SAFE AUTONOMY: CONTINUOUS TESTING AND CONTROLLER COMPENSATION FOR UNRELIABLE PERCEPTION

BY

YAN MIAO

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois Urbana-Champaign, 2024

Urbana, Illinois

Advisor:

Professor Sayan Mitra

Abstract

This thesis addresses two key challenges in designing safe and reliable Autonomous Vehicles (AVs): evaluating the performance of vehicle controllers across different scenarios and designing safe vehicle controllers that can manage unreliable perception.

Testing is essential for AVs as it helps identify potential flaws in controllers that could lead to unsafe actions, and it assists designers in debugging. However, testing is a complex process that involves integrating resources like simulators and various modules, including controllers and perception systems, which is often a cumbersome task. To simplify this, we propose a continuous testing pipeline that automates the evaluation of controllers in diverse simulation environments and provides developers with feedback to continuously improve their controllers. Our continuous testing pipeline has supported over 100 student developers from various universities, enabling them to test their controllers automatically.

Designing controllers with learning-based perception is also crucial, given the popularity of learning-based perception for its scalability. Nevertheless, many existing controller synthesis algorithms assume perfect perception, overlooking the fact that perception can be unreliable and that perception errors can lead to unsafe control actions. To address this, we introduce a two-step approach: an offline phase that identifies perception uncertainties using a preimage perception contract, followed by the real-time implementation of a risk heuristic to ensure safe control. Our simulations, conducted in various adaptive cruise control scenarios across different tracks and weather, have demonstrated the effectiveness of this strategy, resulting in a 73% reduction in safety violations such as collisions and lane departures caused by unreliable perception.

To my parents and friends for their love and support.

Acknowledgments

This thesis could not have been realized without the support and encouragement from numerous individuals. Foremost, I extend my deepest gratitude to my advisor, Prof. Sayan Mitra, who has been both a mentor and a friend over the past two years. His wisdom and critical approach to research problems, coupled with his exemplary leadership in guiding our research group, have been invaluable to my growth.

Additionally, I owe a debt of gratitude to my parents, Beibei Zeng and Yajun Miao, for their unwavering financial and emotional support throughout my journey in higher education. Their backing has been the cornerstone of my achievements.

I am also thankful to my colleagues Yangge Li, Dawei Sun, Kristina Miller, and Ben Yang, for enriching my research and meaningful research discussions.

I'm thankful for my friends, Yuwei Pan, Yurui Cao, and Hellan Lee, for making my time in Champaign enjoyable and rewarding. Their friendship has been a source of joy and has provided intellectual stimulation outside of the academic sphere.

Lastly, special thanks to my girlfriend, Jiawei Song, for her constant companionship and encouragement through the challenging times. Her support has been a beacon of hope and strength.

Table of contents

List of Abbreviations

AV	Autonomous Vehicle.
DL	Decision Logic.
GRAIC	Generalized Racing Intelligence Competition
ML	Machine Learning.
NN	Neural Network.
BNN	Bayesian Neural Network.
CNN	Convolutional Neural Network.
ACC	Autonomous Cruising Control.
PC	Perception Contracct.
PPC	Preimage Perception Contract.

Chapter 1

Architecture of Autonomous Vehicles & Design Challenges

The design of Autonomous Vehicles (AVs) has attracted significant attention in both academic research and industry development, driven by their potential to revolutionize transportation, enhance safety, and improve efficiency. Waymo, a leading company specializing in L4-level autonomous vehicles, has successfully launched hundreds of unmanned taxis across two major cities (as shown in Figure 1.1), San Francisco and Phoenix, and reported an 85% reduction in crash rates involving injuries compared to human driving benchmarks [1]. Another study by the Center for Sustainable Systems at the University of Michigan emphasizes that AVs, when integrated with AI technologies, are expected to reduce vehicular crashes caused by human error by 90%, potentially saving approximately $190 billion annually [2]. Additionally, research suggested that AI algorithms assist AVs in enhancing eco-driving practices, reducing energy consumption by up to 20% [3], and are also believed to increase passenger comfort by minimizing sudden maneuvers, such as harsh acceleration [4].

The advancement of AVs is supported by a broad array of State-of-the-Art Artificial Intelligence (AI) software elements.

Figure 1.1: Waymo simulation uses sensors, including cameras and LiDARs, to reconstruct the environment. This below figure shows the view captured by the camera while the above figure depicts the reconstructed environment, with detected humans and vehicles denoted in blue, the crosswalk in white, and the planned trajectory shown in green.

1.1 Software Elements

This section introduces various elements employed in the software design of AVs, which generally encompasses perception, planning, and control components, as illustrated in Figure 1.2.

The perception module leverages raw data from sensors, such as cameras and LiDARs, to interpret the environment, which involves estimating lane positions and the relative pose of obstacles from raw image arrays and point cloud data. The planning module then uses this data to devise a safe and viable trajectory. Subsequently, the control module determines the necessary maneuvers, including throttle, brake, and steering inputs, based on the planned trajectory.

Figure 1.2: The Autonomous Vehicle Design Pipeline

1.1.1 Perception Module

The perception module is critical as it forms the foundation for planning and control. Inaccuracies in perception can propagate errors, leading to unsafe scenarios. Modern perception modules in AVs typically utilize Machine Learning (ML) due to their scalability and rapid inference capabilities.

YOLO (You Only Look Once) [5], a widely used object detection system, efficiently estimates bounding boxes for traffic signs, pedestrians, and vehicles. YOLO processes object detection as a single regression problem, converting image pixels directly into bounding box coordinates and class probabilities. This efficiency is crucial for real-time applications in AVs. LaneNet [6], designed for lane detection, employs a deep neural network, specifically a U-Net architecture, to accurately identify lane markings under various conditions. The combination of semantic and instance segmentation enables LaneNet to effectively handle complex driving scenarios.

In addition to single-frame detection algorithms, tracking algorithms such as DaSiamRPN [7] and Deep SORT [8] are essential for maintaining object consistency across frames. These algorithms ensure that even if an object is temporarily obscured, it can still be accurately tracked, enhancing the reliability of the perception module.

Moreover, libraries like OpenCV [9] facilitate the integration of complex perception algorithms into the autonomous vehicle ecosystem, supporting the development and optimization of these systems.

1.1.2 Planner Module

The planner module for autonomous vehicles is crucial for navigating complex environments safely and efficiently. It can be broadly categorized into three types: sampling-based planners, graph-based planners, and optimization-based planners, each with distinct methodologies and applications.

Sampling-based planners, such as RRT^* [10], excel in navigating areas dense with obstacles by iteratively refining the path to the goal. Graph-based planners, like Hybrid

A^* [11], consider the vehicle's dynamics, offering paths that are not only short but also navigable. The Frenet Optimal Trajectory Planner [12] represents the vehicle's position in terms of lateral and longitudinaldistances from a reference path or lane centerline, evaluates multiple paths against a cost function, and selects the most optimal path, ensuring safety, legality, comfort, and efficiency in navigation.

Furthermore, existing libraries (for example, pylot [13]) facilitate the integration of diverse planning algorithms into the autonomous vehicle development pipeline, thereby broadening the capabilities of AV systems.

1.1.3 Controller Design

Controller design in autonomous vehicles focuses on precision in following planned paths while maximizing passenger comfort and safety.

PID controllers [14], fundamental yet highly effective, adjust the vehicle's steering and throttle based on the deviation from the desired path. They utilize proportional, integral, and derivative components to correct errors, offering a simple yet robust solution for trajectory maintenance.

Pure Pursuit and the Stanley method [15] calculate the required steering angle to follow the path, with adjustments based on vehicle speed and orientation, ensuring smooth path tracking. Model Predictive Control (MPC) [16], a more advanced technique, predicts and optimizes the vehicle's trajectory, accounting for future states and dynamic constraints, thereby offering a comprehensive solution for complex environments.

1.2 Technical Challenges: Safety & Reliability

Despite advancements in AI and ML technologies, safety challenges in AVs remain a significant concern.

There are many different different driving scenarios due to variation in roads, weather conditions, lighting, obstacles' and the vehicle's position. However, ensuring that AVs remain safe under all scenarios is challenging. For instance, control algorithms may react

inadequately to scenarios where unexpected obstacles appear in road conditions. A notable example from 2022 involved a Waymo Via truck operating in autonomous mode, which, although not at fault, was forced off the road by another semi-truck with road rage [17]. This incident highlights the importance of comprehensive testing across diverse scenarios because testing can reveal the vulnerability of AVs.

Furthermore, achieving flawless perception in AVs is a formidable challenge and is susceptible to failures, especially under adverse weather conditions or in low light. These misinterpretations can lead to unsafe maneuvers. For example, in October 2023, a Cruise unmanned vehicle collided with and subsequently dragged a pedestrian in San Francisco because its perception system failed to detect the pedestrian [18]. In another incident in 2020, a Tesla mistook a Burger King sign for a stop sign, nearly causing the vehicle to stop on a highway [19]. This unreliable perception underscores the urgent need to develop controllers that can operate reliably even with uncertain perception data.

1.3 Thesis Contribution: Continuous Testing & Controller Compensation

In this thesis, we try to focus on two aspects to help improve the safety of autonomous vehicles: continuous testing and designing controller for unreliable perception.

In Chapter 2, we focus on the controller testing problem for AVs and develop an open-source continuous testing pipeline. The key features include:

- It automatically evaluates controller designs across a multitude of scenarios.
- It iteratively provides testing feedback to guide developers to improve controllers.
- It comes with user-friendly code submission portal and leaderboard for displaying results.
- It is used by over 100 student developers across different universities to test their controllers.

In Chapter 3, we focus on the controller design problem with imperfect perception, and proposed a runtime controller correction method. The key features are:

- It characterizes the perception uncertainties of the learning-based perception module.
- It synthesizes safe control action based on the perception uncertainty and a heuristic function.
- Empirically it shows a 73% reduction in safety violations such as collisions and lane departures caused by unreliable perception data in various Adaptive Cruise Control(ACC) scenarios across different tracks and weather conditions.

Chapter 2

Continuous Testing Pipeline for Autonomous Vehicles

In this Chapter, we are going to introduce the continuous testing pipeline, that specifically test the controller component of the autonomous vehicle by assuming perfect perception.

Section 2.1 motivates the continuous testing problem and listed some related work. Section 2.2 explain the detailed design choices of the continuous testing pipeline. Section 2.3 describes two different applications with our continuous testing pipeline. Section 2.4 shows the user experience and feedback. Section 2.5 concludes the work.

2.1 Design & Testing Techniques

Testing of vehicle design has recently emerged as a critical research topic [20] due to the importance of ensuring safety. This section explores existing related work on vehicle design and testing techniques.

2.1.1 Planner & Controller Design

Significant progress has been made in controller synthesis algorithms, which focus on developing safe control algorithms assuming perfect perception [21]–[25]. CommonRoad provides a platform for researchers to evaluate and compare their motion planners [26]. Given the variety of controllers and planners available, effective testing is essential.

2.1.2 Testing through Falsification

Falsification research in vehicle systems aims to identify scenarios or conditions where a vehicle's control systems might fail, thus uncovering bugs in vehicle designs that could lead to failures. Cho and Behl from the University of Virginia propose using reinforcement learning (RL) to generate failure examples and unexpected traffic situations for edge-case testing [27]. Jha from the University of Illinois presents DriveFI, a machine learning-based fault injection engine designed to identify critical faults impacting AV safety [28]. Corso and Du have introduced an adaptive stress testing method to find unavoidable failure scenarios by solving a Markov decision process using reinforcement learning [29].

2.1.3 Field Testing

Field testing of autonomous vehicles involves conducting hardware experiments in specialized facilities to evaluate system performance. Feng and colleagues at the University of Michigan have developed an augmented reality-based testing platform, coupled with a scenario library generation method, tested on an SAE Level-4 ADS vehicle at the M-City test facility [30]. Zhang and colleagues introduced a comprehensive evaluation methodology to assess the performance of roadside perception systems using vehicles equipped with GPS-RTK at Mcity [31]. However, hardware experiments can be time-consuming, risky, and influenced by external factors such as weather and visibility.

