fl_f-2C_rl_r}{I_zV_x}\dot{e}_1 + \frac{2C_fl_f-2C_rl_r}{I_z}e_2 - \frac{2C_fl_f^2+2C_rl_r^2}{I_zV_x}\dot{e}_2 \\ -K_f v_x \\ v_x - \dot{e}_1 e_2 \\ -K_\delta \delta_a \end{bmatrix} \\
 &+ \begin{bmatrix} 0 \\ -v_x(v_x + \frac{2C_fl_f-2C_rl_r}{mv_x}) \\ 0 \\ -\frac{2C_fl_f^2+2C_rl_r^2}{I_z} \\ 0 \\ 0 \\ 0 \end{bmatrix} c + \begin{bmatrix} 0 \\ \frac{4C_f}{m}\delta_a \\ 0 \\ 0 \\ K_p p \\ 0 \\ K_\delta \delta \end{bmatrix}
 \end{aligned} \tag{3.1}$$

where $\mathbf{u} = [\delta \ p]^T$, $\mathbf{x} = [e_1 \ \dot{e}_1 \ e_2 \ \dot{e}_2 \ v_x \ s \ \delta_a]^T$. C_f , C_r are the stiffness coefficients of the front and the rear tires, l_f and l_r are the front and the rear tire distances from the centre of mass (COM), I_z , m are the moment of inertia and mass of the vehicles, and c is

the inverse of the radius of curvature of the reference line at the perpendicular point from the vehicle's position. The control variables are steering angle (δ) and pedal (p). The state variables taken include the lateral position error (e_1), heading angle error (e_2) with respect to the optimal racing line, their first-order derivatives $\dot{e}_1$ and $\dot{e}_2$, longitudinal velocity (v_x), and longitudinal displacement (s). In many existing motion planning and control works, control actions are assumed to be applied instantaneously in an ideal case. Due to this, a delay is caused due to the mismatch between the calculated and the actual steering angle state. To solve this problem, we include the steering angle as a separate state, and model the steering actuator dynamics as follows by a first order ODE (see [Nah+19]): $\dot{\delta}_a = K_\delta(\delta - \delta_a)$, where δ is the desired steering angle, δ_a is the actual steering angle, and K_δ is calculated as the inverse of the time constant. Thus, we control only δ to tell the actuator which state to achieve and use δ_a instead of δ for the dynamics calculation. For acceleration, the delays in pedal dynamics are neglected for simplicity. K_p is used for the mapping between pedal and acceleration. However, its delay can also be modeled in the same way as the steering.

Robust Tube MPC

The dynamic model is described in a discrete-time manner as $\mathbf{x}_{n+1} = f(\mathbf{x}_n, \mathbf{u}_n, \mathbf{w}_n)$. The original non-linear model is linearized around the current state $\mathbf{x}_n$ at every time step into a linear system. However, in the MPC's prediction horizon of n , we assume time-invariant dynamics with Jacobian matrices A_n and B_n , i.e., inside the prediction horizon, the model is $\mathbf{x}_{k+1,n} = A_n \mathbf{x}_{k,n} + B_n \mathbf{u}_{k,n} + \mathbf{w}_{k,n}$. To simplify the notation, henceforth we omit n unless it is necessary to be mentioned explicitly. We assume the matrix pair (A, B) to be controllable, therefore, there exists a stabilizing linear feedback gain K with appropriate dimension such that $A_K = A + BK$ is Hurwitz. This assumption holds empirically true for our experiments and is also common in literature [Gao+14]. The disturbance $\mathbf{w}_k$ is assumed to be bounded as $\mathbf{w}_k \in \mathcal{W}$. $\mathcal{W}$ is assumed to be a convex polyhedron with the origin as an interior point. Also, it must be noted that we can include the linearization error by adding an extra disturbance [Gao+14]. In our work, we assume that the extra disturbance has been enclosed by a sufficiently large disturbance set. The objective is to stabilize this uncertain system while satisfying the state and control constraints as $\mathbf{x}_k \in \mathcal{X}$ and $\mathbf{u}_k \in \mathcal{U}$. For this system, we can define a disturbance invariant set, $\mathcal{Z}$, such that $A_K \mathcal{Z} \oplus \mathcal{W} \subseteq \mathcal{Z}$. A typical MPC controller is defined as a finite horizon optimization problem, P over the future input control sequence and states. We define P on an equivalent nominal system assuming no disturbance to optimize nominal state sequence $\bar{X} = \{\bar{\mathbf{x}}_0, \bar{\mathbf{x}}_1, \dots, \bar{\mathbf{x}}_N\}$ and nominal control sequence $\bar{U} = \{\bar{\mathbf{u}}_0, \bar{\mathbf{u}}_1, \dots, \bar{\mathbf{u}}_{N-1}\}$ as follows (note again the optimization is for the current step n , which is omitted for notation simplicity),

$$\begin{aligned}
\min_U \quad & \sum_{k=0}^{N-1} (\bar{\mathbf{x}}_k - \mathbf{x}_{ref,k})^T Q (\bar{\mathbf{x}}_k - \mathbf{x}_{ref,k}) + \bar{\mathbf{u}}_k^T R \bar{\mathbf{u}}_k \\
& + (\bar{\mathbf{x}}_N - \mathbf{x}_{ref,N})^T Q_N (\bar{\mathbf{x}}_N - \mathbf{x}_{ref,N}) \\
s.t. \quad & \bar{\mathbf{x}}_{k+1} = A\bar{\mathbf{x}}_k + B\bar{\mathbf{u}}_k \\
& \mathbf{x}_0 \in \bar{\mathbf{x}}_0 \oplus \mathcal{Z} \\
& \bar{\mathbf{u}} \in \mathcal{U} \ominus K\mathcal{Z} \\
& \bar{\mathbf{x}} \in \mathcal{X} \ominus \mathcal{Z}
\end{aligned} \tag{3.2}$$

where $X_{ref} = \{\mathbf{x}_{ref,0}, \mathbf{x}_{ref,1}, \dots, \mathbf{x}_{ref,N}\}$ is the reference state sequence, R , Q and Q_N are the control, state and terminal state cost matrices, respectively and all of them are positive semi-definite. We solve the following problem using CasADi's QP solver [And+18]. Now, for this uncertain system, we design a controller $\mathbf{u} = \bar{\mathbf{u}} + K(\mathbf{x} - \bar{\mathbf{x}})$. For such a controller, we can guarantee that $\mathbf{x}^+ \in \bar{\mathbf{x}}^+ \oplus \mathcal{Z}$

for any bounded $\mathbf{w} \in \mathcal{W}$, which implies that all states $\mathbf{x}_k$ will be robustly contained inside $\mathcal{X}$.

Delay-aware robust tube MPC

So far, we assume that the optimization of the command as well as the execution of the optimized command to the vehicle is instantaneous. However, this is not the case in a practical vehicle system, where we have computation delay t_c and control action processing delay t_a . Therefore, the control influences the car at $t_d = t_c + t_a$ after the time when the observed state is used for optimization. This could lead to instability, as the robust tube assumptions no longer hold true. This is especially dangerous when t_d is large. To deal with this time delay in the system, [STL13] proposes a bi-level control scheme where the high-level controller plans robust commands using tube MPC and the low-level unit runs a feedback controller on these reference commands. A buffer is used for communication between the two units. The brief idea is that assuming the input sequence for time frame t to $t + t_h$ is known at time t where t_h is the horizon length, we pass the predicted t_h time-ahead state obtained via simulation to the optimization problem (3.2) starting at time t so as to complete before $t + t_h$ time and apply the optimal input sequence to the system with feedback between times $t + t_h$ to $t + 2t_h$. The above procedure repeats in every cycle with the latest state prediction. In this strategy we need to wait for $t_h - t_c$ time for each cycle where t_c is the computation time taken for optimization.

However, such a common approach in the literature is inappropriate in a fast-changing environment such as driving at high speeds on highways, where immediate action might have to be taken in an emergency case. Fig. 3.3 illustrates our solution: we estimate an upper bound on the computation time, $\hat{t}_c$, calculated locally after each cycle with a high probabilistic guarantee. Instead of updating the buffer from $t + t_h$ to $t + 2t_h$, we update if from $t + \hat{t}_c$ to $t + \hat{t}_c + t_h$. This significantly improves the high-level MPC controller frequency, which would be required for fast-changing scenarios where the path to be followed continually updates. We use an adaptive Kalman filter to calculate $\hat{t}_c$. Section 3.3 describes this in detail. Also, we have an additional constant time delay t_a due to control action processing, hence the total upper bound on delay time is $\hat{t}_d = \hat{t}_c + t_a$. Based on this, we shift the initial state and use the shifted estimated state $\mathbf{x}_{\hat{t}_d|t}$ at time $\hat{t}_d$. We estimate this state at time $\hat{t}_d$ using the nominal commands from the buffer assuming no extra disturbance from the environment. We fill the buffer B from time $t + \hat{t}_c$ to $t + \hat{t}_c + t_h$ with the calculated controls ($\bar{U}$) and nominal states ($\bar{X}$). As we calculate the commands in discrete steps of Δt , the buffer is updated in the following fashion : $\bar{\mathbf{u}}_{[t+\hat{t}_c+k\Delta t, t+\hat{t}_c+(k+1)\Delta t]} = \bar{\mathbf{u}}_k$ for $k \in \{0, 1, 2, \dots, N-1\}$, as demonstrated in Fig. 3.3. The commands are calculated in the following manner as described in Equation 3.3. The pre-compensator acts as a low-level unit that simply executes commands in the buffer at the required times and runs in parallel. It has a higher frequency than the high-level MPC.

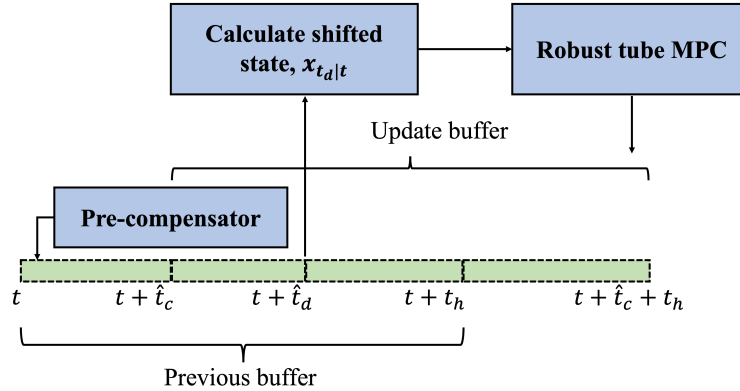


Figure 3.3: Our delay-aware robust MPC with dual cycles

$$\begin{aligned}
\min_U \quad & \sum_{k=0}^{N-1} (\bar{\mathbf{x}}_k - \mathbf{x}_{ref,k})^T Q (\bar{\mathbf{x}}_k - \mathbf{x}_{ref,k}) + \bar{\mathbf{u}}_k^T R \bar{\mathbf{u}}_k \\
& + (\bar{\mathbf{x}}_N - \mathbf{x}_{ref,N})^T Q_N (\bar{\mathbf{x}}_N - \mathbf{x}_{ref,N}) \\
s.t. \quad & \bar{\mathbf{x}}_{k+1} = A\bar{\mathbf{x}}_k + B\bar{\mathbf{u}}_k \\
& \mathbf{x}_{t_d+t} \in \bar{\mathbf{x}}_0 \oplus \mathcal{Z} \implies \mathbf{x}_{t_d|t} \in \bar{\mathbf{x}}_0 \oplus \mathcal{Z} \ominus (\oplus_{j=0}^{s-1} A_K^j \mathcal{W}) \\
& s = \left\lceil \frac{t_d}{\Delta t} \right\rceil \\
& \bar{\mathbf{u}} \in \mathcal{U} \ominus K\mathcal{Z} \\
& \bar{\mathbf{x}} \in \mathcal{X} \ominus \mathcal{Z}
\end{aligned} \tag{3.3}$$

Control limits ($\mathcal{U}$)

For control constraints, we limit throttle amount and steering with their actuation limits.

State CBF Constraints ($\mathcal{X}$)

In (3.1), the longitudinal and lateral errors of the vehicle are s and e_1 . As the free space along the Frenet frame is non-convex in general, we cannot directly set the constraints. This is because it becomes computationally expensive for the optimization problem to set the constraints in the non-convex form. To solve this, we propose the use of the IRIS algorithm to derive a set of convex constraints for the optimization problem to make it efficiently solvable while also ensuring safety for the vehicle [KLD21]. IRIS works by finding a largest possible ellipsoid that fits the non-convex space, which is later converted to a set of convex constraints. The lane boundaries are also fed as constraints to the non-convex free space by sampling points around the vehicle's location. The convex space $\mathcal{X}$ can be expressed as a set of linear state constraints where each bounding line can be expressed by a_i , b_i , and c_i , see (3.4). Further, we formulate each of these constraints with a CBF as given in (3.4), where L is the number of sides of the bounding polygon for the convex space $\mathcal{X}$.

$$h_i(X_k) = a_i s_k + b_i e_{1,k} + c_i, \forall i = 0, 1 \dots L-1 \tag{3.4}$$

Disturbance-invariant set ($\mathcal{Z}$)

We over-approximate $\mathcal{Z}$ at the n -th time step as: $\mathcal{Z}_n = \Sigma_{i=0}^N A_{K,n}^i \mathcal{W}$ [Rak+04], where $A_{K,n} = A_n + B_n K_n$, K_n is the control gain. However, $A_{K,n}$ is state-dependent and time-variant. To ensure robustness, $\mathcal{Z}_n$ must be covered by $\mathcal{Z}_{n+1}$. For detailed calculation and proof, readers

are referred to [KLD22]. The basic idea is to have a union operation of sets given the range of states.

Stability

Using tube-MPC ensures the system stays within a certain tube for bounded disturbance as discussed in Section 3.3. The tube is obtained based on the calculation of the disturbance invariant set. The calculation and the stability proof can be found in [KLD22] and [Lim+10].

Feasibility

A reachable set $\mathcal{R}_k$ is defined as the set of nominal states at the end of k time steps, i.e., at time $t + \hat{t}_c + k\Delta t$ as follows:

$$\begin{aligned} \mathcal{R}_k = \{ & \bar{\mathbf{x}}_k : \forall i = 0, 1..k-1, \bar{\mathbf{x}}_{i+1} = A\bar{\mathbf{x}}_i + B\bar{\mathbf{u}}_i \\ & \bar{\mathbf{u}}_i \in \mathcal{U} \ominus K\mathcal{Z} \} \end{aligned} \quad (3.5)$$

where $\mathcal{R}_0$ is defined according to the constraint of $\bar{\mathbf{x}}_0$ in (3.3). Let us define the set of all states satisfying the CBF constraints as follows:

$$\mathcal{X}_{\text{cbf}} = \{ \mathbf{x} \in \mathcal{X} : \dot{h}(\mathbf{x}) + \gamma(h(\mathbf{x})) \geq 0 \} \quad (3.6)$$

Since the robust tube MPC guarantees that the actual system state, $\mathbf{x} \in \bar{\mathbf{x}} + \mathcal{Z}$ where $\bar{\mathbf{x}}$ is the nominal state obtained from optimization and $\mathcal{Z}$ is the disturbance invariant set defined in Section 3.3, we can define set $\mathcal{X}_{\text{cbf,robust}} = \mathcal{X}_{\text{cbf}} \ominus \mathcal{Z}$. If the nominal state obtained by optimization is present in $\mathcal{X}_{\text{cbf,robust}}$, we can guarantee that the system state will lie within $\mathcal{X}_{\text{cbf}}$, i.e., it will satisfy the CBF constraint.

The feasibility of the system depends on whether the intersection between $\mathcal{R}_k$ and the super-level set $\mathcal{X}_{\text{cbf,robust}}$ is non-empty. If h is a valid CBF, then $\mathcal{X}_{\text{cbf}}$ is non-empty. When we reduce the decay rate γ of the CBF, the set $\mathcal{X}_{\text{cbf}}$ becomes smaller, resulting in a more conservative or safer system. However, there is a tradeoff between feasibility and safety, as described in [ZZS21]. The intersection between $\mathcal{R}_k$ and $\mathcal{S}_{\text{cbf}}$ might become infeasible. Choosing an appropriate γ that ensures a non-empty intersection between the two sets remains an open challenge, and automatically determining the optimal γ for a given scenario is still an open problem. In [LLD21], they addressed this issue by introducing an additional optimization over γ to ensure pointwise feasibility at each step. However, formulating a persistent feasible

solution remains an open problem, which we acknowledge as a limitation and a potential area for future research. In this study, we assume that γ is well-tuned without introducing infeasibility problems. Furthermore, during our experiments, we carefully tuned γ and did not encounter any infeasible corner cases.

INFLUENCE for computation time delay

INFLUENCE is proposed as a variant of the adaptive Kalman filter for the estimation of an upper bound on computation time. The conventional Kalman filter is unsuitable for cases in which the process model and the noise covariance are unknown. These are compensated by using adaptive filters. However, state-of-the-art adaptive filters rely either on known dynamics while assuming unknown noise covariances [HE13; MT76] or on known covariances while assuming unknown model parameters [LT15]. To compensate for this deficiency, INFLUENCE assumes that measurement noise variance r , process noise variance q , and η in the process model are all unknown. We assume a linear unknown process model with parameters η . We assume both process and observation noise distributions to be mutually uncorrelated, independent and Gaussian. INFLUENCE works as described in Algorithm 1, where $t_{c,n}$ is the observed computation time, and $x_{n|n-1}$, $p_{n|n-1}$ are the predicted computation time mean and variance estimates for the n -th time step. The computation complexity taken by each cycle is $O(1)$, i.e., independent of N_r and N_q and thus very low. As observed from Fig. 3.6, the average computation time taken in each cycle is less than 5 ms, so the algorithm is capable of real-time deployment.

The algorithm starts with initialization of all variables. The key idea is that we use the model information η in the current step to estimate the noise information q and r , followed by the update of η itself [HE13; MT76]. In the third box, w and e are measurement and prediction errors. They are updated in an incremental manner. N_q and N_r are tunable parameters for update rates. In the last box, λ and F can be considered as a forgetting factor and a learning rate, respectively.

It is important to note here that for the sake of simplicity, the notation x is used for the computation time to be estimated and is not to be confused with the state notation we used earlier. We use the predicted mean $x_{n|n-1}$ and variance $p_{n|n-1}$ estimates for the n -th time step to get an upper-bound estimate on computation time: $\hat{t}_{c,n} = x_{n|n-1} + \beta p_{n|n-1}$. Assuming a Gaussian distribution, the parameter β dictates the level of confidence in estimating $\hat{t}_{c,n}$ as an upper bound (higher value of β implies higher confidence). For the experiments in this chapter, we use $\beta = 2$.

Algorithm 1 INFLUENCE algorithm

Initialize:

$$x_{0|0} = t_{c,0}, p_{0|0} = 0, q_0 = \epsilon, r_0 = \epsilon, e_0 = 0, w_0 = 0, F_0 = I_2, \eta = [1 \ 0]^T$$

for $n = 1$ **to** T **do**

$$x_{n|n-1} = \eta_{n-1}^T [x_{n-1|n-1}] \quad // \text{ Estimate new state using the current estimated value of } \eta$$

$$p_{n|n-1} = \eta_{n-1,0}^2 p_{n-1|n-1} + q_{n-1}$$

$$e_n = \frac{N_r-1}{N_r} e_{n-1} + \frac{1}{N_r} (t_{c,n} - x_{n|n-1})$$

$$\Delta r_n = \frac{((t_{c,n} - x_{n|n-1}) - e_n)^2}{N_r - 1} - \frac{p_{n|n-1}}{N_r}$$

$$r_n = \left\lfloor \frac{N_r-1}{N_r} r_{n-1} + \Delta r_n \right\rfloor \quad // \text{ Update } r \text{ estimate}$$

$$K_n = \frac{p_{n|n-1}}{p_{n|n-1} + r_n}$$

$$x_{n|n} = x_{n|n-1} + K_n (t_{c,n} - x_{n|n-1})$$

$$p_{n|n} = (1 - K_n) p_{n|n-1}$$

$$w_n = \frac{N_q-1}{N_q} w_{n-1} + \frac{1}{N_q} (x_{n|n} - x_{n|n-1})$$

$$\Delta q_n = \frac{p_{n|n} - \eta_{n-1,0}^2 p_{n-1|n-1}}{N_q} + \frac{((x_{n|n} - x_{n|n-1}) - w_n)^2}{N_q - 1}$$

$$q_n = \left\lfloor \frac{N_q-1}{N_q} q_{n-1} + \Delta q_n \right\rfloor \quad // \text{ Update } q \text{ estimate}$$

$$\text{Let } \phi \text{ be } [x_{n-1|n-1} \ 1]^T$$

$$F_n = \frac{1}{\lambda} \left(F_{n-1} - \frac{F_{n-1} \phi \phi^T F_{n-1}}{\lambda + \phi^T F_{n-1} \phi} \right)$$

$$\eta_n = \eta_{n-1} + F_n \phi (x_{n|n} - x_{n|n-1}) \quad // \text{ Update } \eta \text{ estimate}$$

end for

Controller Plan A

Putting everything together, we propose the first controller design as illustrated in Fig. 3.4. The steering dynamic delay is compensated by adding the actual steering state as a new state and by using a first-order ODE to model steering dynamics (see section 3.3). The computation and actuator dynamic delays are handled by initial state shifting (see 3.3 and 3.3). After the optimization, the calculated commands from robust tube MPC are used to update the buffer from $t + \hat{t}_c$ to $t + \hat{t}_c + t_h$ filled with the nominal states and the commands. Observed computation time is used to update the new local upper bound for the next cycle by using our filter. The pre-compensator unit runs in parallel with high frequency as a feedback controller using the nominal states and commands from the buffer.

Controller Plan B

We also propose a new controller plan aimed towards safeguarding blackbox controllers whose internal mechanism is not known, like an LE controller which has been trained in a simplistic simulation environment where practical delays are not considered. We call the LE controller the nominal controller (see Fig. 3.5). To compensate for the computation time and control

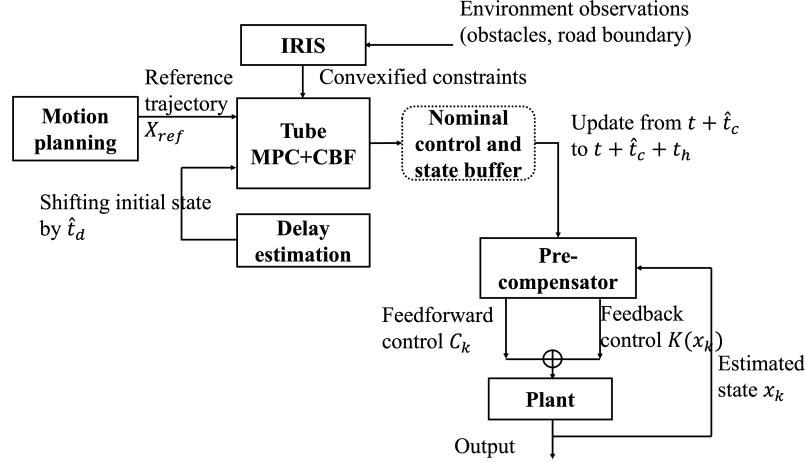


Figure 3.4: Controller Plan A for delay-aware robust tube MPC.

action processing delay, we use a similar design to that of plan A through initial state shift. However, for compensating actuator dynamic delay, we propose the use of a separate process that uses QP to output control commands that closely track the desired nominal commands inputted to it while also avoiding collision and respecting lane constraints.

First, we estimate the computation and control action processing delay and use it to shift the initial state, which is inputted to the LE controller. Rollouts are conducted to obtain sequential commands as output from the LE controller expressed as $\hat{U} = \{\hat{u}_1, \hat{u}_2, \dots, \hat{u}_N\}$. After solving the QP problem in (3.10), we get the refined commands as output expressed as $U = \{u_1, u_2, \dots, u_N\}$. Here, u_{start} is the observed steering angle at the start before optimization and r_k is obtained by taking output at time $k\Delta t$ in response to a unit step input to the steering actuator at time 0. Q_{ac} and R_{ac} are positive semidefinite weight matrices for tracking controller commands $\hat{U}$ and minimizing controller effort, respectively. Along with actuator dynamic delay compensation, we also add CBFs to offer safety constraints to the blackbox controller.

Lane constraints

We assume variable left and right lane width as a function of the longitudinal displacement s along the racing line. Let the approximate linear functions $L_l(s)$ and $L_r(s)$ govern the left and right lane width functions, respectively. We have

$$\begin{aligned}
h_{left,k}(\mathbf{x}) &= L_l(s_0) - e_{1,k} \\
h_{right,k}(\mathbf{x}) &= e_{1,k} + L_r(s_0) \\
- L_r(s_0) &\leq \frac{1}{\lambda^2}(\ddot{e}_{1,k} + \lambda \dot{e}_{1,k} + \lambda^2 e_{1,k}) \leq L_l(s_0)
\end{aligned} \tag{3.7}$$

Collision avoidance constraints

For collision avoidance with each vehicle, we set the following second-order CBF since the dynamic model has a relative degree of two. The detailed justification and expansion can be found in our code repository.

$$h_{veh,k}(\mathbf{x}, \mathbf{x}_{opp}) = \left(\frac{s - s_{opp}}{d_s} \right)^2 + \left(\frac{e_1 - e_{opp}}{d_e} \right)^2 - 1 \tag{3.8}$$

where s_{opp} and e_{opp} are the longitudinal and lateral error of the opponent vehicle, and d_s and d_e are the longitudinal and lateral safe distances from the target opponent vehicle. $\mathbf{x}_{opp}$ is the opponent vehicle's state. The safety constraint of this second-order CBF is:

$$h_{veh,k}(\mathbf{x}, \mathbf{x}_{opp}) + 2\lambda \dot{h}_{veh,k}(\mathbf{x}, \mathbf{x}_{opp}) + \lambda^2 \ddot{h}_{veh,k} \geq 0 \tag{3.9}$$

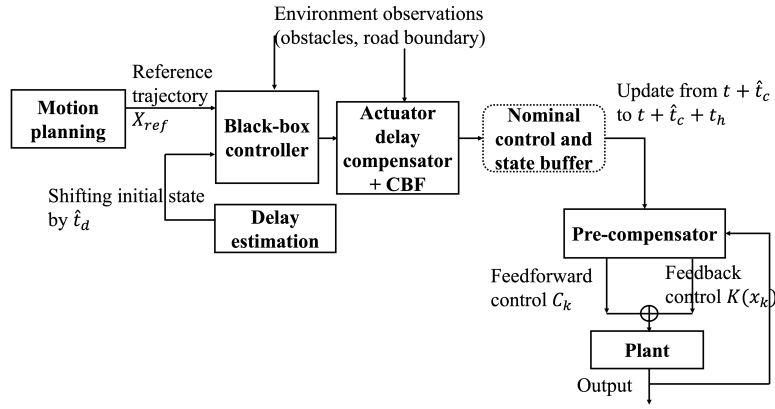


Figure 3.5: Controller Plan B, delay compensation for an LE controller.

$$\begin{aligned}
\min_U \quad & \sum_{k=1}^N \left\| \hat{\delta}_k - \left(\delta_0 + \sum_{i=1}^k (\delta_i - \delta_{i-1}) r_{k-i+1} \right) \right\|_Q + \|\delta_k\|_R \\
s.t. \quad & \mathbf{u}_0 = \mathbf{u}_{start} \\
& \mathbf{x}_{k+1} = A_k \mathbf{x}_k + B_k \mathbf{u}_k \\
& h(\bar{\mathbf{x}}_k, \mathbf{x}_{opp}) + \lambda \dot{h}(\bar{\mathbf{x}}_k, \mathbf{x}_{opp}) + \lambda^2 h(\bar{\mathbf{x}}_k, \mathbf{x}_{opp}) \geq 0
\end{aligned} \tag{3.10}$$

where $r_i = (1 - e^{-Ki\Delta t})$.

3.4 Experimental Results

The Gazebo simulator [KH04] is used to test normal autonomous driving scenarios (which we refer to as non-racing in the rest of the chapter). The racing scenarios are tested using the CARLA simulator [Dos+17b]. Validation of controller plan A and controller plan B is divided into two subsections. For all the simulation experiments, we have used $m = 1600$ kg, $C_f = 16000$ kg·m·s⁻²·rad⁻¹, $C_r = 16000$ kg·m·s⁻²·rad⁻¹, $I_z = 1800$ kg·m², $K_p = 5$ m·s⁻², $K_f = -0.134$ s⁻¹, $L = 3$ m, $w_{veh} = 1$ m, $Q = \text{diag}[1 \ 0 \ 10 \ 0 \ 1 \ 0 \ 0]$, $R = \text{diag}[10 \ 10]$, $W := w||w|_\infty < 0.1$, $\Delta t = 0.05s$, $N = 8$, $K_\delta = 11$ s⁻¹, $N_r = 30$, $N_q = 30$, $\beta = 2$. For all the experiments with dynamic obstacle vehicles, their future trajectories are considered by assuming constant velocities.

Controller plan A

Static obstacle avoidance (non-racing)

We conduct a static obstacle avoidance experiment to test controller plan A. The trajectories can be found in [KLD22]. As shown in Fig. 3.6, INFLUENCE can safely bound the computation delay in this experiment. Also, the overhead due to the addition of delay estimation using INFLUENCE is significantly smaller than the computation time taken by the optimization solver.

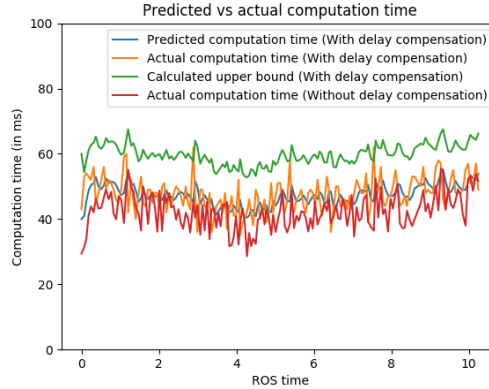


Figure 3.6: Experiment 3.4, blue: predicted computation time using INFLUENCE without variance; green: predicted computation time using INFLUENCE with variance; orange: actual computation time including INFLUENCE’s computation cost; red: actual computation time without INFLUENCE’s computation cost. ROS time at the x axis is the timestamp in the ROS operation system.

Racing scenario

We set up a racing scenario in CARLA to validate the controller plan A. A part of the track from the Town06 map is used for the experiments. It is a closed track suitable for setting up a racing scenario. The optimal racing line for the track is given in Fig. 3.7. In the case when no delay compensation is considered (see Fig. 3.7), the vehicle crashes once during the lap at a sharp turn. The reason for this is that as the vehicle operates at its friction limit at the turn, tracking error accumulates due to computation, and actuator delays lead the vehicle to hit the boundary (see the "without delay compensation" trajectory in the zoomed-in plot), thus leading to high lap time. On the other hand, with the delay compensation plan included (see Fig. 3.7), the vehicle safely and closely tracks the racing line, leading to faster lap time. We further compare race times and the number of crashes with the lane boundary (see Tab. 3.1). As can be observed, with delay compensation, the vehicle takes less time to complete the race, while without delay compensation due to crash, the vehicle takes more time to complete the race.

Also, to evaluate the robustness of our algorithm, we perform different runs by manually adding extra computation time. Fig. 3.8a and Fig. 3.8b show the trajectories followed with and without delay compensation. As shown in the figures, our delay compensation is able to handle high computation time up to 0.2 s; while without delay compensation, the vehicle loses control with only 0.06 s additional computation time. Also, the vehicle remains very stable with delay compensation, as shown in the lap time and steering variance in Fig. 3.8c and Fig. 3.8d), respectively.

Run	Lap time (in s)	No of lane boundary violations
Without delay com- pensation	35.46	1
With delay compensation	35.34	0

Table 3.1: Comparison in lap times with and without delay compensation included.

Sudden change scenario (racing)

We also test controller plan A against a sudden change racing scenario. For this scenario, an opponent vehicle ahead of the ego vehicle suddenly loses control, and thus applies sudden

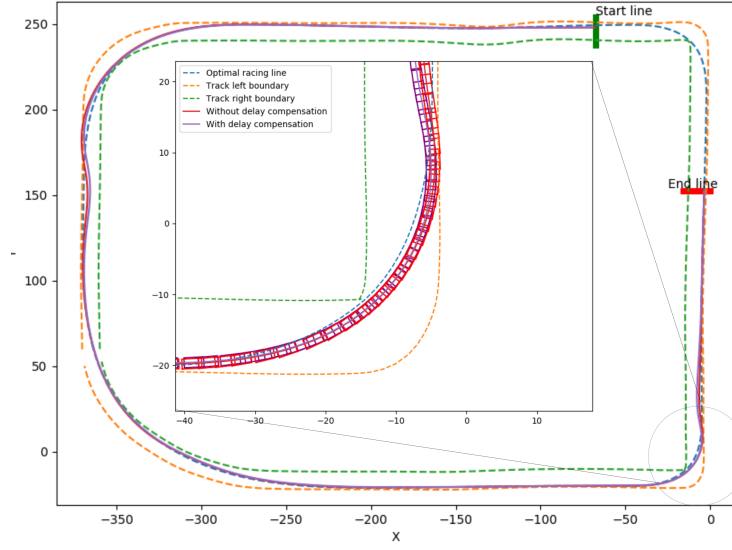
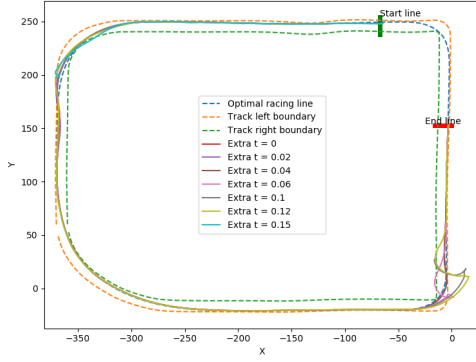
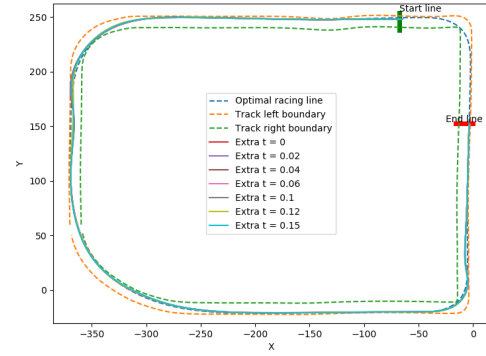


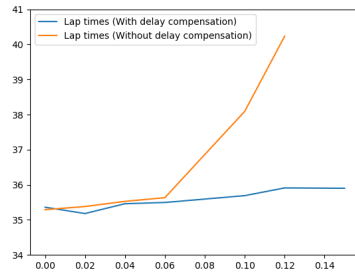
Figure 3.7: Performance comparison: red boxes (without delay compensation) show that the vehicle ran outside the road boundary while the vehicle controlled with delay compensation (blue boxes) always stays inside the boundary. The vehicle moves counterclockwise.



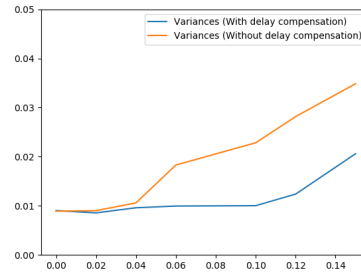
(a) Trajectories without delay computation



(b) Trajectories with delay computation



(c) Lap times



(d) Steering variances

Figure 3.8: Robustness test with additional computation time from 0.02 s to 0.15 s.

braking at $t = 2s$, forcing the ego vehicle to respond suddenly to avoid collision. If delay compensation is not considered for this scenario (see Fig. 3.9b), the vehicle collides with the opponent vehicle. On the other hand, with delay compensation considered, the vehicle makes a more informed decision to make a sudden turn with braking (see Fig. 3.9a) to avoid collision with the opponent vehicle.

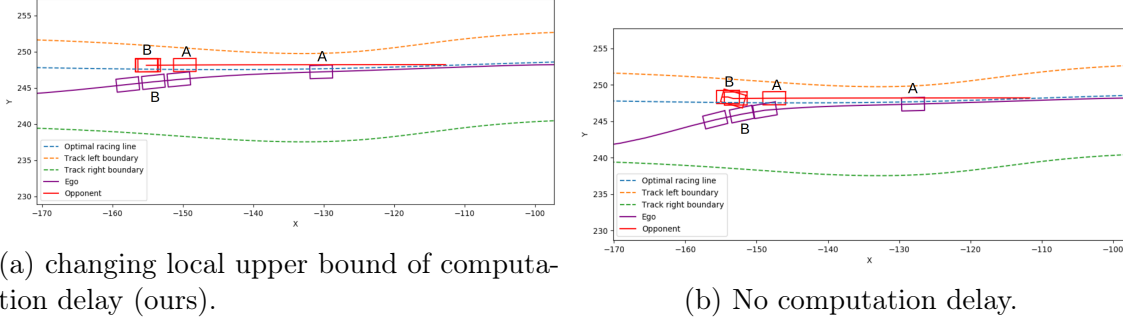


Figure 3.9: Comparison in Experiment 3.4. The opponent vehicle starts braking at point A. The ego vehicle crashes into the opponent vehicle at point B without delay compensation.

Overtaking in a turn scenario (racing)

Finally, we test the controller plan A in a scenario where the vehicle has to overtake at a turn, forcing it to leave the optimal racing line at the turn. Such a scenario requires highly precise controls, as the vehicle operates at its friction limits. As the vehicle deviates from the optimal racing line to overtake and avoid collision, the vehicle has to apply braking to avoid moving outside the track. However, without the delay compensation (see Fig. 3.10b), the vehicle applies braking late, leading to the vehicle CBF constraint being unable to retain the vehicle within the lane boundaries. With delay compensation considered, the vehicle makes more effective control decisions taking the delay into account and thus is able to perform the overtaking maneuver safely at turns (see Fig. 3.10a).

Controller plan B

We test the controller plan B in two scenarios described below. Imitation learning is used in our work for training a blackbox LE controller, which has a demonstrator (a model predictive controller) to obtain state-to-control mapping data [CDS19]. We use a fully connected feed-forward neural network (FNN) as an LE controller.

Path following in racing scenario

We test controller plan B in the path following racing scenario without obstacles. As discussed, as the track requires highly precise controls at turns to stay within the track, the

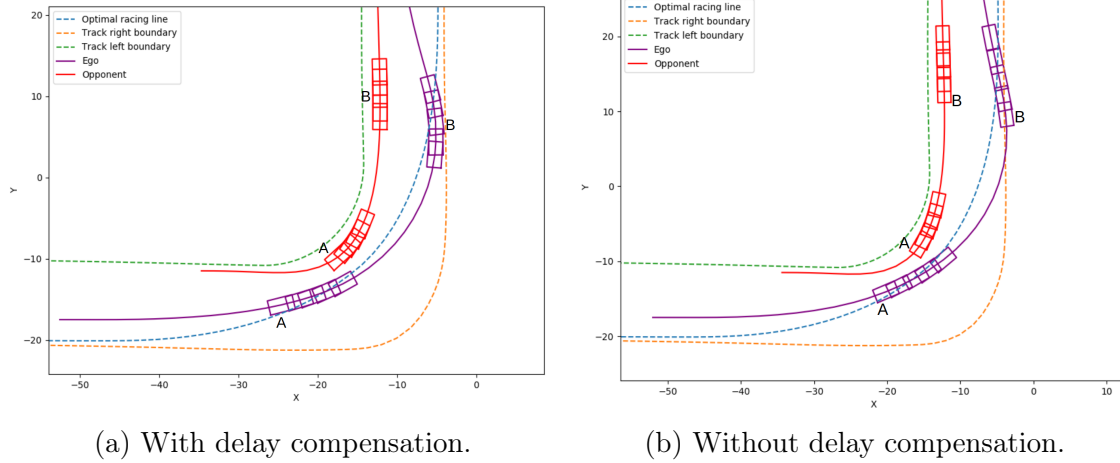


Figure 3.10: Comparison in Experiment 3.4. The ego vehicle crashes at point A in the case in which delay compensation is not considered.

safety controller in the form of lane constraints has to come into play to compensate for errors in current state measurements and delays. As shown in Fig. 3.11, when delay compensation is not considered the vehicle moves out of the track boundary as the safety constraints are not able to safeguard the vehicle due to inconsistency in vehicle state caused by delay errors. However, with delay compensation (see Fig. 3.11 and Fig. 3.13), the vehicle is able to safely stay within the lane constraints.

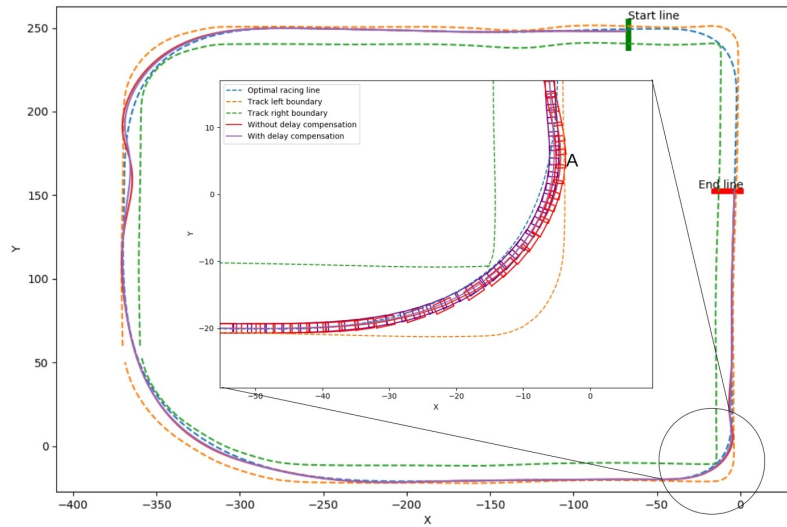


Figure 3.11: Performance comparison: red boxes (w/o delay compensation) show that the vehicle ran outside the road while the vehicle controlled with delay compensation (blue) always stays inside the boundary. The vehicle moves clockwise.

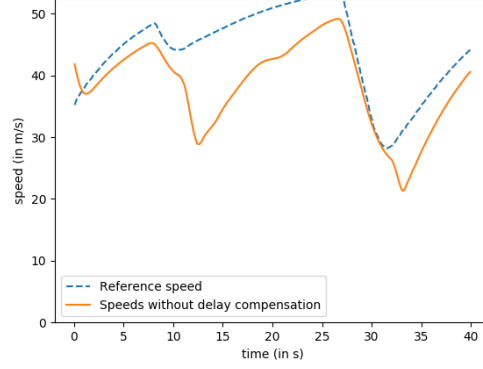


Figure 3.12: Ego vehicle speed for the case when delay compensation is not considered.

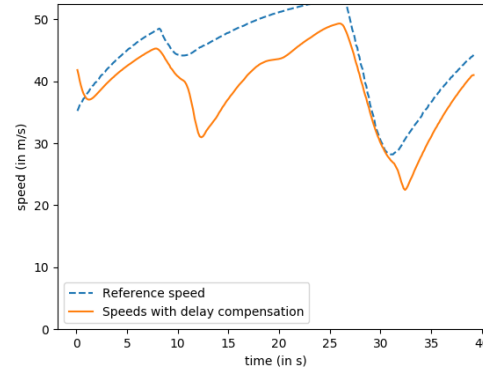


Figure 3.13: Ego vehicle speed for the case when delay compensation is considered.

Run	Lap time (in s)	No of lane boundary violations
Without delay compensation	38.04	2
With delay compensation	36.51	0

Table 3.2: Comparison in lap times and safety: w/ and w/o delay compensation

Overtaking at turn in racing scenario

Finally, to validate the controller plan A, we design a scenario where the vehicle has to overtake at a turn, forcing it to leave the optimal racing line at the turn. As the vehicle deviates from the optimal racing line to overtake and avoid collision, the vehicle has to apply braking to avoid moving outside the track. However, without the delay compensation (see Fig. 3.14b), the vehicle applies braking late, leading to the vehicle CBF constraint being unable to retain the vehicle within the lane boundaries. With delay compensation considered, the vehicle makes more effective control decisions taking the delay into account and thus is able to perform the overtaking maneuver safely at the turn (see Fig. 3.14a).

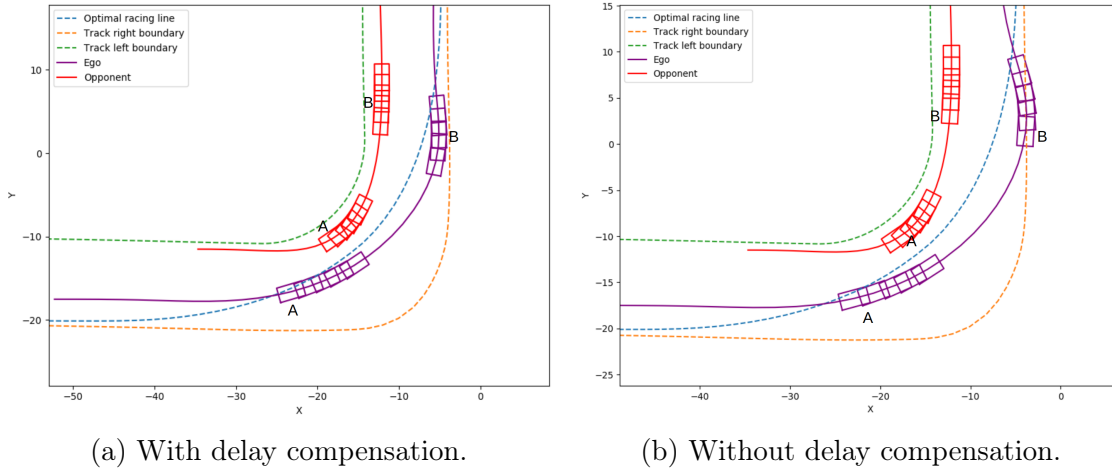


Figure 3.14: Comparison in Experiment 3.4. The ego vehicle crashes at point A without any delay compensation.

Hardware experiments in an RC car

We have tested the control plan A on a real 1/10-scale RC car (see Fig. 3.15a) that is equipped with a Lidar, a depth dual camera, an IMU and wheel encoders. The onboard computation platform is an NVIDIA Jetson TX2 with 8GB of RAM and 256 core NVIDIA Pascal GPU.

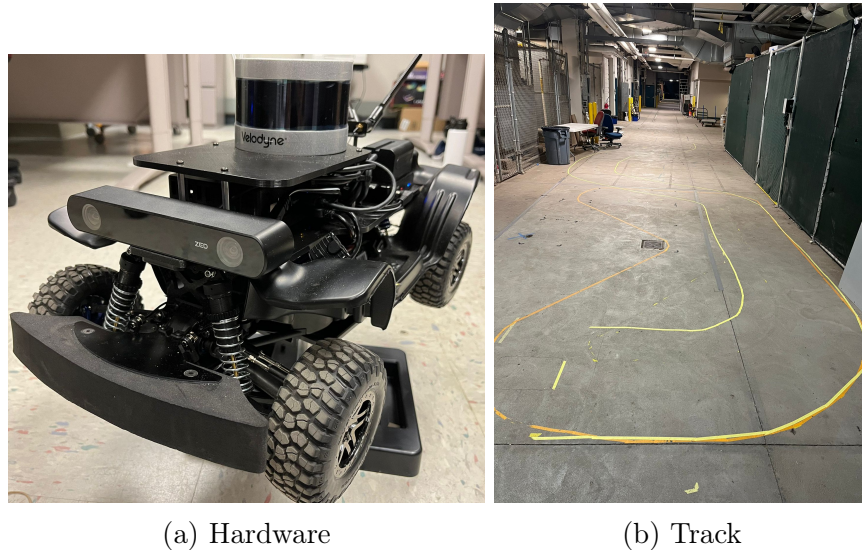


Figure 3.15: RC car setup

Experiment setup

We perform the experiment on a track as shown in Fig. 3.15b. The lane width is about 0.8 m. We use hector SLAM [Saa+19] for pre-mapping and Monte Carlo localization (MCL) for

localization [Del+99]. The uncertainty estimate obtained from MCL is used in our framework assuming Gaussian distribution of the state estimate from MCL.

Results

Due to the inherent non-determinism caused by various factors such as slight unavoidable changes in starting position, the uncertainty of Lidar sensor observations, etc., we conduct 5 runs with and without using our dynamic delay compensation. The trajectory results are shown in Fig. 3.16. As can be observed, without considering delay compensation, the vehicle moves out of the lane at point A most times while with the compensation included, the vehicle is able to stay safely within the lane limits. The delay-estimates for one of the runs are plotted in Fig 3.17. The video recording of the runs made with the RC car is also included in the video demonstration.

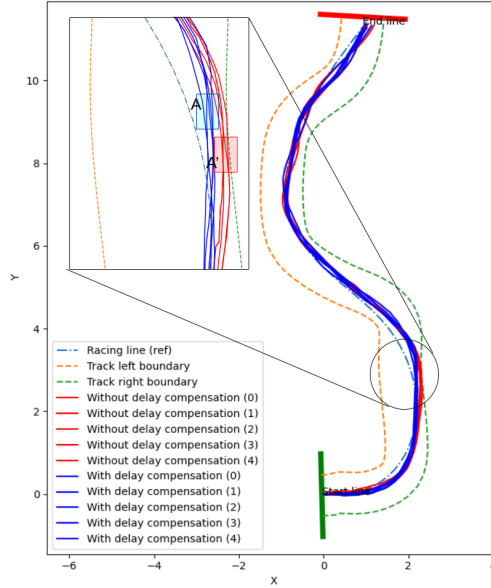


Figure 3.16: Performance comparison for 5 runs each: red lines (without delay compensation) show that the vehicle ran outside the road boundary while the vehicle controlled with delay compensation (blue lines) always stays inside the boundary. The vehicle moves from bottom to top.

3.5 Conclusion

We propose a delay-aware robust controller design for safe autonomous driving and racing. We also compensate for delays of an LE controller and employ it in the closed-loop system as a primary controller. The simulation results in high-fidelity simulators and on a real racing car show the efficacy of our designs. However, one limitation of this work is the need to know

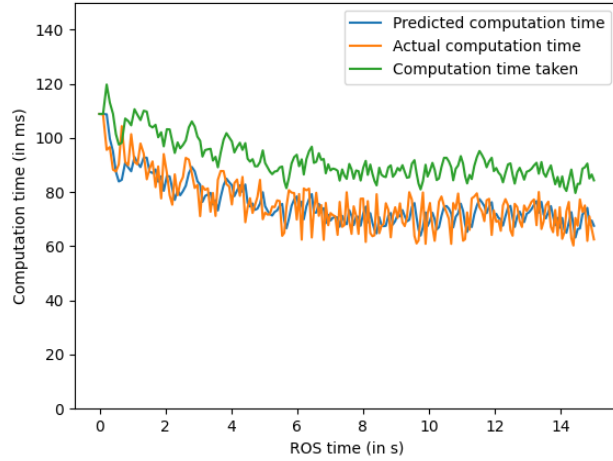


Figure 3.17: Experiment 3.4, predicted computation time (blue): predicted value using INFLUENCE without adding variance; computation time taken (green): predicted value using INFLUENCE with variance; actual computation time (orange): ground truth delay.

vehicle parameters. It may not be feasible to always be able to calculate these parameters accurately plus in case if they change during the run in real-time, it may not be possible to handle large model discrepancy from the actual model. Hence, we need a way to also adapt the parameters based on the feedback during the run. In the next chapter, we will extend this work to also include model adaptation in real-time based on online collected data.

Chapter 4

ONLINE MODEL-BASED ADAPTIVE LEARNING CONTROLLER FOR AUTONOMOUS RACING

We saw in the previous chapter how we can formally guarantee a bound on vehicle’s safety if we have a nearly correct vehicle model with bounded error wrt the nominal model. But if the nominal model is very inaccurate, the provable bound becomes infeasible. Autonomous racing is a challenging problem, as the vehicle needs to operate at the friction or handling limits in order to achieve minimum lap times. Autonomous race cars require highly accurate perception, state estimation, planning, and control. Adding to this complexity is the need to accurately identify vehicle model parameters governing lateral tire slip effects, which can evolve over time due to factors such as tire wear and tear. Current approaches to this problem typically either propose offline model identification methods or rely on initial parameters within a narrow range (typically within 15-20% of the actual values). However, these approaches fall short in accounting for significant changes in tire models that can occur during actual races, particularly when pushing the vehicle to its handling limits. We present a unified framework in this chapter that not only learns the tire model in real time from collected data but also adapts the model to environmental changes, even when the model parameters exhibit substantial deviations. The friction estimation, obtained as a byproduct from the learning results, facilitates the selection of the optimal racing line from a library for adaptive speed planning. We validate our approach through testing in simulators, encompassing a 1:43 scale race car and a full-size car, and also through experiments with a physical F1/10 autonomous race car.

4.1 Introduction

The estimation of tire conditions is critical for the minimal lap times and safety of car racing [WL20]. Race drivers adapt their driving strategies during each lap [BTI23], adjusting speeds at the turns based on their estimations of how well the tire traction will support them. Estimating the vehicle’s handling capacity is particularly challenging for autonomous racing, owing to the multitude of variables present in an actual race, including temperature, weather conditions, and more.

We provide a summary of our comparison with other approaches in Table 4.1. Traditional model-based planning and control frameworks for autonomous racing [LDM15; KIR+19],

e.g., model predictive control (MPC), heavily rely on accurate vehicle dynamic models. However, achieving precise dynamics estimation, particularly concerning lateral tire forces, necessitates intensive system identification and verification processes involving interaction with the road material.

Numerous studies have focused on estimating the tire lateral friction given the slip angles [Pac05; SG03; Sve07; GRG08; Xu+20; MPP08; Sri+20; Spi+19]. Some of these research efforts involve conducting system identification to determine parameters of the Pacejka tire model [Pac05] and the Fiala brush model [Fia54]. [Dje+23] characterizes tire models using Neural Ordinary Differential Equations (NODEs). Additionally, certain studies employ more sophisticated neural networks to fit the tire curve [AK18; GRG08; Xu+20; MPP08], as well as the entire vehicle dynamics [Sri+20; Spi+19]. However, it's essential to note that the computational time required for these approaches can become a limiting factor when attempting to apply them in real-time for high-speed racing control.

[Jai+20] proposes an iterative procedure that uses data from previous laps to correct the vehicle model. [HLZ17] proposes using Gaussian process (GP) and linear MPC to enable learning vehicle parameters offline. [RB19; RB16] proposes adjusting linear vehicle model online from collected data. But it assumes the availability of nearly-accurate tire parameters to start with and also all the methods assume the tire parameters to be constant throughout the run and not adapt to any changes. [Jai+20] proposes using an extended kinematic model and applies GP to learn the difference between the high-fidelity dynamic model and the extended kinematic model from collected data offline. However, using GP in optimization is computationally expensive (0.25s-0.3s according to [Jai+20]), which makes it practically infeasible to update online for an actual car. It is noteworthy that all of these works do not test on an online adaptive setting where the tire-surface interaction can change over time.

Methods	Tolerant to model errors	Adaptive planning to varied friction	Learning	Ref.
Traditional	✗	✗	-	[LDM15] [KIR+19]
Learning based	✓	✗	Offline	[Jai+20] [RB16]
			Online	[HLZ17] [RB19]
Ours	✓	✓	Online	-

Table 4.1: Our motivation and comparison with others

An overview of our framework is illustrated in Fig. 4.1. The process begins with an offline optimization step to generate a library including minimum-curvature optimal racing lines for

different surface frictions (see the blue block). Within our learning component (highlighted in red), we adopt a relatively simple kinematic *nominal model* described in Section 2.2. The discrepancy between the nominal model and the more complex dynamic model, which takes into account tire dynamics, aerodynamic forces, and longitudinal friction, is captured by an extreme learning machine (ELM). The significance of our learning is two-fold: 1) It facilitates model error compensation for the model-based controller (shown in the green block) utilizing the nominal model; 2) Our framework allows the retrieval of friction estimates from the learning results, thereby enabling the selection of the corresponding optimal racing line from the library for adaptive speed planning. This stands in contrast to existing methods that assume time-invariant friction.

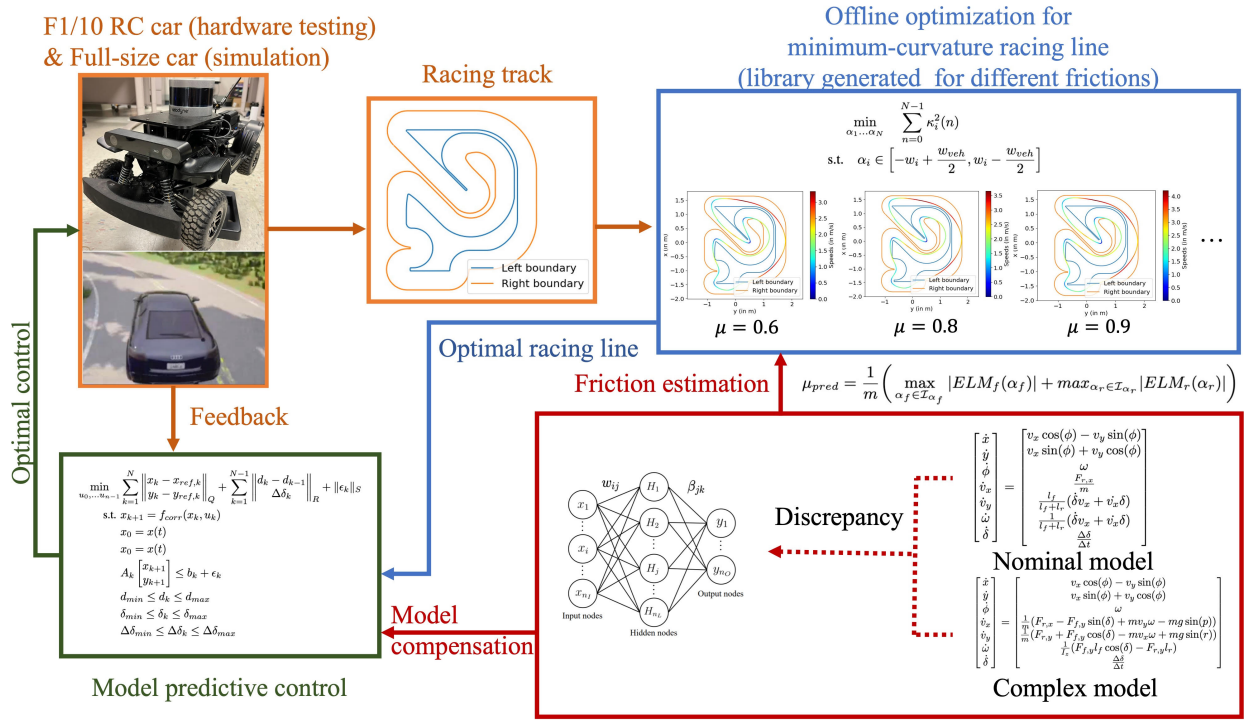


Figure 4.1: Overview of our approach. Our approach consists of an online learning component marked in red with fits the residual between actual model and the nominal e-kinematic model and estimates the friction coefficient. The estimated friction coefficient is used to select the raceline that is used by the controller with the updated vehicle model to track.

We first demonstrate our results by using a 1:43-scale open-source numerical simulator, allowing for a comparison with a state-of-the-art approach [LDM15]. We then demonstrate in the high-fidelity Carla simulator, incorporating measurement noise and a sloped racetrack. Finally, we test our approach on a real F1/10 RC car.

The contributions of our work are summarized as follows:

- We propose a novel unified optimization framework for online model error compensation and online adaptation for autonomous racing. Our approach outperforms the state-of-the-art GP controller [Jai+20] in terms of lap time, safety, and computation efficiency.
- Unlike existing approaches that presume time-invariant friction, our ELM model estimates time-varying friction online to empower adaptive speed planning along the race line for safe racing.

The rest of the chapter is organized as follows: Section 4.2 briefly goes through some preliminaries. Section 4.3 elaborates on our proposed method. Section 4.4 reports the experimental results. In Section 4.5, we make concluding remarks and discuss future directions.

4.2 Preliminaries

Extreme Learning Machine

For modeling tire friction curves, we use extreme learning machines (ELMs). An architecture of a single-layer ELM is shown in Fig. 4.2. Let the number of input nodes be n_I , the number of hidden nodes n_L , and the number of output nodes n_O . The mapping from input and output is:

$$\mathbf{y} = \mathbf{H}(\mathbf{x})\beta = g(\mathbf{w}^T \mathbf{x})\beta \quad (4.1)$$

where $\mathbf{H} = [H_1, H_2, \dots, H_j, \dots, H_{n_L}]$ is a nonlinear transformation, $\mathbf{w} = [w_1, \dots, w_{n_I}]$ are input weights, and $\beta = [\beta_1, \dots, \beta_{n_L}]$ are output weights. Furthermore, $g(\cdot)$ corresponds to a kernel function that fulfills the condition for universal approximation [HCS+06].

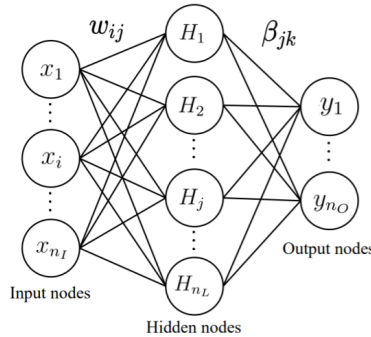


Figure 4.2: Architecture of a single layer ELM.

For training, let us consider the dataset to be of size N with each element of the form $(\mathbf{x}_n, \mathbf{t}_n)$, where $\mathbf{t}_i \in \mathbb{R}^{n_O}$ is the true label corresponding to the i -th datapoint. ELM has a fixed input weight denoted as $\hat{\mathbf{w}}$, unlike a traditional feedforward single-layer neural network. The *first*

reason for using ELM instead of a single-layer neural network is that the training of ELM is more efficient [HZS06]: if we initialize input weight $\mathbf{w}$ with random values drawn from a distribution and keep them fixed, it can theoretically get the same results even if it is not optimized. The objective of ELM is to obtain the optimal β^* which satisfies the following objective:

$$\beta^* = \arg \min_{\beta} \mathcal{L}(\mathbf{y}(\beta), \mathbf{t}) = \arg \min_{\beta} \|\mathbf{H}(\mathbf{x})\beta - \mathbf{t}\|_2 \quad (4.2)$$

In our case, we use separate ELMs for rear and front tire slip-curve estimation. Here the inputs are the slip angle $\alpha_{f|r}$ and unity 1 to allow adding bias on the first linear layer output, thus $n_I = 2$ and the output is the lateral force, $F_{[f|r]y}$, thus $n_o = 1$. n_h is a hyperparameter which we choose as 40 for the experiments. We will discuss further in Sec. 4.3 how we pass the gradient signal across both front and rear tire ELMs.

4.3 Methodology

We start our pipeline with an E-kinematic model as the nominal model $f_{\text{e-kin}}$. Later, through the collected data, we get the residual error f_{err} , i.e., $f_{\text{corr}} = f_{\text{e-kin}} + f_{\text{err}}$. The learnable parameters here are the ELM tire slip models, longitudinal friction C_{lf} , and aerodynamic force constant C_d .

Learning based control

Data collection

The collected data can be represented as a pair of states from consecutive time steps, i.e., as $\{\mathbf{x}_k, \mathbf{u}_k, \mathbf{x}_{k+1}\}$. Hence, the error between the actual state x_{k+1} and the one estimated from the discretized E-kinematic model (using 6th-order Runge-Kutta) can be represented as:

$$e(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{x}_{k+1} - f_{\text{e-kin}}(\mathbf{x}_k, \mathbf{u}_k) \quad (4.3)$$

We collect the data online, hence the dataset at the n -th time step is: $D_n = \{e(\mathbf{x}_k, \mathbf{u}_k) | k \in \{1, 2, \dots, n-1\}\}$.

Training and online adaptation

Our goal is to learn the model discrepancy, $e(\mathbf{x}_k, \mathbf{u}_k)$. The predicted value $e_{\text{pred}}(\mathbf{x}_k, \mathbf{u}_k)$ is obtained by computing the difference between (2.3) and (2.4):

$$\begin{aligned} \begin{bmatrix} e_{\dot{v}_x} \\ e_{\dot{v}_y} \\ e_{\dot{\omega}} \end{bmatrix}_{\text{pred}} &= \begin{bmatrix} \frac{1}{m}(-C_{lf} - C_d v_x^2 - F_{f,y} \sin(\delta) + m v_y \omega) \\ \frac{1}{m}(F_{r,y} + F_{f,y} \cos(\delta) - m v_x \omega) \\ \frac{1}{I_z}(F_{f,y} l_f \cos(\delta) - F_{r,y} l_r) \end{bmatrix} \\ &\quad - \begin{bmatrix} m \sin(p) \\ -m \sin(r) + \frac{l_f}{l_f + l_r}(\dot{\delta} v_x + \dot{v}_x \delta) \\ \frac{1}{l_f + l_r}(\frac{\Delta \delta}{\Delta t} v_x + \dot{v}_x \delta) \end{bmatrix} \end{aligned} \quad (4.4)$$

where $F_{f,y} = ELM_f(\alpha_f)$, $F_{r,y} = ELM_r(\alpha_r)$.