2.1.4 Vehicle Simulator

Simulation compensates for some disadvantages of field testing, such as fewer hardware constraints. The CARLA Simulator is an open-source platform that provides extensive facilities for developing, training, and validating autonomous driving systems, including sensor simulation and various urban scenarios [32]. Scenic, a domain-specific probabilistic programming language, facilitates the design and testing of systems in autonomous vehicles and robotics by allowing specifications of scenario distributions [33]. The F1-Tenth simulator integrates the Robot Operating System (ROS) and the Gazebo simulator to test vehicle algorithms in a competitive racing context [34]. However, setting up these simulators can also be challenging as they often require advanced technical skills and resources, such as powerful GPUs and knowledge about Linux.

2.1.5 Continuous Integration

Research involving Continuous Integration (CI) and continuous testing is crucial for advancing the reliability and functionality of autonomous systems. Drake software highlights the importance of automated unit testing in robotics for maintaining software quality [35]. Other notable CI frameworks combine version control tools, like GitHub, with build tools, like AWS, to automate testing upon new code commits [36]. However, many of these advanced testing frameworks are not open-source, limiting their accessibility to the wider research community.

Inspired by the innovative work previously mentioned, this thesis aims to implement a continuous testing pipeline for controllers in simulation, assuming perfect perception. This pipeline is designed to be user-friendly and quick in producing testing results, requiring minimal technical knowledge. The next section will detail the construction of this continuous testing pipeline.

2.2 Design Choices of Continuous Testing Pipeline

Our continuous testing pipeline is architecturally composed of three distinct servers: the submission server, the testing server, and the output server, depicted in Figure 2.1. The code setup and instructions on how to replicate the continuous testing pipeline with three servers could be found at this Github repository.

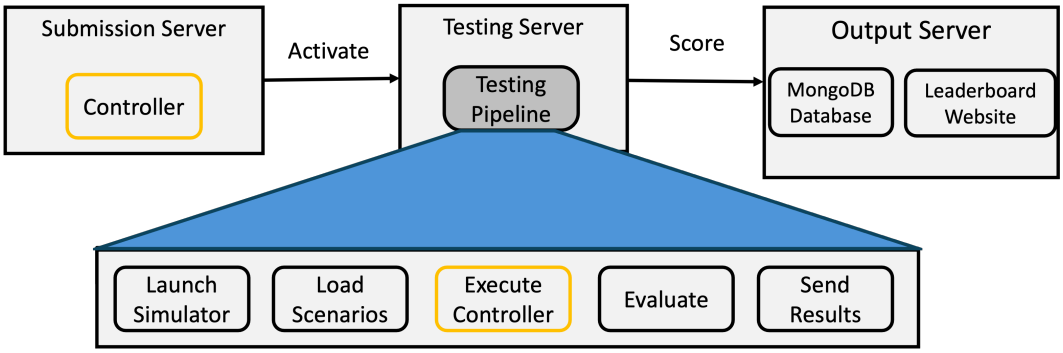


Figure 2.1: Block diagram showing the flowchart of the code-level continuous testing pipeline. The diagram highlight three main components: the submission server, the testing server, and the output server. The controller submitted to the submission server by developers will be automatically trigger the testing server for evaluation, and the output server will display the performance result on a leaderboard website when the testing server finishes the testing process.

The submission server facilitates an interface for developers to upload their scripts to be tested as well as some basic information. Following this, the testing server undertakes the evaluation of the submitted scripts within varied simulation environments. Ultimately, the output server is responsible for presenting the performance results of these evaluations on a publicly accessible leaderboard website.

Our continuous testing pipeline primarily operates in the background, thereby alleviating developers from the complexities of the testing process. Upon submitting their scripts via the interface illustrated in Figure 2.2, developers can expect to receive email notifications regarding their submissions in a matter of minutes, and they can also view their performance metrics displayed on the leaderboard website, as depicted in Figure 2.3.

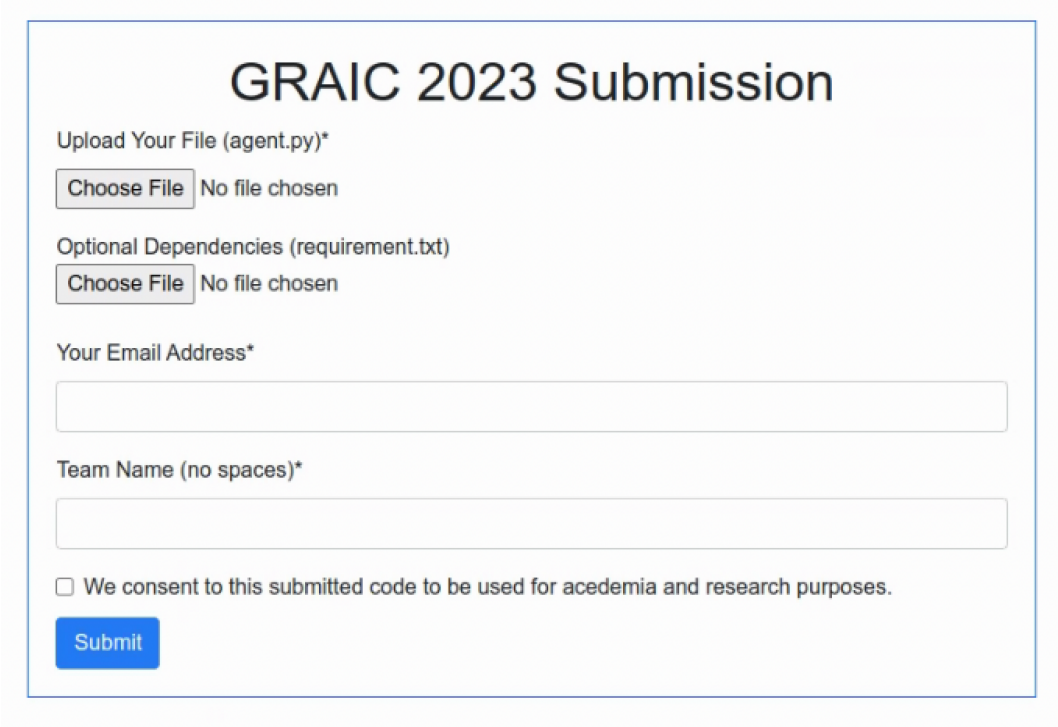


Figure 2.2: The website interface for the submission server, where developers could submit the controller script for evaluation

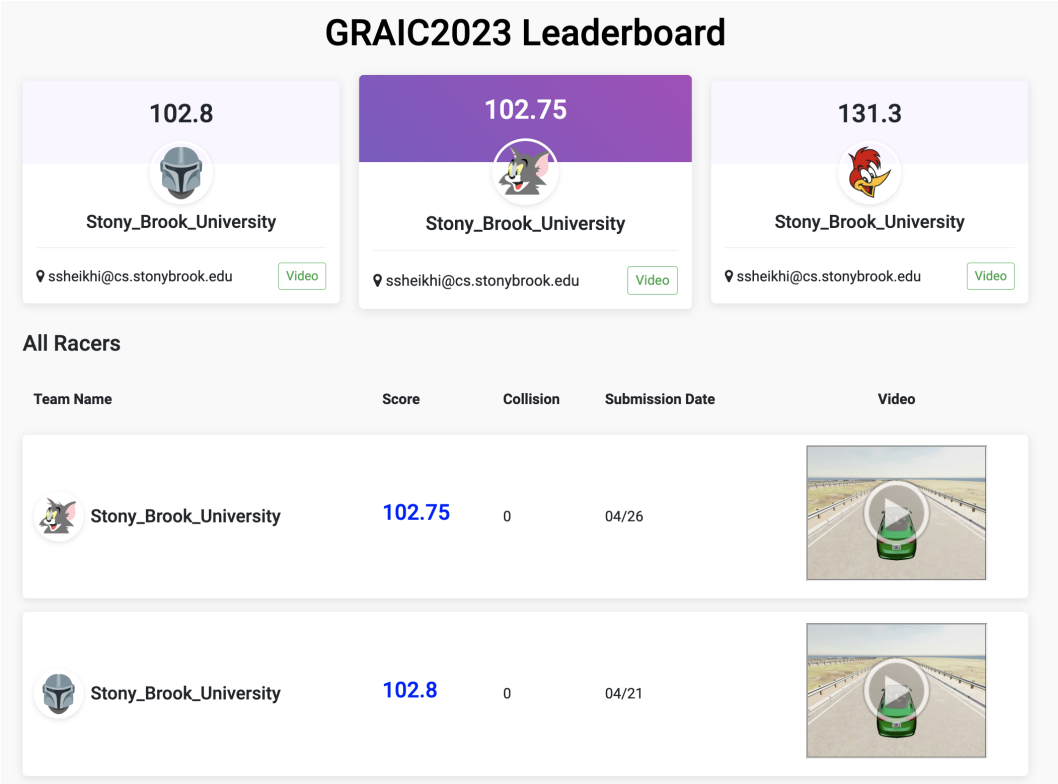


Figure 2.3: The leaderboard website with performance metric displayed as well as video log and score.

2.2.1 Submission Server

The first stage of the automated testing pipeline is the submission server. The purpose of this server is to provide a modern user interface for developers to easily submit their files, which are then securely stored on a server and later retrieved by the testing server for processing.

The website interface, shown in Figure 2.2, facilitates file submission and can be accessed here. Should this link become inactive, a cached snapshot of the website as of March 25, 2024, is available here.

Our website offers a straightforward platform where developers can submit their scripts along with their email addresses. The front end uses HTML and Bootstrap CSS to ensure the site is navigable and aesthetically pleasing on any device, from smartphones to desktop computers. Bootstrap's responsive design features help maintain alignment and responsiveness without the need for extensive custom CSS.

The back end is powered by the Python Flask web framework, which serves dual purposes: it renders the HTML page when someone visits the site (using a GET request) and processes the data when a user submits a script along with their email address (using a POST request). Upon submission, the Flask server securely stores the script, the email address, and a timestamp on our server, maintaining a record of each submission.

Both the front end and the back end are hosted on the University of Illinois' cPanel web hosting service, ensuring that the site is reliable and can handle significant traffic without requiring us to manage server infrastructure directly. The source code and a simple readme to replicate the submission server setup can be found on our GitHub repository.

2.2.2 Testing Server

The testing server functions as the core component of our testing pipeline, specifically designed to support a range of testing applications, such as GRAIC and Verse, which are detailed further in Section 2.3. It operates by hosting a specialized script that continuously monitors the submission server for new submissions. When a new submission is detected, this script initiates the testing process by launching the simulation environment, loading the

designated simulation scenarios, and executing the tests. Upon test completion, it generates various outputs, including videos, images, HTML files, and score logs.

To efficiently manage these outputs, another script is triggered to automatically upload the test results to Google Drive using the Google Cloud API, integrating cloud services to enhance data storage and accessibility. Simultaneously, the MongoDB database is updated with links to these results and the submitter's email address using the MongoDB API.

Additionally, an automatically activated script employs the smtplib library to send detailed test results via email to developers. This use of the smtplib library automates email communications directly from Python scripts, providing developers with immediate feedback on their submissions. This level of automation from submission to feedback ensures that the entire process is seamless and requires no manual intervention, significantly enhancing the user experience.

For more details about configuring the setup to accommodate inputs/outputs for different simulations, please refer to Section 2.3.

2.2.3 Database & Leaderboard Website

The output server represents the final stage of our testing pipeline and plays a crucial role by offering developers a comprehensive and accessible view of their performance through a leaderboard website, accessible via this link.

This website converts the raw data logs from the testing server into a user-friendly format, fostering competition and inspiration among developers. The architecture of the leaderboard utilizes the MERN stack, primarily focusing on MongoDB, Express.js, and Node.js for our implementation.

MongoDB, a NoSQL database, is selected for its ability to manage large volumes of unstructured data efficiently and flexibly. It stores results updated by the testing server and supports dynamic queries for retrieving the latest test outcomes. Express.js, a streamlined Node.js web application framework, manages the rendering of the webpage. It processes requests and dynamically serves the leaderboard content, ensuring the information is current and accurate.