The loss to be minimized is defined as:

$$\begin{aligned} L = \sum_{k \in \{i, \dots, j\}} &\left((e_{\dot{v}_x}(\mathbf{x}_k, \mathbf{u}_k) - e_{\dot{v}_x, \text{pred}}(\mathbf{x}_k, \mathbf{u}_k))^2 \right. \\ &\quad + (e_{\dot{v}_y}(\mathbf{x}_k, \mathbf{u}_k) - e_{\dot{v}_y, \text{pred}}(\mathbf{x}_k, \mathbf{u}_k))^2 \\ &\quad \left. + (e_{\dot{\omega}}(\mathbf{x}_k, \mathbf{u}_k) - e_{\dot{\omega}, \text{pred}}(\mathbf{x}_k, \mathbf{u}_k))^2 \right) \end{aligned} \quad (4.5)$$

We minimize L iteratively using stochastic gradient descent with momentum. We use a mini-batch with size K_{batch} for a robust update. The training cycle can be briefly visualized as in Fig. 4.3. Letting the update rate be γ_{lr} and the momentum be γ_p , the model parameters are updated as:

$$\begin{aligned} p &= \begin{bmatrix} C_{lf} & C_d & \beta_{\text{elm},f}^T & \beta_{\text{elm},r}^T \end{bmatrix}^T \\ v(k+1) &= \gamma_p v(k) + (1 - \gamma_p) \frac{\partial L}{\partial p} \\ p(k+1) &= p(k) - \gamma_{lr} v(k) \end{aligned} \quad (4.6)$$

Adaptive speed planning

In addition to updating the vehicle model, it is crucial to adjust the reference speeds along the race line. For instance, if the friction coefficient decreases, the vehicle's ability to navigate turns successfully is compromised, necessitating a reduction in the reference speeds. To

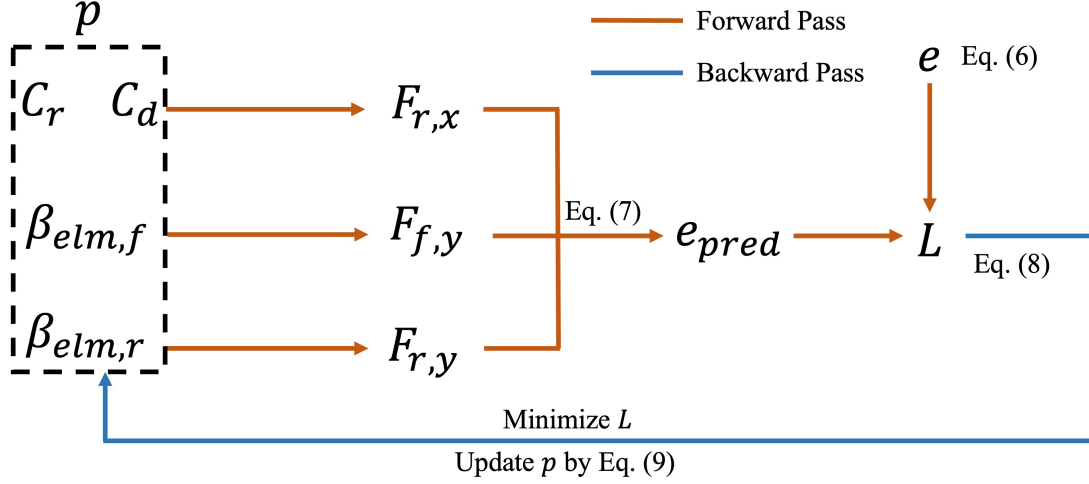


Figure 4.3: Training cycle. In the forward pass, given C_{lf} , C_d , and two ELMs, we calculate forces, then a residual error can be obtained to further update the model in the backward pass.

achieve this, we first create a speed profiles library for varied frictions, as elaborated in Section 2.1. We then employ the estimated friction for adaptive speed planning, dynamically assigning reference speeds in real time based on the varying friction conditions. We use the traction circle assumption as described in [SS21] for raceline calculation, where the friction coefficient dictates the maximum ratio of the force exerted to the normal force. We obtain the predicted friction coefficient as the ratio of the maximum force exerted to the available normal force $N = mg$. As we predict the tire slip vs. force curve, we can obtain maximum force as the maximum magnitude of predicted values from the ELM with slip angle inputs ranging from the observed slip angle values. This is inline with what we use for raceline calculation, as described in [SS21] (cf. CH. 3.1.3).

The predicted friction coefficient is calculated as:

$$\mu_{\text{pred}} = \frac{1}{mg} \left(\max_{\alpha_f \in \mathcal{I}_{\alpha_f}} |ELM_f(\alpha_f)| + \max_{\alpha_r \in \mathcal{I}_{\alpha_r}} |ELM_r(\alpha_r)| \right) \quad (4.7)$$

where $\mathcal{I}_{\alpha_f} = [\alpha_{f,min}, \alpha_{f,max}]$, $\mathcal{I}_{\alpha_r} = [\alpha_{r,min}, \alpha_{r,max}]$, $\alpha_{(f/r),min}$ and $\alpha_{(f/r),max}$ are the min and max values of the slip angles observed at any of the tires.

The optimization for control is formulated as follows:

$$\begin{aligned}
& \min_{u_0, \dots, u_{n-1}} \sum_{k=1}^N \left\| \begin{matrix} \mathbf{x}_{k,x} - x_{ref,k} \\ \mathbf{x}_{k,y} - y_{ref,k} \end{matrix} \right\|_Q + \sum_{k=1}^{N-1} \left\| \begin{matrix} d_k - d_{k-1} \\ \Delta \delta_k \end{matrix} \right\|_R + \|\epsilon_k\|_S \\
& \text{s.t. } \mathbf{x}_{k+1} = f_{\text{corr}}(\mathbf{x}_k, \mathbf{u}_k) \\
& \mathbf{x}_0 = \mathbf{x}(t) \\
& A_k \begin{bmatrix} \mathbf{x}_{k+1,x} \\ \mathbf{x}_{k+1,y} \end{bmatrix} \leq b_k + \epsilon_k \\
& d_{\min} \leq d_k \leq d_{\max} \\
& \delta_{\min} \leq \delta_k \leq \delta_{\max} \\
& \Delta \delta_{\min} \leq \Delta \delta_k \leq \Delta \delta_{\max}
\end{aligned} \tag{4.8}$$

where A_k and b_k are the tangent line coefficient matrices for the left and the right track boundaries at the nearest point from the vehicle's position. The reference trajectory is obtained by sampling points along the target racing line at distances according to reference velocities at those points, similar to [Jai+20]. The positive semi-definite matrices Q , R , and S characterize the tracking, actuation, and slack penalties, respectively. Slack variable ϵ_k is introduced to prevent infeasibility for the safety constraint, i.e., linearizing the track boundary at the nearest point. $f_{\text{corr}} = f_{\text{e-kin}} + e_{\text{pred}}$ is the corrected model, which outputs the next state. $f_{\text{corr}} = f_{\text{e-kin}}$ for $t \leq N$ when data are limited. Beyond N , we will adopt an online update for the model. $d_{\min}, d_{\max}, \delta_{\min}, \delta_{\max}$ are the actuation limits and $\Delta \delta_{\min}, \Delta \delta_{\max}$ are the steering rate limits.

4.4 Experimental Results

We perform experiments on the 1:43 numeric simulator, the Carla high-fidelity vehicle simulator [Dos+17a], and a real 1/10-scale RC car. Additional results are presented on a webpage¹ due to limited space.

Numeric simulator

We use the 1:43 autonomous open-source race car simulation [LDM15] to validate our algorithm. We demonstrate our results on the ETHZ track [LDM15] and the ETHZMobil track [Jai+20].

¹<https://dvij542.github.io/apacrace/>

Constant friction decay

We compare path-following performance without and with the use of adaptive model compensation in Fig. 4.4a and Fig. 4.4c. We modify the actual μ , i.e., change parameters D_f and D_r , as shown in Fig. 4.4d, and the predicted values as well. As can be seen, using our approach with adaptive model compensation clearly yields better performance. Using adaptive speed planning also allows the vehicle to stay within track boundaries. If speed references are not changed (see Fig. 4.4b), the vehicle is still able to complete its maneuvers, but leaves the track several times. This is because even with a compensated model, unadjusted reference speeds push the vehicle beyond its limits at turns. Not updating the vehicle model leads to the vehicle’s completely failing and losing control at a region on the 3rd lap. Finally, we also compare with an oracle (Fig. 4.5b), where we use the ground truth Pacejka friction parameters. As can be observed, the trajectory followed with our method (Fig. 4.4c) and the lap times (shown in Table 4.2) are almost the same as those of the oracle (Fig. 4.5b).

Moreover, we also compare against the use of adaptive Gaussian processes [Jai+20], where we adjust the speeds according to the ground truth value of μ (see Fig. 4.4d), and this yields slightly worse (very close) results than ours but with much higher computation times. Also, as can be observed from Fig. 4.4c and Fig. 4.5a, the trajectory followed when using GP goes further away from the reference with decreasing μ while with our method, the trajectory remains nearly the same similar to the oracle (see Fig. 4.5b). It should also be noted that using GP requires a separate μ prediction process to replace and obtain an estimate close to the ground truth value, while in our method we inherently obtain the predicted value of μ according to (4.7).

We summarize run statistics in Table 4.2. As can be seen: 1) our lap time is slightly worse than the oracle but the safety (i.e., the violation time) has been significantly improved; 2) the mean deviation from the racing line is least when our method is used with adjusted speeds (close to the oracle); 3) our approach outperforms the state-of-the-art GP-based controller [Jai+20] in terms of every criterion as well as the computation efficiency. The computation times for our method are within $0.05s \pm 0.02s$ and for GP is $0.30s \pm 0.05s$ when used online on our machine (AMD Ryzen 7 5000 series CPU, 16 GB RAM).

Sudden friction change

We test our approach in a situation where the friction coefficient suddenly changes. This is possible, for example, when a sudden heavy rain changes the road’s friction coefficient. For

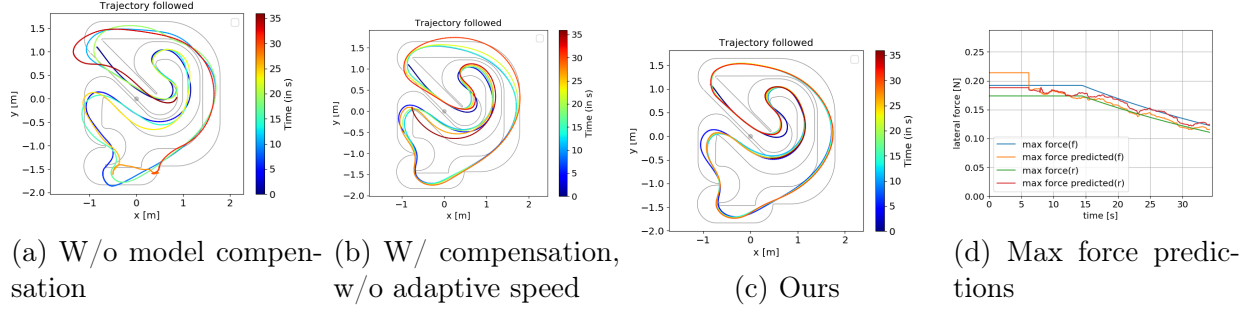


Figure 4.4: Comparison for w/o and w/ learning on numeric sim

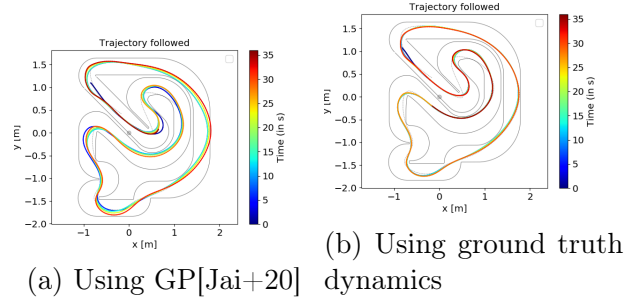


Figure 4.5: Comparison with other approaches on numeric sim

Method	Lap 0 time	Lap 1 time	Lap 2 time	Lap 3 time	Mean dev. from racing line	Violation time	Avg. cycle computation time
Without adaptation	8.44s	8.34s	8.72s	-	0.141m	4.32s	0.02s
With adaptation (Constant speeds)	8.26s	7.26s	7.56s	8.14s	0.1104m	2.84s	0.05s
With adaptation (Var speeds, ours)	8.26s	7.42s	7.76s	8.46s	0.0832m	0.16s	0.05s
GP	8.28s	7.44s	7.84s	8.56s	0.0989m	0.62s	0.32s
Oracle	<i>7.92s</i>	<i>7.40s</i>	<i>7.72s</i>	<i>8.40s</i>	<i>0.0511m</i>	<i>0s</i>	0.03s

Table 4.2: Track 1 (ETHZ track [LDM15])/Track 2 (ETHZMobil track [Jai+20]). Comparison in statistics for numeric simulator. Violation time is the total time the vehicle was outside the track limits.

this case, we demonstrate the path followed by our approach (See Fig. 4.6b) and compare it to when the model compensation is not considered (See Fig. 4.6a). As observed, with model compensation, the vehicle is able to estimate the change in friction coefficient quite well and rapidly. It does cross the outer boundary at the beginning but is able to adjust the speeds to stay safe within the lane limits later, whereas if compensation is not considered, the vehicle completely loses control at several points after μ changes.

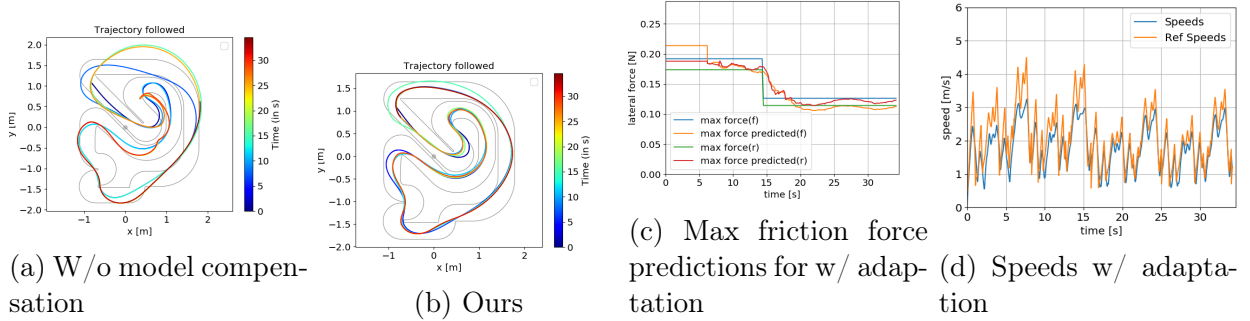


Figure 4.6: Online run comparison with model compensation & adaptive speed planning vs. if none are used (sudden friction change experiment)

Carla

We perform experiments in Carla [Dos+17a] with time-varying tire friction parameters to test our online adaptation algorithm. The online run with friction decay is shown in Fig. 4.7c. Both tires have the same tire model. For comparison, we show the results with and without adaptive model compensation (See Fig. 4.7b and Fig. 4.7a). Without adaptive model compensation, the vehicle moves out of bounds after about 60s once the friction coefficient starts reducing and collides with an obstacle. Due to limited space, more results are provided on the webpage.

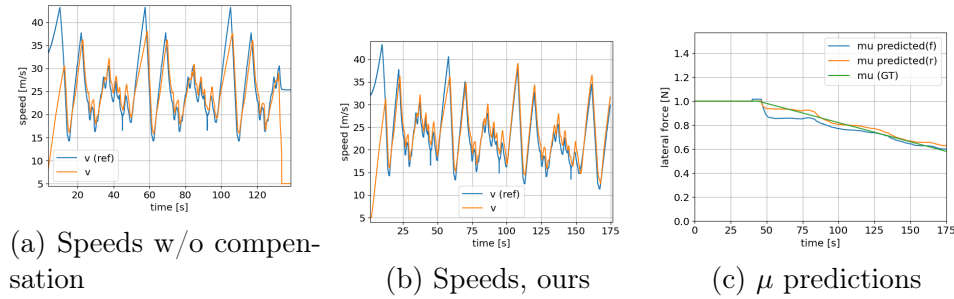


Figure 4.7: Online run with model compensation in Carla

Hardware experiments in an RC car

We test on a real F1/10 RC car (see Fig. 3.15) using Vicon [Mer+17] for localization.

Experiment setup

We perform the experiment on a track with a lane width of 0.5 m. The reference line is taken to be the centerline with limiting speeds rendered according to [Hei+20]. We conduct the experiment with an initial friction coefficient 1.5 and incrementally increase target speeds from 50% to 100% of the reference speeds from $t = 10s$ to $t = 35s$. This is to ensure physical safety and also allow sufficient time and data for the adaptation algorithm.

Results

The final trajectories for a rounded rectangle reference are shown in Fig. 4.8a and Fig. 4.8b for the comparison between w/o model and reference speed adaptation vs. ours. As can be clearly observed, without model adaptation, the vehicle goes outside the track and the car physically hits the wall, while with adaptation, the algorithm is able to correctly estimate the friction coefficient of the surface online (see Fig. 4.8c) and adjust the vehicle model and speeds accordingly. In the beginning, the car does exceed the track boundaries, but is able to effectively use this experience to recover quite well from its subsequent laps. For the experiment with model adaptation, we stop the car at $t = 70s$ and change the surface along the track from a rough original surface to a smoother surface obtained by placing wooden slabs on it. Then, we restart the same experiment but now on this new surface *starting with the previous trained model for testing the adaptation and transition*. The adaptation algorithm is able to reasonably adapt to the new friction parameters and estimate the friction coefficient to be around 15% less than the earlier predicted value of the original surface (see Fig. 4.9b). The trajectory for the new surface is shown in Fig. 4.9a. The statistical results are tabulated in Table 4.3. The website displays the results of another, diamond-shaped track with consistent conclusions, as space limits the presentation.

	Avg. lap time (Last 3 laps)	Mean deviation from reference	Violation time
Without adaptation	5.4s	0.52m	35.6s
With adaptation	5.05s	0.28m	2.1s
With adaptation (new surface)	5.08s	0.29m	7.0s

Table 4.3: Statistics for w/ and w/o model adaptation

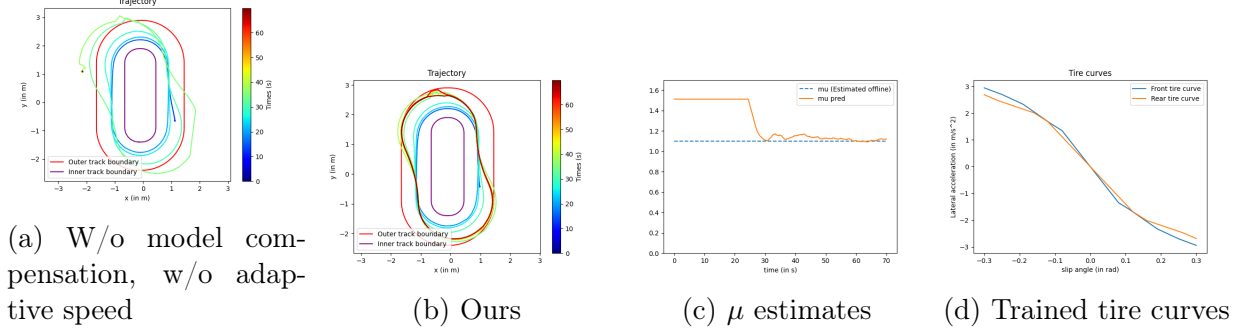


Figure 4.8: RC car experiment on rounded rectangle trajectory

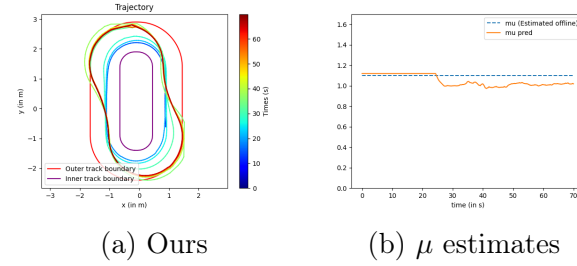


Figure 4.9: Adaptation to new surface

4.5 Conclusion

In this chapter, we introduce a novel learning-based control algorithm that not only significantly reduces the effort required for system identification but also enables real-time adaptation. Our approach commences with a simple geometric model and collects training data, which the algorithm utilizes iteratively to compensate discrepancy between the nominal and actual vehicle models. It adapts dynamically to any changes in the system. This capability empowers racing algorithms to craft aggressive strategies without the need for constant concern about vehicle model parameters and tire conditions. We showcase the algorithm's robust performance in scenarios featuring model uncertainty and simulated friction variations, leveraging both simulators and a physical testbed. One limitation of this work is that it is very specific to autonomous racing for a specific form of dynamics as described in this work. The same design may not work for completely different type of vehicles like if the rear axle is coupled to both rear tyres in which case both of them are constrained resulting into an opposing torque that needs to be added. Also if the car is front-driven or all-wheel driven, the same model structure will not hold that cannot be inherently learned by just learning tire curves and aerodynamic parameters but need to re-design the structure for them. Also if the car is moving in uneven terrains which cannot be directly physically modelled, or if the car is towing an object, is on a muddy terrain which has considerable

longitudinal slip as well, it requires good amount of engineering to propose a stable learnable model. To address this limitation, in chapter 7, we will show how we can use meta-learning to initialize a fully end-to-end model that makes no assumptions, takes as input history of states and actions and outputs the next state. The initialized model would be such that it can quickly adapt to any type of car with very few data but is generic enough to work on any car or any robotic system in general.

However, in the next 2 chapters (Chapter 5 and Chapter 6) we will see how we can use the model-based learning as we discussed in this chapter as a filter on reinforcement learning and imitation learning based task learning model-free controllers respectively.

SAFE REINFORCEMENT LEARNING

In the earlier 2 chapters, we designed classical and semi-learning approaches to handle autonomous racing control scenarios. We do need a task specification in form of the cost function and a planned trajectory to achieve the task. Our performance was tied to how good we are able to track the planned trajectory but we assume that planned trajectory will guarantee us optimal performance. This constraints the possibility of being able to go beyond the planned trajectory to directly achieve the task. Reinforcement learning is a popular approach to achieve direct task learning instead of learning the model first given a simulator where it can learn a direct optimal policy with continuous trial and error. Reinforcement learning (RL) agents need to explore their environment in order to learn optimal behaviors to achieve maximum reward. But exploration may be fundamentally risky in some situations when deploying RL on real systems, for example, in autonomous driving. Some recent approaches have leveraged safety filters like Control Barrier Functions (CBFs) to filter actions from RL during training. However, the strong safety guarantee of CBF relies on a precise dynamic model. In practice, there always exist uncertainties, including internal disturbance from the dynamic model as well as external disturbances like wind or measurement noise for the system state. Several works attempt to quantify these uncertainties to make the CBF more robust. However, this interferes with the RL policy and prevents exploration, thereby leading to a conservative policy.

In this chapter, we combine online residual model learning as we discussed in previous chapter and disturbance observers (DOB) in our CBF formulation to compensate for these model errors and external disturbances during learning while not compromising on optimality. We demonstrate our results on the Safety-gym benchmark for Point and Car robots on all tasks where we can outperform state-of-the-art approaches that use only residual model learning or DOB. We later also demonstrate our results on realistic scenarios like head-to-head racing and uneven terrain navigation. Later, we also demonstrate our results on a real RC car.

5.1 Introduction

Recently, reinforcement learning has been successful in solving various challenging control tasks on real systems, such as legged locomotion [Kum+21], manipulation [Mor+21], and autonomous driving [Kir+20]. Most of these approaches use some kind of sim-to-real transfer

to deploy the policies trained on simulators with varying model parameters to mimic the uncertainty in the real world and deploy the same policy on the real robot with the hope of being robust to real-world disturbances. Some very recent works have moved towards performing RL training on real systems directly [Wu+22]. Such systems should be able to remain safe while training [Pad+21], since obstacle collisions to inform the policy gradient to enforce safety are very expensive. To tackle this fundamentally, safe RL algorithms that respect safety while learning have been developed in recent years. There may be different definitions of safety, for example, in [GF15; Bru+21], but we focus on satisfying hard safety constraints like avoiding certain states as safety specifications.

Among different safe RL paradigms, the most common approach is to use safety filters on the actions generated by RL agents and modify them whenever necessary to satisfy the constraints. An advantage of such a design is that the safety filter can be used as a plug-in to any RL algorithm without much modification. These safety filters require precise dynamic models for a strong safety guarantee. System identification is difficult for most robots. Hence, some approaches have employed online model learning from environment interactions to learn the uncertain dynamics. The basic idea is to start with an initial nominal model and incrementally learn the discrepancy between the nominal and the true model. This is referred to as *residual model learning* in this thesis. However, when the learned model is poor, e.g., the machine learning has not converged or suffers from sampling inefficiency, the safety filter based on the sum of the nominal model and the learned discrepancy still fails to provide a safety guarantee. Some robust control methods that consider worst-case error bound often make the safety filters overly conservative. This in turn leads to high interference with RL, restricting it from learning an agile policy.

In [CZH22], the authors claim to use a simple nominal model for efficient RL training without learning the high-fidelity model. Instead, they employ a Disturbance Observer (DOB) to compensate for disturbances, including model errors. As a result, the RL approach is almost model-free, eliminating the need for continuous learning of the dynamic model, as the model discrepancy is addressed by the DOB, which provides stronger guarantees for uncertainty quantification. The limitation is that when the nominal model has large discrepancy from the true model and we also have some external disturbance, the DOB is required to compensate for all of them, which leads to slow estimation convergence.

Our work is motivated by the fact that, as we operate the system over time, we gain a better understanding of the disturbances. Once we collect data from these disturbances and attempt to model them, we can use this information as prior knowledge in the Disturbance Observer (DOB). This reduces overall uncertainty, benefiting the DOB by enabling it to

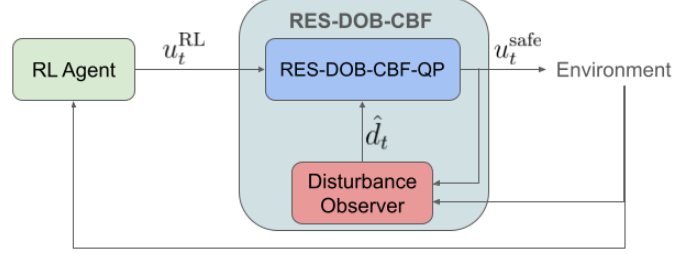


Figure 5.1: Proposed safe RL framework via using RES-DOB-CBF. At each time step t , the RL agent action u_t^{RL} is filtered by RES-DOB-CBF based safety to avoid any unsafe actions

estimate fewer disturbances and converge faster

The contribution of our work is that we propose a seamless integration of residual model learning and DOB to leverage the advantages of both. This integration is expected to provide stronger convergence guarantees than using residual model learning alone, as the imperfections of the model learning can be compensated for by the DOB, providing an additional layer of assurance. Furthermore, the integration is expected to achieve faster disturbance estimation convergence compared to using only the DOB, thanks to the prior information on the disturbances captured by the machine learning model. We comprehensively validate our approach on a modified safety-gym benchmark [Ope], in which we outperform vanilla PPO_lagrangian, DOB+CBF [CZH22], and Residual Model+CBF approaches on all tasks.

5.2 Preliminaries

We consider a non-linear control affine system:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t)) + g(\mathbf{x}(t))\mathbf{u}(t) + \mathbf{d}(t) \quad (5.1)$$

where $\mathbf{x}(t) \in \mathcal{X} \subset \mathbb{R}^n$ is the state vector, $\mathbf{u}(t) \in \mathcal{U} \subset \mathbb{R}^m$ is the input vector, $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ and $g : \mathbb{R}^n \rightarrow \mathbb{R}^n$ are the Lipschitz-continuous model functions, and $\mathbf{d}(t) \in \mathcal{D} \subset \mathbb{R}^n$ represents nonlinear time-varying disturbance.

Disturbance Observer (DOB)

DOB has been widely used in control theory for uncertain systems [Che+16]. The idea is to address all unknown uncertainties from internal disturbances (model errors) and external disturbances as lumped disturbances. In this work, we use the same DOB formulation as used in [CZH22]. DOB consists of two components: a state predictor and a piecewise-constant (PC) estimation law. The state predictor is given by:

$$\dot{\hat{\mathbf{x}}}(t) = f(\mathbf{x}) + g(\mathbf{x})\mathbf{u} + \hat{\mathbf{d}}(t) - a\tilde{\mathbf{x}} \quad (5.2)$$

where $\tilde{\mathbf{x}} = \hat{\mathbf{x}} - \mathbf{x}$ denotes the state prediction error, $a > 0$ is a constant, and $\hat{\mathbf{d}}(t)$ is the estimated disturbance. The disturbance estimation is updated according to

$$\begin{cases} \hat{\mathbf{d}}(t) = \hat{\mathbf{d}}(iT), & t \in [iT, (i+1)T), \\ \hat{\mathbf{d}}(iT) = -\frac{a}{e^{aT} - 1} \tilde{\mathbf{x}}(iT), & i = 0, 1, \dots, \end{cases} \quad (5.3)$$

where T is the estimation sampling time. Under reasonable assumptions, the estimation error from DOB, $\|\hat{\mathbf{d}}(t) - \mathbf{d}(t)\|$ can be bounded by a pre-computed estimation bound [CZH22].

Model Residual learning

Moving back to (5.1), let's say we do not have an accurate model for f and g but a nominal model $\hat{f}$ and $\hat{g}$. The difference between the actual models f, g and nominal models $\hat{f}, \hat{g}$ maybe due to many reasons like unknown or inaccurate model parameters, dynamically changing environmental parameters like surface friction, weather, etc. It may also be because of difficult-to-determine dynamic model parameters like tire friction, in which case the nominal model can be chosen to be the kinematic model of the robot. Hence, we can modify (5.1) as follows:

$$\dot{\mathbf{x}}(t) = \hat{f}(\mathbf{x}(t)) + \hat{g}(\mathbf{x}(t))\mathbf{u}(t) + \Delta f(\mathbf{x}(t), \theta) + \Delta g(\mathbf{x}(t), \theta) + \mathbf{d}(t) \quad (5.4)$$

Where θ is represented by a feed-forward neural network which takes the state, $\mathbf{x}$ as input and outputs $(\Delta \bar{f}, \Delta \bar{g})$. We can estimate θ from online collected data by optimizing the following loss function

$$\begin{aligned} \theta^* &= \underset{\theta \in \Theta}{\operatorname{argmin}} \mathcal{L}(\hat{S}(\mathbf{x}, \mathbf{u}, \theta), \dot{h}(\mathbf{x}, u)) \\ \dot{S}(\mathbf{x}, \mathbf{u}, \theta) &= \hat{f}(\mathbf{x}(t)) + \hat{g}(\mathbf{x}(t))\mathbf{u} + \Delta \bar{f}(\mathbf{x}(t), \theta) \\ &\quad + \Delta \bar{g}(\mathbf{x}(t), \theta)\mathbf{u}(t) + \hat{d}(t) \end{aligned} \quad (5.5)$$

We can update θ iteratively with each new collected data element with the following update rule: $\theta_{k+1} = \theta_k - \beta \frac{\delta L_k}{\delta \theta}$ where β is the learning rate. This formulation is similar to [Mun+22]. For the DOB part in (5.2) and (5.3), we take $f = \hat{f} + \Delta f$ and $g = \hat{g} + \Delta g$. We assume the error in model prediction will be covered by disturbance $d(t)$.

5.3 Methodology

Residual model and DOB-based Control Barrier Function (RES-DOB-CBF)

The high-order CBF having $\Delta\bar{f}$, $\Delta\bar{g}$ and $\hat{\mathbf{d}}$ is

$$\psi_m(\mathbf{x}, \mathbf{u}, t) \triangleq L_f^m h(\mathbf{x}) + L_g L_f^{m-1} h(\mathbf{x}) \mathbf{u} + [\mathcal{L}_f^{m-1} h(\mathbf{x})]_{\mathbf{x}} \hat{\mathbf{d}}(t). \quad (5.6)$$

where $h_{\mathbf{x}}(\mathbf{x}) \triangleq \frac{\partial h(\mathbf{x})}{\partial \mathbf{x}}$, $L_f h(\mathbf{x}) \triangleq \frac{\partial h(\mathbf{x})}{\partial \mathbf{x}} f(\mathbf{x})$ and $L_g h(\mathbf{x}) \triangleq \frac{\partial h(\mathbf{x})}{\partial \mathbf{x}} g(\mathbf{x})$. For more details on the derivation, readers are referred to [Mun+22] with the only difference being that we replace $f = \hat{f} + \Delta\bar{f}$ and $g = \hat{g} + \Delta\bar{g}$.

Safe-RL policy training with RES-DOB-CBFs

To obtain a safe control, we solve the following quadratic programming (QP) problem based on the Res-DOB-CBF (5.6):

$$\begin{aligned} u_{\text{safe}} &= \underset{u \in \mathcal{U}}{\operatorname{argmin}} \frac{1}{2} (u - u_{\text{RL}})^T P (u - u_{\text{RL}}) \\ \text{s.t. } &\mathcal{K}(t, x, u) + \beta_m (\phi_{m-1}(x)) \geq 0 \end{aligned} \quad (5.7)$$

where P is the positive-definite weighting matrix, u_{RL} is the nominal control, u_{safe} is the final safe control. To have an efficient and safe RL controller, it is critical to obtain a safety filter based on an accurate dynamic model.

5.4 Experimental Results

We perform experiments on the Safety-gym benchmark [AA19] on the Point and Car robots. The environment selection includes Goal1, Goal2, Button1, Button2, Push1, and Push2. The obstacles include hazards, pillars, vases, and gremlins. Safety-gym is a strong benchmark to evaluate the performance of Safe-RL algorithms. We use the *PPO_Lagrangian* algorithm [AA19] that regulates the negative reward received from violating constraints to achieve a target cost. We set the target cost to 25 for all the experiments, the same as [AA19], which means the agent is allowed to violate constraints for 25 time steps in an episode of length 2000. To validate our approach's efficacy for uncertainties, we include internal disturbance caused by model uncertainty, e.g., 50% model errors in the kinematic models. We also add wind as an external disturbance of fixed magnitude (0.001 for point robot and 0.01 for Car robot) and continually changing direction with a constant angular rate of 5 Hz.