The server-side content rendering uses EJS (Embedded JavaScript templates), enabling HTML markup creation with plain JavaScript. This templating engine is crucial for handling GET requests, fetching necessary data from MongoDB, and dynamically producing the HTML content that users see on the website. This method allows for smooth integration of database content into the webpage, providing a dynamic and interactive experience for users.

For hosting, we chose Heroku due to its robust integration with GitHub, which supports continuous deployment and integration. Heroku's platform not only simplifies development but also accelerates the deployment process, allowing for quick updates and iterations. Its scalability ensures the leaderboard website remains accessible, current, and capable of managing data flow from our testing server efficiently.

2.3 Applications in Testing & Verification

As mentioned in the last section, the testing pipeline can be easily modified to different testing applications. In this section, we are going to introduce the usage of the continuous testing pipeline with two applications: a vehicle race called GRAIC [37] and a verification framework called Verse [38].

2.3.1 Controller Testing with GRAIC

In this subsection, we will demonstrate how the continuous testing pipeline integrates with the Generalized Racing Intelligence Competition (GRAIC) [37].

GRAIC utilizes the Carla simulator to provide a specialized environment for testing vehicle controllers in racing scenarios, assuming perfect perception to focus solely on controller performance. The competition offers a diverse array of tracks and obstacles, meticulously crafted using RoadRunner for track creation and Scenario Runner for simulating obstacle behaviors, as highlighted in Figures 2.4 and 2.5. A key feature of GRAIC is the provision of a perception oracle, which supplies ground truth perception data, allowing participants to concentrate on refining and testing their controller designs, as illustrated in Figure 2.6. The primary challenge for participants is to develop a controller that can navigate all waypoints

Figure 2.4: An example of the GRAIC Scenario. The blue box in front of the ego vehicle is the waypoint it requires to reach. The two vehicles ahead are the dynamic obstacles that the ego vehicle is expected to avoid. One controller design choice is to change to the left lane and overtake the obstcales using the inside lane of the left turn ahead of the ego vehicle.

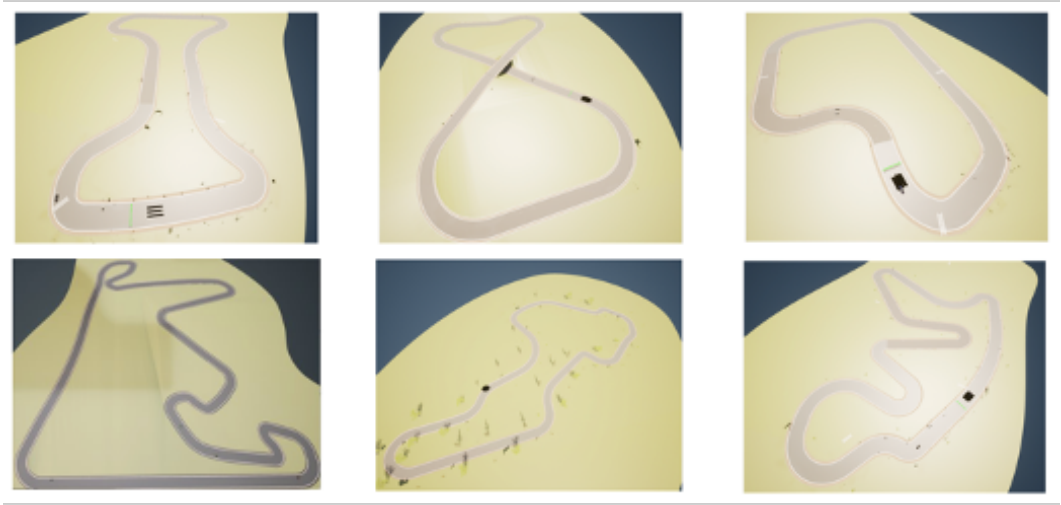


Figure 2.5: Six different tracks we have created using RoadRunner for GRAIC testing. Top row has three relatively simple tracks with less turns. The bottom row shows three complicated tracks with sharp turns. The left bottom one is a reconstruction of a real Formula One track – Shanghai Audi International Circuit

```
def run_step(self, filtered_obstacles, waypoints, vel, transform, boundary):
    """
    Execute one step of navigation.

    Args:
    obstacles
        - Type:        List[carla.Actor(), ...]
        - Description: All actors except for EGO within sensoring distance
    waypoints
        - Type:         List[[x,y,z], ...]
        - Description:  List All future waypoints to reach in (x,y,z) format
    vel
        - Type:         carla.Vector3D
        - Description:  Ego's current velocity in (x, y, z) in m/s
    transform
        - Type:         carla.Transform
        - Description:  Ego's current transform
    boundary
        - Type:         List[List[left_boundary], List[right_boundary]]
        - Description:  left/right boundary each consists of 20 waypoints,
                        they defines the track boundary of the next 20 meters.

    Return: carla.VehicleControl() <Steering + Throttle + Brake>
    """
```

Figure 2.6: This is the Input-Output interface of the controller. developers are given the ground truth perception information, including the obstacles global location, ego vehicle's global location and velocity, reference trajectory to follow. And developers are expected to design controller logic that output steering, throttle and brake based on the given perception information.

```python
def decisionLogic(ego: State, other: State):
    output = copy.deepcopy(ego)

    # TODO: Edit this part of decision logic

    if ego.agent_mode == VehicleMode.Normal and other.dist < 12:
        output.agent_mode = VehicleMode.Brake

    ##########################################

    assert other.dist > 2.0

    return output
```

Figure 2.7: This is the Decision Logic interface provided to user. Input is the distance to the pedestrian, represented with other.dist, the output DL should assign in output.agent_mode. The example DL is assigned the Brake mode when the ego vehicle is previously in the Normal driving mode and the distance to the pedestrian is less than 12 meters.

as swiftly as possible while avoiding obstacles.

Within our continuous testing pipeline, GRAIC functions as the testing server, which will generate a score that considers the time to finish the lap and the collision penalty. We have customized the submission server to accept controller scripts in Python, as depicted in Figure 2.2.

The output server has been adapted to accommodate GRAIC's specific outputs. In addition to displaying score metrics such as completion time and the number of collisions, it also showcases videos of the simulation runs. This visual representation not only provides insight into each participant's controller performance but also adds an engaging element to the competition. This allows both participants and observers to analyze and appreciate the nuances of each controller's performance. Alongside the publicly available leaderboard data, detailed run logs are sent to developers in the form of Carla Recorder, providing further insights into each controller's behavior during the competition.

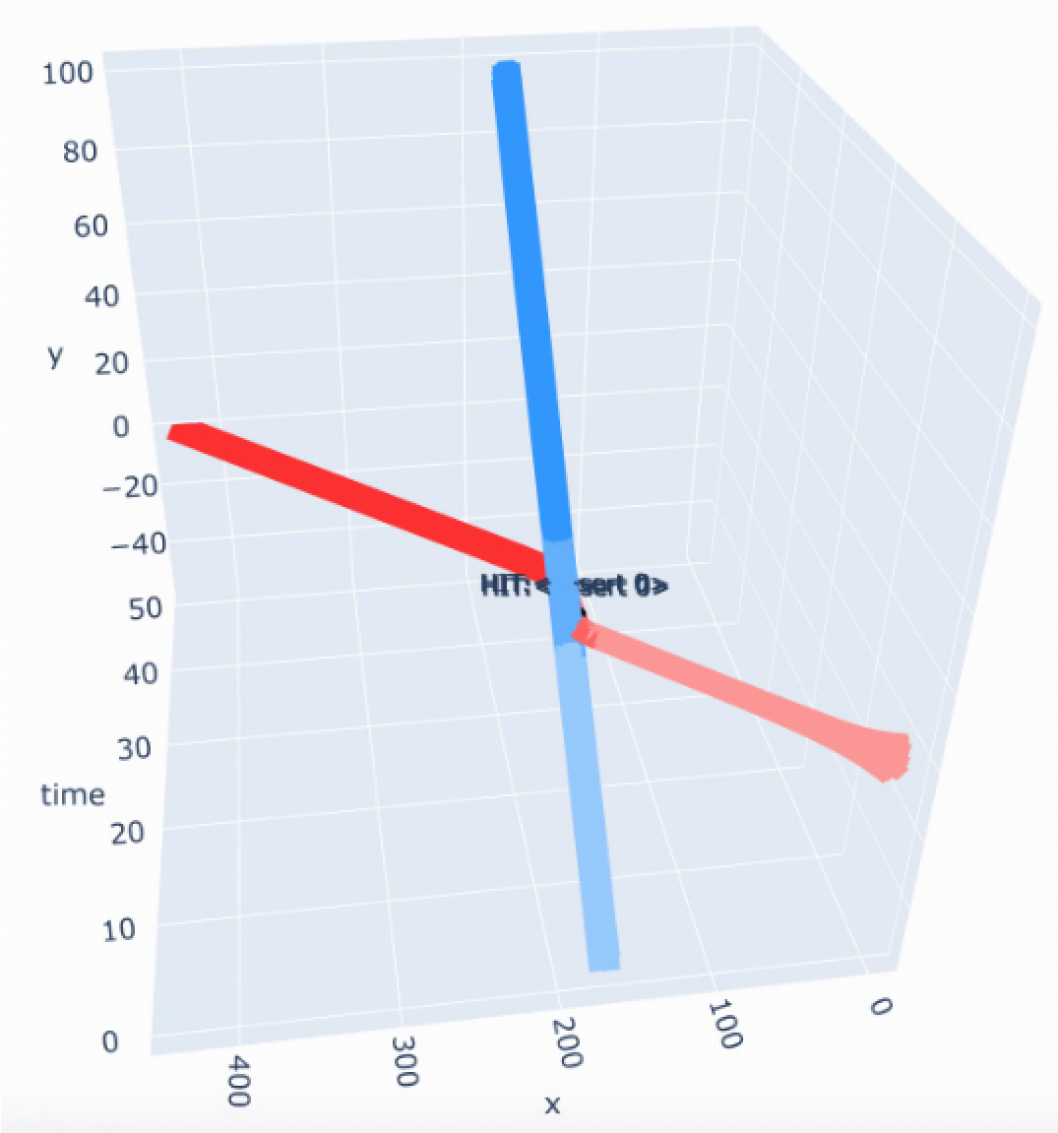


Figure 2.8: The blue tube is the pedestrian's reachable sets, while the red tube is the ego vehicle's reachable sets under a certain DL; "HIT" in the graph indicates that Verse determines the DL will result in unsafe behaviors.

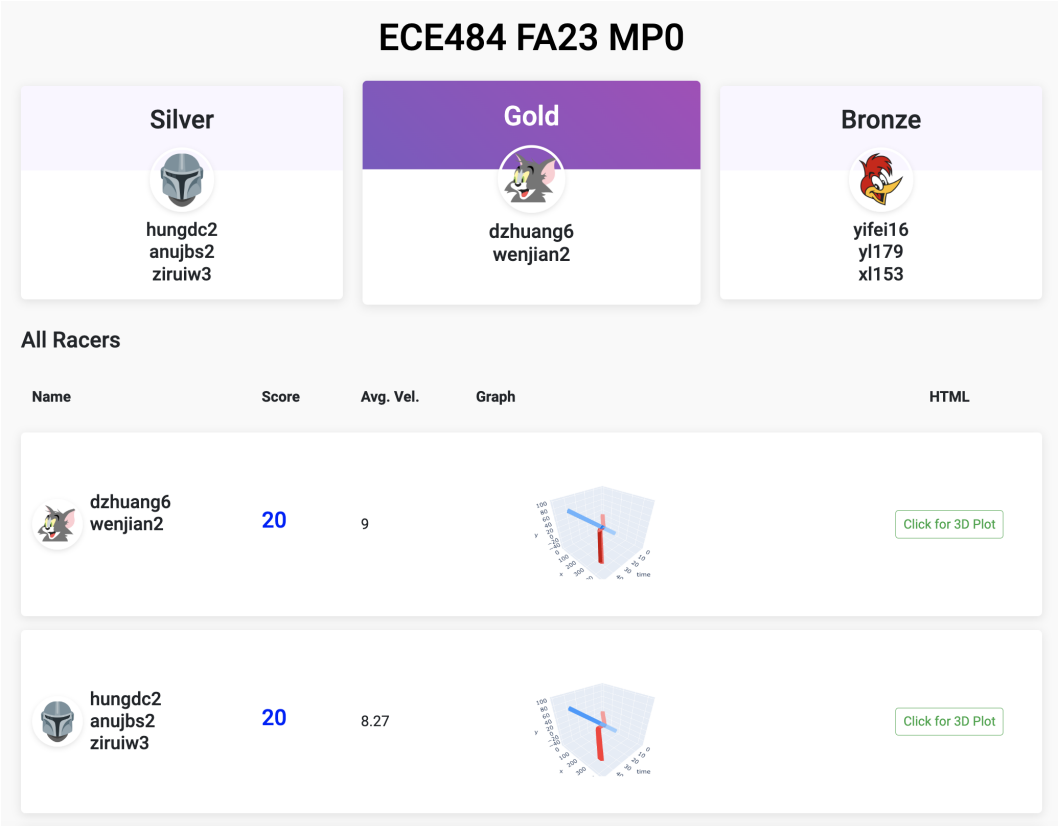


Figure 2.9: The leaderboard for Verse testing, with a slight modification to the original GRAIC leaderboard, as shown in Figure 2.3

2.3.2 Decision Logic Verification with Verse

Our testing pipeline can also work with verification tools like Verse [38]. Verse is a Python-based verification tool for hybrid systems, which calculates the reachable set of the system using its dynamics and user-submitted decision logic. Given a set of initial states, the reachable set is defined as all terminal states that the system can transition to under the specified decision logic and dynamics. Understanding reachable sets is crucial because proving that they do not intersect with unsafe sets at any time confirms that the decision logic ensures safe execution.

One specific scenario tested using Verse is the automatic emergency braking (AEB) decision logic (DL) for a car when a pedestrian suddenly appears in front. Although many scenarios can be tested, we focus here on introducing the AEB testing setup. The pipeline can be extended to other scenarios with appropriate modifications.

In this AEB scenario, the input to the DL is the ground truth distance to the pedestrian. Based on this input, developers are tasked with designing DL that outputs various driving modes, such as Normal, Accelerate, Brake, and Hard Brake. An example of the decision logic is illustrated in Figure 2.7.

In this setup, the submission server retains its role as the initial contact point for developers, who submit their decision logic scripts intended to control the vehicle in pedestrian crossing scenarios.

The testing server, now integrated with the Verse library instead of the Carla simulator used in previous setups, calculates the reachability of the decision logic script.

The results from this reachability analysis are then relayed to the output server, as shown in Figure 2.9. Unlike the score metrics of time and collision count used in the GRAIC competition, the outputs for decision logic testing are more complicated. The server generates a detailed reachability HTML report, depicted in Figure 2.8, which visually represents the reachability of the decision logic and the unsafe area, along with a counterexample file (if one exists) to highlight any flaws in the decision logic. These outputs provide developers with clear, accessible feedback on whether their vehicle's decision-making logic has successfully navigated the pedestrian-crossing scenario without incident.

2.4 User Experience

We have successfully applied our continuous testing pipeline to various events.

During the 2022 CPS-IoT Week, we launched our continuous testing pipeline with the GRAIC competition. We received a total of 16 submissions from six teams spanning three universities (University of Illinois at Urbana-Champaign, Stony Brook University, University of North Carolina) and one company (Galois, Inc.). The effectiveness of our testing is evidenced by every team's final submission score being higher than their initial one, indicating that our feedback helped them improve. Specifically, the Stony Brook University team identified and fixed a memory leak in their planner code based on our feedback, which significantly improved their score.

Additionally, we deployed the continuous testing pipeline for MP0 in UIUC's ECE484: Principles of Safe Autonomy course during the Fall 2023 semester, reaching both undergraduate and graduate students (46 and 20 students, respectively). Student feedback from our survey included positive remarks such as "It is clear and it can detect unsafe execution." Notably, two groups that did not achieve perfect scores initially were able to do so after receiving our feedback with counterexample files.

In both cases, our continuous testing pipeline proved capable of delivering feedback within 15 minutes, identifying potential collisions or counterexamples in the designs, and aiding student developers in debugging their design.

Moreover, we have extended the application of our continuous testing pipeline to community outreach events, where it received widespread support. At the 2022 Engineering Open House, we set up the simulation for students of all age groups to experience our testing pipeline, receiving 52 submissions[1]. We also used the continuous testing pipeline at the 2022 Summer Camp for six high school students[2].

[1] During the 2022 EOH, our continuous testing pipeline won three awards: 1st place in the Spirit of Innovation, 1st place in Most Engaging, and 2nd place in Outstanding Tech Exhibit

[2] For more details, check UIUC ECE News: https://ece.illinois.edu/newsroom/students-explore-autonomous-vehicles-through-camp

2.5 Summary

In this chapter, we introduced an automated continuous testing pipeline designed specifically for evaluating control algorithms. This pipeline improves the user experience by allowing users to easily submit their controller designs through a website. Once submitted, the controllers are tested across various scenarios within a simulator, and the results and feedback are quickly provided via email and displayed on a public leaderboard. Furthermore, we have shown that this continuous testing pipeline can assist developers in debugging their controllers using the feedback provided.

Chapter 3

Controller Compensation for Unreliable Perception

In the previous chapter, we focus on the testing of the controller, while assume the input (perception information) to the controller is perfect. However, in real life application, perfect perception is often not realizable. Therefore its common to rely on learning-based perception module in many autonomous systems. Yet, the boundary where ML perception does or does not work is poorly characterized. Incorrect perception can lead to unsafe or overtly conservative downstream control actions. Therefore, in this chapter, we are going to propose a method to correct controller for unreliable perception.

Section 3.1 provides motivation to the safe autonomous system while Section 3.2 conducts a literature review. Section 3.3 defines the closed loop and the controller compensation problem while Section 3.4 propose a two-step strategy for correcting ML-based state estimation using Preimages of Perception Contracts (PPC) and a risk heuristic function. Section 3.5 perform extensive simulation-based evaluation of this runtime controller compensation strategy on different vision-based adaptive cruise controllers (ACC modules), in different weather conditions, and road scenarios. Finally, Section 3.6 summarizes the work.

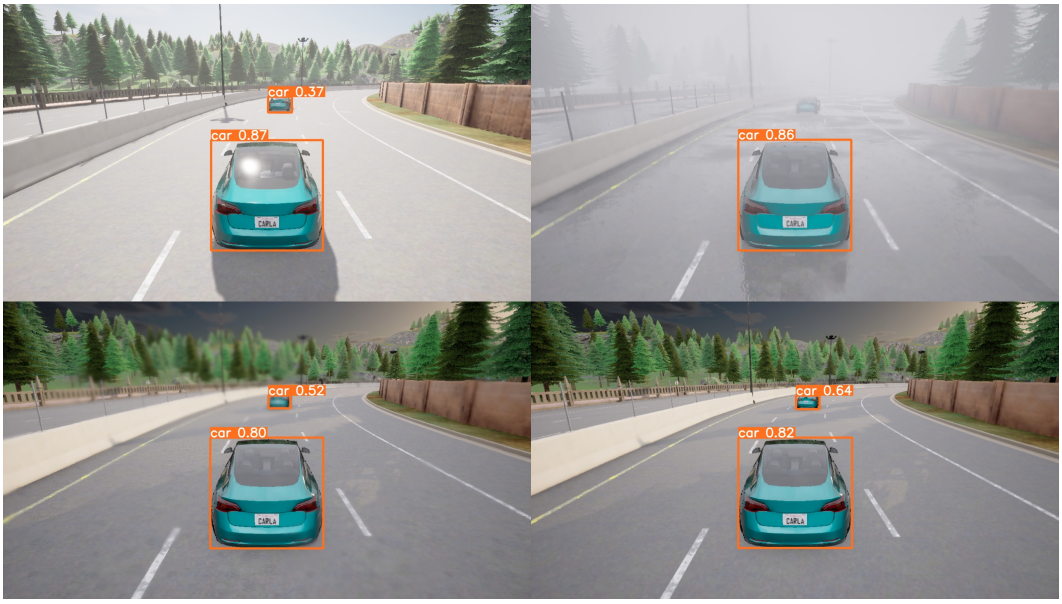


Figure 3.1: Screenshot of simulation of Autonomous Cruise Control (ACC) scenarios under different weather conditions. Under rain and fog, ML-enabled perception designed to estimate crosstrack error and distance to leading vehicles, can have larger errors. For example, it cannot detect a vehicle in the the top-right scenario, and the confidence score (labeled outside the orange box) for detecting nearby vehicle is lower. Our approach corrects for such perception errors for a family of controllers.

3.1 Overview on Perception Contract Methodology

Machine learning (ML) can play an important role in the creation of cyber-physical and autonomous systems that operate in complex environments. Inexpensive sensors coupled with powerful pre-trained ML models can serve as an attractive alternative to traditional sensing and state estimation methods. At the same time, it is also well-known that ML models suffer from fragile decision boundaries and adversarial examples [39]. Indeed, a major AI safety concern is the potentially out-sized impact of this lack of robustness in safety critical applications. On the other hand, for traditional control systems and cyber-physical systems (CPS), there is a rich body of techniques for model-based design and analysis of systems that are robust to certain types of disturbances [40]–[43]. These methods provide rigorous guarantees about safety, robustness, and stability, but only in relatively structured environments and with simple sensor models. In this paper, we explore the middle-ground and aim to provide semi-formal safety guarantees for AI-enabled CPS.

Consider a vision-based Autonomous Cruising Control (ACC) system in which a vehicle (controller) relies on perception for lane keeping and maintaining safe distance from a leading vehicle. The perception module h (specifically Yolo [44] and LaneNet [45] in this example) provides estimates of distance to the leading vehicle (provided there is one) and crosstrack error with respect to the lane center. The vehicle controller g uses these observations or estimates to compute the steering, throttle, and brake inputs for the vehicle (see Figure 3.2). Setting aside the ML-based state estimator h for a moment, we observe that the rest of this system is a classical cyber-physical system (CPS). If only we could assume that the state estimator were perfect h^* or that it came with reasonable error bounds, then a whole arsenal of tools would become available for design and analysis: We could get stability envelops using Lyapunov analysis, we could compute invariants using reachability, and so on. However, like other machine learning models, h as implemented in Yolo and LaneNet do not have error specifications and is fragile. Further, its output estimates depend on environmental factors like lighting and weather in complex ways, which can violate safety of ACC (see Figure 3.1). This near slip from grasp motivates us to investigate the following problem: Given a controller g that preserves some invariant R with perfect perception h^*, and given a real perception module h (implemented using ML and thus afflicted by fragility, environment, lack of specs), can we modify or correct the output of h so that the resulting system preserves the safety invariant R. We call this the *controller compensation problem*, and the formal statement appears in Definition 3.1.

We propose a solution to this problem using *preimages of perception contracts*. A perception contract M [46], [47] bounds the output of an ML-based state estimator *as a function of the ground truth state*, so that it preserves a closed-loop invariant such as R (see Definition 3.1). Different representations for perception contracts have been proposed using piece-wise affine set-valued functions [46] and decision trees [47]. Perception contracts have been used to verify a vision-based lane-keeping system [46], automated landing for a drone [48], and distributed formation flight [49]. A closely related notion of weakest preconditions has been used to analyze vision-based taxiing in the probabilistic setting [50]. While these ideas have been fruitful for offline verification, at runtime, the ground truth

state is not available, and therefore, perception contracts cannot be used for monitoring or for taking corrective actions.