Point robot experiments

First, we present results on a Point robot. We model the point robot model with a kinematic unicycle model:

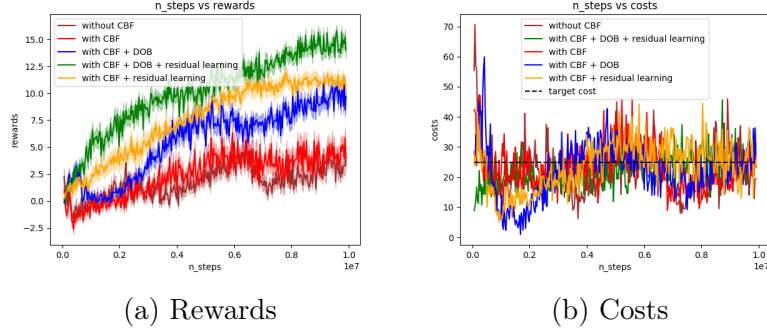


Figure 5.2: Comparisons for Goal1 task on Point robot

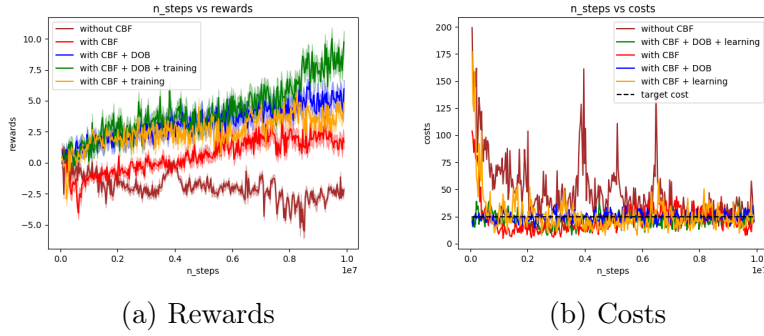


Figure 5.3: Comparisons for Goal2 task on Point robot

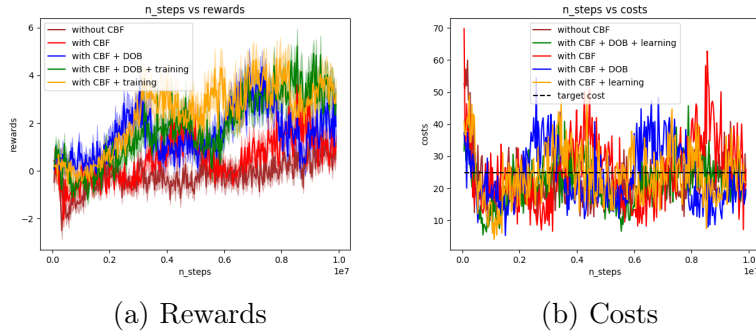


Figure 5.4: Comparisons for Button1 task on Point robot

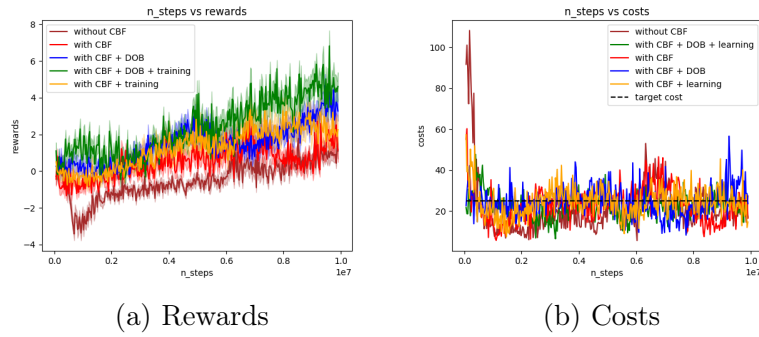


Figure 5.5: Comparisons for Button2 task on Point robot

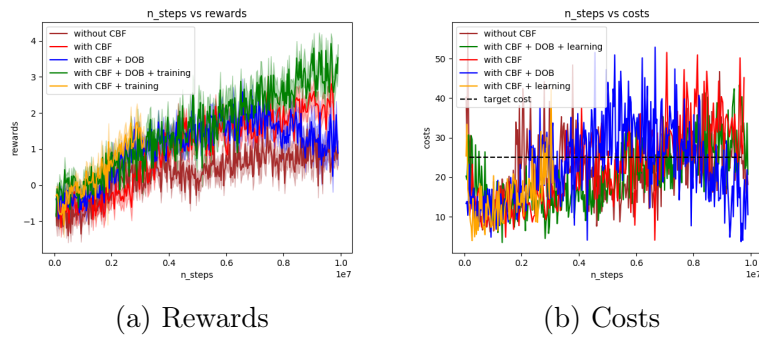


Figure 5.6: Comparisons for Push1 task on Point robot

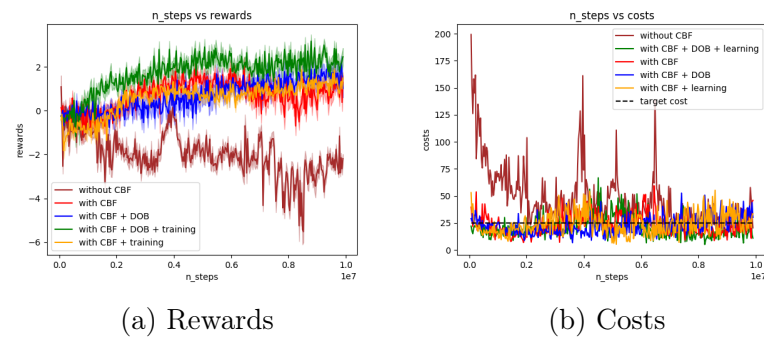


Figure 5.7: Comparisons for Push2 task on Point robot

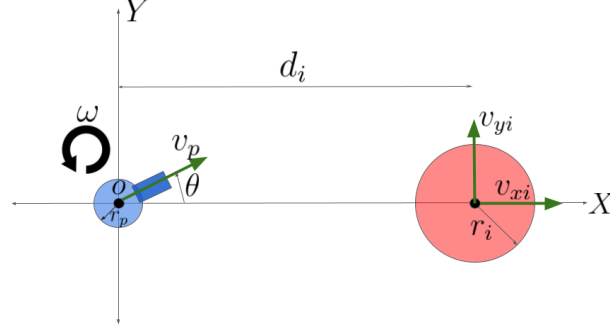


Figure 5.8: Reference frame of the robot for obstacle avoidance. The blue circle is the robot, and the orange circle is the moving obstacle.

$$\begin{bmatrix} \dot{x}_p \\ \dot{y}_p \\ \dot{\theta}_p \\ \dot{v}_x \\ \dot{v}_y \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_p \cos(\theta_p) \\ v_p \sin(\theta_p) \\ K_\omega c_\omega \\ K_{v2}(K_{v1}c_v - v_x) \\ 0 \\ 0 \end{bmatrix} \quad (5.8)$$

where the state $\mathbf{x} = [x_p, y_p, \theta_p, v_x, v_y, \omega]^T$, the control $\mathbf{u} = [v_t \ \omega]^T$, (x_p, y_p) is the center position of the robot, θ_p is the yaw angle, v_x, v_y are the body frame longitudinal and lateral velocities and ω is the angular velocity. c_v, c_ω are the target velocity and angular velocity commands. Note that we have lateral and angular velocities included as states but they are not used in the kinematic model. This is just to enable the residual model learning algorithm to learn the difference between the actual dynamic model and the approximated kinematic model from the collected data online.

We formulate the safety specification of a robot to avoid any i -th circular region in space that is moving with a constant velocity. For any moving obstacle, we assume it would move with a constant velocity v_{xi} and v_{yi} with the frame of reference visualized in Fig. 5.8. Let the radius of the moving obstacle be r_i and of the robot be r_p . Without loss of generality, we derive the CBF for a frame of reference with origin at the robot and the X axis direction along the circular region to be avoided. Let the angle made with the X axis (i.e., the direction towards the obstacle) be θ . Let the distance between the obstacle and the robot be d_i . To avoid a collision, we need to have $d - r_i - r_p \geq 0$. Therefore, we define $h_i(X) = d_i - r_i - r_p = \sqrt{(x_p - x_{ci})^2 + (y_p - y_{ci})^2}$.

The Point robot results in all environments are given in Figures 5.2-5.7 (due to space constraints, some results have been moved to the appendix). Without CBF and in the presence

of external disturbance the RL algorithm is not able to learn anything useful towards the task while remaining safe, as the reward is negative. Having CBF assistance helps the RL agent to learn something better, but as it is not perfect due to external disturbances and model error, it fails to learn an aggressive policy. Adding DOB and residual model learning help in achieving better rewards, but using only one of them still lags in achieving an aggressive policy. Using both residual model learning and DOB to compensate for model error and external disturbances separately is able to achieve the maximum reward in all environments.

Car robot experiments

For a Car robot, the control commands are the left and right wheel angular velocities, c_l and c_r respectively. For the nominal kinematic model, we simply change $\dot{v}_x = K_{v1}(K_{v2}(c_l + c_r) - v_x)$ and $\dot{\omega} = K_{\omega}(c_r - c_l)$. K_{v1} , K_{v2} and K_{ω} are approximated offline. Learning an optimal policy for the Car robot is more challenging, as the left and right wheel commands need to be coordinated. Also, there is more difference between the actual dynamic model and the kinematic model for the Car robot than the Point robot, meaning there is more skidding for the Car robot. The results are consistent with Figures 5.9-5.13.

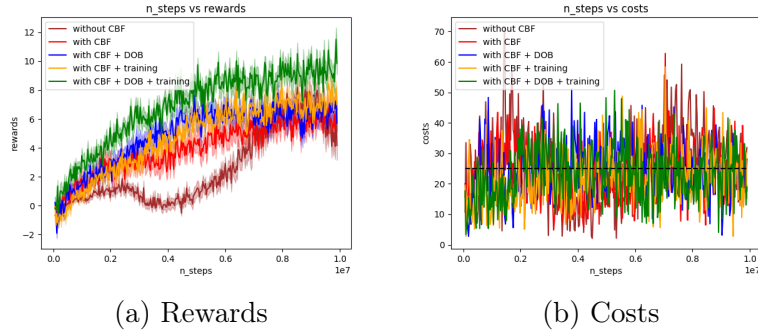


Figure 5.9: Comparisons for Goal1 task on Car robot

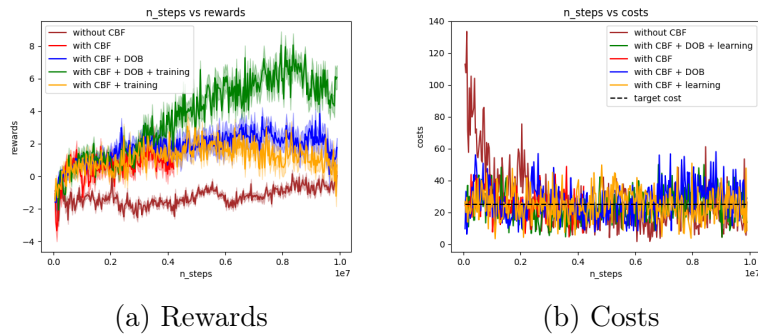


Figure 5.10: Comparisons for Goal2 task on Car robot

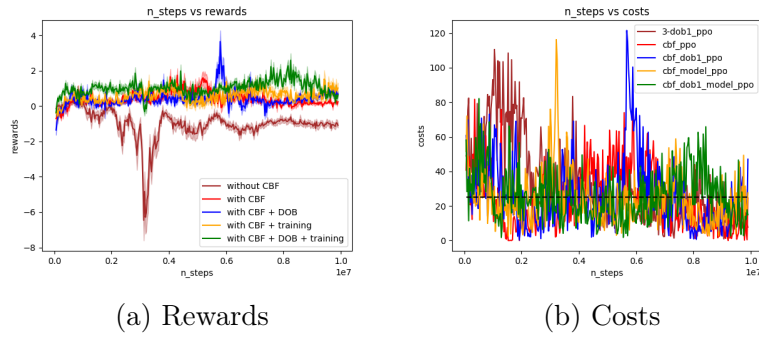


Figure 5.11: Comparisons for Button1 task on Car robot

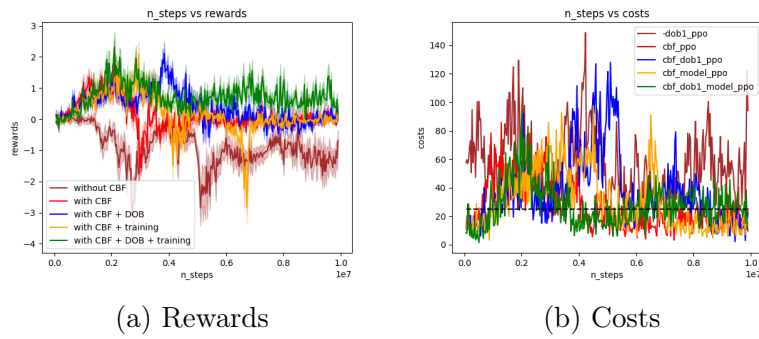


Figure 5.12: Comparisons for Button2 task on Car robot

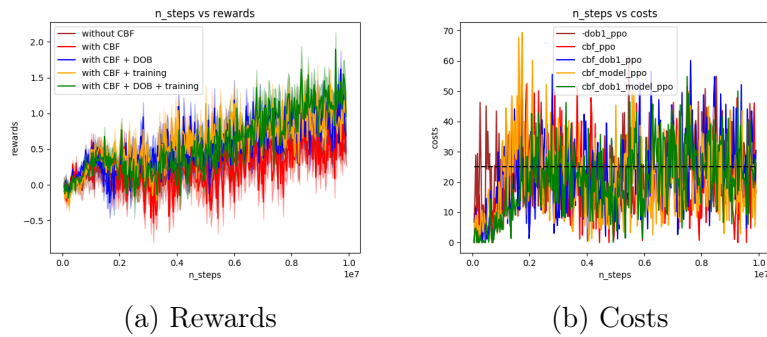


Figure 5.13: Comparisons for Push1 task on Car robot

	w/o CBF	RES-CBF	DOB-CBF	RES-DOB-CBF
Avg. commute time	5.42s	3.42s	3.47s	3.37s

Table 5.1: Average commute times for RC car experiment

RC car results

At last we also deploy our solution on hardware. For this we design a simple task of going back and forth between 2 goals similar to Goal1 task of safety-gym but with only 1 hazard in the middle and also with the added constraint of staying within the square arena of $4.2m \times 4.2m$. Hence, we have CBF constraints for the 4 walls and for the hazard at the center of radius $0.8m$ with the same formulation as seen earlier in safety-gym experiments.

To save time with RL on the hardware, we first train a base model in Unity simulator (see Figure 5.14) with the same dimensions as the physical arena but with the only objective to reach the goal while not hitting the walls but not caring about the hazard. The pre-trained model on the simulator is then deployed on the hardware with the added task of avoiding the hazard and trying to minimize the commute time between the goals on the hardware model i.e. sim-to-real policy transfer. We test the same baselines as in safety-gym on this setup. We run all baselines for 10 minutes on the hardware. We set the violation goal of 0 for PPO_lagrangian. The trajectories followed for all baselines are given in Figure 5.15. The average commute times for all approaches for last 5 mins are listed in Table 5.1. As can be seen in Figure 5.15a, without CBF it struggles to learn a good policy and ends with 5.42s average commute time. With *DOB - CBF*, *RES - CBF* and *RES - DOB - CBF*, it achieves good performance while avoiding hazards with the best average time achieved by *RES - DOB - CBF*. All videos are in [LINK](#)

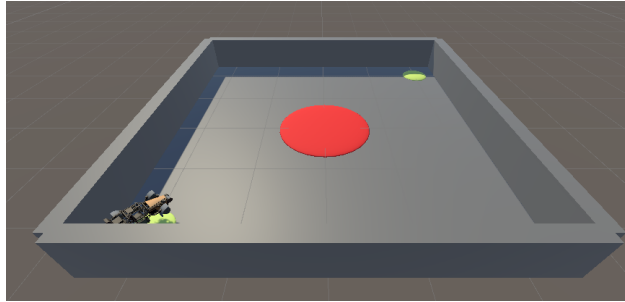


Figure 5.14: Snapshot of twin RC car simulator in unity

5.5 Case study: Head-to-head autonomous racing

We aim to apply and test the CBF-guided safe RL training solution on head-to-head racing problem. The environment is similar to [Tha+22], where 2 agents race against each other

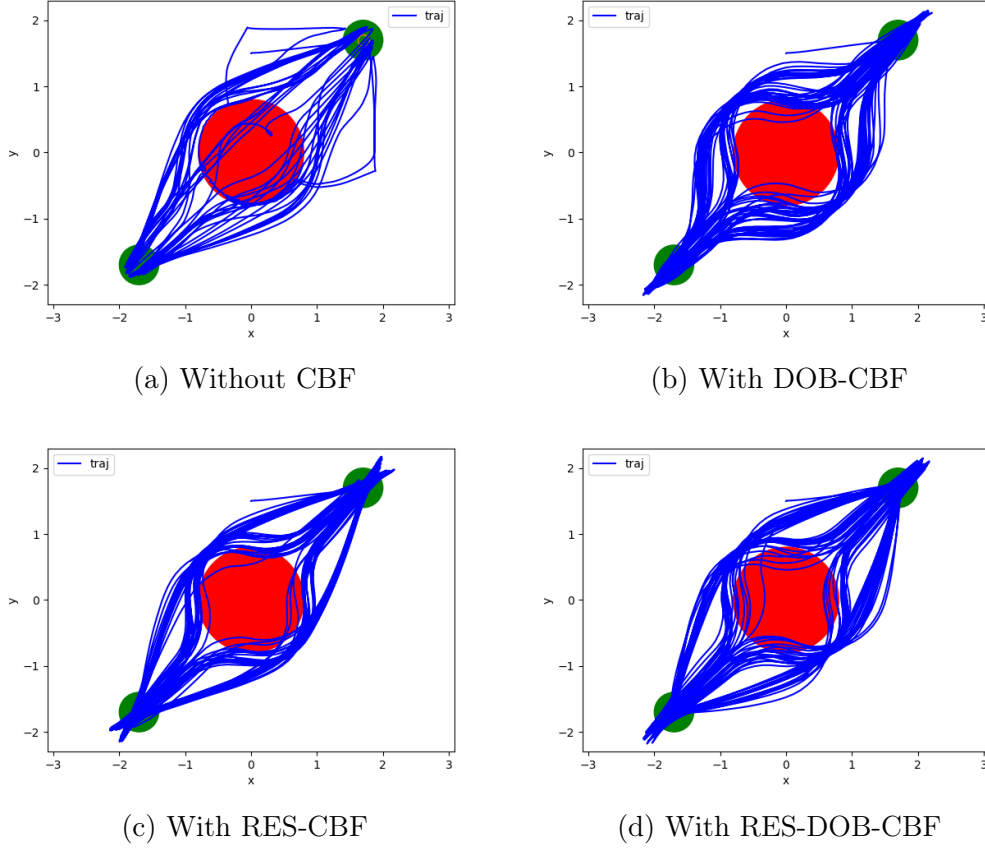


Figure 5.15: Trajectories followed for different approaches for RC car experiment

with the objective to win or gain a time advantage over its opponent. We modify the vehicle model f used in [Tha+22] from a simple kinematic model to a more challenging and realistic dynamic bicycle model. The state of the car x_t is a 6- d vector including global coordinates p_x , p_y , and yaw rotation ϕ in the global frame; longitudinal velocity v_x , lateral velocity v_y , and yaw angular velocity ω in the vehicle's body frame. Throttle d and steering angle δ are control actions. The dynamic model is defined in Eqn 2.3

We employ a hierarchical design with a high-level planner for discrete checkpoint planning followed by a low-level tracking controller. Having a decoupled planning approach facilitates a real-time implementation and achieving long-term strategic plans, e.g., overtaking [Tha+22].

High-level planner

The high-level tactical planner outputs a high-level discrete plan. This can be obtained by first discretizing the state, defining the state transition model, and then solving the

optimization problem, as discussed in the remainder of this subsection.

State discretization

We first transform a continuous state of the vehicle into a discrete state. We encode the position of the vehicle into a pair of discrete variables, i.e., lane ID and last passed checkpoint (or longitudinal segment). The longitudinal velocity is also represented using a discrete interval. Fig. 5.16 exemplifies the state conversion from continuous to discrete.

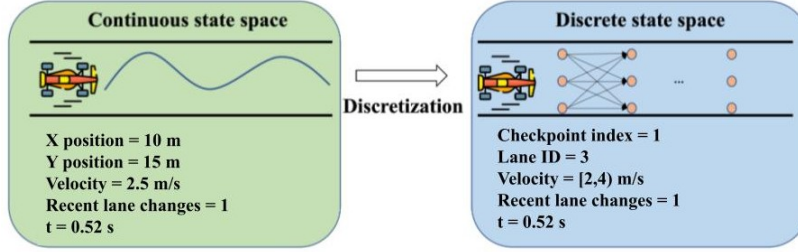


Figure 5.16: Converting a continuous state to a discrete state

State transition model

As shown in Fig. 5.16, a vector of discrete state includes: lane ID, velocity interval, recent lane changes, checkpoint index, and time state. A transition is feasible if all the continuous states from the current discrete state are able to reach at least one discrete state in the next step. If all the jump conditions are not satisfied, we rule out the transition. Our update implementation is similar to [Tha+22] where for every state transition, we strictly increase the checkpoint index (or longitudinal segment number) by 1. A transition to a given lane and speed range happens only if it is admissible within the speeds supported by the longitudinal acceleration and friction constraints. The number of lane changes counter is increased by the absolute difference in the new and old lane IDs.

The game is played with both players starting at an initial checkpoint and progressively updating each player's choices with the smallest time state at each point. A lower time state value implies that the player in question reached the particular checkpoint before the other player and hence gets to choose the next action. This gives the other player a chance to make a resultant strategic action, like whether and how to overtake, etc. A collision avoidance rule is incorporated in the high-level planner by restricting actions that result in the same checkpoint and the same lateral lane with time difference less than mT .

High-level optimization

The high-level problem of minimizing racing time against the opponent is solved using Monte Carlo tree search (MCTS). The optimal performance is defined as follows:

$$C_{X_a, X_{a+1}, \dots, X_{b-1}} = \sum_{i=a}^{i=b-1} (o_i - X_{i, \text{lane}})^2 - k_{\text{time}} X_{b-1, t} \quad (5.9)$$

where o_i is the optimal lane and X_i is the high-level state at the i -th checkpoint. The optimal lane is determined by finding the (segment, lane) pair at which the optimal racing line passes. The optimal racing line is computed by solving a minimum-time trajectory optimization [JM20]. The solution of MCTS is a sequence of discrete states of the player and the adversarial opponent. Note that we assume that the ego vehicle and opponent vehicle try to optimize based on the same objective function. Our formulation based on MCTS is different from [Tha+22], since our optimal solution is obtained by prioritizing staying closer to the racing line rather than minimizing time. We set a very low value of k_{time} to first prioritize choosing a trajectory close to the racing line and then for that same trajectory choose the one with least time. We believe this leads to a better optimal solution, as explained in Section 4.4.

Low-level controller

Reward design

The reward for the low-level controller is defined as follows:

- Positive reward for passing through a checkpoint and additional reward for reaching the target lane and the target speed: $k_{\text{target}} e^{-\max(d_{tc} - \frac{L}{2n_l}, 0)} + k_{\text{target}, v} e^{-\max(|v - v_{tc}| - \frac{v_w}{2}, 0)}$, where d_{tc} is the distance from the target checkpoint center, L is the track width, n_l is the number of lateral lanes, v is the speed when passing the checkpoint, v_{tc} is the mean of the target speed ranges and v_w is the target window size.
- Negative reward for taking too long to pass two consecutive checkpoints: $-k_{\text{time}} \Delta t$, where Δt is the passing time of two consecutive checkpoints.
- Negative reward for swerving too frequently on straight lanes: $-k_{\text{swerve}} \mathbb{1}_{(x, y) \in S \wedge l_{k-1} \neq l_k}$, where S is the set of straight section checkpoints, k is the current checkpoint number, l_k is the lane at the k^{th} checkpoint. $\mathbb{1}$ is an indicator function used for checking if the

current passed lane is not the same as the last passed checkpoint lane and if the car is on a straightaway.

- Negative reward for colliding with the wall. We use an indicator function $\mathbb{1}_{I_j \leq h \wedge I_j \text{ wall-hit}}$ that determines if the j^{th} Lidar reading is less than h and if Lidar bounced off the wall:
 $-\sum_{j=1}^{n_{\text{lidar}}} k_{\text{wall-hit}} \mathbb{1}_{I_j \leq h \wedge I_j \text{ wall-hit}}$ where n_{lidar} is the number of rays from the Lidar.
- Negative reward for collision with other players. We use the indicator function $\mathbb{1}_{I_j \leq h \wedge I_j \text{ opponent}}$ to check if the j^{th} Lidar’s ray touches the opponent and we have a set ϕ containing all Lidar rays that point to the front of the car for which we impose an additional penalty:
 $-\sum_{j=1}^{n_{\text{lidar}}} k_{\text{opp},1} \mathbb{1}_{I_j \leq h \wedge I_j \text{ opponent}} + k_{\text{opp},2} \mathbb{1}_{I_j \leq h \wedge I_j \text{ opponent} \wedge j \in \phi}$
- Negative reward for braking unnecessarily, i.e., when speed is lower than the target window, high lateral slips:

$$k_{\text{brake}} \mathbb{1}_{v \leq v_{\text{target}} \wedge d \leq 0} + k_{\text{slip}} (\alpha_f^2 + \alpha_r^2)$$

Network architecture

The PPO RL algorithm is used to obtain the optimal policy. The neural network used for estimating the value function and the policy is a simple feedforward neural network with 8 hidden layers and 128 neurons in each layer. There is a **Tanh** layer at the end to reshape the output steering and throttle into their legitimate ranges. The input consists of the following values: 1) The vehicle’s body frame state (v_x, v_y, w) ; 2) The relative state w.r.t. the racing line reference, i.e., signed lateral distance from the racing line e_1 , relative angle w.r.t. the closest point on the racing line e_2 ; 3) Relative position of the opponent vehicle; 4) Discrete high-level target state. All the values within the range are passed by the average of the lower and upper limits; 5) Raw Lidar data consisting of distances of 32 rays cast from the extreme left of the car to the right.

Training environment

Training was conducted on 16 parallel tracks, 8 clock-wise and 8 counter-clockwise with different turn radii so that the agent does not overfit its policy to a specific track shape but can generalize across different track shapes. This setup is similar in design to [Tha+22].

Vehicle model transition

With the problem formulation and the hierarchical control design in place, we now define the proposed curriculum-based learning framework to teach the RL agent an optimal policy

progressively. Let us define a parameter t_s denoting the time scale. We vary t_s from 0 to 1, where $t_s = 0$ means the easiest initial environment setup and $t_s = 1$ is when the original complex environment is used for training. We will elaborate later in this section on how we use t_s to modify the environment. We transition t_s linearly from 0 and 1 (i.e., t_{start} training steps to t_{end} training steps) so that after t_{end} training steps, the environment is the same as the original. Mathematically, we vary t_s as:

$$t_s = \max \left(0, \min \left(1, \frac{t - t_{\text{start}}}{t_{\text{end}} - t_{\text{start}}} \right) \right) \quad (5.10)$$

The highly non-linear dynamics (defined in (2.3)) pose a significant challenge for the RL agent to learn an effective policy directly. This is primarily because it is difficult to learn how to move at optimal speeds while respecting the friction limits and skidding caused due to high lateral slips at higher speeds. Hence, we define a transition from a relatively simpler dynamic model (close to a kinematic model with fewer slips at the same speeds and much higher friction limits) to the complex model with the actual parameters. The parameters of the Pacejka tire model change with the time scale (given in (5.10)) as:

$$\begin{aligned} D_{ft_s} &= 2^{1-t_s} D_{f0} \\ C_{ft_s} &= C_{f0}^{2^{t_s}-1} \\ B_{ft_s} &= 2^{1-t_s} \frac{D_{f0} C_{f0} B_{f0}}{D_{ft_s} B_{ft_s}} \end{aligned} \quad (5.11)$$

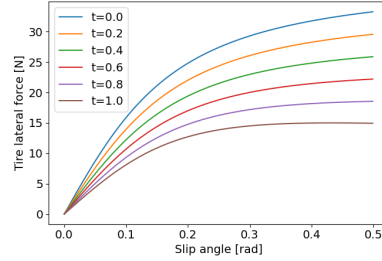


Figure 5.17: Tire curve variation for curriculum learning

The tire curves obtained at these Pacejka tire parameters are plotted in Fig. 5.17. Steeper slopes at $t_s = 0$ mean smaller slip angles to achieve the same force and hence less drifting. Also, the larger the maximum of the curve, the higher speed the vehicle can attain without skidding at the same curvature. Hence, it is easier to control at higher speeds when the tire curve is steeper with a larger maximum. As can be seen from Fig. 5.17, we transition from steeper slopes with higher force limits to the original tire curve to transition from an easier-to-control vehicle model to the original one.

Safety CBF

We use a control barrier function [Ame+19] (CBF) as a shield to guide the agent to learn a safe policy [Bru+21]. We observe that the RL agent struggles at the beginning, hitting the

walls, which debar the agent from learning meaningful behavior later in the episode. To get rid of this, we define a CBF for the wall boundary as a set of safety constraints. We define the safety function h and derive a 2nd-order CBF as follows:

$$\begin{aligned}
h_{\text{right}}(x) &= -e_{\text{center}} + w \\
\dot{h}_{\text{right}}(x) &= -v_x \sin(\theta - \theta_{\text{center}}) - v_y \cos(\theta - \theta_{\text{center}}) \\
\ddot{h}_{\text{right}}(x, u) &= -\dot{v}_x \sin(\theta - \theta_{\text{center}}) - \dot{v}_y \cos(\theta - \theta_{\text{center}}) - \omega \cdot (v_x \cos(\theta - \theta_{\text{center}}) - v_y \sin(\theta - \theta_{\text{center}}))
\end{aligned} \tag{5.12}$$

$$\begin{aligned}
h_{\text{left}}(x) &= e_{\text{center}} + w \\
\dot{h}_{\text{left}}(x) &= v_x \sin(\theta - \theta_{\text{center}}) + v_y \cos(\theta - \theta_{\text{center}}) \\
\ddot{h}_{\text{left}}(x, u) &= \dot{v}_x \sin(\theta - \theta_{\text{center}}) + \dot{v}_y \cos(\theta - \theta_{\text{center}}) + \omega \cdot (v_x \cos(\theta - \theta_{\text{center}}) - v_y \sin(\theta - \theta_{\text{center}})) \\
C_{\text{right}}(x, u) &= -\min(0, \lambda_1 \lambda_2 \ddot{h}_{\text{right}} + (\lambda_1 + \lambda_2) \dot{h}_{\text{right}} + h_{\text{right}}) \\
C_{\text{left}}(x, u) &= -\min(0, \lambda_1 \lambda_2 \ddot{h}_{\text{left}} + (\lambda_1 + \lambda_2) \dot{h}_{\text{left}} + h_{\text{left}})
\end{aligned} \tag{5.13}$$

where $C_{\text{left}}(x, u)$ and $C_{\text{right}}(x, u)$ are two certificates for the right and left boundary, respectively. θ_{center} is the angle at the closest point on the centerline of the track and e_{center} is the lateral error.

The safe control is obtained via the following optimization program, where K_{viol} is set to a high value and u_{ref} is the nominal control obtained from policy that is potentially unsafe:

$$\begin{aligned}
\min_u \quad & K_{\text{viol}}(C_{\text{right}}^2 + C_{\text{left}}^2) + \|u - u_{\text{ref}}\|^2 \\
\text{s.t.} \quad & u_{\text{min}} \leq u \leq u_{\text{max}}
\end{aligned} \tag{5.14}$$

This new safe action is fed to the environment for controlling the agent. With this, the RL agent would be able to stay safe while learning, as the CBF shield defined in (5.14) will modify the unsafe controls from the nominal u_{ref} to their corresponding nearest safe commands u if admissible within control limits u_{min} and u_{max} . To inform the policy update to generate safe action on its own and thus not be overridden by CBF, we also add a negative reward for constraint violation proportional to the change in command caused by the CBF shield as follows: $R_{\text{constraint}} = k_{\text{constraint}} \|u - u_{\text{ref}}\|^2$

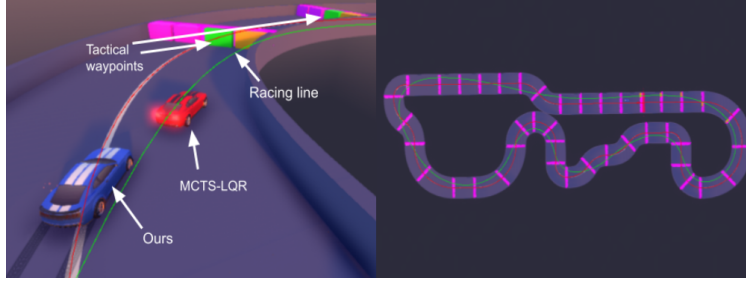


Figure 5.18: A snapshot of the racing simulator on the left where two racecars compete against each other to win the race. The blue vehicle is controlled by our algorithm, while the red vehicle’s controller is based on combining Monte Carlo tree search and linear quadratic regulator. The green line is the racing line, moving along which guarantees best lap times. The right image shows the top-down view of the track map on which the races are conducted.

Simulation Results

We upload some videos of the runs on this anonymous webpage ¹. Our racing simulator is implemented in the Unity Game Engine [Haa14], with an example image shown in Fig. 5.18.

Training results

We train our controllers on the oval tracks of different sizes as described in Section 5.5. We conduct test races on the track shown in Figure 5.18. We race with two agents with different controllers for comparison. The races are conducted with the initial position set randomly either on the left or the right at the starting line with an equal chance, since one position may be at an advantage if it is closer to the racing line. We first give training rewards obtained by curriculum learning to show the advantage of using curriculum learning in Fig. 5.19b. As can be observed, using curriculum learning with only model changes clearly outperforms the controller trained without using it in terms of the rewards obtained during training. t_{start} and t_{end} are chosen to be 500000 and 1500000. It is unfair to compare till 1500000 steps, as the controller trained with model-based curriculum runs on a different RL environment with a superior vehicle model. But after 1500000 steps both environments have the same configuration and our controller clearly outperforms the non-curriculum-based RL controller in reward. Adding RES-DOB-CBF filter further helps in getting even better rewards, as can be seen in Figure 5.2a. Please note that with the RES-DOB-CBF filter added, due to negative reward for each CBF constraint violation, the reward is less at the beginning but it eventually improves at the end, achieving an even larger reward. Now, for validation, we conduct 4 races each between the 3 pairs of agents (so effectively 12 races) every 250000 steps on the complex track shown in Figure 5.19b. It clearly shows our controller with both

¹<https://sites.google.com/view/rlc-submission>

model and CBF curricula used during training wins most races at all times. Also, it is better than only using model-based curriculum learning in terms of the number of races won during validation. It can be observed that the curriculum-based approach is able to completely beat the non-curriculum approach at the beginning, since it learns quickly at the beginning on a simpler environment. Later, with more training, the without-curriculum approach does start winning some races, as it starts learning a competitive policy which wins in some situations where it has an advantageous starting position. But the curriculum-based approach is still better in being able to win most races.

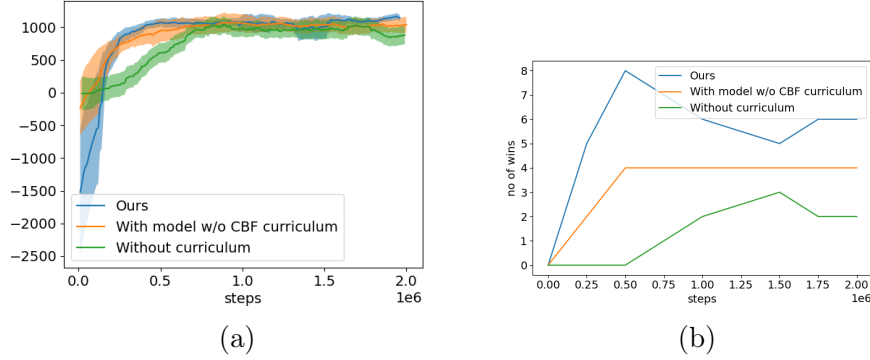


Figure 5.19: (a) Training rewards (b) No of wins across training steps. Ours : With using both model curricula and RES-DOB-CBF filter during learning

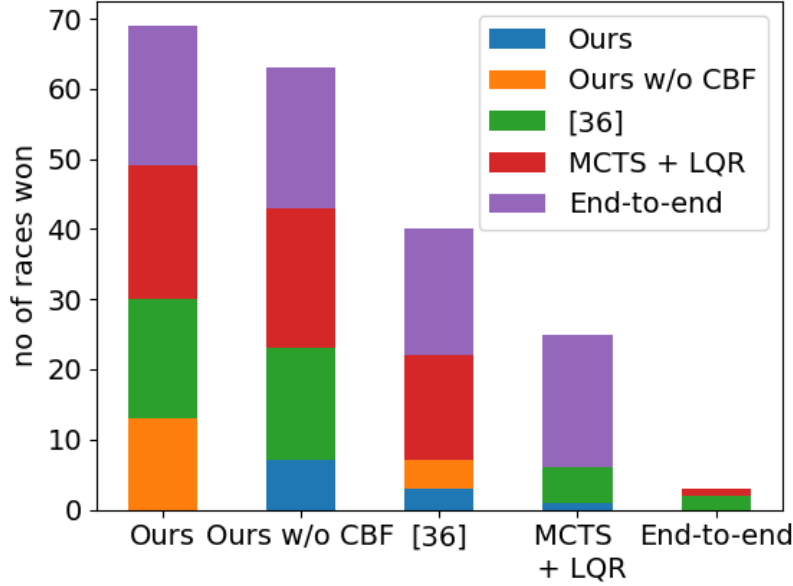
Comparison with baselines

Finally, we conduct races among other baselines for comparison. All races consist of 3 laps, with the car which crosses the finish line first after 3 laps winning the race. In total 20 races are conducted with each pair for comparison, hence 200 races in total for 10 pairs. We compare against the following baselines:

1. Ours: This is with CBF and model-curriculum.
2. Ours w/o CBF: This is with model-curriculum but without RES-DOB-CBF.
3. [Tha+22]: This is without using any curriculum during training. Also, the raceline information is omitted in the input to the policy and value estimation networks.
4. MCTS + LQR: This is to compare with a classical rule-based controller. We design an LQR controller to track the high-level checkpoints planned by the MCTS algorithm. We use the same LQR design as in [Raj12].

5. End-to-end: This is without using any hierarchical controller. The high-level state observations are omitted as input to the policy and value estimation networks described in Section 5.5. Also, the target lane-specific reward term described in Section 5.5 is changed to just a constant reward for passing any checkpoint irrespective of the lane it passes through. Also, no curriculum learning is used for this.

We compare the runs against the following metrics: 1) Number of wins; 2) Average lap time; 3) Average lateral distance from the race line; 4) Number of collisions with walls; 5) Number of collisions with opponents from behind.



	Avg. lap time (in s)	Avg. lat. dist. from raceline	No of collisions	
			Wall	Opp.
Ours	28.8	1.03	250	85
Ours w/o CBF	29.2	1.05	360	92
Ours w/o raceline, w/o curriculum learning [Tha+22]	29.5	1.80	589	78
MCTS + LQR	29.6	0.53	212	64
End-to-end	30.3	1.67	670	56

Figure 5.20: The race win statistics. For example, the first bar is for the comparison between ours with other approaches. We won all 20 races (see the value from 49 to 68 in the purple block) against the end-to-end approach.

Fig.5.20 contains the win statistics for all methods. As can be observed, our approach outperforms all other methods in most races. Especially with races against [Tha+22], we observe that using racing lines in the optimization and curriculum learning is beneficial as

compared to training without them. Fig. 5.20 contains all the other statistics. As can be seen, we achieve better lap times as compared to other methods. Also, we stay much closer to the raceline as compared to [Tha+22] after adding explicit raceline state as input. However, using classical control like LQR is able to achieve better raceline tracking but may not be able to achieve optimal speeds as the RL controller does. Also, clearly, using the hierarchical controller yields better results than the end-to-end approach. In terms of the number of collisions, it suggests that using RES-DOB-CBF filter helps the agent in learning a more robust safe policy, as it has fewer collisions as compared to not using it. Using the classical LQR controller tracks the reference path well with fewer collisions, but is not agile enough. Using an RL-based controller helps in learning an agile controller aimed at winning the race, but leads to slightly more tracking error.

5.6 Case study: Offroad driving

Next, we show how our approach is pretty effective and useful for another scenario which is smooth offroad traversal on a trail avoiding obstacles. We first test on a custom-built simulator in Unity game engine. Off-road navigation is a challenging problem both at the planning level to get a smooth trajectory and at the control level to avoid flipping over, hitting obstacles, or getting stuck at a rough patch. There have been several recent works using classical approaches involving depth map prediction followed by smooth trajectory planning and using a controller to track it. We design an end-to-end reinforcement learning (RL) system for an autonomous vehicle in off-road environments using a custom-designed simulator in the Unity game engine. We warm-start the agent by imitating a rule-based controller and utilize Proximal Policy Optimization (PPO) to improve the policy based on a reward that incorporates Control Barrier Functions (CBF), facilitating the agent’s ability to generalize effectively to real-world scenarios. The training involves agents concurrently undergoing domain-randomized trials in various environments. Videos and additional results: <https://sites.google.com/view/wroom-utd/home>.

We first delve into the different components of this proposed environment. This encompasses the design of the expert controller for executing imitation learning, and the training process for our PPO-based agent, alongside detailed descriptions of the reward functions and CBF constraints that we implement. Additionally, we touch upon the approach employed for distillation, facilitating the agent’s ability to generalize effectively to real-world scenarios.

Simulator design

We use the Unity Game engine [Haa14] to design a novel off-road driving simulator, which we call *Off-road Terrain Simulator* (**OffTerSim**). The open-source code for the environment

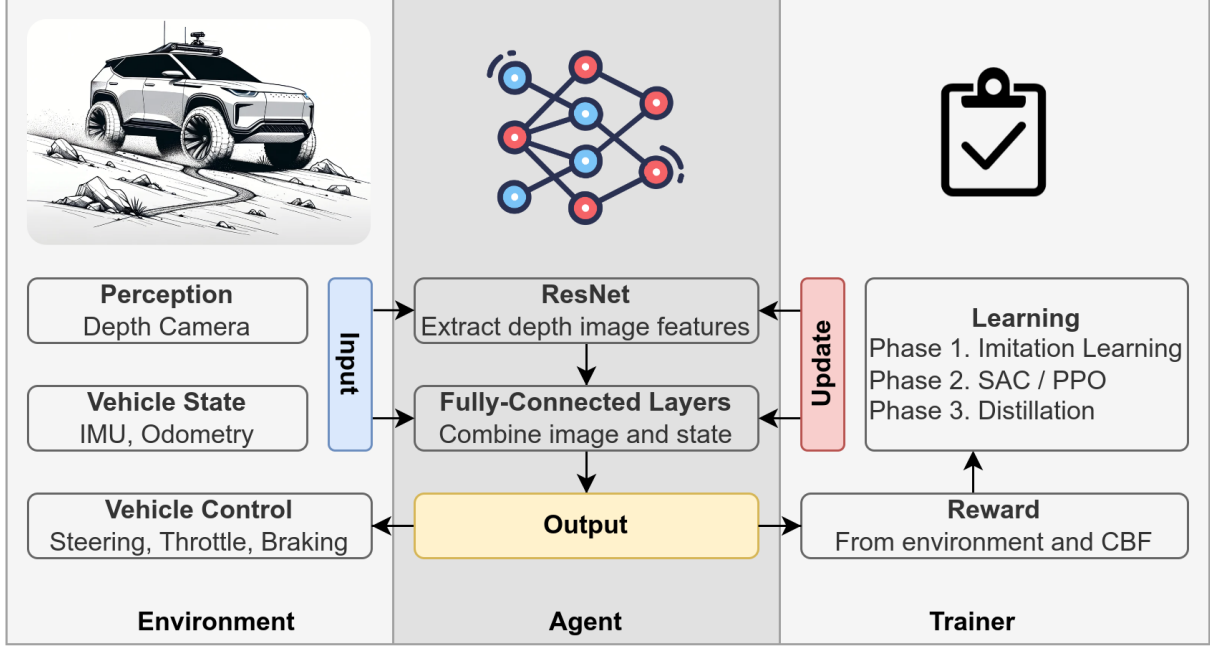


Figure 5.21: Overview of the proposed approach, **WROOM**: The end-to-end RL agent collects depth camera and IMU measurements from the environment and outputs steering, throttle, and braking commands. The reward function evaluates not only the agent’s progress in the environment but also the smoothness and safety of its maneuvers, as reported by the control barrier function (CBF). Imitation learning is utilized to kick-start the agent, followed by Proximal Policy Optimization (PPO) to refine its policy, and finally, policy distillation is employed for real-world deployment.

can be found here:

<https://github.com/dvij542/OffTerSim> We procedurally generate an environment for our simulator aiming to mimic a forest trail environment where the agent can be spawned at the start position and tasked to traverse it while taking a smooth path free of any obstacles. A random terrain is designed to mimic a forest trail. We define the centerline of the trail by a 4th-order equation. Without loss of generality, let us assume our origin to be the starting point of the trail where the agent will be spawned for each episode. We define our trail shape by:

$$y = f(x) = M/2 + bx + cx^2 + dx^3 \quad (5.15)$$

where (x, y) is a point on the trail, and $a, b, c, d \sim \mathcal{U}(-1, 1)$ are randomly chosen coefficients that vary for each episode. The expression $w \sin \mathcal{U}(W_{\min}, W_{\max})$ is the width of the trail, also randomly chosen. We choose incline angles in the x and y directions as $\alpha_x \sim \mathcal{U}(\alpha_{\min}, \alpha_{\max})$



Figure 5.22: Scandots (in purple) as privileged information to the expert controller.



Figure 5.23: Real-world deployment using an RC car.

and $\alpha_y \sim \mathcal{U}(\alpha_{\min}, \alpha_{\max})$ to incline the terrain in x and y directions. We also bound the non-trail region on the left and right of the trail with a steepness defined by β_{left} and $\beta_{\text{right}} \sim \mathcal{U}(-\beta_{\max}, \beta_{\max})$. Based on the value of β , we can end up with hills on both sides, one side or no side to simulate all forms of trails. We also add smooth random noise to simulate unevenness on the terrain which is defined by augmenting each point on the terrain with a height value. The height map is defined by a smooth continuous $2d$ function as follows:

$$d(x, y) = \sum_{i=1..M, j=1..M} \gamma_{ij} \sin\left(\frac{2\pi x}{i} + \frac{2\pi y}{j}\right) \quad (5.16)$$

where γ_{ij} are Fourier coefficients chosen randomly from $\mathcal{U}(0, \gamma_{\max})$, and $M \times M$ is the terrain image size with resolution d . On top of these, we also add Gaussian noise with variance σ_{trail} for in-trail points and $\sigma_{\text{non-trail}}$ for out-of-trail points. $\sigma_{\text{non-trail}} > \sigma_{\text{trail}}$ to make the trail region smoother than the non-trail region. The vehicle model is defined as follows:

$$\begin{aligned} F_{rx} &= K_{\text{throttle}} c_t + K_{\text{brake}} c_b & \alpha_f &= c_\delta - \arctan\left(\frac{v_y + \omega l_f}{v_x}\right) \\ F_{ry} &= D_r \sin(C_r \arctan(B_r \alpha_r)) \\ F_{fy} &= D_f \sin(C_f \arctan(B_f \alpha_f)) & \alpha_r &= \arctan\left(\frac{\omega l_r - v_y}{v_x}\right) \end{aligned} \quad (5.17)$$

where K_{throttle} and K_{brake} are constants that dictate the longitudinal force F_{rx} . l_f and l_r are the forward and rear lengths of the vehicle from the COM. $B_f, C_f, D_f, B_r, C_r, D_r$ are the Pacejka friction coefficients of front and rear tires. α_f, α_r are the front and rear slip angles. ω_z is the angular velocity in the body frame's z axis. v_x, v_y are the body frame's longitudinal and lateral velocities. F_{ry}, F_{fy} are the lateral forces from the front and rear tires perpendicular to the tires. All these parameters are also varied for each episode. Unity's physics engine dictates the motion of a rigid body under the effect of all these forces. Randomly sampled

obstacles of various sizes and shapes are also placed in the terrain uniformly with trees only in non-trail regions. Domain randomization includes randomly sampling all these parameters as discussed to make the agent robust to surface and trail shape unevenness. Some stills from the simulator are depicted in Figure 5.24.

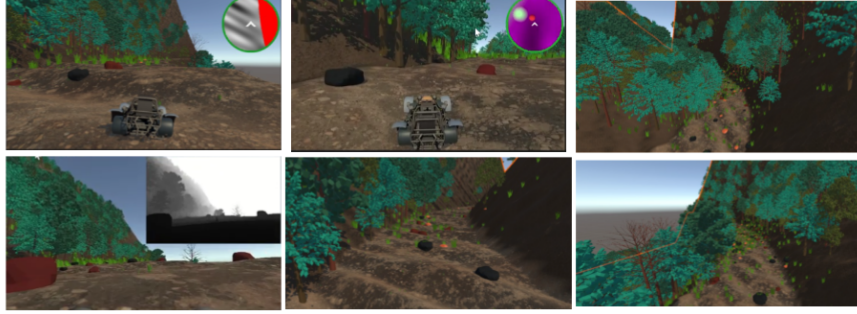


Figure 5.24: Stills from the proposed simulation environment, **OffTerSim**.

Reinforcement Learning (RL)

We employ RL to train a policy that receives environmental inputs and generates actions to guide the agent along a seamless trajectory, avoiding obstacles and adhering to designated trails. This section delves into the specifics of our observation and action spaces, as well as the two different RL algorithms used for agent training.

Observation Space. Our policy model integrates various input variables, including inertial state data from the IMU sensor (acceleration, angular velocity, roll, and pitch), Frenet state [Wer+10] (See Figure 6.2) to track lateral displacement from the center line of the trail, depth images from vehicle-mounted sensors for obstacle and path detection, and scandots [Aga+22] for privileged information on depth values. These inputs facilitate the agent’s inference of critical information such as visible region height, Frenet frame state, lateral displacement, and heading angle from the center line.

Action Space. We test two action space variations for the agent. The first is continuous, offering unrestricted movement in all directions with a continuous throttle. The second variant limits heading directions to n discrete values from -1 to 1 with a continuous throttle control as before. The latter proves simpler for learning, as optimization becomes less challenging. Allowing continuous control in the first variant leads to the agent getting stuck due to the symmetric nature of the local obstacle avoidance problem. This results in the need to learn a discontinuous policy. In contrast, the second variant allows for easier optimization and yields qualitatively cleaner behavior.

Policy Model Architecture. To determine the action for the agent based on the depth image, we employ a policy model featuring a ResNet architecture [He+15] for encoding the image input, which is then followed by several fully connected layers combining information from the depth image with inputs from the scandots and the IMU sensor values.

Safety CBF

We use the CBF as a shield to guide the agent to learn a safe policy similar to [Ema+22]. We observe that the RL agent struggles at the beginning, going off-trail, which debars the agent from learning meaningful behavior later in the episode. To get rid of this, we use the CBF to refine commands from the policy to obey the safety constraints. We use the same track boundary constraints as described in Section 5.5 and use RES-DOB-CBF based filter design as specified in Section 5.3.

This new safe filtered action is fed to the environment for controlling the agent. With this, the RL agent would be able to stay safe while learning, as the CBF shield defined in (5.14) will modify the unsafe controls from the nominal u_{ref} to their corresponding nearest safe commands u if admissible within control limits u_{min} and u_{max} . To inform the policy update to generate safe action altogether and thus not get this override from CBF, we also add a negative reward for constraint violation proportional to the change in command caused by the CBF shield as follows: $R_{\text{constraint}} = k_{\text{constraint}} \|u - u_{\text{ref}}\|^2$.

Reward Design

Our reinforcement learning agent is trained using a reward function comprised of five key terms. These terms are designed to incentivize the agent’s movement along trails, encourage progress, avoid obstacles, and ensure a safe traversal toward the goal. The specific components of the reward function are as follows:

- **Progress Reward:** This term promotes the advancement of the agent along the trail by providing positive rewards for progress made.
- **Smoothness Reward:** We also emphasize the smoothness of the agent’s trajectory by penalizing the magnitudes of pitch and roll of the vehicle.
- **Boundary Reward:** To maintain the agent within the trail boundaries, we penalize movement outside the designated path.
- **Collision Reward:** Heavy penalties are imposed to ensure that the agent actively avoids collisions with obstacles along its path.

Privileged	Discrete Actions	CBF	Pretrain	# col-lisions	Collision time (s)	Progress	Cum. unevenness	# CBF Violations
X	✓	✓	-	91.8	53.7	336.8	3095.14	13.0
✓	✓	X	-	23.9	8.77	514.17	2369.3	-
✓	✓	✓	-	12.4	5.61	588.4	2300.5	3.6
✓	X	X	-	26.7	8.81	489.21	2311.6	-
✓	X	✓	-	19.5	6.97	585.12	2240.6	2.3
X	✓	✓	Dagger	13.5	5.88	569.2	2282.1	4.3
X	✓	X	GAIL	22.7	6.33	531.14	2123.7	-
X	✓	✓	GAIL	15.2	4.68	595.14	1702.2	0

Table 5.2: Quantitative ablation of individual components of **WROOM** using various metrics.

- **CBF Reward:** Any violations of the CBF conditions are penalized to encourage adherence to safety constraints.

Policy distillation

We observe that when directly trying to train an agent end-to-end, it takes a very long time to learn a meaningful policy on the high-dimensional depth image as observation. To address the given challenge, we implement a distillation step. Initially, we train an agent with privileged information, specifically the scandots, following previous works [Aga+22]. We first train an expert controller that would have privileged information for driving through rough terrain. Subsequently, we employ this expert policy trained with privileged information as the teacher and train a student policy network. The student network uses the teacher’s outputs as ground truth observations. It takes depth images from the environment as input and learns to navigate relying solely on IMU values and depth images. This approach facilitates the student network in generalizing to real-world scenarios that it has not been explicitly trained on.

Results

In this section, we highlight the efficacy of the proposed approach, **WROOM**, on both the simulation environment, **OffTerSim**, as well as on the real-world deployment of a real RC car using a set of five quantitative metrics and the training average reward.

Metrics. (1) # collisions are the average number of collisions with the obstacles over 10 runs. (2) Collision time (in seconds) is the total time the car was in a collided state with the obstacle during an episode. (3) Progress is the total longitudinal distance traversed along

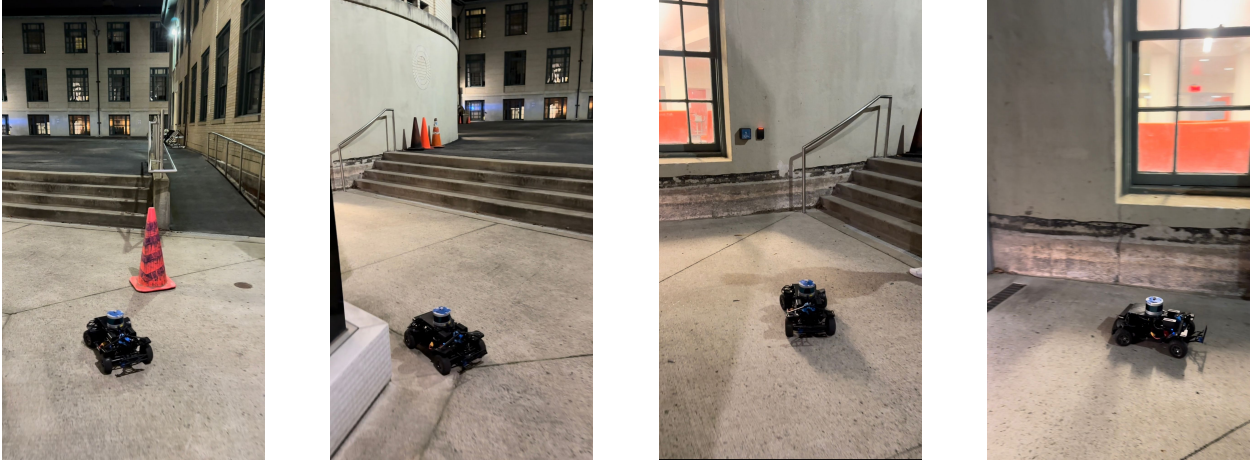


Figure 5.25: Real-world deployment of our proposed approach, **WROOM**.

the trail direction. (4) Cumulative unevenness (measure of rough drive) is the sum of squares of roll and pitch over all time steps in an episode. (5) # CBF Violations is the number of times the CBF constraints were violated in an episode.

Quantitative Results. We experimented with various approaches to train the agent, exploring different observations, network architectures, and output spaces. Comparative analyses of training rewards were conducted across different settings. Initially, we sought to train a policy directly using the privileged scandots information from the simulator (see Figure 5.22). We employ a continuous action space for steering and throttle commands. The training involved 32 agents concurrently undergoing domain-randomized trials in various environments.

However, we encountered challenges in learning obstacle avoidance when using a continuous action space, primarily due to symmetry issues (plot 4 in Fig 5.26). Subsequently, we transitioned to a discrete action space for steering, incorporating only $n = 7$ action commands: from left to right. This adjustment resulted in significantly improved rewards, as illustrated by plot 2 in Fig 5.26. The success of this approach can be attributed to the neural network’s struggle with learning a discontinuous observation-to-action mapping when faced with obstacles directly ahead of the agent, as detailed in [AKR23].

The agent’s direct training using depth-based raw observations proves ineffective due to the high dimensionality of the input. In line with the methodology outlined in [Aga+22], we explored an alternative approach to distill the policy learned from privileged scandots information. Specifically, we employed DAgger-based imitation learning, as described in [RGB10], and improved metrics like # collisions by 85.2% and smoothness in drive-by $\approx 27\%$. Additionally, we experimented with GAIL [HE16], where the expert experience was collected

for each episode, and the expert controller was utilized to label the data. The collected data were then used to train a generator-discriminator model, imposing penalties on RL actions for substantial deviations from expert actions. The loss term scale was gradually reduced as the agent enhanced its alignment with the expert’s actions. Subsequently, other RL reward terms were employed to further refine the agent’s performance. This strategy closely resembles the policy distillation process depicted in [Aga+22]. This led to a CBF-abiding policy with no CBF violations and 82.21% improvement in traversing along the set trail.

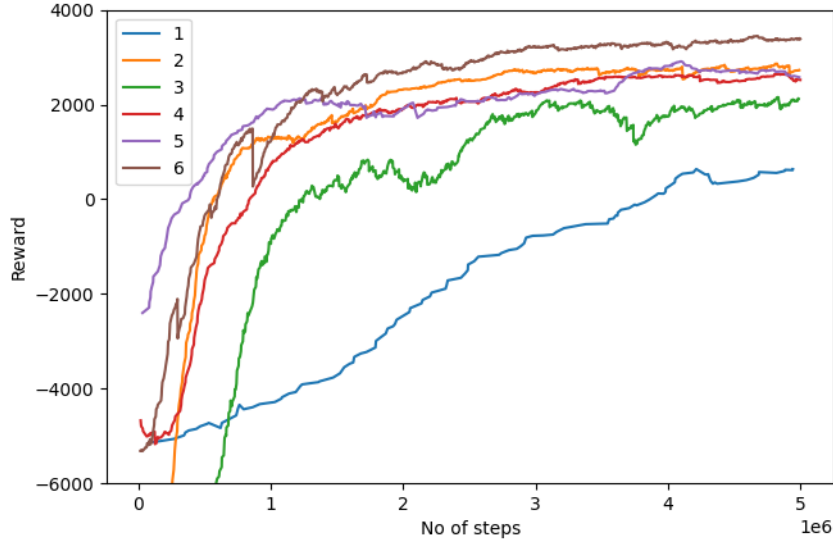


Figure 5.26: Training reward plots highlighting the significance of each component of the proposed approach, **WROOM**.

5.7 Conclusion

In this chapter, we saw how we can formulate a CBF barrier with residual model learning and DOB-based rejection to guard the RL training to enforce safety constraints while not compromising on the optimality of the solution learned. In the next chapter, we will see how a similar framework can be extended to automated imitation learning which we will discuss in the next chapter.

Chapter 6

DECOUPLED SAFETY AND OPTIMALITY FOR END-TO-END AUTONOMOUS RACING WITH AUTOMATED-IMITATION LEARNING

Autonomous car racing is a challenging task, as it requires precise applications of control while the vehicle is operating at cornering speeds. Traditional autonomous pipelines require accurate pre-mapping, localization, and planning, which make the task computationally expensive and environment-dependent. Recent works propose use of imitation and reinforcement learning to train end-to-end deep neural networks and have shown promising results for high-speed racing. However, the end-to-end models may be dangerous to be deployed on real systems, as the neural networks are treated as black-box models devoid of any provable safety guarantees. In this chapter we propose a decoupled approach where an optimal end-to-end controller and a state prediction end-to-end model are learned together, and the predicted state of the vehicle is used to formulate a control barrier function for safeguarding the vehicle to stay within lane boundaries. We validate our algorithm both on a high-fidelity Carla driving simulator and a 1/10-scale RC car on a real track. The evaluation results suggest that using an explicit safety controller helps to learn the task safely with fewer iterations and makes it possible to safely navigate the vehicle on the track along the more challenging racing line.

6.1 Introduction

As we saw in previous chapters, high-speed autonomous racing presents unique challenges that have gained recent attention after significant progress in urban autonomous driving. The following items make autonomous racing more challenging than traditional urban driving: 1) The vehicle operates near its handling limits in a highly non-linear regime [LDM17]. 2) It has to avoid collisions in a rapidly changing environment [KVR19; KLD22]. 3) It needs to have very low response time given its high-speed nature. The classical pipeline splits the task into various traditional components, i.e., pre-mapping, localization, planning, and control [KVR19; JMS21]. However, such a pipeline requires significant onboard computation resources. Also, these methods are prone to environment changes and require pre-mapping of the race track to operate safely and efficiently [JMS21]. In the last chapter, we saw how we can leverage reinforcement learning for learning how to get around the track as fast as possible on the simulator and how we can use CBF safety wrapper around the RL output to safeguard it during training and how it also helps in maximizing its performance. In this

chapter, we will see how we can decouple end-to-end racing networks that take as input the image and inertial state to output commands and a safety end-to-end network that outputs a CBF state that can be used to safeguard the controller commands.

Recent advances in deep learning have brought new approaches to handling autonomous driving/racing based on an end-to-end solution. Based on the powerful representation abilities of deep neural networks, end-to-end methods directly take as input only the raw high-dimensional sensory data (e.g., RGB images and/or Lidar pointclouds) and output low-level control commands (i.e., steering and throttle). Most of these DNNs representing end-to-end control policies are trained with imitation learning (IL), which can efficiently use expert driver demonstrations (i.e., observation-action data pairs) to train efficient models, but suffers from the distribution mismatch problem (a.k.a. covariate shift). Specifically, IL assumes the dataset is independent and identically distributed (i.i.d.) and trains networks to predict actions offline that do not affect future states. However, driving a car is inherently a sequential task, and the actions predicted by the network indeed affect the future. This phenomenon breaks the i.i.d. assumption and can lead to a distribution shift between training and testing. Thus, the policy network may make mistakes on unfamiliar state distributions after deployment. To address this issue, recent approaches use DAgger [RGB11] or online supervision [Pan+17], but they bring further problems of using human demonstrations for accurately labeling the data. To alleviate this, some approaches use supervised control, which takes the complete accurate state of the vehicle as input to output control commands to train the end-to-end model, which then learns to autonomously navigate the track.

However, these approaches have the following limitations: 1) They treat the end-to-end trained model as a black box, the performance is not robust to environmental changes, and they do not provide any safety guarantees. 2) DNNs require a lot of training data whose collection may be inconvenient and practically difficult on a real race track. 3) The end-to-end network does not explicitly take vehicle dynamics into account for its training or for ensuring safety. Hence, we require a **data-efficient approach which can safeguard the end-to-end controller to run on a wide range of environments**. To this end, we propose the use of two different deep neural networks which are learned to output optimal control and the state of the vehicle respectively coupled with use of an IMU sensor and wheel encoder sensors to get the dynamic state of the vehicle. This predicted dynamic state is used to formulate a probabilistic control barrier function (CBF) which makes use of the vehicle dynamics to guarantee safety of the vehicle. The epistemic uncertainty in the DNN predictions is estimated using a dropout-based approach [GG16] to account for uncertainty in unknown observations. Another common drawback in previous works is that most of

them [Wad+21; Pan+17; Cai+21] aim to train the end-to-end controller on the center line, which we believe makes the task relatively easy, as there is ample scope for the vehicle to recover from moving toward the lane boundaries. We, however, aim to teach the end-to-end controller to run on the racing line at operating limits, which turns out to be more challenging, as the racing line is mostly located near the lane boundaries, leaving less scope for the end-to-end controller to make mistakes. Hence, we decouple the optimal end-to-end controller, which aims to track the racing line in order to achieve minimum lap time, and the safety controller, which tries to ensure vehicle safety.

Finally, we demonstrate our framework running on a 1/10-sized RC car with limited computation power and sensors. To summarize, we make the following contributions:

- 1) We propose an uncertainty-aware framework which decouples the safety aspect of the controller and the optimality of the controller. Probabilistic CBFs are used to explicitly impose probabilistic safety constraints on the end-to-end controller.
- 2) We run the racing car on the racing line at near-cornering speeds instead of the center line with constant speed, unlike most recent works.
- 3) We propose use of other dynamic model inputs like lateral velocity and angular velocity of the vehicle as input to the end-to-end DNN architecture.
- 4) We validate the proposed method both in a realistic driving simulation (Carla) and on a real-world 1/10-scale RC car for autonomous racing, showing that it outperforms previous traditional implicit end-to-end learning approaches.

To the best of our knowledge, this is the first work proposing use of CBFs on predicted state from a learned model to safeguard end-to-end models. The rest of the chapter is organized as follows: Sec. 6.2 discusses the proposed imitation learning framework we use. Sec. 6.3 describes the safety controller design we use to safeguard the end-to-end optimal controller. Sec. 6.4 describes the overall training cycle and finally Sec. 4.5 concludes the chapter.

6.2 Imitation learning for autonomous racing

Problem definition

We first mathematically formulate the autonomous racing task. Let us consider a discrete-time continuous-valued optimal control problem. Let $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{O}$ denote the state, action and observation spaces respectively. In our setting, we assume that the observations of the system are the RGB camera image from the front camera and the longitudinal, lateral, and

angular velocities from the wheel encoders and the IMU sensor. The action space consists of the continuous-valued steering and throttle commands.

The goal is to find a policy $\pi : \mathcal{O} \rightarrow \mathcal{A}$ which minimizes the cumulative cost over a finite horizon of length T denoting the whole episode as follows:

$$\min_{\pi} \quad \sum_{i=0..T-1} C_i \quad (6.1)$$

in which $s_t \in \mathcal{S}$, $a_t \in \mathcal{A}$ and ρ_{π} is the distribution of the trajectory $(s_0, o_0, a_0, \dots, a_{T-1})$ under policy π . For the racing scenario, we usually choose function C to encourage that the car stays within the track and completes the lap in minimum time or makes maximum progress along the track safely. Our goal is to learn optimal policy π such that taking $a_t = \pi(o_t)$ sequentially leads to minimizing the cumulative cost $J(\pi)$.