Our proposed method uses the simple observation that the *preimage M^{-1} of a preception contract (PPC)*—which outputs the uncertainty in state given an input observation—can be used at runtime. Specifically, if the set of states $M^{-1}(y)$ corresponding to an observation y shows no risk of violating the target invariant R, then no corrective action is needed. On the other hand if $M^{-1}(x)$ could potentially compromise safety, then some corrective action may be necessary.

Calculating the preimage perception contract (PPC) directly is a complex task, and hence, we use a Bayesian Neural Network (BNN) to construct the PPC based on data.

We then introduce a concept of risk, which defines a monotonous function relative to the safety property across different states. Our proposed risk heuristic, guides control actions by prioritizing the states within the inferred uncertain set that pose the highest level of risk, as illustrated in Figure 3.3

By integrating the correction module at runtime, we were able to effectively recover 33 out of the 45 unsafe scenarios, resulting in a 73% recovery rate. It is important to note that our inability to address all unsafe scenarios may be attributed to conformance violations during the empirical construction of the preimage perception contract (PPC) from data.

Additionally, our evaluations demonstrate that the intervention of the module is minimal and does not impose excessive or overly conservative control measures. In fact, our experiments reveal only a 2.8% increase in the time required to complete tasks when our module is integrated.

In summary, our approach to addressing the runtime controller compensation problem involves the incorporation of the preimage perception contract (PPC) and a risk heuristic into the existing closed-loop system. Initial findings have yielded promising results, indicating that this method is effective in enhancing the system's safety.

3.2 Related Work on Controller Design with Imperfect Perception

With the emergence of increasingly sophisticated sensors, such as cameras, LiDAR, and radar, coupled with the development of advanced perception algorithms, the issue of seamlessly integrating these sensors and their associated machine learning-based algorithms into the controller pipeline has become a prominent subject of research. In recent studies, innovative methods like imitation learning [51] and reinforcement learning utilizing RGB cameras [52] have been introduced to tackle the challenge of vision-based control for autonomous vehicles. However, it is important to note that these approaches are data-driven, and they do not effectively characterize or bound the potential errors in perception, which ultimately limits their capacity to guarantee safety.

Recent research efforts have been primarily directed towards ensuring safety through vision-based control. In the work by Dean et al. [53], the authors synthesized a vision-based controller for autonomous vehicles and carried out theoretical analyses to establish a robust safety guarantee. However, their approach involved simplifying the vehicle model to a linear one. In a subsequent study, as presented in [54], the author proposed a Measurement-Robust Control Barrier Function (MR-CBF) that incorporates an optimization method for synthesizing a safe controller. Dawson et al. [55] focused on working with high-dimensional sensors like LiDAR. They proposed a method for learning a control Lyapunov function (CLF) and a control barrier function (CBF) within the observation space, without making assumptions about the perception module. Additionally, in the work by Chou et al. [56], the authors designed a neural network-based perception module capable of outputting a set of potential states. Subsequently, they applied contraction theory and robust motion planning algorithms to synthesize a robust and safe vision-based controller. This work is closely related to our research; however, our approach involves generating a set of potential states by learning the behavior of a black-box perception model, in contrast to their method, which has to constructs such set of potential states while designing the perception model from data. In [48], authors use the concept of perception contract to design a drone controller

for a safe landing problem, which is very close to the topic we are focusing on; however, their method requires the drone to be static for some time for perception error to decrease.

3.3 Control compensation Problem

In this section, we introduce the different parts making up the perception-based control system and then define the runtime control compensation problem.

The closed-loop system, comprises of three components: the plant with dynamics f, the controller g, the learning-based perception module h.

Plant dynamics

The state of the physical part of the system is denoted by vector $x \in \mathcal{X} \subset \mathbb{R}^n$, where $\mathcal{X}$ is called the state space. We denote by $x[i]$ the i^{th} component of x. For example, for an autonomous vehicle, the state vector x may include its position, velocity, heading, distance to front vehicle, etc.

System-level safety requirements are given in terms of a set of *unsafe states*, $Unsafe \subset \mathcal{X}$, that the overall system must stay away from. The set of safe states, $Safe = \mathcal{X} \setminus Unsafe$, is the complement of the unsafe states.

Example 3.1. *Consider a vehicle (ego) following curvy lanes on a highway with objective of ensuring that the vehicle stays within its lane and does not deviate. The state vector is defined by valuations of several variables:* $\{p_E, v_E, \theta, d_L\}$*, where* p_E *is ego vehicle's pose (position and heading),* v_E *is ego vehicle's velocity,* θ *is the angle between ego's heading and the lane's heading,* d_L *is vehicle's cross-track error with respect to the center of the lane. Since the vehicle enters an unsafe state when a lane departure occurs, then we can define the* unsafe states *as a set,* $Unsafe = \{(p_E, v_E, \theta, d_L) : |d_L| \geq \frac{L}{2}\}$*, where* L *is the lane width.*

The evolution of the plant state is described by a *dynamic function* $f : \mathcal{X} \times \mathcal{U} \mapsto \mathcal{X}$, where $\mathcal{U}$ is the *control input space*. In Example 3.1, the control inputs for the vehicle are throttle $t \in [0, 1]$, brake $b \in [0, 1]$, and steering $s \in [-1, +1]$, here -1 stands for the maximum

left steering input and +1 stands for maximum right steering. Given a state $x \in \mathcal{X}$ and an input $u \in \mathcal{U}$, the next state of the vehicle $x_{t+1} = f(x_t, u_t)$.

Perception and control

The method developed in this paper targets systems in which the control input u is computed in two stages: first, a *perception module* h interprets the signals generated in state x via sensors to produce an observation $y \in \mathcal{Y}$, and then, the *controller* $g : \mathcal{Y} \to \mathcal{U}$ takes as input this observation y and computes the control input (for the plant). A diagram is shown in Figure 3.2. In Example 3.1, h could output cross-track-error d_L as the observation y, and the controller could applies a hard brake whenever the d_L is above some threshold.

The perception module h generates observation y from the actual plant state x. We name a perfect observer $h^* : \mathcal{X} \mapsto \mathcal{Y}$. In Example 3.1, if x is known, the observation y (cross-track error d_L) can be directly obtained from x by dropping the extra state components and retaining d_L as observation. However, in most autonomous systems, usually state information x cannot be directly obtained. Therefore, we rely on an observer h, which encapsulates the behavior of the sensors that generate the raw signals (e.g., images, LIDAR returns) as well as the algorithms (e.g., machine learning models, filters), to generate the observation y from those signals. The signals also critically depend on certain environmental factors (e.g., lighting, fog, rain, etc.). The space of all possible such environmental conditions is denoted by $\mathcal{E}$. Thus, the perception module is modeled as a function $h : \mathcal{X} \times \mathcal{E} \to \mathcal{Y}$. In Example 3.1, to achieve the lane following task, ego vehicle needs to first rely on the camera sensor to generate an RGB image. Then, the analysis of this image is done using a ML algorithm (E.g. LaneNet [45]), to produce lane information and subsequently cross-track-error d_L.

There are several reasons for this two-stage architecture for the computation of u. First, from the control theory point of view, it is standard to think of the whole pipeline as the composition of a state estimator (h) and a controller (g). Loosely speaking, the certainty equivalence principle assures that the optimality of controller design can be preserved by this decomposition, under appropriate assumptions. The access to privileged state information, like the observables, have also been noted to benefit the development of reinforcement

learning-based controllers [57].

Identifying all possible environmental factors that influence h can be a complex problem. This work is based on the premise that domain experts prescribe the dominant factors in $\mathcal{E}$ with respect to which runtime control compensation should be applied.

Closed-loop system

The discrete time evolution of the *closed-loop system* or simply the *system* S, in an environment $e \in \mathcal{E}$, as shown in Figure 3.2, is given by the following:

$$x_{t+1} = f(x_t, g(h(x_t, e)). \tag{3.1}$$

An *execution* in an environment e, is a sequence of states $\alpha(e) = x_0, x_1, \ldots,$ such that for each t, x_{t+1} and x_t satisfy (3.1). With respect to an unsafe set *Unsafe*, the system S is safe over an environment $E' \subseteq \mathcal{E}$ and a set of initial states $X_0 \subseteq \mathcal{X}$, if for each $e \in E'$ and $x_0 \in X_0$, none of the states in $\alpha(e)$ are in *Unsafe*, i.e., the reachable states of S are disjoint from *Unsafe*.

Definition 3.1. *A* control invariant set $R \subseteq \mathcal{X}$ *for the system* S *is a set such that:*

1. $R \subseteq$ *Safe*
2. $\forall x \in R,\ \exists\, u \in \mathcal{U}$ *s.t.* $f(x, u) \in R$.

There has been substantial progress in computing the control invariant sets for systems. Notable techniques include barrier certificates[58], [59], formal controller synthesis[60], control barrier functions[61], and more recently neural barrier functions[62].

These techniques vary in terms of the levels of knowledge needed about f, g, h, their computational complexity, and the level of formal guarantee that they provide. However, our system S includes learning-enabled perception h, which depends on the environment in complex ways, and therefore, some of the existing techniques will not be directly applicable. Instead, our sufficient condition for proving safety of the overall system is based on using control invariant sets for an *idealized controller-observer pair* g^*, h^*. In the two-staged

observer-controller design paradigm, it is indeed common for the controller design to assume that the observer is at least asymptotically correct. The following codifies this assumption about such an idealized observer-controller pair.

Assumption 3.1 (Safety with perfect observer-controller)**.** *There exists a* perfect observer $h^* : \mathcal{X} \to \mathcal{Y}$ *and controller* $g^* : \mathcal{Y} \to \mathcal{U}$ *pair for a given safe invariant set* $R \subseteq Safe$*. That is, for any* $x \in R$*,* $f(x, g^*(h^*(x)) \in R$*.*

Due to environmental uncertainty, sensor noise, and inaccuracies in the Deep Neural Network, the observation $h(x, e)$ may not be accurate and deviate from $h^*(x)$, which could lead to unsafe conditions (e.g., incorrect lane detection causing lane invasion in foggy conditions). As a result, in certain cases, even though the system has a safe controller under perfect perception (Assumption 3.1), there is no guarantee the system is safe with the neural-network based perception module h.

Problem 3.1 (Runtime control compensation problem)**.**

Given:

- *a perfect observer-controller pair* g^*, h^* *and a corresponding control invariant set* R *that proves safety of the closed-loop system* S *with respect to Safe.*
- *an actual (learning-enabled) observer module* h *that depends on environment factors in* $\mathcal{E}$*.*

The objective is to synthesize a controller $\hat{g} : Y \mapsto U$ *such that* $\forall x_0 \in \mathcal{X}_0, e \in \mathcal{E}_0$*, the new closed loop system with* h*,* g *is safe.*

In addition, to ruling out trivial solutions (e.g., always brake and stop), we have a soft-requirement that the new system with g and h should be minimally invasive over S. Metrics for invasiveness are not straightforward to define, and we will consider some examples in Section 3.5.

3.4 Preimage of Perception Contract for Safe Control

Our idea for solving the above problem involves two stages: the first stage infers the uncertainty in the state of the system at runtime from the observations, and the second stage takes control action based on the riskiest states in inferred uncertain set. For reasons that will become clear below, the first stage is called *Preimage of a Perception Contract (PPC)* and the latter is called a *risk heuristic*. The difficulty of inferring the uncertainty in the actual state from observations is addressed using using the recently invented notion of perception contracts

3.4.1 Perception Contracts M

Safety analysis of machine learning-enabled and perception-based control systems is hard since we do not have specifications for the ML modules. Specification are not only necessary for formal verification, but they form the basis for modern, large-scale software engineering by enabling unit tests, modular design, and assume-guarantee reasoning. In [46], [47] the authors introduce the notion of *perception contracts* for addressing this problem. A perception contract M for an actual perception module h, in the context of a closed loop system S, and its control invariant set R, captures two ideas: First, M is an over-approximation of h, over at least some part of the relevant environment space $\mathcal{E}$. This is called the *conformance* of the contract. Second, M preserves system-level correctness of S with respect to R. That is, if M is plugged-in to S, then the resulting closed-loop system, S_M, should preserve R.