Dagger

Directly optimizing Eq. 6.1 is challenging for high-speed autonomous racing. Also, model-free RL techniques are not very effective in our case, which involves a physical robot, as they are quite sample-inefficient and thus would be quite inconvenient. Using model-based RL may require fewer samples, but it can lead to unstable results, as it is difficult to learn from the RGB image and speed inputs as the state space due to its high dimension.

Considering these limitations, we propose to use Imitation Learning. Given an expert policy π^* , we would like to match our policy π as closely to π^* as possible. For training the policy π , we use an expert controller to get the policy π^* . We assume access to additional sensors like Lidar, GPS, etc. and computation for deriving the expert controller, i.e., the expert may be a computationally intensive optimal controller that relies on GPS and Lidar for accurate state estimation. Our goal is to learn the policy π which performs similarly to π^* , but only takes low-cost camera sensors and IMU sensors as input with limited computation power for it to run during test time. Put formally, the state s^* used by the expert consists of the computed accurate position, orientation of the vehicle on the track using GPS and Lidar, speed inputs from IMU, and wheel encoders; while the state s used by policy π consists of just the camera RGB image(s) and speed inputs. Here, we assume that the localized position and heading angle should map one-to-one to the camera observation, which means that we have a unique image observation for a given state of the vehicle, i.e., there should exist a function F that performs a one-to-one mapping $s^* \rightarrow s$. This would be reasonably possible for a camera attached on the car in a fixed frame of reference w.r.t. the car's body.

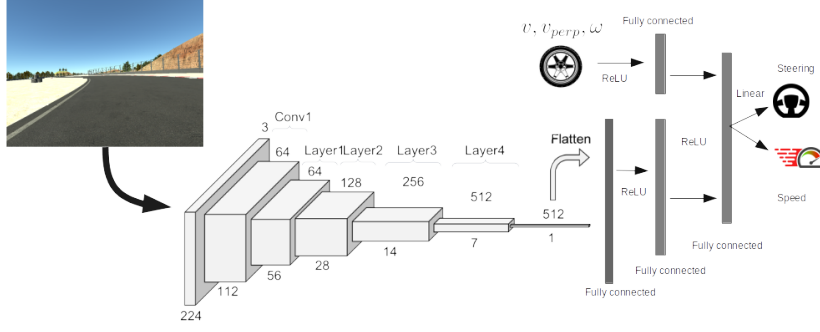


Figure 6.1: Control prediction model architecture

To train the policy π , we use the DAgger algorithm [RGB11]. We start with an empty dataset, $D = \phi$, then we run the vehicle with an optimal controller following policy π^* described later. The optimal policy makes use of ample computation resources and extensive sensors for its training. Also, a camera is attached to the vehicle whose observations are used for creating the dataset for training the policy π . π_i is the trained policy obtained after the i^{th} iteration. The collected data are used to train the policy π_i learned by a DNN described later in Section 6.2. The trained policy π_i is used to run the race car on the track and the collected data are added to the dataset D for re-training for the next iteration.

Network architecture

For the policy π , we use a DNN with network architecture similar to [Pan+17], as shown in Fig. 6.1. It consists of a modified ResNet18 architecture with removal of the last classification layer being fed by the camera RGB image as input to get a one-dimensional feature vector. The lateral and longitudinal speeds and angular velocity from the IMU sensor, and the wheel encoder are appended to this feature vector, and the resultant feature vector is passed through a fully connected layer to get the steering angle and target speed as outputs. The target speed is achieved through a simple PID longitudinal controller that outputs the required throttle.

Expert controller

The expert controller using MPC [KVR19] basically minimizes the cost C formally defined as:

$$\begin{aligned}
\min_U \quad & \sum_{k=0}^{N-1} (\bar{\mathbf{x}}_k - \mathbf{x}_{ref,k})^T Q_k (\bar{\mathbf{x}}_k - \mathbf{x}_{ref,k}) + \bar{\mathbf{u}}_k^T R_k \bar{\mathbf{u}}_k \\
& + (\bar{\mathbf{x}}_N - \mathbf{x}_{ref,N})^T Q_N (\bar{\mathbf{x}}_N - \mathbf{x}_{ref,N}) \\
s.t. \quad & \bar{\mathbf{x}}_{k+1} = A\bar{\mathbf{x}}_k + B\bar{\mathbf{u}}_k \\
& \mathbf{x}_0 = \mathbf{x}_{start} \\
& \mathbf{x}_{k+1} = A_k \mathbf{x}_k + B_k \mathbf{u}_k + C_k
\end{aligned} \tag{6.2}$$

where, A_k , B_k , c_k are the state, control matrices respectively obtained by linearizing the vehicle dynamics at the current state. $\mathbf{x}_{ref,k}$ are the reference states obtained to follow the racing line reference described in Section 2.1 and F is the vehicle model.

6.3 Safety controller

In this section, we describe the design of the safety controller using CBF with predicted state as input.

Control Barrier Functions (CBFs)

CBFs ensure safety by rendering a forward-invariant safe set. We define a continuous and differentiable safety function $h(x) : \mathcal{X} \rightarrow \mathbb{R}$. The superlevel set $\mathcal{C} \in \mathbb{R}^n$ can be named as a safe set. Let the set $\mathcal{C}$ obey

$$\mathcal{C} = \{\mathbf{x} \in \mathcal{X} : h(\mathbf{x}) \geq 0\} \tag{6.3}$$

$$\partial\mathcal{C} = \{\mathbf{x} \in \mathcal{X} : h(\mathbf{x}) = 0\} \tag{6.4}$$

$$\text{Int}(\mathcal{C}) = \{\mathbf{x} \in \mathcal{X} : h(\mathbf{x}) > 0\}. \tag{6.5}$$

A control affine system has the form $\dot{\mathbf{x}} = f(\mathbf{x}) + g(x)\mathbf{u}$, such that

$$\exists u \quad s.t. \quad \dot{h}(\mathbf{x}) \geq -\kappa_h(h(\mathbf{x})) \tag{6.6}$$

where $\kappa_h \in \mathcal{K}$ is particularly chosen as $\kappa_h(a) = \gamma a$ for a constant $\gamma > 0$. The time derivative of h is expressed as

$$\dot{h}(\mathbf{x}) = L_f h(\mathbf{x}) + L_g h(\mathbf{x})\mathbf{u}. \tag{6.7}$$

where $L_f h(\mathbf{x})$ and $L_g h(\mathbf{x})$ represent the Lie derivatives of the system denoted as $\nabla h(\mathbf{x})f(\mathbf{x})$ and $\nabla h(\mathbf{x})g(\mathbf{x})$, respectively. The safety constraint for a CBF is that there exist a $\gamma > 0$ such that

$$\inf_{\mathbf{u} \in \mathcal{U}} (L_f h(\mathbf{x}) + L_g h(\mathbf{x})\mathbf{u}) \geq -\gamma(h(\mathbf{x})) \quad (6.8)$$

for all $\mathbf{x} \in \mathcal{X}$. The solution $\mathbf{u}$ assures that the set $\mathcal{C}$ is a forward invariant, i.e., $x(t \rightarrow \infty) \in \mathcal{C}$. In a practical collision avoidance task, the safety function can be designed as the relative distance between the ego system and a dynamic obstacle.

Let us consider the state of the system as X . For our case, we define the state X as the vehicle's state relative to the Frenet frame as $[x \ \theta \ \omega \ v \ v_{\text{perp}} \ c]$, where x is the closest signed distance from the center line with positive sign if on the right of the center line and negative otherwise. v is the longitudinal velocity, v_{perp} is the lateral velocity, θ is the relative angle wrt the center line's nearest-point tangent, and ω is the angular velocity of the vehicle. $R = \frac{1}{c}$ is the curvature of the center line at the nearest point. The full state representation is shown in Fig. 6.2. The dynamics of the vehicle are defined as follows, where C_f and C_r are the lateral stiffness coefficients of the front and the rear tires respectively, l_f and l_r are the distances of front and rear tires from the COM of the vehicle, m is the mass and I_z is the moment of inertia about the COM of the vehicle.

$$\begin{bmatrix} \dot{x} \\ \dot{\theta} \\ \dot{\omega} \\ \dot{v} \\ \dot{v}_{\text{perp}} \end{bmatrix} = \begin{bmatrix} v \sin \theta + v_{\text{perp}} \cos \theta \\ \omega - vc \\ \frac{-2(l_f^2 C_f + l_r^2 C_r)}{I_z v} \omega + 2 \frac{l_f C_f}{I_z} \delta \\ a_x \\ \frac{-2(C_f + C_r)}{mv} v_{\text{perp}} - 2 \frac{l_f C_f^2 + l_r C_r^2}{I_z v} + 2 \frac{l_f C_f \delta}{I_z} \end{bmatrix} \quad (6.9)$$

We consider lane violation constraints for both left and right lane boundaries (see Eq. 5.13). We assume constant lane widths L throughout the track. We consider using 2^{nd} -order CBF, as we have a system with a relative degree of 2. For Higher-Order CBF (HOCBF) expression derivations, readers are referred to [XB19]. We also constrain the relative angle with respect to the Frenet frame, θ between $-\theta_{\max}$ and $\theta_{\max}$. This helps to stabilize the vehicle at high speeds. To impose this constraint, we use 2^{nd} -order CBF (see Eq. 6.11). We take $\alpha_1 = \alpha_2 = \alpha_3 = \lambda$ as per notations in [XB19].

$$\begin{aligned}
h_{\text{left}}(X, U) &= \frac{L}{2} - x, h_{\text{right}}(X, U) = \frac{L}{2} + x \\
\dot{x} &= v_{\text{perp}} \cos(\theta) + v \sin(\theta) \\
\dot{v}_{\text{perp}} &= \frac{-2(C_f + C_r)}{mv} v_{\text{perp}} - 2 \frac{l_f C_f^2 + l_r C_r^2}{I_z v} + 2 \frac{l_f C_f \delta}{I_z} \\
\dot{\theta} &= \omega - vc \\
\ddot{x} &= \dot{v}_{\text{perp}} \cos(\theta) + a_x \sin(\theta) + (\omega - vc)(-v_{\text{perp}} \sin(\theta) + v \cos(\theta)) \\
\dot{\omega} &= \frac{-2(l_f^2 C_f + l_r^2 C_r)}{I_z v} \omega + \left(2 \frac{l_f C_f}{I_z} \right) \delta
\end{aligned} \tag{6.10}$$

2-order CBF constraint for left boundary is:

$$\begin{aligned}
&\ddot{h}_{\text{left}}(X, U) + 2\lambda \dot{h}_{\text{left}}(X) + \lambda^2 h_{\text{left}}(X) \geq 0 \\
&-\ddot{x} - 2\lambda \dot{x} - \lambda^2 \left(\frac{L}{2} - x \right) \geq 0
\end{aligned}$$

2-order CBF constraint for right boundary is:

$$\begin{aligned}
&\ddot{h}_{\text{right}}(X, U) + 2\lambda \dot{h}_{\text{right}}(X) + \lambda^2 h_{\text{right}}(X) \geq 0 \\
&\ddot{x} + 2\lambda \dot{x} + \lambda^2 \left(\frac{L}{2} + x \right) \geq 0
\end{aligned}$$

$$h_{\theta, \text{right}} = \theta + \theta_{\text{max}}, h_{\theta, \text{left}} = -\theta + \theta_{\text{max}}$$

2-order CBF constraints for relative angle restriction are:

$$\begin{aligned}
&\ddot{h}_{\theta, \text{right}}(X, U) + 2\lambda \dot{h}_{\theta, \text{right}}(X) + \lambda^2 h_{\theta, \text{right}}(X) \geq 0 \\
&\text{Or } \dot{\omega} - a_x c + 2\lambda(\omega - vc) + \lambda^2(\theta + \theta_{\text{max}}) \geq 0 \\
&\text{And } \ddot{h}_{\theta, \text{left}}(X, U) + 2\lambda \dot{h}_{\theta, \text{left}}(X) + \lambda^2 h_{\theta, \text{left}}(X) \geq 0 \\
&\text{Or } -\dot{\omega} + a_x c - 2\lambda(\omega - vc) + \lambda^2(-\theta + \theta_{\text{max}}) \geq 0
\end{aligned} \tag{6.11}$$

Probabilistic Control Barrier Functions

If the state X is uncertain, we can use probabilistic control barrier functions which ensure the satisfaction of the constraints by a confidence margin η [LLD21]

$$\begin{aligned}
&Pr(\ddot{h}_{\text{left}}(X, U) + 2\lambda \dot{h}_{\text{left}}(X) + \lambda^2 h_{\text{left}}(X) \geq 0) \geq \eta \\
&Pr(\ddot{h}_{\text{right}}(X, U) + 2\lambda \dot{h}_{\text{right}}(X) + \lambda^2 h_{\text{right}}(X) \geq 0) \geq \eta \\
&Pr(\ddot{h}_{\theta, \text{left}}(X, U) + 2\lambda \dot{h}_{\theta, \text{left}}(X) + \lambda^2 h_{\theta, \text{left}}(X) \geq 0) \geq \eta \\
&Pr(\ddot{h}_{\theta, \text{right}}(X, U) + 2\lambda \dot{h}_{\theta, \text{right}}(X) + \lambda^2 h_{\theta, \text{right}}(X) \geq 0) \geq \eta
\end{aligned} \tag{6.12}$$

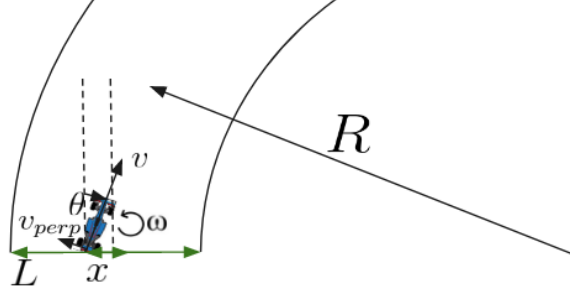


Figure 6.2: Vehicle state representation

If we represent the equation in the form:

$$\begin{aligned}
 a^T U &\leq b \text{ where } a \sim \mathcal{N}(\bar{a}, \sigma_a) \\
 Pr(a^T c \leq b) &= \phi\left(\frac{b - \bar{a}^T c}{\sqrt{c^T \Sigma c}}\right) \\
 Pr(a^T c \leq b) &\Leftrightarrow b - \bar{a}^T c \geq \phi^{-1}(\eta) \|\sigma_a^{1/2} c\|^2
 \end{aligned} \tag{6.13}$$

In order to derive hard constraints on the control u we conduct n rollouts on the function $a(x)$ with different values of x chosen at random and obtain the mean $\bar{a}$ and variance Σ . We fit a gaussian distribution to these, with the values $\bar{a}$ and σ_a obtained as: $\bar{a} = \frac{\sum_{i=0,1..n-1} a(X_i)}{n}$ and $\sigma_a = \sqrt{\frac{\sum_{i=0,1..n-1} (a(X_i) - \bar{a})^2}{n}}$, where X_i is the state obtained from the i^{th} rollout of the state obtained from the trained network and sensor inputs.

Accounting for uncertainty in DNN predictions

As we use a neural network to predict a subset of the state X , i.e., $[x \ \theta \ c]^T$, there are 2 types of uncertainties associated with the prediction from the trained network on a given dataset D [HW21]: aleatoric and epistemic. Aleatoric uncertainty is caused when the dataset is not accurate, i.e., for a single input, there is a distribution of outputs in the dataset and not a deterministic output. In other words, there are data elements with the same input but different outputs. In our case, this uncertainty would exist for the control prediction model if the expert driver strategy is stochastic. Epistemic uncertainty is the uncertainty caused due to the test input being outside the training distribution, i.e., the model is uncertain of the prediction if the input is unseen. In our case, the more prominent uncertainty is the epistemic one, as we use the MPC for expert feedback which is deterministic w.r.t. the given state and expect more uncertainty in the prediction if the input test image is outside the training data distribution. We use a common scheme used in the literature to estimate this epistemic uncertainty [GG16], which uses dropout layers in the network and then takes the mean X_μ and variance X_σ (assuming all state variables to be independent) of the predictions from n different rollouts of the model.

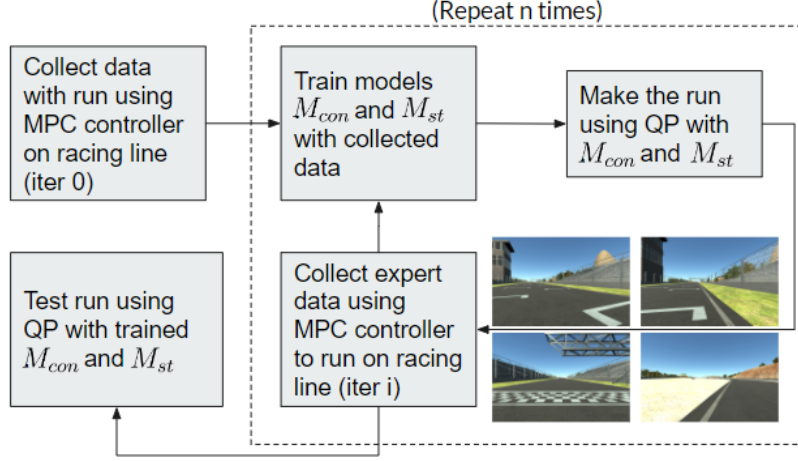


Figure 6.3: Training and testing in Carla. M_{con} : end-to-end control network; M_{st} : end-to-end state network

$$\begin{aligned}
 X_\mu &= \frac{\sum_{i=0 \dots n-1} F_i(X_{im})}{n} \\
 X_\sigma &= \sqrt{\frac{\sum_{i=0 \dots n-1} (F_i(X_{im}) - X_\mu)^2}{n}}
 \end{aligned} \tag{6.14}$$

6.4 Training cycle

Putting both the imitation learning framework and the safety controller together, the process and training and testing are illustrated in Fig. 6.3. 1) An MPC and full access to sensors are leveraged to control the vehicle in the first lap (iter 0) to collect data. 2) The data are used to train the end-to-end control neural network M_{con} (mapping from image to control action) and the end-to-end state neural network M_{st} (mapping from image to state). 3) In the following training, a safe controller using a quadratic programming-CBF (QP-CBF) is used to control the vehicle. The state for the QP-CBF is from the output of M_{st} . The MPC does not control the vehicle but provides control actions as labels for continuously training M_{con} . Meanwhile, M_{st} is trained using the ground truth states from the sensors. 4) In the testing part, we directly use the two neural networks to obtain the QP-CBF formulation to control the vehicle.

The QP-CBF is formulated as follows:

$$\begin{aligned}
& \min_{U, \epsilon_i, \epsilon_2, \epsilon_3, \epsilon_4} (U - U_{ref, \mu})^T U_\sigma^{-1} (U - U_{ref, \mu}) + \sum_{i=0}^3 k_{err}(\epsilon_i^2) \\
& s.t. \quad Pr(\ddot{h}_{left}(X, U) + 2\dot{h}_{left}(X, U) + \lambda^2 h_{left}(X) \geq -\epsilon_1) \geq \eta \\
& \quad Pr(\ddot{h}_{right}(X, U) + 2\lambda \dot{h}_{right}(X) + \lambda^2 h_{right}(X) \geq -\epsilon_2) \geq \eta \\
& \quad Pr(\ddot{h}_{\theta, left}(X, U) + 2\lambda \dot{h}_{\theta, left}(X, U) \\
& \quad \quad + \lambda^2 h_{\theta, left}(X, U) \geq -\epsilon_3) \geq \eta \\
& \quad Pr(\ddot{h}_{\theta, right}(X, U) + 2\lambda \dot{h}_{\theta, right}(X, U) \\
& \quad \quad + \lambda^2 h_{\theta, right}(X, U) \geq -\epsilon_4) \geq \eta
\end{aligned} \tag{6.15}$$

where ϵ_1, ϵ_2 are added to impose soft constraints on the lane violation. Constant k dictates how severely to penalize if the constraints are violated. $U_{ref, \mu}$ and U_σ are the mean and variance obtained from the control prediction obtained by n rollouts on M_{con} . There are two things to note here: 1) When M_{con} is uncertain, the magnitude of U_σ is large, hence when the vehicle is at the boundary, the violation of lane constraints is given more priority, which drives the vehicle away from the boundary; 2) When M_{con} is more certain and there is more uncertainty of the state from M_{st} , more weight is given to following the reference state $U_{ref, \mu}$ obtained from M_{con} .

6.5 Experimental results

We perform experiments on both the Carla simulator and an RC car. A video link to all the experiment recordings is available online¹.

Carla simulator

For the Carla simulator experiments, we use the position obtained from the simulator and pre-labeled track boundaries to obtain the expert MPC controller.

Track

Our test track is shown in Fig. 6.4a We use only about 70% of the track for training and the rest for testing. The optimal racing line is shown in Fig. 6.4a with velocity profile in Fig. 6.4b.

¹Please check the anonymous video from supplementary

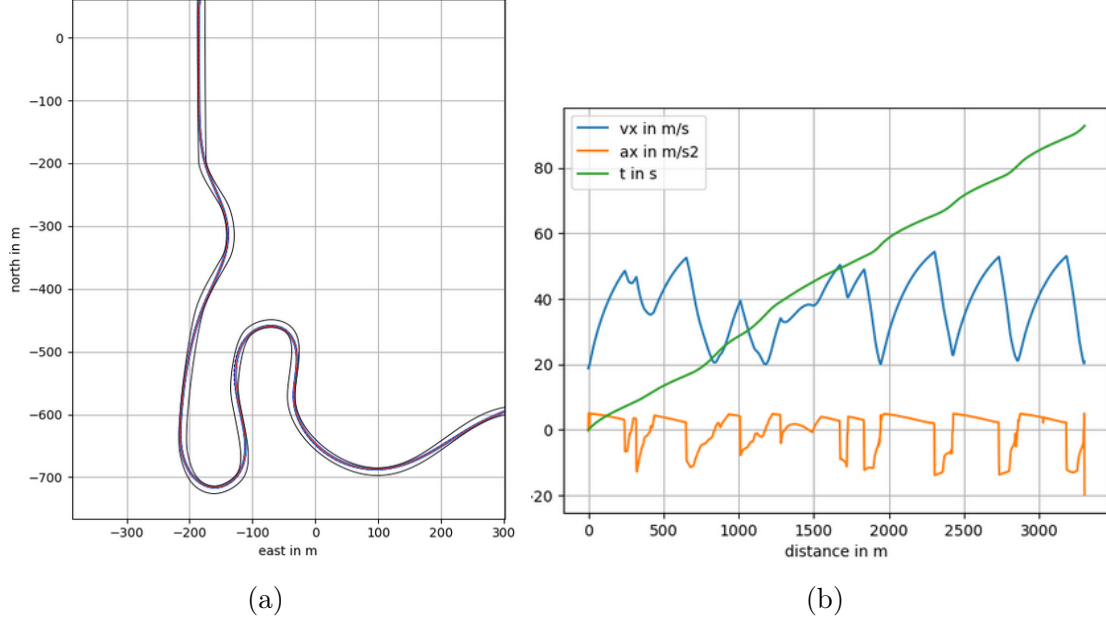


Figure 6.4: Racing line

Vehicle model and Hyperparameter settings

We use a dynamic vehicle model 2.3 for the expert controller and CBF constraint due to high speed. We use $\lambda = 2.5$, $\theta_{max} = \frac{\pi}{4}$, $\eta = 0.95$

Experiments

We demonstrate our results both on the center line and the racing line as reference. First, we train an end-to-end policy to run on the center line of the track. The results with racing line reference without CBF and with CBF are shown in Fig. 6.5a and Fig. 6.5b, respectively. Plots for center line reference are omitted due to lack of space. As expected, as it is more challenging to learn to run with the racing line as reference, without using the safety CBF, the end-to-end policy learns to complete the track in about 11 iterations, whereas using CBF, it completes the track in only 5 iterations and with improved lap times later. The lap times are shown in Table 6.1. Hence, we can expect that using the safety CBF safely drives the vehicle along the track with fewer iterations required for training.

RC car

Finally, we also perform the experiments on a real 1/10-scale RC car which is fitted with Lidar, depth dual camera, IMU and wheel encoders. It has an NVIDIA Jetson TX2 attached to it with 8GB of RAM and 256 core NVIDIA Pascal GPU for onboard computation. There is

Iter	CBF?	Ref : Center line (k=2)		Ref : Racing line (k=4)	
		Lap time (in s)	Mean deviation from ref	Lap time (in s)	Mean deviation from ref
0	×	70.9*	0.05	59.4	0.1
1	×	10.1*	2.94*	5.7*	3.62*
2	×	9.5*	0.96*	4.65*	4.63*
3	×	37.0*	0.91*	11.55*	1.76*
	✓	72.8	3.49	-	-
4	×	54.4*	0.54*	14.1*	1.81
	✓	71.6	1.19	-	-
5	×	68.9*	0.61*	42.0*	3.62*
	✓	70.9	0.54	24.75*	1.49*
6	×	43.4*	1.17*	19.65*	2.08*
	✓	70.9	0.35	60.6	2.17
7	×	71.0	0.39	37.2*	0.65*
	✓	70.8	0.26	59.85	1.386
8	×	70.9	0.35	18.45*	1.99*
	✓	70.9	0.27	59.85	1.01
9	×	-	-	40.35*	0.40*
	✓	-	-	59.7	1.22
10	×	-	-	60.55	1.41
	✓	-	-	59.7	0.741
11	×	-	-	59.8	1.16
	✓	-	-	59.7	0.437

Table 6.1: Statistics for all iterations (Carla simulator). * : Didn't complete the track

a twin ZED camera attached to the vehicle which also outputs depth information. However, we only take the first 3 channels of the input excluding the depth information of the camera. Also, as there are two camera image feeds obtained from the twin camera, we merge both the images to obtain a 6-channel image and pass it as input to the state and control prediction networks.

Track

We perform the experiment on the track shown in Fig. 6.7a. The lane width is about 0.8 m. The racing line and the reference speeds are shown in Fig. 6.7a and Fig. 6.7b, respectively.

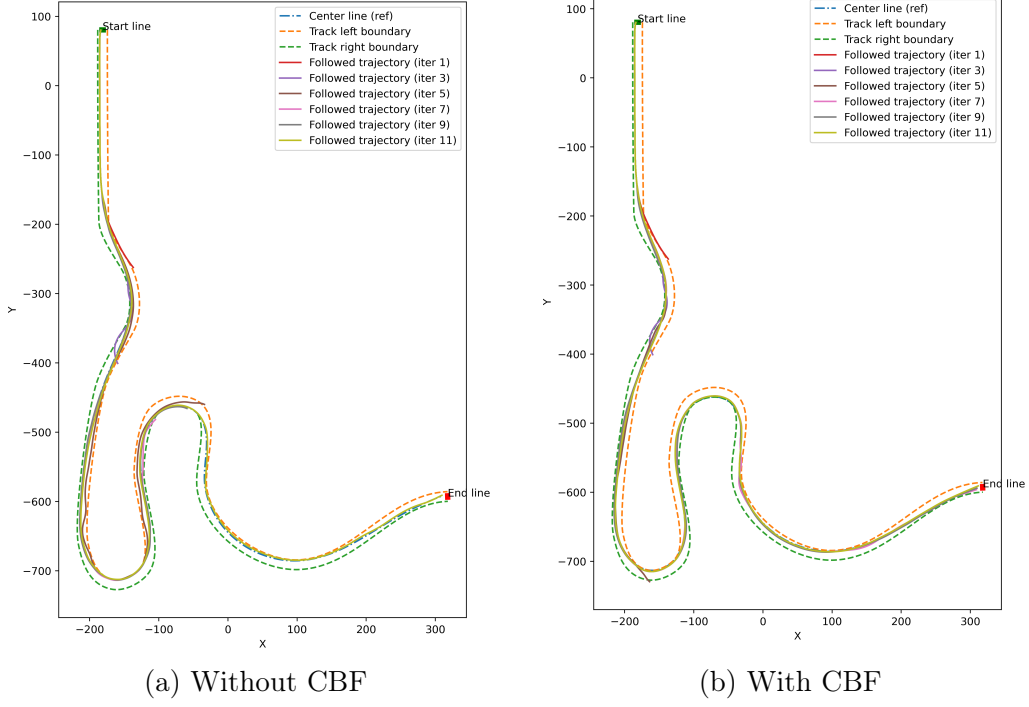


Figure 6.5: Trajectories for all iterations (Carla, Racing line)

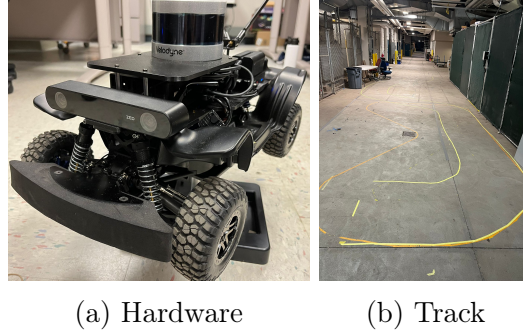


Figure 6.6: RC car setup

Localization for training

We use hector SLAM [Saa+19] for pre-mapping and Monte Carlo localization (MCL) for localization [Del+99].

Vehicle model

We use the kinematic vehicle model 2.2 for the expert controller and CBF due to the low-speed capacity of the vehicle.

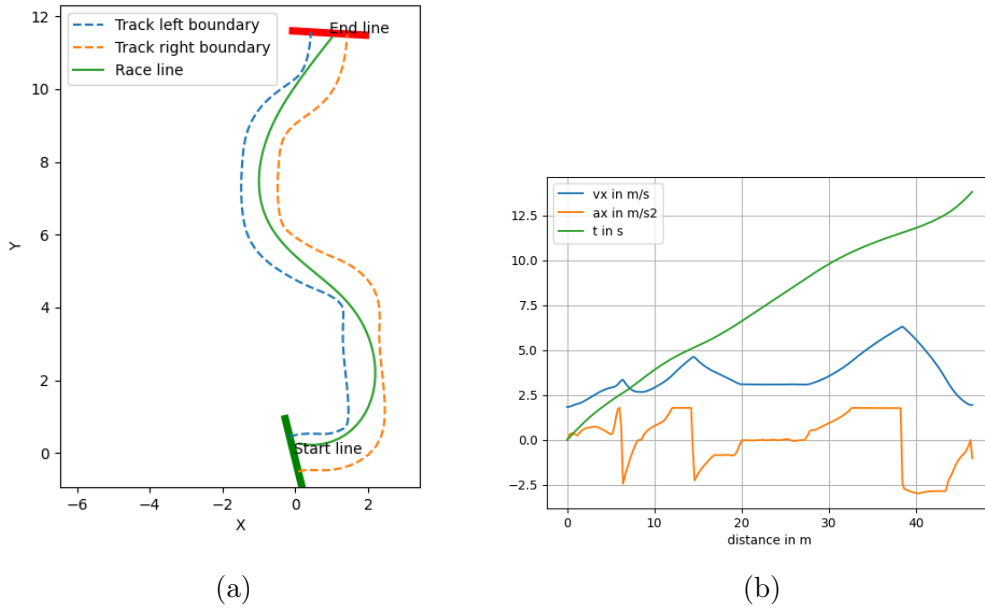


Figure 6.7: Racing line

Iter	CBE	Ref : Center line (k=3)		Ref : Racing line (k=3)	
		Mean deviation from ref	Lap time (in s)	Mean deviation from ref	Lap time (in s)
0	×	0.02	30.98	0.024	20.73
1	×	0.328*	5.13*	0.341*	2.41*
2	×	0.509*	5.78*	0.217*	3.24*
3	×	0.210*	8.77*	0.223*	9.26*
4	×	0.379*	16.56*	0.278*	11.85*
	✓	0.175	33.63	0.334	24.91
5	×	0.340*	18.58*	0.260*	12.31*
	✓	0.092	31.15	0.277	24.80
6	×	0.110	31.23	0.223*	12.9*
	✓	0.048	31.03	0.201	24.91
7	×	0.048	30.96	0.141*	14.92*
	✓	0.044	31.17	0.291	23.69
8	×	-	-	0.159*	15.62*
	✓	-	-	0.223	23.47
9	×	-	-	0.291	23.68
	✓	-	-	0.113	23.28

Table 6.2: Statistics for all iterations (RC car). *: Didn't complete the track

Experiments

Finally, the results using the racing line as the reference are tabulated in Table 6.2 and Figures 6.8a and 6.8b. We observe a similar trend to that of the simulator experiments. The center line reference results are also included in Table 6.2.

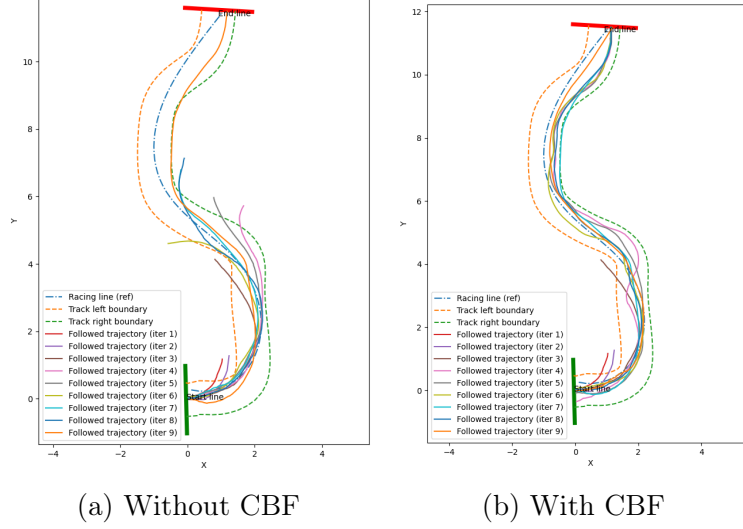


Figure 6.8: Trajectories for all iterations (RC car, Racing line)

Visualizing feature maps

We also visualize what parts of the image maximally contribute to the steering and speed predictions in the trained network of the RC car (See Fig. 6.9) using GradCAM [Sel+17]. GradCAM is a popular tool used in the literature to visualize which part of the image contributes the maximum to the class label predicted by the model. In the case of a regression model, we use the reduction function as the identity of the output value, as discussed in [WY]. This would mean finding the region of the image contributing the maximum to the regression output, which is steering and speed in our case. As can be observed from the figures, most of the contribution is made by the track boundary markings, which means that the model learns to derive its control by looking at the lane markings relative to the RC car. In our case, as we have two images stacked across channels as input to the network, we project the activation maps on both the left and right camera images in Fig. 6.9.

6.6 Conclusion

This chapter presents a novel algorithm that uses end-to-end DNN for controlling a racing vehicle and a safety controller to explicitly safeguard the commands predicted from the end-to-end model. The epistemic uncertainty in the predictions of states from DNN is also

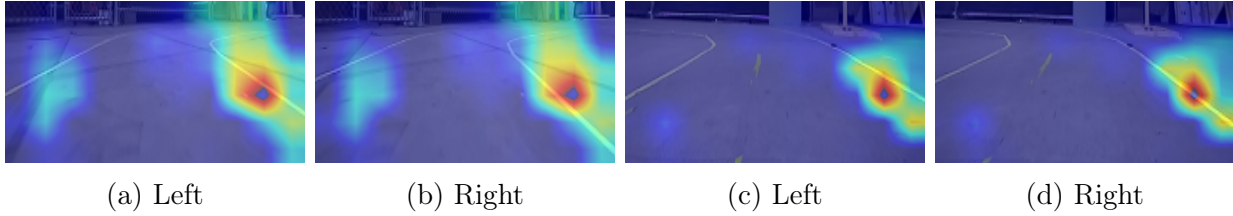


Figure 6.9: GradCAM projections on two example images

considered as part of the framework. In the last 2 chapters (Chapter 5 and Chapter 6), we discussed how a robust and learnable model-based filter can be designed on top of task-learning based controller. In the next chapter, we will move back to model-learning framework in continuation to Chapter 4 where we will see how learning the whole end-to-end vehicle model can be used in conjunction with MPPI to track any trajectory for any car. The same framework can be extended easily to any other system like drones etc as well.

TOWARDS FOUNDATION MODEL FOR FULLY LEARNABLE MODEL-BASED CONTROL FOR AUTONOMOUS RACING

In Chapter 4, we had discussed an efficient model parameter learning scheme assuming a dynamic bicycle model structure. In this chapter, we finally move towards a meta-learning approach for high speed dynamic control. We aim towards an approach that can be generalized to any vehicle parameters including system delay which can be controlled with throttle and steering. We devise a meta learning approach where we train an end-to-end sequence prediction model that can be quickly adapted to any new vehicle model with very few data. The sequence prediction model takes history of states and actions as input and outputs the state derivative which can be integrated to obtain the next state. We describe our approach in detail as follows:- Section 7.1 describes some previous works along this. Section 7.2 describes the methodology in detail for achieving this aforementioned objective of an uncertainty-aware meta-learning approach to get an initial fully end-to-end model that can be quickly adapted to a new vehicle with any model parameters and also to vehicles that do not essentially follow conventional dynamics like towing an object, axle couple wheels, vehicle on uneven terrain that cannot be modelled with a dynamic vehicle model in Eqn 2.3 etc. Section 7.3 describes some results on various simulators and real world RC-car. Section 7.4 concludes our discussion and lays ground for future works.

7.1 Previous works

There have been numerous works on model-learning based control across different domains in recent times. Many of these works aim towards directly learning a policy that can perform a certain task or execute a skill. Some of these works directly deploy the trained policy in simulator with domain randomization to make the policy robust against model parameter errors [Che+21]. Some works employ sim-to-real techniques which modifies the training objective to be able to predict model parameters first from the history and then use the parameters to condition the policy [Kum+21]. These approaches work well if the actual parameters are within the training distribution in the simulator. If the parameters drift significantly which is common in racing due to tire degradation, these approaches fail to generalize to them. To address this challenge, there is a need to also adapt the parameters in a closed loop based on feedback from environment. This is addressed in [Xia+23]. However, these approaches are still trained to work on a specific task. To train on a different task,

large scale training will need to be carried with the same procedure. There is other line of works which aims towards learning model parameters and then use them to simulate the learned model. The learned model is used to formulate optimal control using MPC [LDM15] or MPPI [Wil+16]. Using MPPI has become more popular as it allows highly non-linear models. Most works like [Kum+21] use end-to-end model with no specific model structure while some have a specific structure like [CNB23] and also the previous described work in chapter 5. Complete end-to-end model makes it flexible even if the model used to represent the system dynamics is not accurate as it does not assume any structure. This model is however prone to over-fitting to a specific scenario if trained directly online as online data may not have enough diversity. Towards this end, we propose a meta learning approach to initialize the weights such that it can easily adapt to any environment or task. The task in MPPI can be defined by defining the cost function to achieve that task. We also propose an uncertainty-aware approach to systematically learn to perform well on states it is less confident on as well as be able to prefer assured safe actions rather than focus on states that might offer better optimality but poor confidence hence avoid taking that action

7.2 Methodology

Meta learning

We first formulate our problem into a meta-learning problem. Let us define the state as X and an action u as described in Section 2.2. Our trainable model denoted as f takes the history of states and actions denotes as $H_i = \{(x_{i-n}, u_{i-n}), (x_{i-n+1}, u_{i-n+1}), \dots, (x_i, u_i)\}$ and output the next state, x_{i+1} . The network outputs $y_i = \frac{x_{i+1} - x_i}{\Delta t}$. For each new environment r , the optimal weights representing the environment is denoted as f_r . We use model agnostic meta learning [FAL17] algorithm to learn weights which if initialized with can quickly adapt from online data to a new scenario in K steps. For this we need second order derivatives as shown in algorithm 2. For training, we generate a wide range of scenarios by varying all model parameters including $m, l_f, l_r, I_z, D_{f/r}, C_{f/r}, B_{f/r}, C_{m1}$ and C_{m2}, K_δ and also system delay, data is generated with dynamic model in Section 2.2. A task r denoted by τ_r consists of data sample pair (X_r, Y_r) with size N where $X_r = \{H_1, H_2 \dots H_N\}$ and $Y_r = \{y_1, y_2 \dots y_N\}$

Model architecture

As the next state prediction task is sequential in nature, we use LSTM encoder to exploit the sequential nature of the problem. Then we use a fully connected network to output $\dot{v}_x, \dot{v}_y, \dot{\omega}$. These derivatives are assumed to be constant for Δt . The model architecture is depicted in Figure 7.1. We also append auxiliary information to the encoded history as shown in the

Algorithm 2 Model-Agnostic Meta-Learning (MAML)

Initialize:

θ = initialize parameters

for each episode t **do**

$\tau_i \sim \mathcal{T}$ // Sample a task τ_i from batch $\mathcal{T}$

$\theta_i^1 = \theta$ // Initialize inner loop parameters

for $k = 1$ to K **do**

$\hat{y}_i^k = \text{Model}(\theta_i^k, x_i)$ // Evaluate model on task data

$\ell_i^k = \text{Loss}(\hat{y}_i^k, y_i)$ // Calculate loss

$\theta_i^{k+1} = \theta_i^k - \alpha \nabla_{\theta_i^k} \ell_i^k$ // Update inner loop parameters

end for

$\mathcal{L} = \frac{1}{|\mathcal{T}|} \sum_{\tau_i \in \mathcal{T}} \text{Loss}(\text{Model}(\theta_i^K, X_i), Y_i)$ // Calculate meta-loss

$\theta \leftarrow \theta - \beta \nabla_{\theta} \mathcal{L}$ // Update model parameters

end for

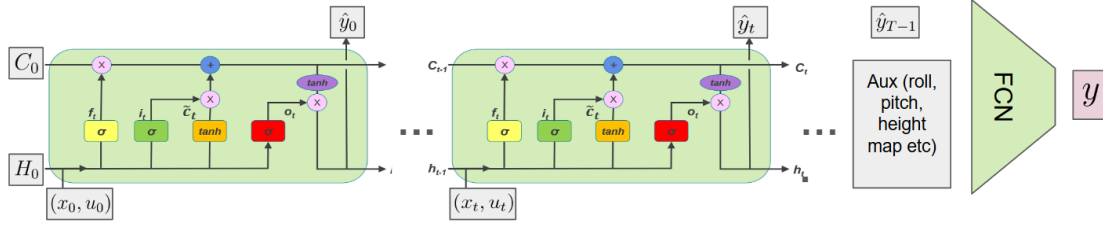


Figure 7.1: Model architecture

figure which may include features like roll, pitch, longitudinal slip etc. that may be relevant for prediction but not included in dynamic state. This is explained in detail in Section 7.3.

Uncertainty estimation

For a safety-critical problem especially with autonomous racing where safety is also closely tied to the performance, it may also be necessary to quantify if the model is confident about the prediction. The uncertainty can also be used to gauge if taking some particular actions will lead the model to learn and gain more information about the actual model. We use ensemble based epistemic uncertainty prediction where we randomly initialize weights $f^{(0)}, f^{(1)}, \dots, f^{(k-1)}$ and train all of them along the same objective. Now when we make a prediction, we take mean of all predictions as μ_y and variance σ_y .

$$\mu_y = \frac{1}{k} \sum_i f^{(i)}(x) \sigma_y = \frac{1}{k} \sum_i (f^{(i)}(x) - \mu_y)^2 \quad (7.1)$$

As all the randomly initialized models are trained towards the same objective, if the input

is within the training distribution, all of the models should output the same y but if input is outside of training distribution, all models will output a different y resulting into larger variance σ_y as they have not been trained on it. Hence we exploit this I.I.D. nature of training to estimate if the input H is outside of training distribution.

Model Predictive Path Integral (MPPI) Control

MPPI is a popular for control when the model is highly non-linear as in our case. Consider our non-linear dynamical system where :-

$$x_{t+1} = f(x_t, u_t) \quad (7.2)$$

where $x_k \in \mathcal{R}^{n_x}$ and $u_k \in \mathcal{R}^{n_u}$ is the control input. With the control horizon length k , we can denote trajectory of states, $\mathcal{X} = [x_0^T, \dots, x_K^T]^T$, the mean control sequence $F = [z_0^T, \dots, z_K^T]^T$ and the injected Gaussian noise $\epsilon = [\epsilon_0^T, \dots, \epsilon_{K-1}^T]$. The control sequence, $\mathbf{u} = \mathbf{v} + \epsilon$. The original MPPI problem aims to solve the following problem:-

$$\min_{\mathbf{v}} J(\mathbf{v}) = \mathbb{E} \left[\sum_{k=0}^{K-1} \left(q(x_k) + \frac{\lambda}{2} v_k^\top \Sigma_\epsilon^{-1} v_k \right) \right] \quad (7.3)$$

where $x(t)$ is the measured current state at time t . Note that the state dependent cost $q(x_k)$ can take any form which can be set accordingly based on the task. In our case, we will mostly look at the task of tracking, hence $q(x_k)$ is defined as follows:-

$$q(x_k) = (x_k - x_{\text{ref},k})^2 \quad (7.4)$$

where $x_{\text{ref},k}$ is the reference state at time $t + k\Delta t$ which needs to be tracked as closely as possible. The MPPI controller is derived by minimizing the KL-divergence between the current controlled trajectory distribution and the optimal distribution as in [Wil+18] to solve the problem. We sample a large no of simulation trajectories using our learned model f . The cost of the m^{th} sample trajectory, S_m is defined as follows:-

$$S_m = \sum_{k=0}^{K-1} q(x_k^m) + \gamma (v_k^m)^\top \Sigma_\epsilon^{-1} (v_k^m + \epsilon_k^m) \quad (3)$$

The MPPI algorithm calculates the optimal control v^+ by taking weighted average of all sample trajectories as follows:-

$$\mathbf{v}^+ = \sum_{m=1}^M \omega_m \mathbf{u}^m / \sum_{m=1}^M \omega_m \omega_m = \exp \left(-\frac{1}{\lambda} (S_m - \beta) \right) \quad (5)$$

where $\beta = \min_{m=1,\dots,M} S_m$ is the smallest cost among m sampled trajectories and it is used to avoid numerical overflow while keeping the same solution. For the next control cycle, the mean control is set to the previous mean but shifted by 1 as suggested in [Yi+24] as we apply the first control command in each iteration in receding horizon fashion. The mean of the last step is set the same as the last command of v^+ which is same as last second command of mean v for next cycle.

Uncertainty-aware MPPI

We do not want to prioritize actions where the model is not confident on. There are multiple approaches to tackle this, one of which is as given in [YZT22] where the worst-case cost trajectory of a sample of trajectories sampled from each action sample with uncertainty is chosen. But to practically implement this, we need to sample additional trajectories to get the worst case sample for each sampled action sequence. We however adopt a cost-efficient approach by adding uncertainty variance as calculated in Section 7.2 of each sample to discourage choosing actions with high uncertainty. The net cost after adding this additional cost is given as follows:-

$$q(x_k) = \gamma \sigma_k + (x_k - x_{\text{ref},k})^2 \quad (7.5)$$

Online adaptation

Once we have the control algorithm in place, what becomes necessary is to adapt the model to new conditions. The pre-trained model that we got from Section 7.2 needs to be fine-tuned on the on-policy data. For this we first need a good enough model to be used that gives some meaningful predictions. For this, we use a simple controller at the beginning to adapt the pre-trained model on the given environment with some few shot data. Then, we switch to use the trained model for the task and continue adapting the model with on-policy data. On a continuously evolving environment like in autonomous racing, fast adaptation is also as necessary. We use simple gradient descent to fine-tune our model where we keep the last b state-action pairs in buffer $B = \{(x_{t-b+1}, u_{t-b+1}), (x_{t-b+2}, u_{t-b+2}) \dots, (x_t, u_t)\}$ and use them to take the gradient descent step as follows:-

$$L = \sum_{i=n}^{i=b} \text{Loss}(\text{Model}(\theta, X_i), Y_i) \theta^+ = \theta - \lambda \frac{\partial L}{\partial \theta} \quad (7.6)$$

The training should be fast as it needs to be carried while the car is running in real-time, hence we need to choose the buffer size B accordingly

7.3 Results

We evaluate our results on a wide range on environment on multiple platforms. We first pre-train our model on numeric sim generated data as per vehicle model defined in Section 2.2. The pre-training data is generated as described in Section 7.2. All videos of results can be viewed from [Link](#).

Numeric sim

We first evaluate our approach on numeric sim. The dynamics for numeric sim are defined as described in Section 2.2. We evaluate our results on an oval trajectory as shown in Figure 7.2a. The oval trajectory is a good platform to test drifting in one direction for any control algorithm. We present comparisons in results across various baselines and ablations as follows:- 1. Using same controller used for data collection at the beginning throughout, the one with incorrect parameters 2. Controller with correct model parameters but assuming 0 delay. This is to show how delay plays an important role in designing an accurate control at limiting speeds which is not very trivial to estimate 3. Using proposed approach without considering uncertainty 4. Using proposed controller with initializing randomly rather than using pre-trained model 5. Using proposed approach with both meta-learning based pre-training and proposed controller

The lateral error plots for all approaches are given in Figures 7.3 the trajectory followed for an instance is given in Figures 7.2. As can be seen in Figure 7.3d where our proposed approach is used without uncertainty estimation, the lateral error suddenly jumps at some points. This is because at those points the model overfits because of only availability of on-policy data with left turns. Due to this at some points, it mistakes right steers as potential commands for high angular velocities and thus ends up making them leading to sudden jumps in lateral errors. If pre-training is not used as seen in Figure 7.3c, it takes a large time to settle for a well-trained model that can track the trajectory while with pre-training, it very rapidly adapts to the new environment and brings down the lateral error immediately as seen in Figure 7.3e. To understand how pre-training helps more clearly, we present a predicted angular velocity plot in Figure 7.4. We compare the angular velocities achieved if

	Avg. lateral error (in m)	Avg. speed (in m/s)	Mean No of laps covered
Without adaptation	1.12	2.12	12.3
Without adaptation correct params	0.94	2.24	14.2
With adaptation random initialization	1.03	2.04	11.6
With adaptation + meta-learning + without uncertainty estimation	0.43	2.18	21.1
With adaptation + meta-learning + with uncertainty estimation	0.35	2.18	23.2

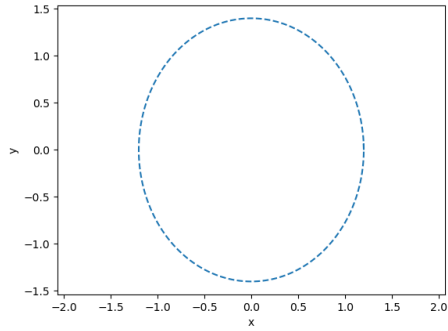
Table 7.1: Numerical results for numeric sim

the same steering thaw was applied at that time step is applied further for 10 more time steps and if negative of that steering was applied for the next 10 time steps. This is to show that if the steering is reversed, if the model has learned to successfully predict negative angular velocity of what is achieved with current steering eventually after sufficient no of time steps 10. If it does, it means it has understood that the steering is symmetrical on both sides. As can be seen in Figure 7.4a, it predicts opposite angular velocities for left and right steering right from the beginning, it just adjust the rate magnitude to match with the observation. While as seen in Figure 7.4b, with random initialization it struggles a lot to learn this detail. This means that with pre-training it has learned that all cars mostly perform symmetrically on left and right steers hence it replicates it's learnings on the other side even if it has seen only left steer on-policy data. It is important to note here that this does not mean it cannot handle steering bias, it can learn to predict that as well from on-policy data as we will see later in experiments.

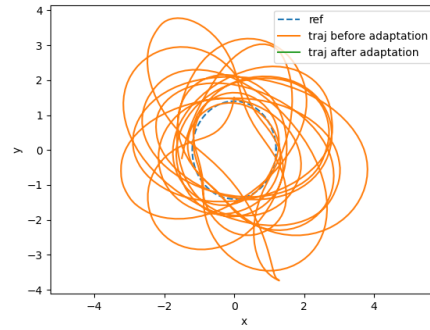
The numerical results are presented in Table 7.1 where we compare our mean lateral error, average speed and no of laps covered. As can be seen, in all metrics, our proposed approach excels

Unity sim

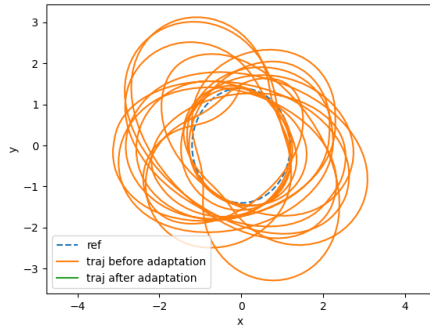
Later, we test our approach on a custom built simulator in Unity game engine on a wide range of scenarios as described below. All videos can be viewed on [Link](#). The comparison in results between our approach from chapter 4, without adaptation and the presented approach



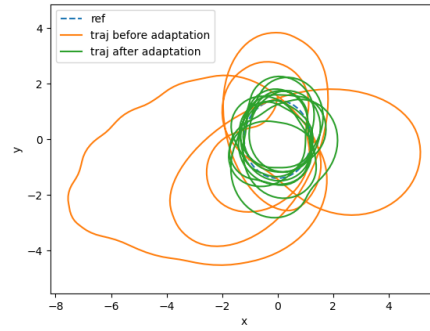
(a) Reference oval trajectory



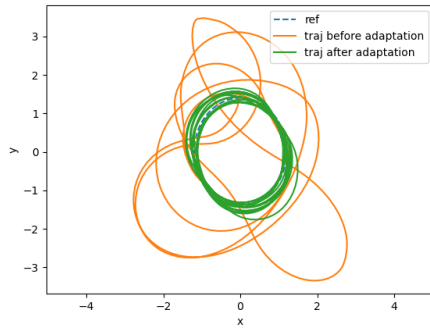
(b) With no adaptation using incorrect model params



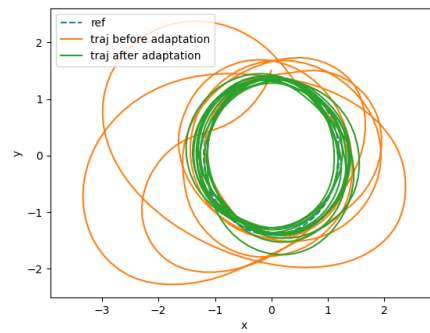
(c) With no adaptation using correct params but assuming no delay



(d) With adaptation starting with random initialization



(e) With adaptation with pre-training initialization no uncertainty estimation



(f) With adaptation with pre-training initialization with uncertainty estimation

Figure 7.2: Trajectory followed for an instance

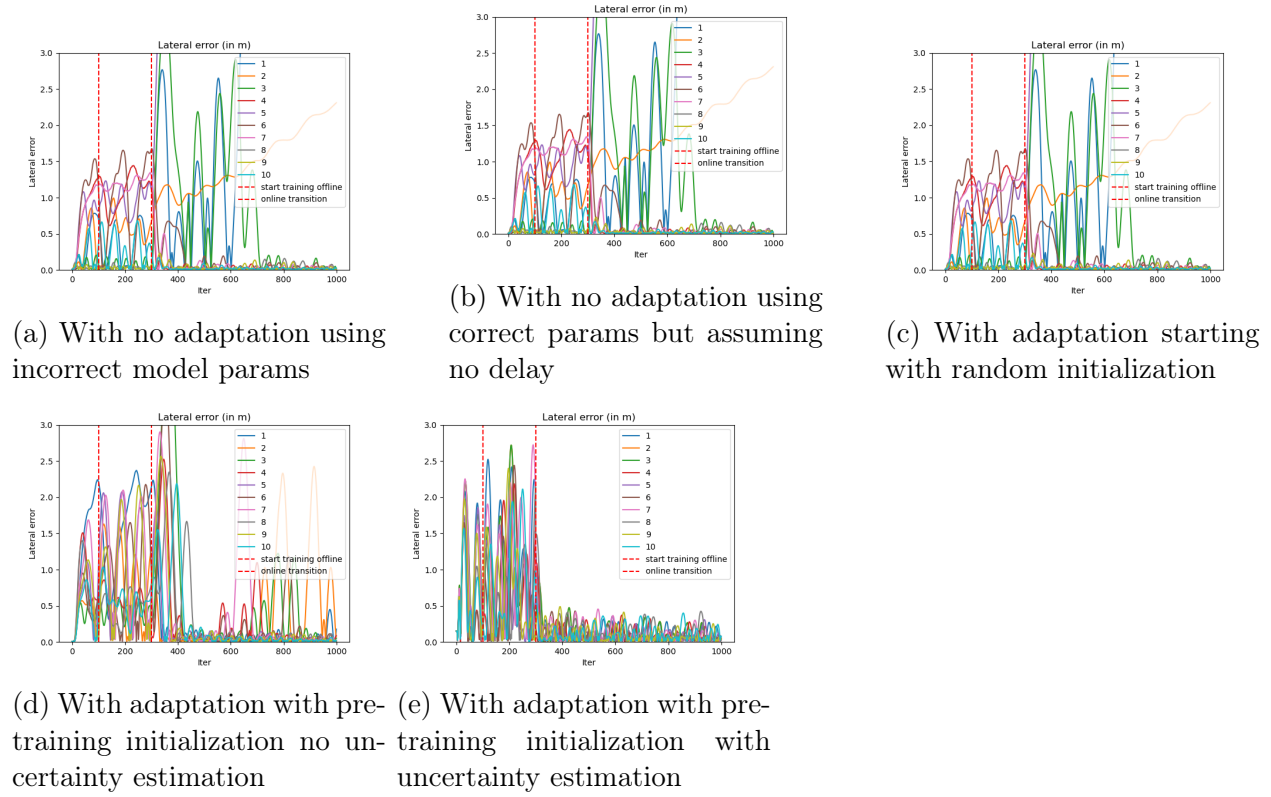
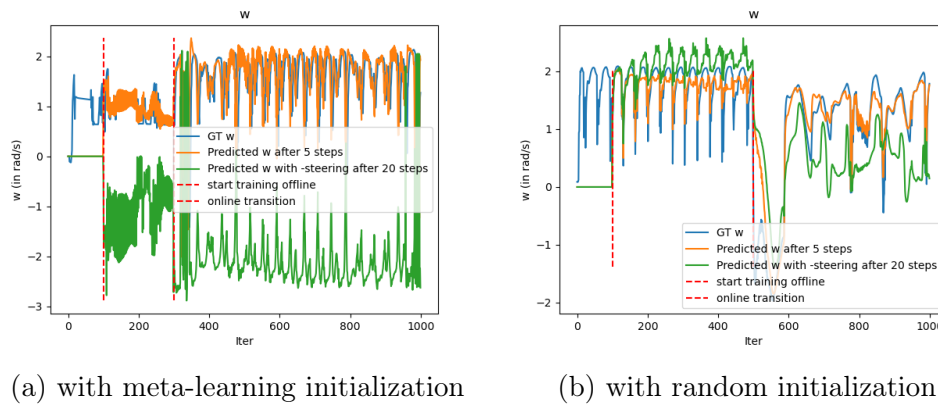


Figure 7.3: Lateral errors

Figure 7.4: Predicted ω

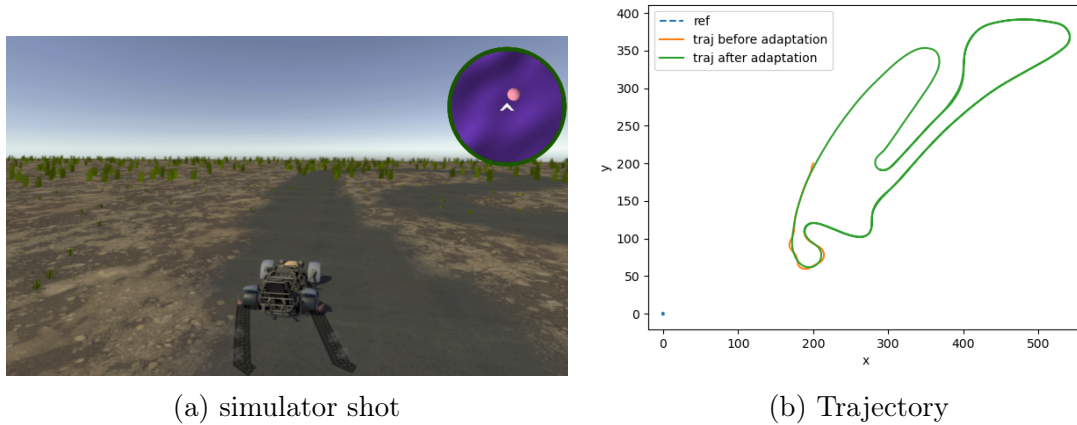


Figure 7.5: Circuit track

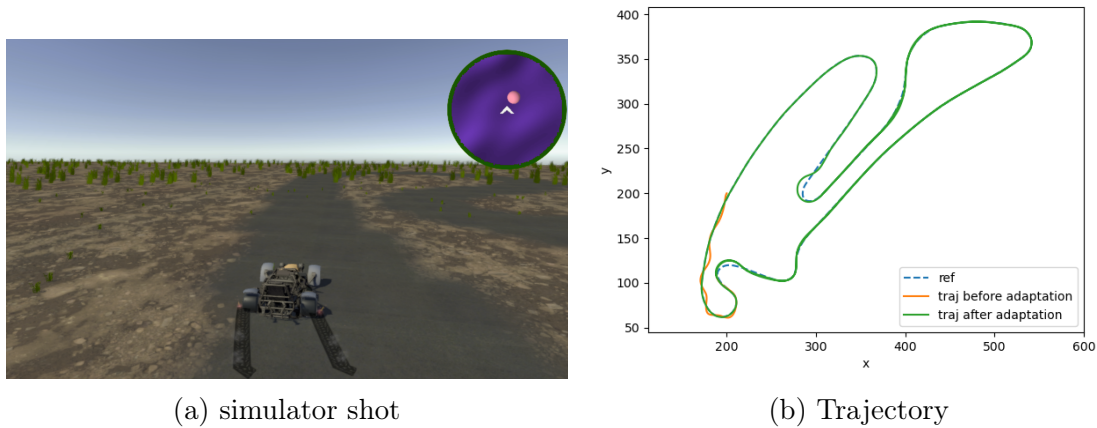


Figure 7.6: Steer bias

in this paper are given in Table 7.2.

1. Circuit track: We test our approach against an autonomous racing scenario where it has to track a trajectory at it's limits on a circuit track which involves coordination between longitudinal and lateral control to successfully execute a turn.
2. Steer bias: We add a left steer bias of 0.1 to the earlier scenario and see if the approach can handle such a scenario.
3. Decaying friction: We decay the coefficient on friction from 1 to 0.7 to test if the model can adapt to it's change
4. Go-kart: We then change the car to go-kart where the rear wheels are coupled by an axle constraining them to move at the same wheel speeds. This results into an opposing torque which ultimately leads to a much different dynamics form than in Section 2.2.