Definition 3.2. *For a given perception module $h : \mathcal{X} \times \mathcal{E} \to \mathcal{Y}$, an invariant set $R \subseteq \mathcal{X}$, and an environment $\mathcal{E}' \subseteq \mathcal{E}$, a* perception contract *is a map $M : \mathcal{X} \to 2^{\mathcal{Y}}$ that satisfies the two conditions:*

1. *Conformance: $\forall x \in R, e \in E'$, $h(x, e) \subseteq M(x)$.*
2. *Correctness: $\forall x \in R, f(x, g(M(x)) \subseteq R$.*

Note that here we use $2^{\mathcal{Y}}$ to denote a powerset of $\mathcal{Y}$.

In the previous works, the authors have shown that it is possible to construct such contracts from data for vision-based lane keeping systems and for automated landing systems. The constructed contracts can indeed be used to rigorously prove system-level safety (e.g., car does not leave the lane boundaries). In [46], for example, $M(\cdot)$ is a piece-wise affine set-valued function constructed from data, and the correctness condition is verified using program analysis. Owing to the complexity of the actual perception pipeline h and its complex dependence on the environment E, the conformance property is empirically validated based on input-output data.

3.4.2 Preimage of Perception Contracts M^{-1}

Inspired by perception contracts, in this work we explore how such contracts can be used at runtime to possibly correct perception errors. The challenge we face is that the ground truth state x is not available at runtime to use $M(x)$, even though, $M(\cdot)$ is computed offline. Our key idea is to use the *preimage* of perception contracts, that is, $M^{-1} : Y \to 2^X$. Conceptually, for a given observation $y \in \mathcal{Y}$, its preimage $M^{-1}(y)$ gives the set of all possible states that could generate y, in some environment $\mathcal{E}'$.

For a given perception contract $M : \mathcal{X} \to 2^{\mathcal{Y}}$ we define $M^{-1} : \mathcal{Y} \to 2^{\mathcal{X}}$ as $M^{-1}(y) := \{x \mid y \in M(x)\}$. It follows that, if that for any realizable $y \in \mathcal{Y}$, if observer function h conforms to M over $\mathcal{E}'$, then $h^{-1}(y) \subseteq M^{-1}(y)$.

Proposition 3.1. *Consider any realizable observation $y \in \mathcal{Y}$, such that there exists $x \in \mathcal{X}, e \in \mathcal{E}'$ with $h(x, e) = y$. If h conforms to the perception contract M, then $h^{-1}(y) \subseteq M^{-1}(y)$.*

Proof. Follows from the definitions. Consider any realizable $y \in \mathcal{Y}$ and let $h^{-1}(y) := \{x \mid \exists e \in \mathcal{E}', h(x, e) = y\}$. Consider any $x \in h^{-1}(y)$. Since, h conforms to M over $\mathcal{E}'$, $h(x, e) = y \in M(x)$, for some $e \in \mathcal{E}'$. By definition of M^{-1}, then $x \in M^{-1}(y)$. □

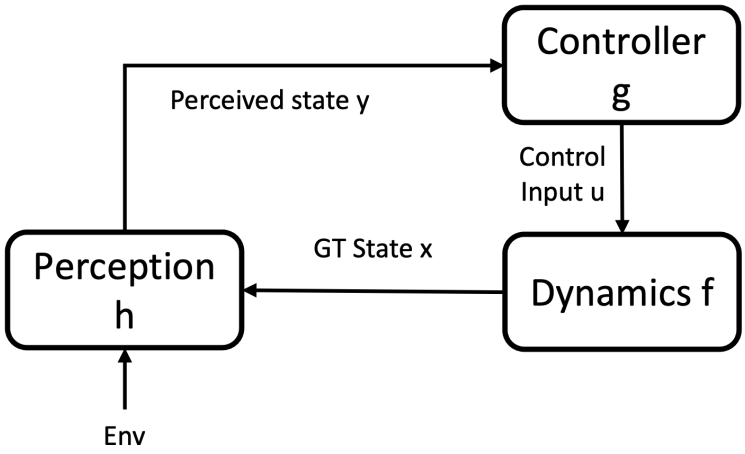


Figure 3.2: This is the block diagram of the closed loop system S, which is mathematically described in Equation (3.1)

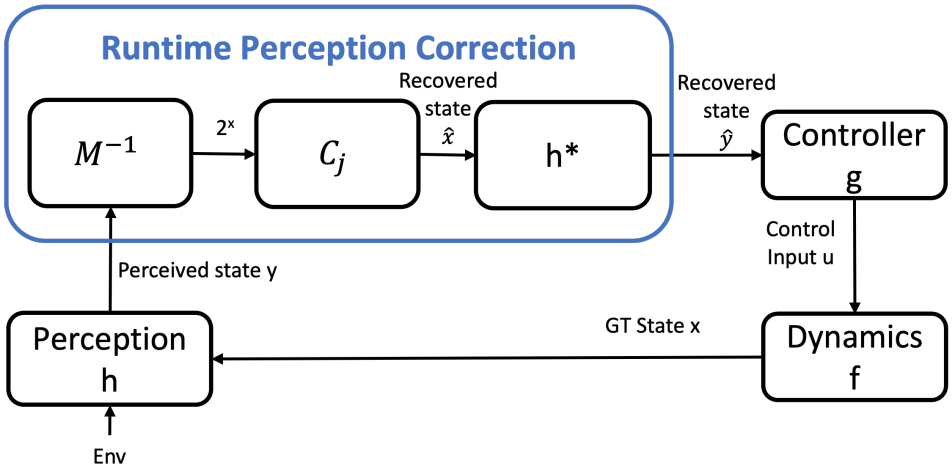


Figure 3.3: This is the block diagram of the new closed-loop system S_{MJ} with our runtime control compensation module from y to $\hat{y}$, which is mathematically described at Equation (3.6)

3.4.3 Constructing PPC from data

Ideally, PPC should achieve perfect conformance, i.e., Proposition 3.1 should always hold. However, this perfect conformance requirement of PPC might be too strong for real autonomous system involving perception. In this paper, instead of directly learning M^{-1} from M, we propose a method to empirically learn a PPC from data using Bayesian Neural Network(BNN). We choose BNN as our model architecture due to its ability to model uncertainty and its ability to generalize from limited training data.

Let dataset $\mathcal{D} := \{(y_i, (x_i, e_i))\}$, where y_i is the input data, (x_i, e_i) is the output label. Traditional neural networks output a point estimate for a given input, that is, they produce one deterministic value (or vector of values) given an input vector, i.e. $(x, e) = f_{\theta^*}(y)$, where f is the neural network parameterized by a deterministic optimal θ^*, and θ^* in practise is obtained by conducting gradient descend to minimize the loss function.

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \sum_{(x_i, e_i, y_i) \in \mathcal{D}} \mathcal{L}(f_\theta(y_i), (x_i, e_i))). \tag{3.2}$$

A Bayesian Neural Network(BNN), in contrast, produces a distribution over possible outputs for an input vector, since the parameters of the BNN are also random variables, i.e. $\widetilde{x}, \widetilde{e} = f_{\widetilde{\theta}}(y)$ where $\widetilde{x}, \widetilde{e}, \widetilde{\theta}$ are both random variables. This probabilistic approach allows for the modeling of uncertainty, which can be essential in many applications, especially when decisions based on the output have significant implications, like in autonomous driving[63].

In this paper, we use a specific kind of BNN, where the prior distribution of the BNN weights follow a Gaussian distribution, i.e., $\theta \sim \mathcal{N}(\mu, \Sigma)$, the optimal weights are parameterized by μ^*, Σ^*. Similarly to traditional neural networks, in practice, the optimal weight is found by doing gradient descend on the loss function.

$$\mu^*, \Sigma^* = \underset{\mu, \Sigma}{\operatorname{argmin}} \sum_{(x_i, e_i, y_i) \in \mathcal{D}} \mathcal{L}(f_\theta(x_i, e_i), y_i) - KL(p(\theta), p(\theta_0)). \tag{3.3}$$

where θ_0 is a normal distribution, i.e., $\theta_0 \sim \mathcal{N}(0, 1)$. The KL divergence is added to to the loss function to prevent the weight from drifting away from normal distribution. We use

a Gaussian distribution as a prior for weight of the BNN, because the Gaussian prior acts as a natural form of regularization to avoid overfitting, and have been used as a common technique in training the BNN. [64]–[66].

During the inference time of the BNN, we can get a set of $\hat{x}, \hat{e}$ given y, simply by sampling the querying the trained BNN N times. We include both x and e during training process to increase the training accuracy. But for the runtime control compensation problem, we only need to infer for x, so we will drop the e dimension when making up M^{-1} as follows:

$$M^{-1}(y) = \{x_i | (x_i, e_i) \sim f_{\theta^*}(y)\}_{i=1}^{N}. \tag{3.4}$$

3.4.4 Risk Heuristic

We introduce a notion of risk which defines a monotonic function over states with respect to the safety property. A similar notion "Monotonic Safety" was used in [67] to remove the uncertainty of non-deterministic models for Statistic Model Checking.

Definition 3.3. *For a given control invariant set $R \subseteq \mathcal{X}$, a* risk function $J : \mathcal{X} \mapsto \mathbb{R}_{\geq 0}$ *assigns a real value to each state such that for any two safe states $x_1, x_2 \in$ Safe, if $J(x_1) \geq J(x_2)$ and there exists an R-preserving control $\bar{u} \in \mathcal{U}$ for x_1, then $\bar{u}$ also preserves R for x_2, that is, $f(x_2, \bar{u})) \in R$.*

Informally, if a more risky state preserves the invariant R, then a less risky state also preserves the same invariant by taking the same control action. In Examaple 3.1, consider the ego vehicle following lanes on curvy highway and two states x_1=$(p_{E_1}, v_E = 5, \theta = 0, d_L = \frac{L}{2})$, x_2=$(p_{E_2}, v_E = 5, \theta = 0, d_L = 0)$, and a risk function $J(x) = |d_L|$, which measures the risk as distance to lane center. It's obvious that $J(x_1) \geq J(x_2)$ since x_1 is on the lane boundary while x_2 is in the lane center, and if the next state of x_1 with full throttle $u = (t = 1, b = 0, s = 0)$ under dynamics is in R, then x_2 with the same full throttle control will also stay in R.

Using the risk heuristic J, we can define a corresponding function C_J that chooses the riskiest state from the intersection of the PPC M^{-1} and the invariant set R. Here we take the intersection because, according to definition 3.1, states outside R cannot guarantee the

existence of control values that will preserve the invariant set R.

$$C_J(M^{-1}(y)) = \underset{x \in M^{-1}(y) \cap R}{\operatorname{argmax}} J(x). \tag{3.5}$$

Finally, we apply the perfect observer h^* and subsequently the control function g to the state returned by $C_j(M^{-1}(y))$ to compute the control input to the plant. The evolution of the resulting corrected closed-loop system S_{MJ} is given by (shown in Figure 3.3):

$$x_{t+1} = f(x_t, g(h^*(C_J(M^{-1}(h(x_t, e)))))). \tag{3.6}$$

We claim that S_{MJ} indeed preserved the safety invariant R provides h conforms to M, and therefore, solves the runtime control compensation problem.

Theorem 3.1. *Given a preimage of a perception contract M^{-1} for the actual perception function h such that h conforms to M and a risk heuristic J for R, the corrected system S_{MJ} described by Equation (3.6) preserves the invariant R for $\mathcal{E}' \in \mathcal{E}$.*

Proof. Consider any $x_t \in R$ to be the input of h. From Proposition 3.1, we know that for any $x_t \in \mathcal{X}, e \in \mathcal{E}'$ with $h(x_t, e) = y$, if h conforms to the perception contract M, we have

$$x_t \in M^{-1}(h(x_t, e)). \tag{3.7}$$

Since $x_t \in R$ then $M^{-1}(h(x_t, e)) \cap R \neq \emptyset$. From Equation (3.5), we have the output of the heuristic function, calling it $\hat{x}$, satisfying

$$\hat{x} = C_J(M^{-1}(h(x_t, e))) \in (M^{-1}(h(x_t, e)) \cap R) \subseteq R. \tag{3.8}$$

From Assumption 3.1, since $\hat{x} \in R$, we know that there exists g such that

$$f(\hat{x}, g(h^*(\hat{x})) \in R. \tag{3.9}$$

From Equation (3.5), since $\hat{x}$ is chosen to maximize the Risk Function J within $(M^{-1}(y) \cap R)$,

then $J(\hat{x}) \geq J(x_t)$. From Definition 3.3, we hence have that

$$x_{t+1} = f(x_t, g(h^*(\hat{x})) \in R \subseteq Safe. \tag{3.10}$$

thus concluding the proof. □

3.5 Case Study: Adaptive Cruising Control

In this section, we introduce the details of an autonomous cruise control system (ACC) as well as presenting evaluation and result on our methods with various ACC scenarios.

3.5.1 Autonomous Cruise Control Problem

The driver assistance feature we study is a combination of lane keeping and adaptive cruise control. Similar autonomy features commercially go by other names such as AutoPilot, Travel Assist, AutoCruise, etc. In typical operation, the ego vehicle moves at a set speed behind a lead vehicle. Both vehicles follow (possibly curving) lanes. If the lead vehicle slows down then the ego vehicle has to maintain safe separation. The ego vehicle's autonomy pipeline uses vision-based perception. We explore three lane configurations: Track 1, a lane with left curve; Track 2, a lane with right curve; Track 3, a straight lane.

The state of the whole system includes $(p_E, p_F, v_E, v_F, \theta, d_L, d_F)$, where p_E, p_F are the pose (position and heading) of the ego vehicle and the leading vehicle respectively, v_E, v_F are the velocity of the ego vehicle and the leading vehicle respectively, θ is the angle between ego's heading and the lane's heading, d_L is ego vehicle's distance to lane center(also known as cross track error), d_F is ego's distance to the leading vehicle. The unsafe set is defined as states where the ego vehicle is out of the lane boundaries or there is a collision, i.e. $Unsafe = \{(p_E, p_F, v_E, v_F, \theta, d_L, d_F) | d_L \geq \frac{L}{2} \vee d_F \leq W\}$, where L is the lane width and W is the vehicle length. The observations given by sensors and ML-based perception will consist of (θ, d_L, d_F), which have the same meaning as the state variables with the same names above.

We use CARLA [68] to create and run the scenarios. CARLA is an open-source platform designed specifically to support the development and validation of autonomous driving systems. For the ego vehicle and the leading vehicle, we use the Carla built-in Tesla Model 3 vehicle's dynamics. We use a reachability tool [69] to approximate the invariant R.

A key advantage of CARLA lies in its ability to create realistic and diverse real-world scenarios, as well as different weather conditions (ranging from clear skies and rain to fog and snow) that could affect the sensor readings. During runtime testing, we consider six weather conditions ranging from demanding environments such as late night, heavy fog, and rain to clear skies (optimal driving conditions): Weather 1 to 5 represent decreasing levels of fog and rain, with Weather 6 characterized by clear skies. We intentionally chose 5 extreme weather conditions to assess their impact on perception h.

Perception h We use two learning-based perception module to detect the front vehicle and the lane: YOLO v8n [44] and LaneNet [45]. YOLO v8n, an evolution of the 'You Only Look Once' series, is utilized for its swift and accurate object detection capabilities, particularly for identifying front leading vehicles. It is known for offering high detection accuracy while ensuring minimal latency. On the other hand, LaneNet is employed specifically for its prowess in lane detection. This architecture combines semantic segmentation with instance segmentation to precisely distinguish between individual lane lines, even in challenging conditions. Together, YOLO v8n and LaneNet form a routine perception backbone, ensuring our autonomous vehicle is consistently aware of its surroundings, which is also a commonly adopted by the research community for autonomous vehicle systems [70]. The two modules will output location of leading vehicle and lanes in the camera frame. Together with a depth camera and the known intrinsic and extrinsic matrix of the camera sensor, we can get the observation $y := (\theta, d_L, d_F)$. We fine-tuned the two models on a customized Carla image dataset.

Controller g We experiment with 3 lateral and 1 longitudinal controller. The controllers are implementations of racing controllers submitted by leading participants of the GRAIC competition [71]. All controllers take the same observations, and produce the control values,

namely throttle, brake and steering.

The lateral controllers we have from GRAIC are a modified verison of Pure Pursuit[72], a modified version of Stanley[73] and a simplified kinematic steering control. The Pure Pursuit algorithm computes the required steering angle based on the vehicle's lookahead distance; the Stanley controller computes the steering based on the vehicle's orientation relative to the path and cross-track-error; the simplified kinematic steering control uses simple geometric solution to the path-following problem.

For longitudinal control, we utilize a PID controller, designed to maintain a constant vehicle speed with high precision. In scenarios demanding instant decisions, such as potential collisions, we incorporate the Responsibility Sensitive Safety (RSS) formula. This ensures the vehicle brakes promptly and safely, taking into account both the vehicle's dynamics and the surrounding environment.

The vehicle controller g is a combination of both lateral control and longitudinal control. We name the combination of Pure Pursuit and PID Controller C_1, the combination of Stanley and PID C_2, the combination of kinematic control and PID C_3.

The combination of three road geometries, six set of weather, and three controllers define 54 different ACC *scenarios*. Each scenario can be modeled as a closed loop system S satisfying Equation (3.1) if we are using perception h. Moreover, each scenario can also be modeled as a new closed loop system S_{MJ} satisfying Equation (3.6) if we use both perception h and our runtime controller compensation (M^{-1}, C_J, h^*).

3.5.2 Construction of Preimage Perception Contract M^{-1}

To obtain training data for generating preimage perception contract, we run a safe controller across 20 sets of different weather conditions and 3 different lane configurations to collect pairs of ground truth states x and observation $y = h(x, e)$. We run the controller for 3 hours and collect 5k set of such pairs to make up the dataset $\mathcal{D} = \{(y_i, (x_i, e_i))\}$, where y_i is the input feature, and (x_i, e_i) is the output label.

We divide the dataset into 80% training dataset, and the rest 20% for validation, as this is a common practice in neural network training. We trained the data for 3000 epoches with

batch size 32 on a BNN, discussed at Section 3.4.3. The BNN has 3 hidden layers with 32, 128, 16 neurons respectively. We assume the prior distribution of each layer's weights follow a Gaussian distribution $\mathcal{N}(0, 0.3)$, we use learning rate 0.01. The layers, neurons and prior distributions are hyper-parameters subject to tuning. We find such hyper-parameters after empirical tryouts.

3.5.3 Choice of Risk Function

The high-level idea behind the specific risk function $J : X \mapsto \mathbb{R}_{\geq 0}$ we chose in this case study is that the closer the vehicle is to unsafe state, the more risk the state will have. This function is monotonically increasing with respect to the inherent risk of the state.

We define the risk as the L_∞ norm of the weighted inverse of the distance to the unsafe region, taken over each dimension of the state space.

$$J(x) = \left\| \begin{bmatrix} \frac{w_1}{x[1]-x^u[1]} \\ \frac{w_2}{x[2]-x^u[2]} \\ \vdots \\ \frac{w_n}{x[n]-x^u[n]} \end{bmatrix} \right\|_\infty \tag{3.11}$$

where $x^u = (0, 0, 0, 0, \frac{\pi}{2}, \frac{L}{2}, W)$ is the unsafe boundary for the state variables $(p_E, p_F, v_E, v_F, \theta, d_L, d_F)$ and the corresponding weight $w = (0, 0, 0, 0, 0.2, 0.4, 0.4)$. w is carefully chosen after empirically tryouts to satisfy Definition 3.3. The idea behind such choice of weight is because we want to assign high risks to vehicle states where it's too close to the leading vehicle (i.e. $d_F = W$ is where collision will happen) or cross track error is too high (i.e. $d_L = \frac{L}{2}$ is where out-of-lane will happen) or heading error is too large(i.e. $\theta = \frac{pi}{2}$ is where ego is almost perpendicular to the lanes).

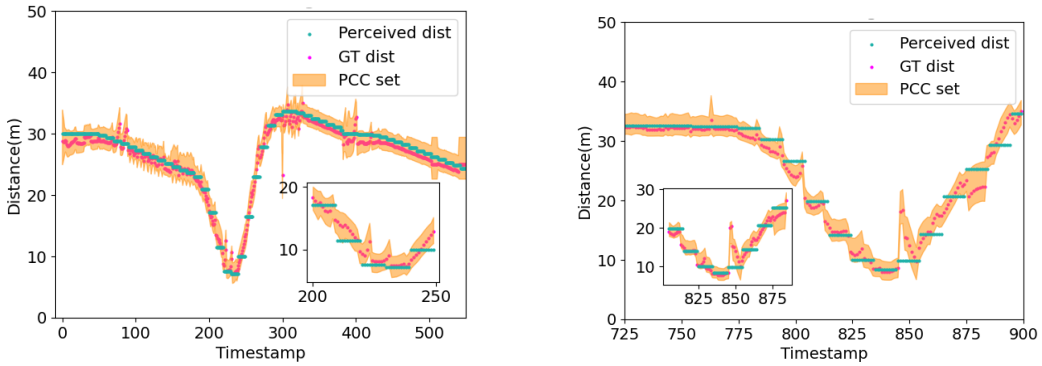
3.5.4 Result: PPC corrects 73% unsafe scenarios

We first ran ego with perfect observer h^* (i.e., ground truth observation provided by Carla simulator), and all 54 scenarios result in safe execution, which indicates that Assumption 3.1

		Track 1		Track 2		Track 3	
Scenarios		h	M^{-1}	h	M^{-1}	h	M^{-1}
Weather 1	C1	X	X	X	X	X	X
	C2	X	X	X	✓	X	✓
	C3	X	✓	X	✓	X	✓
Weather 2	C1	X	X	X	✓	X	✓
	C2	X	✓	X	✓	X	✓
	C3	X	✓	X	✓	X	✓
Weather 3	C1	X	X	X	X	X	X
	C2	X	X	X	✓	X	✓
	C3	X	✓	X	✓	X	✓
Weather 4	C1	X	X	X	X	X	✓
	C2	X	X	X	✓	X	✓
	C3	X	✓	X	✓	X	✓
Weather 5	C1	X	✓	X	✓	X	✓
	C2	X	✓	X	✓	X	✓
	C3	X	✓	X	✓	X	✓
Weather 6	C1	✓	✓	✓	✓	✓	✓
	C2	✓	✓	✓	✓	✓	✓
	C3	✓	✓	✓	✓	✓	✓

Table 3.1: Safety evaluation under a variety of road conditions, weather and controllers. "X" indicates unsafe scenarios while "✓" signifies ego can finish the scenario safely

holds for the three controllers we have. Next, we ran ego with perception module h on the same 54 scenarios. We discovered that, ego safely finished 9 scenarios while the rest 45 scenarios resulted in an unsafe finish (e.g., either a collision with the front vehicle or steering out of lanes). Last, we ran the ego with h complemented by our runtime controller compensation module (M^{-1}, C_J, h^*), and we see that our method maintains safety in 9 scenarios where using h alone is safe; furthermore, it can recover 33 out of the 45 (73 % recover rate) unsafe scenarios in which h failed. We looked into the log of 12 scenarios where our approach cannot recover, and we noticed that it's due to that h^{-1} does not fully conform to the constructed PPC from data(i.e. $x = h^{-1}(y) \notin M^{-1}(h(x, e))$ for some $x \in \mathcal{X}, e \in \mathcal{E}'$ in these 12 scenarios) , which violated the conformance assumption of Theorem 3.1 and, thus, safety can not be guaranteed. Detailed results are provided in Table 3.1.



(a) (b)

Figure 3.4: The two graphs show PPC's prediction on one of the state dimension(distance to front vehicle d_F). Red dot indicates ground truth distance given by h^*, while green dots indicate observed distance given by h. Shaded area represented the constructed PPC M^{-1}

.