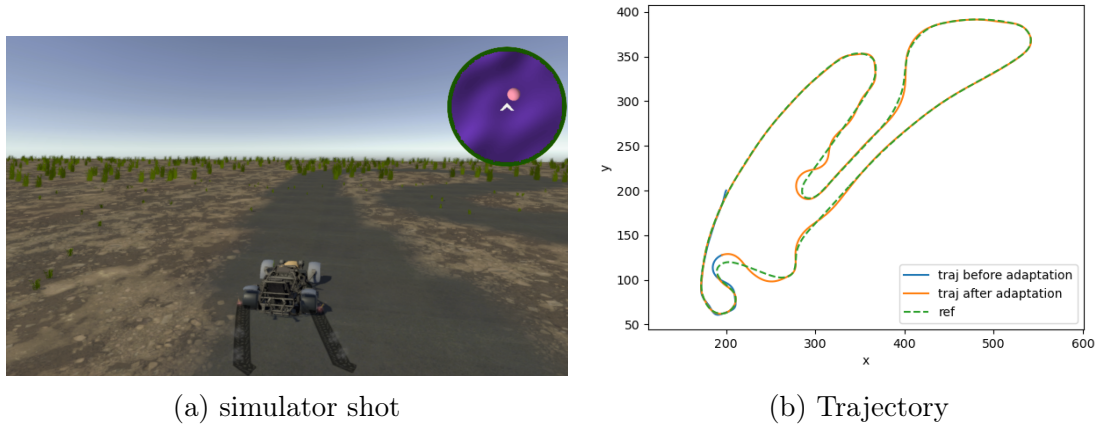


Figure 7.7: Decaying friction

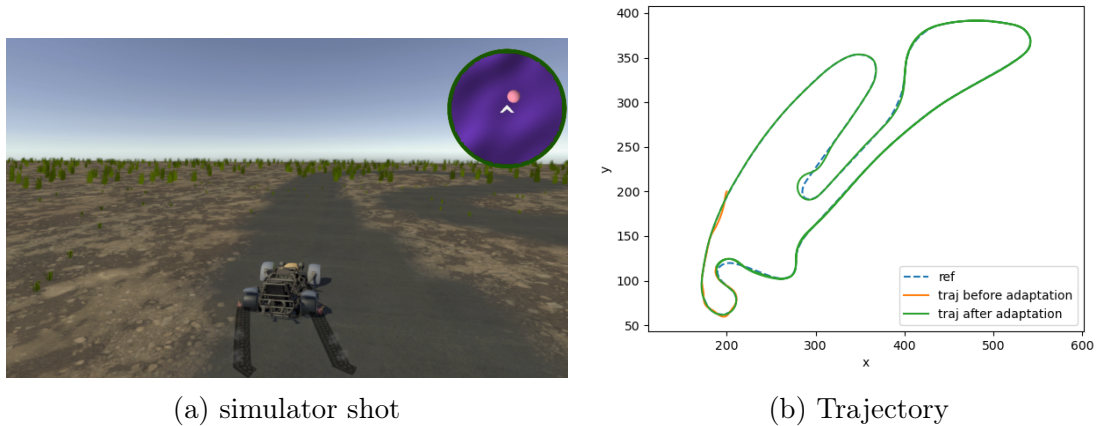
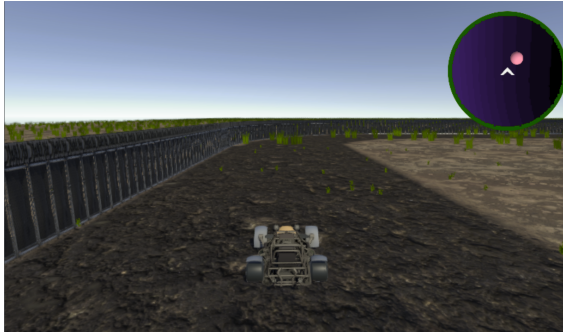
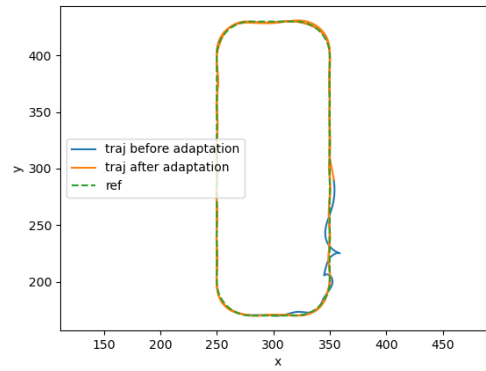


Figure 7.8: Go-kart

5. Mud: We then test the approach for a car moving on mud. Mud has very low grip due to which there is a considerable amount of longitudinal slip. This is ignored in Section 2.2. It is very challenging to perform in this environment as it will need to learn to skid in control
6. Mars: We then test it on a different gravity field. It becomes particularly challenging as it reduces traction. This is not equivalent to just reducing friction coefficient with earth's gravitational field as the moment of inertial to effective mass ratio changes on different gravitational field leading to a different form of controlled skidding that needs to be executed
7. Grass: We then test on a bit uneven terrain to give it a grassy essence. This test would ensure that the model training is robust model noises caused by uneven terrain which cannot be modelled.
8. Banked-oval track: Then we finally test on a track with bank angle. Moving against an

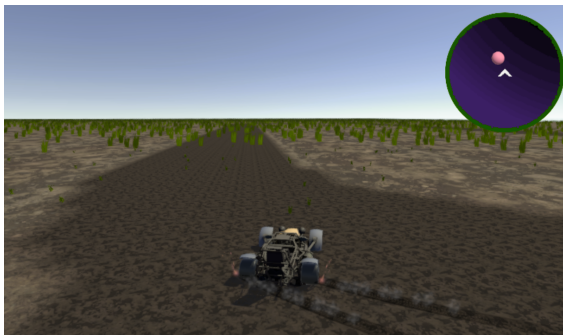


(a) simulator shot

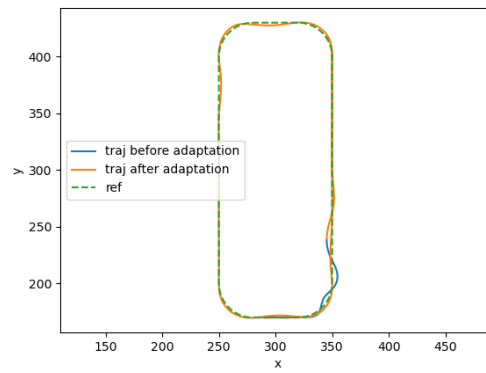


(b) Trajectory

Figure 7.9: Mud



(a) simulator shot

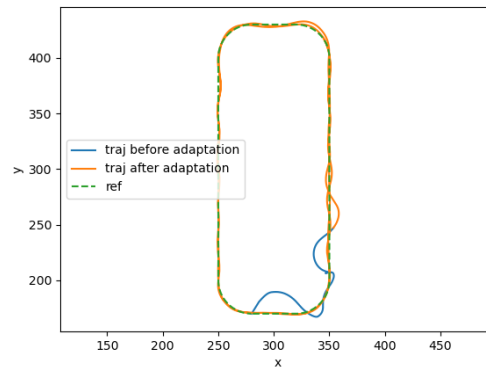


(b) Trajectory

Figure 7.10: Mars



(a) simulator shot



(b) Trajectory

Figure 7.11: Grass

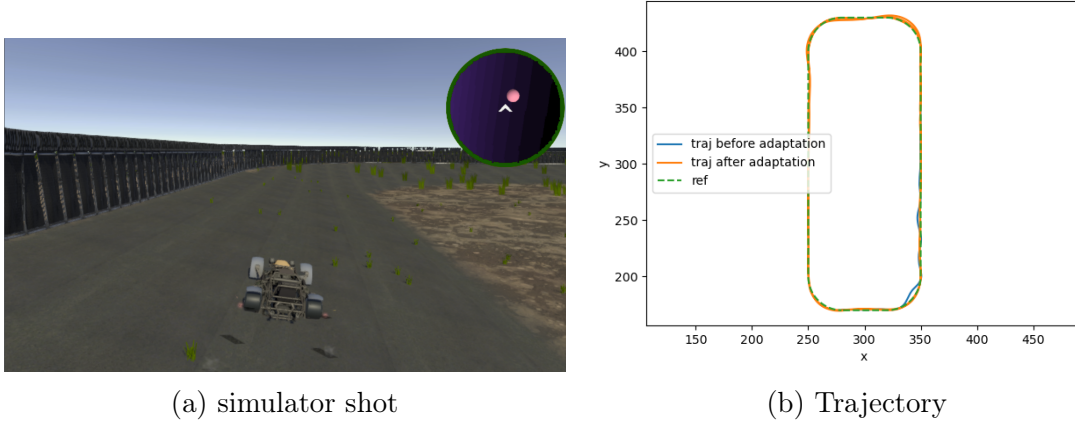


Figure 7.12: Banked-oval track

	Without adaptation	With model parameter adaptation (Work 2)	With full model adaptation (ours)
Circuit	1.31/-	0.12/137.2	0.14/140.1
Circuit with go-kart	0.94/-	0.23/154.7	0.16/143.6
Circuit with steer bias	1.62/-	0.38/152.3	0.16/142.9
Circuit with tire degradation	1.99/-	0.13/148.9	0.15/150.2
Mud	1.43/183.8	0.89/75.4	0.42/44.7
Mars	1.36/-	0.32/66.5	0.38/70.1
Grass	1.29/166.5	0.29/42.3	0.36/45.3
Banked-oval	1.22/186.8	0.53/52.9	0.35/43.1
Front-driven	1.15/-	0.14/137.1	0.16/139.8

Table 7.2: [Lateral error (in m) / Lap time (in s)] Results table for Unity sim experiments

inclined track is particularly challenging to learn as the vehicle needs to move at an oblique angle towards the slope in order to be able to move straight. Our approach is able to learn that to be able to stay at the reference and move at an oblique angle

Hardware platform

We finally also test our approach on hardware RC car platform same as what we used under Section 6.5. We test on an oval track similar in size to what we tested for numeric sim in Section 4.4. We test on different configurations including towing an object, attaching tape on the wheel to alter friction, increasing weight of the car by attaching lidar and changing surface to mat. Trajectories followed for all of them are in Figure 7.13.

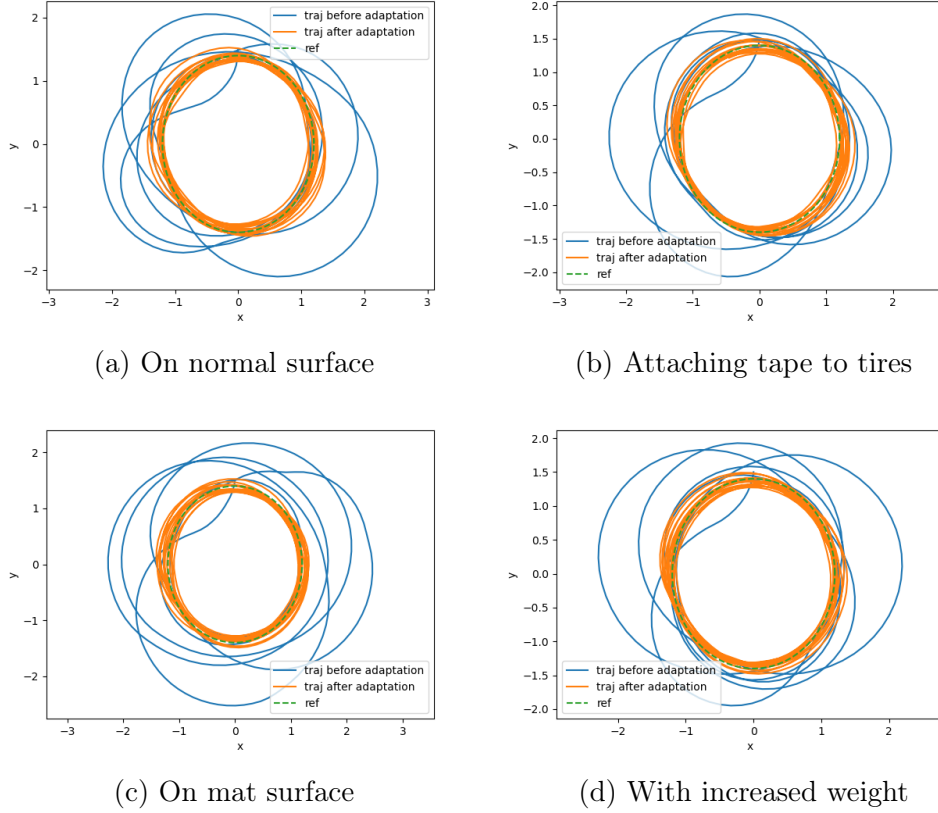


Figure 7.13: Trajectory followed by RC car with vicon

7.4 Conclusion

In this last chapter, we saw how using meta-learning can be used for end-to-end trainable LSTM based vehicle model to allow fast adaptation to any new scenario and changes within the scenario online. We saw how using epistemic uncertainty estimation and formulation uncertainty -aware cost function for MPPPI further improves the tracking performance. The same structure can be extended to other robots like quadrupeds, drones, manipulator etc. as well

CONCLUSION AND FUTURE WORK

We presented work along two directions. The first is task-free model learning. We first formally guaranteed if the vehicle model is nearly known with some bounded model error and system noise and extended robust-tube MPC to also compensate for system delays. In Chapter 4 we learn the lateral friction model and aerodynamic parameters from online data, which can be used to learn the model as well as quickly adapt it to changes in the environment. In Chapter 7 we use meta-learning to pre-train an end-to-end LSTM-based vehicle model making no assumptions about the model. This allows quick adaptation to any unseen scenario, demonstrating similar performance to Chapter 4 in scenarios that Chapter 4 can handle and the ability to handle scenarios that Chapter 4 fundamentally cannot. There are still some failure cases for Chapter 7: it can get stuck at a place where the model is not confident about all actions at a state. Over those states it continues to take only one action which it gets confident about and hence never takes other actions, as they incur more cost due to more uncertainty, and there is no notion of exploration to avoid getting stuck in those states. Potential future directions include using a ϵ -greedy solution where an action with high variance is chosen with ϵ probability to avoid getting stuck. Also the framework described in Chapter 7 only considers aleatoric uncertainty. Epistemic uncertainty arising from sensor noises is not considered as it assumes perfect state. It turns out to be somewhat robust to noises but does not explicitly reason about them. Interesting future direction would be to predict a gaussian distribution of state derivatives and not just a deterministic value and train on mean likelihood estimation loss as seen in [VS22]. Also, it maybe a good idea to extend the idea to enforce hard safety constraints with a similar framework as seen in Chapter 3. Also, it would be interesting to see how the method scales to higher degree of freedom systems such as manipulator, quadraped etc.

The second line of work was along task-learning model-free controllers. We showed how we can use model-based CBF to boost learning as well as reduce the number of constraint violations while learning for reinforcement learning and automated imitation learning-based controllers. Currently the safety filter ignores non-continuous state noises arising from sensor errors which is completely random. Making the filter more robust based on the noise estimates is an interesting future direction to explore. Also, currently in Chapter 5, we assumed feasible control barrier functions but the feasibility of CBFs depend on the vehicle model. There have been some recent works which iteratively update the CBF parameters based on

the environment model to make it feasible [Che+24]. It would be interesting to incorporate that into the framework to build a more robust adaptable filter. Also both works assume no sensor noise. Inculcating them would also be an interesting future direction. Finally, testing these works on more complex systems like quadruped, manipulators etc. for various safety critical tasks would be an interesting extension to see if the method works for higher DOF systems

BIBLIOGRAPHY

- [AA19] Joshua Achiam and Dario Amodei. “Benchmarking Safe Exploration in Deep Reinforcement Learning”. In: 2019.
- [Aga+22] Ananye Agarwal et al. *Legged Locomotion in Challenging Terrains using Ego-centric Vision*. 2022. arXiv: 2211.07638.
- [AK18] Manuel Acosta and Stratis Kanarachos. “Tire lateral force estimation and grip potential identification using Neural Networks, Extended Kalman Filter, and Recursive Least Squares”. In: *Neural Computing and Applications* 30 (2018), pp. 3445–3465.
- [AKJ21] Imoleayo Abel, Miroslav Krstić, and Mrdjan Janković. “Safety-Critical Control of Systems with Time-Varying Input Delay”. In: *IFAC-PapersOnLine* 54.18 (2021), pp. 169–174.
- [AKR23] Khawla Saif Almazrouei, Ibrahim Kamel, and Tamer Rabie. “Dynamic Obstacle Avoidance and Path Planning through Reinforcement Learning”. In: *Applied Sciences* (2023). URL: <https://api.semanticscholar.org/CorpusID:259908264>.
- [Ame+19] Aaron D Ames et al. “Control barrier functions: Theory and applications”. In: *2019 18th European Control Conference*. IEEE. 2019, pp. 3420–3431.
- [And+18] Joel A E Andersson et al. “CasADi – A software framework for nonlinear optimization and optimal control”. In: *Mathematical Programming Computation* (2018).
- [Bao+23] HanQiu Bao et al. “Moment-Based Model Predictive Control of Autonomous Systems”. In: *IEEE Transactions on Intelligent Vehicles* 8.4 (2023), pp. 2939–2953. DOI: 10.1109/TIV.2023.3238023.
- [Bru+21] Lukas Brunke et al. “Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning”. In: *ArXiv abs/2108.06266* (2021).
- [BTI23] Andrea Bonomi, Evelyn Turri, and Giovanni Iacca. “Evolutionary F1 Race Strategy”. In: *Proceedings of the Companion Conference on Genetic and Evolutionary Computation* (2023). URL: <https://api.semanticscholar.org/CorpusID:260119494>.
- [Cai+21] Peide Cai et al. “Vision-Based Autonomous Car Racing Using Deep Imitative Reinforcement Learning”. In: *IEEE Robotics and Automation Letters* 6 (2021), pp. 7262–7269.
- [CDS19] Arthur Claviere, Souradeep Dutta, and Sriram Sankaranarayanan. “Trajectory tracking control for robotic vehicles using counterexample guided training of neural networks”. In: *Proceedings of the International Conference on Automated Planning and Scheduling*. Vol. 29. 2019, pp. 680–688.

- [Che+16] Wen-Hua Chen et al. “Disturbance-Observer-Based Control and Related Methods—An Overview”. In: *IEEE Transactions on Industrial Electronics* 63 (2016), pp. 1083–1095.
- [Che+21] Xiaoyu Chen et al. “Understanding Domain Randomization for Sim-to-real Transfer”. In: *ArXiv* abs/2110.03239 (2021). URL: <https://api.semanticscholar.org/CorpusID:242784472>.
- [Che+24] Rui Chen et al. “Real-time Safety Index Adaptation for Parameter-varying Systems via Determinant Gradient Ascend”. In: *ArXiv* abs/2403.14968 (2024). URL: <https://api.semanticscholar.org/CorpusID:268666977>.
- [CNB23] John Chrosniak, Jingyun Ning, and Madhur Behl. “Deep Dynamics: Vehicle Dynamics Modeling with a Physics-Informed Neural Network for Autonomous Racing”. In: *ArXiv* abs/2312.04374 (2023). URL: <https://api.semanticscholar.org/CorpusID:266044227>.
- [Cor+11] Patricio Cortes et al. “Delay compensation in model predictive current control of a three-phase inverter”. In: *IEEE Transactions on Industrial Electronics* 59 (2011), pp. 1323–1325.
- [CZH22] Yikun Cheng, Pan Zhao, and Naira Hovakimyan. “Safe and Efficient Reinforcement Learning using Disturbance-Observer-Based Control Barrier Functions”. In: *Conference on Learning for Dynamics & Control*. 2022.
- [Del+99] Frank Dellaert et al. “Monte Carlo localization for mobile robots”. In: *Proceedings 1999 IEEE International Conference on Robotics and Automation* 2 (1999), 1322–1328 vol.2.
- [Dje+23] Franck Djeumou et al. “Autonomous Drifting with 3 Minutes of Data via Learned Tire Models”. In: *2023 IEEE International Conference on Robotics and Automation*. 2023, pp. 968–974.
- [Dos+17a] Alexey Dosovitskiy et al. “CARLA: An Open Urban Driving Simulator”. In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, pp. 1–16.
- [Dos+17b] Alexey Dosovitskiy et al. “CARLA: An open urban driving simulator”. In: *Conference on Robot Learning*. PMLR. 2017, pp. 1–16.
- [Ema+22] Yousef Emam et al. “Safe reinforcement learning using robust control barrier functions”. In: *IEEE Robotics and Automation Letters* 99 (2022), pp. 1–8.
- [FAL17] Chelsea Finn, P. Abbeel, and Sergey Levine. “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks”. In: *International Conference on Machine Learning*. 2017. URL: <https://api.semanticscholar.org/CorpusID:6719686>.
- [Fia54] Ernst Fiala. “Seitenkräften am rollenden Luftreifen”. In: *VdI* 96 (1954), pp. 973–979.

- [FL21] Alexander Heilmeier Fabian Christ Alexander Wischnewski and Boris Lohmann. “Time-optimal trajectory planning for a race car considering variable tyre-road friction coefficients”. In: *Vehicle System Dynamics* 59.4 (2021), pp. 588–612. DOI: 10.1080/00423114.2019.1704804. eprint: <https://doi.org/10.1080/00423114.2019.1704804>. URL: <https://doi.org/10.1080/00423114.2019.1704804>.
- [Gao+14] Yiqi Gao et al. “A tube-based robust nonlinear predictive control approach to semiautonomous ground vehicles”. In: *Vehicle System Dynamics* 52.6 (2014), pp. 802–823.
- [GF15] Javier García and Fernando Fernández. “A comprehensive survey on safe reinforcement learning”. In: *J. Mach. Learn. Res.* 16 (2015), pp. 1437–1480.
- [GG16] Yarin Gal and Zoubin Ghahramani. “Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning”. In: *ArXiv* abs/1506.02142 (2016).
- [GRG08] Paolo Guarneri, Gianpiero Rocca, and Massimiliano Gobbi. “A Neural-Network-Based Model for the Dynamic Simulation of the Tire/Suspension System While Traversing Road Irregularities”. In: *IEEE Transactions on Neural Networks* 19 (2008), pp. 1549–1563.
- [Haa14] John K Haas. “A history of the unity game engine”. In: (2014).
- [HCS+06] Guang-Bin Huang, Lei Chen, Chee Kheong Siew, et al. “Universal approximation using incremental constructive feedforward networks with random hidden nodes”. In: *IEEE Transactions on Neural Networks* 17.4 (2006), pp. 879–892.
- [He+15] Kaiming He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 770–778.
- [HE13] Iyad Hashlamon and Kemalettin Erbatur. “An improved real-time adaptive Kalman filter with recursive noise covariance updating rules”. In: *Turkish Journal of Electrical Engineering and Computer Sciences* (Dec. 2013). DOI: 10.3906/elk-1309-60.
- [HE16] Jonathan Ho and Stefano Ermon. “Generative Adversarial Imitation Learning”. In: *Neural Information Processing Systems*. 2016. URL: <https://api.semanticscholar.org/CorpusID:16153365>.
- [Hei+20] Alexander Heilmeier et al. “Minimum curvature trajectory planning and control for an autonomous race car”. In: *Vehicle System Dynamics* 58.10 (2020), pp. 1497–1527. DOI: 10.1080/00423114.2019.1631455.
- [HL23] Xiangkun He and Chen Lv. “Towards Energy-Efficient Autonomous Driving: A Multi-Objective Reinforcement Learning Approach”. In: *IEEE/CAA Journal of Automatica Sinica* 10.5 (2023), pp. 1329–1331.

- [HLZ17] Lukas Hewing, Alexander Liniger, and Melanie Nicole Zeilinger. “Cautious NMPC with Gaussian Process Dynamics for Autonomous Miniature Race Cars”. In: *2018 European Control Conference* (2017), pp. 1341–1348.
- [HW21] Eyke Hüllermeier and Willem Waegeman. “Aleatoric and epistemic uncertainty in machine learning: an introduction to concepts and methods”. In: *Machine Learning* 110 (2021), pp. 457–506.
- [HZZ06] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. “Extreme learning machine: theory and applications”. In: *Neurocomputing* 70.1-3 (2006), pp. 489–501.
- [Jai+20] Achin Jain et al. “BayesRace: Learning to race autonomously using prior experience”. In: *Proceedings of the 4th Conference on Robot Learning*. 2020.
- [JM20] Achin Jain and Manfred Morari. “Computing the racing line using Bayesian optimization”. In: *2020 59th IEEE Conference on Decision and Control (CDC)* (2020), pp. 6192–6197.
- [JMS21] Minghao Jiang, Kristina Miller, and Dawei Sun. “Continuous integration and testing for autonomous racing software: An experience report from GRAIC”. In: *Workshop on Opportunities and Challenges with Autonomous Racing, ICRA* (2021).
- [KH04] Nathan Koenig and Andrew Howard. “Design and use paradigms for Gazebo, an open-source multi-robot simulator”. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vol. 3. IEEE. 2004, pp. 2149–2154.
- [KI11] Nazli E. Kahveci and Petros A. Ioannou. “Automatic steering of vehicles subject to actuator saturation and delay”. In: *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. 2011, pp. 119–124.
- [KIR+19] Juraj Kabzan, Miguel de la Iglesia Valls, Victor Reijgwart, et al. “AMZ Driverless: The full autonomous racing system”. In: *Journal of Field Robotics* 37 (2019), pp. 1267–1294.
- [Kir+20] Bangalore Ravi Kiran et al. “Deep Reinforcement Learning for Autonomous Driving: A Survey”. In: *IEEE Transactions on Intelligent Transportation Systems* 23 (2020), pp. 4909–4926.
- [KLD21] Shivesh Khaitan, Qin Lin, and John M. Dolan. “Safe Planning and Control Under Uncertainty for Self-Driving”. In: *IEEE Transactions on Vehicular Technology* 70.10 (2021), pp. 9826–9837. DOI: 10.1109/TVT.2021.3108525.
- [KLD22] Dvij Kalaria, Qin Lin, and John M Dolan. “Delay-aware robust control for safe autonomous driving”. In: *2022 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2022, pp. 1565–1571.
- [Kum+21] Ashish Kumar et al. “RMA: Rapid Motor Adaptation for Legged Robots”. In: *ArXiv abs/2107.04034* (2021). URL: <https://api.semanticscholar.org/CorpusID:235650916>.

- [KVR19] Juraj Kabzan, Miguel de la Iglesia Valls, and Victor Reijgwart. *AMZ Driverless: The Full Autonomous Racing System*. 2019. DOI: 10.48550/ARXIV.1905.05150.
- [Lan+04] Wilbur Langson et al. “Robust model predictive control using tubes”. In: *Automatica* 40.1 (2004), pp. 125–133.
- [LDM15] Alexander Liniger, Alexander Domahidi, and Manfred Morari. “Optimization-based autonomous racing of 1:43 scale RC cars”. In: *Optimal Control Applications and Methods* 36 (2015), pp. 628–647.
- [LDM17] Alexander Liniger, Alexander Domahidi, and Manfred Morari. “Optimization-based autonomous racing of 1:43 scale RC cars”. In: *Optimal Control Applications and Methods* 36 (2017), pp. 628–647.
- [Li+22] Xuefang Li et al. “Robust Adaptive Learning-Based Path Tracking Control of Autonomous Vehicles Under Uncertain Driving Environments”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.11 (2022), pp. 20798–20809. DOI: 10.1109/TITS.2022.3176970.
- [Lim+10] D. Limon et al. “Robust tube-based MPC for tracking of constrained linear systems with additive disturbances”. In: *Journal of Process Control* 20.3 (2010), pp. 248–260. ISSN: 0959-1524. DOI: <https://doi.org/10.1016/j.jprocont.2009.11.007>.
- [LL18] Yonglong Liao and Fucheng Liao. “Design of preview controller for linear continuous-time systems with input delay”. In: *International Journal of Control, Automation and Systems* 16.3 (2018), pp. 1080–1090.
- [LLD21] Yiwei Lyu, Wenhao Luo, and John M. Dolan. “Probabilistic Safety-Assured Adaptive Merging Control for Autonomous Vehicles”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)* (2021), pp. 10764–10770.
- [LT15] Changliu Liu and Masayoshi Tomizuka. “Safe exploration: Addressing various uncertainty levels in human robot interactions”. In: *2015 American Control Conference (ACC)*. IEEE. 2015, pp. 465–470.
- [Mer+17] Pierre Merriaux et al. “A Study of Vicon System Positioning Performance”. In: *Sensors* 17.7 (2017). ISSN: 1424-8220. DOI: 10.3390/s17071591.
- [MNM22] Majid Mazouchi, Subramanya Nagesh Rao, and Hamidreza Modares. “Conflict-Aware Safe Reinforcement Learning: A Meta-Cognitive Learning Framework”. In: *IEEE/CAA Journal of Automatica Sinica* 9.3 (2022), pp. 466–481. DOI: 10.1109/JAS.2021.1004353.
- [Mor+21] Eduardo F. Morales et al. “A survey on deep learning and deep reinforcement learning in robotics with a tutorial on deep reinforcement learning”. In: *Intelligent Service Robotics* 14 (2021), pp. 773–805.
- [MPP08] Jadranko Matuško, Ivan Petrović, and Nedjeljko Perić. “Neural network based tire/road friction force estimation”. In: *Engineering Applications of Artificial Intelligence* 21.3 (2008), pp. 442–456.

- [MT76] Kenneth Myers and Byron Tapley. “Adaptive sequential estimation with unknown noise statistics”. In: *IEEE Transactions on Automatic Control* 21.4 (1976), pp. 520–523.
- [Mun+22] Emanuel Munoz et al. “Online Adaptive Compensation for Model Uncertainty Using Extreme Learning Machine-based Control Barrier Functions”. In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 10959–10966. DOI: 10.1109/IROS47612.2022.9981680.
- [Nah+19] Asal Nahidi et al. “A study on actuator delay compensation using predictive control technique with experimental verification”. In: *Mechatronics* 57 (2019), pp. 140–149.
- [Ope] Open AI. *Safety Gym*. <https://openai.com/research/safety-gym> [Accessed: 3-27-2024].
- [Pac05] Hans B. Pacejka. “Tire and Vehicle Dynamics”. In: Elsevier, 2005.
- [Pad+21] Cosmin Paduraru et al. “Challenges of real-world reinforcement learning: definitions, benchmarks and analysis”. In: *Machine Learning* 110 (2021), pp. 2419–2468.
- [Pan+17] Yunpeng Pan et al. “Agile Off-Road Autonomous Driving Using End-to-End Deep Imitation Learning”. In: *ArXiv* abs/1709.07174 (2017).
- [Raj12] Rajesh Rajamani. “Steering Control for Automated Lane Keeping”. In: 2012. URL: <https://api.semanticscholar.org/CorpusID:60147235>.
- [Rak+04] Saša V. Raković et al. *Invariant approximations of robustly positively invariant sets for constrained linear discrete-time systems subject to bounded disturbances*. University of Cambridge, Department of Engineering Cambridge, 2004.
- [RB16] Ugo Rosolia and Francesco Borrelli. “Learning Model Predictive Control for Iterative Tasks. A Data-Driven Control Framework”. In: *IEEE Transactions on Automatic Control* 63 (2016), pp. 1883–1896.
- [RB19] Ugo Rosolia and Francesco Borrelli. “Learning How to Autonomously Race a Car: A Predictive Control Approach”. In: *IEEE Transactions on Control Systems Technology* 28 (2019), pp. 2713–2719.
- [RB20] Ugo Rosolia and Francesco Borrelli. “Learning How to Autonomously Race a Car: A Predictive Control Approach”. In: *IEEE Transactions on Control Systems Technology* 28 (2020), pp. 2713–2719.
- [RGB10] Stéphane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. “A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning”. In: *International Conference on Artificial Intelligence and Statistics*. 2010.
- [RGB11] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. “A reduction of imitation learning and structured prediction to no-regret online learning”. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. JMLR Workshop. 2011, pp. 627–635.

- [Saa+19] Shahrizal Saat et al. “HECTORSLAM 2D MAPPING FOR SIMULTANEOUS LOCALIZATION AND MAPPING (SLAM)”. In: *Journal of Physics: Conference Series* 1529 (2019).
- [Sel+17] Ramprasaath R. Selvaraju et al. “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization”. In: *International Journal of Computer Vision* (2017), pp. 336–359.
- [SG03] Jacob Svendenius and Magnus Gäfvert. “Construction of Novel Semi-Empirical Tire Models for Combined Braking and Cornering”. In: *Technical Reports; TFRT* 7606 (2003).
- [Spi+19] Nathan A. Spielberg et al. “Neural network vehicle models for high-performance automated driving”. In: *Science Robotics* 4 (2019).
- [Sri+20] Sirish Srinivasan et al. “End-to-End Velocity Estimation for Autonomous Racing”. In: *IEEE Robotics and Automation Letters* 5 (2020), pp. 6869–6875.
- [SS21] TOBIAS SJÖLIN and ALEXANDER SUNDBERG. “Trajectory planning and control of an autonomous race vehicle”. PhD thesis. Stockholm, Sweden: KTH Royal Institute Of Technology School Of Industrial Engineering and Management, June 2021.
- [STL13] Yang Su, Kok Kiong Tan, and Tong Heng Lee. “Computation delay compensation for real time implementation of robust model predictive control”. In: *Journal of Process Control* 23.9 (2013), pp. 1342–1349.
- [Sve07] Jacob Svendenius. “Tire modeling and friction estimation”. PhD thesis. Lund University, 2007.
- [Tha+22] Rishabh Saumil Thakkar et al. “Hierarchical Control for Multi-Agent Autonomous Racing”. In: *ArXiv* abs/2202.12861 (2022).
- [Ves15] Richard W. Vesel. “Racing line optimization @ race optimal”. In: *ACM SIGEVOlution* 7 (2015), pp. 12–20.
- [VS22] Matias Valdenegro-Toro and Daniel Saromo. “A Deeper Look into Aleatoric and Epistemic Uncertainty Disentanglement”. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2022), pp. 1508–1516. URL: <https://api.semanticscholar.org/CorpusID:248266412>.
- [Wad+21] Shakti Nagnath Wadekar et al. “Towards End-to-End Deep Learning for Autonomous Racing: On Data Collection and a Unified Architecture for Steering and Throttle Prediction”. In: *ArXiv* abs/2105.01799 (2021).
- [Wer+10] Moritz Werling et al. “Optimal trajectory generation for dynamic street scenarios in a Frenét Frame”. In: *2010 IEEE International Conference on Robotics and Automation*. 2010, pp. 987–993.
- [WH04] Adrian G. Wills and William P. Heath. “Barrier function based model predictive control”. In: *Automatica* 40.8 (2004), pp. 1415–1422. ISSN: 0005-1098. DOI: <https://doi.org/10.1016/j.automatica.2004.03.002>.

- [Wil+16] Grady Williams et al. “Aggressive driving with model predictive path integral control”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)* (2016), pp. 1433–1440. URL: <https://api.semanticscholar.org/CorpusID:18322548>.
- [Wil+18] Grady Williams et al. “Robust Sampling Based Model Predictive Control with Sparse Objective Information”. In: *Robotics: Science and Systems XIV* (2018). URL: <https://api.semanticscholar.org/CorpusID:46964005>.
- [WL20] Wilhelm J. West and David J.N. Limebeer. “Optimal Tyre Management of a Formula One car”. In: *IFAC-PapersOnLine* 53.2 (2020). 21st IFAC World Congress, pp. 14456–14461. ISSN: 2405-8963. DOI: <https://doi.org/10.1016/j.ifacol.2020.12.1446>.
- [Wu+22] Philipp Wu et al. “DayDreamer: World Models for Physical Robot Learning”. In: *Conference on Robot Learning*. 2022.
- [WY] Zhiguang Wang and Jianbo Yang. “Diabetic Retinopathy Detection via Deep Convolutional Networks for Discriminative Localization and Visual Explanation”. In: *ArXiv abs/1703.10757* ().
- [XB19] Wei Xiao and Calin A. Belta. “Control Barrier Functions for Systems with High Relative Degree”. In: *2019 IEEE 58th Conference on Decision and Control (CDC)* (2019), pp. 474–479.
- [Xia+23] Wenli Xiao et al. “Safe Deep Policy Adaptation”. In: *ArXiv abs/2310.08602* (2023). URL: <https://api.semanticscholar.org/CorpusID:264128072>.
- [XJ22] Zhanrui Xu and Xiaohong Jiao. “Robust Control of Connected Cruise Vehicle Platoon With Uncertain Human Driving Reaction Time”. In: *IEEE Transactions on Intelligent Vehicles* 7.2 (2022), pp. 368–376. DOI: 10.1109/TIV.2021.3131641.
- [Xu+20] Nan Xu et al. “Tire Force Estimation in Intelligent Tires Using Machine Learning”. In: *IEEE Transactions on Intelligent Transportation Systems* 23 (2020), pp. 3565–3574.
- [Yi+24] Zeji Yi et al. “CoVO-MPC: Theoretical Analysis of Sampling-based MPC and Optimal Covariance Design”. In: *ArXiv abs/2401.07369* (2024). URL: <https://api.semanticscholar.org/CorpusID:266999348>.
- [YZT22] Ji Yin, Zhiyuan Zhang, and Panagiotis Tsiotras. “Risk-Aware Model Predictive Path Integral Control Using Conditional Value-at-Risk”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)* (2022), pp. 7937–7943. URL: <https://api.semanticscholar.org/CorpusID:252532183>.
- [Zho+23] Qiuxin Zhong et al. “Robust Control of Manned Submersible Vehicle With Non-linear MPC and Disturbance Observer”. In: *IEEE CAA J. Autom. Sinica* 10 (2023), pp. 1349–1351.

- [ZZS21] Jun Zeng, Bike Zhang, and Koushil Sreenath. “Safety-Critical Model Predictive Control with Discrete-Time Control Barrier Function”. In: *2021 American Control Conference (ACC)* (2021), pp. 3882–3889.