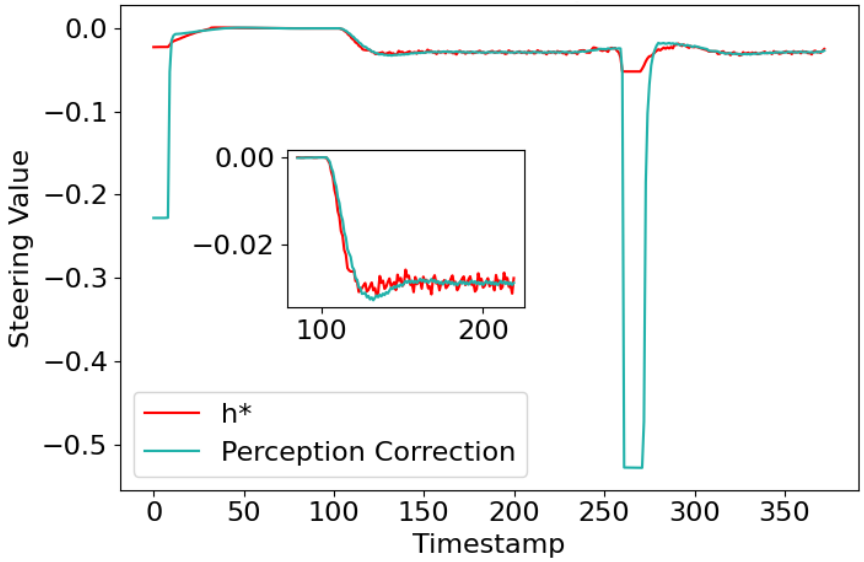


Figure 3.5: Vehicle's steering profile on one particular scenario.

3.5.5 Result: Compensation Intervention are minimally evasive

While the runtime controller compensation module helps to maintain safety, it can also be integrated without inducing overly unnecessary behaviors, such as frequent stops or excessive adjustments. We look at the vehicle's steering profile of one particular example(Track 1, Weather 6, C3) in Figure 3.5. We noticed that most of the time, our runtime controller compensation results in a similar steering value as ones produced by the perfect observer h^*. Most of the time, runtime controller compensation module does not result in unnecessary movement except for timestamp 270 to 280, where the curvature of the map slightly increase and our PPC captures a large uncertainty in perception errors, thus results in a large control value.

Given the large number of experiments, analyzing each scenario by directly comparing control values can be both time-consuming and potentially misleading, due to temporal shifts in these values. As such, we employ a more informative quantitative metric – time to finish the scenario, which helps in gauging whether the vehicle engages in unnecessary behaviors. Due to the fact that each track has different length and potentially different time to finish, we compared the time to finish using perception h and our runtime controller compensation module $(M^{-1}, C_J, h*)$ with time to finish with perfect observer h^*. Then in Table 3.2, we show the percentage of increase in time to finish using our runtime controller compensation module. As presented, the percentage increases in completion time are, for the most part, negligible. The average percentage increase in time to finish the 33 scenarios is 2.8%. This shows that our approach is minimally invasive to the original closed loop system.

3.5.6 Result: PPC can be empirically constructed by BNN

We evaluated whether the constructed PCC is effective by applying the trained BNN model on scenarios not in its training data, and we see that for 91.2% of the testing data, h^{-1} conforms to the constructed M^{-1} (i.e. $x = h^{-1}(y) \in M^{-1}(h(x, e))$ for x, e in the testing dataset). We show an example of how PCC visualizes in Figure 3.4a, 3.4b. We see that, for most of the time, the constructed PCC through BNN always include the ground truth, which

		Track 1	Track 2	Track 3
Weather 1	C1	X	X	X
	C2	X	+1.5%	+4.1%
	C3	+3.2%	+2.6%	+4.6%
Weather 2	C1	X	+1.0%	+3.0%
	C2	+1.3%	+1.5%	+3.6%
	C3	+3.2%	+2.6%	+4.1%
Weather 3	C1	X	X	X
	C2	X	+1.5%	+4.6%
	C3	+3.2%	+2.6%	+5.2%
Weather 4	C1	X	X	+2.5%
	C2	X	+1.5%	+3.6%
	C3	+3.2%	+2.6%	+4.1%
Weather 5	C1	+2.6%	+1.0%	+2.5%
	C2	+1.3%	+1.5%	+3.6%
	C3	+3.2%	+2.6%	+5.2%
Weather 6	C1	+2.6%	+1.0%	+2.5%
	C2	+1.3%	+1.5%	+4.6%
	C3	+3.2%	+2.6%	+4.1%

Table 3.2: Percentage of increase in time to finish by comparing system use runtime controller compensation $(M^{-1}, C_J, h*)$ with system that uses perfect observer h^*. "X" indicates our runtime controller compensation module cannot finish that scenario safely

satisfies the conformance property of M^{-1}. For example, in Figure 3.4a, between timestamp 200 and 250, although observed distance, generated by perception h, slightly differ from ground truth distance, the PCC we constructed(yellow shaded part) through BNN always contain the ground truth.

3.6 Summary

We presented our method, which leverages the preimage perception contract and a risk heuristic, to correct learning-based perception errors for safety during runtime, assuming perfect conformance. Empirically, our approach demonstrated the capability to rectify 73% of unsafe Adaptive Cruise Control (ACC) scenarios stemming from perception errors, while minimizing unnecessary behavior.

3.6.1 Limitation

Ideally, by theorem 3.1, we should be able to recover any unsafe scenarios caused by noisy perception module. However, the empirical result only showed 73% success. This is due to the fact that constructing the Preimage Perception Contract M^{-1} through BNN does not guarantee 100% conformance. In other words, Proposition 3.1 might not always hold; however, in the same time, constructing M^{-1} using any data-driven method(E.g. quantile regression) will face the same issue.

Chapter 4

Conclusions

As the research and industry sectors increasingly focus on the development of autonomous systems, the challenge of creating safe and reliable designs has come to the forefront. This thesis has tackled two major topics within the realm of safe autonomous vehicle systems: continuous testing and controller correction for unreliable perception. First, we introduced a continuous testing pipeline that abstracts away the perception module by assuming perfect perception. This approach enables us to test and evaluate the controller across various scenarios and provide targeted feedback on controller design to users, thereby helping them enhance the robustness and safety of the controller designs. Subsequently, we addressed the controller correction problem through synthesis, aiming to adjust a controller's behavior under real-life, ML-based perception modules. This effort further improves the robustness and safety of controllers when interacting with real-world perception data.

4.1 Future Work

For future work, we plan to enhance the continuous testing pipeline in several key areas:

1. **Scaling the system for a larger number of submissions:** The current system handles a relatively small number of submissions, approximately 10 even during peak times. We aim to improve testing efficiency by parallelizing testing tasks. Utilizing

Kubernetes containers across different machines appears to be a promising approach to facilitate this scaling.

2. **Providing more meaningful feedback to users:** While we currently provide comprehensive data logs for debugging, we intend to offer more insightful interpretations of these results. Identifying scenarios in which the controller is most likely to fail could provide valuable insights. Integrating large language models into the testing pipeline could enhance our ability to analyze and interpret test outcomes more effectively.

We also plan to improve our method for correcting controllers under unreliable perception in the following ways:

1. **Constructing fully conformant PPC:** Our current BNN-based training method does not always guarantee conformance, making our method less than 100% sound. Exploring alternative architectures or models to construct a fully conformant PPC is a critical next step.

2. **Scaling to different autonomous systems:** Since our method currently operates only in simulations for autonomous vehicles, its feasibility on actual hardware remains untested. It would be intriguing to see if this approach can generalize to other systems, such as robotic systems in hardware, including drones. Given that all current experiments are conducted in software, transitioning to hardware applications would be a significant advancement.

References

[1] T. W. Team, "Waymo significantly outperforms comparable human benchmarks over 7+ million miles of rider-only driving," 2023. [Online]. Available: https://waymo.com/blog/2023/12/waymo-significantly-outperforms-comparable-human-benchmarks-over-7-million/.

[2] U. of Michigan Center for Sustainable Systems, "Autonomous vehicles factsheet, pub. no. css16-18," 2023. [Online]. Available: https://css.umich.edu/publications/factsheets/mobility/autonomous-vehicles-factsheet.

[3] J. Gawron, G. Keoleian, R. De Kleine, T. Wallington, and H. C. Kim, "Life cycle assessment of connected and automated vehicles: Sensing and computing subsystem and vehicle level effects," *Environmental Science & Technology*, vol. 52, Feb. 2018. DOI: 10.1021/acs.est.7b04576.

[4] Z. Wadud, D. Mackenzie, and P. Leiby, "Help or hindrance? the travel, energy and carbon impact of highly automated vehicles," *Transportation Research Part A Policy and Practice*, vol. 86, pp. 1–18, Apr. 2016. DOI: 10.1016/j.tra.2015.12.001.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, *You only look once: Unified, real-time object detection*, 2016. arXiv: 1506.02640 [cs.CV].

[6] Z. Wang, W. Ren, and Q. Qiu, *Lanenet: Real-time lane detection networks for autonomous driving*, 2018. arXiv: 1807.01726 [cs.CV].

[7] Z. Zhu, Q. Wang, B. Li, W. Wu, J. Yan, and W. Hu, *Distractor-aware siamese networks for visual object tracking*, 2018. arXiv: 1808.06048 [cs.CV].

[8] N. Wojke, A. Bewley, and D. Paulus, *Simple online and realtime tracking with a deep association metric*, 2017. arXiv: 1703.07402 [cs.CV].

[9] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[10] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt*," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 1478–1483. DOI: 10.1109/ICRA.2011.5980479.

[11] M. Montemerlo, J. Becker, S. Bhat, *et al.*, "Junior: The stanford entry in the urban challenge," in *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*, M. Buehler, K. Iagnemma, and S. Singh, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 91–123, ISBN: 978-3-642-03991-1. DOI: 10.1007/978-3-642-03991-1_3. [Online]. Available: https://doi.org/10.1007/978-3-642-03991-1_3.

[12] M. Werling, J. Ziegler, S. Kammel, and S. Thrun, "Optimal trajectory generation for dynamic street scenarios in a frenet frame," Jun. 2010, pp. 987–993. DOI: 10.1109/ROBOT.2010.5509799.

[13] I. Gog, S. Kalra, P. Schafhalter, M. A. Wright, J. E. Gonzalez, and I. Stoica, "Pylot: A modular platform for exploring latency-accuracy tradeoffs in autonomous vehicles," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 8806–8813.

[14] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems (8th Edition) (What's New in Engineering)*. Pearson, 2018, ISBN: 0134685717. [Online]. Available: https://www.amazon.com/Feedback-Control-Dynamic-Systems-Engineering/dp/0134685717?SubscriptionId=AKIAIOBINVZYXZQZ2U3A&tag=chimbori05-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0134685717.

[15] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun, "Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing," in *2007 American Control Conference*, 2007, pp. 2296–2301. DOI: 10.1109/ACC.2007.4282788.

[16] P. Falcone, F. Borrelli, J. Asgari, H. E. Tseng, and D. Hrovat, "Predictive active steering control for autonomous vehicle systems," *IEEE Transactions on Control Systems Technology*, vol. 15, no. 3, pp. 566–580, 2007. DOI: 10.1109/TCST.2007.894653.

[17] M. Harris, *Behind the scenes of waymo's worst automated truck crash self-driving semi was hit and run in may, then everything went quiet*, 2022. [Online]. Available: https://techcrunch.com/2022/07/01/behind-the-scenes-of-waymos-worst-automated-truck-crash/.

[18] Cruise, *Cruise releases third-party findings regarding october 2*, 2023. [Online]. Available: https://www.getcruise.com/news/blog/2024/cruise-releases-third-party-findings-regarding-october-2/.

[19] G. Rapier, *Tesla's autopilot confused a burger king sign for a stop sign. the fast-food chain turned it into an ad*, 2020. [Online]. Available: https://www.businessinsider.com/tesla-autopilot-mistakes-burger-king-stop-sign-new-ad-2020-6/.

[20] S. Tang, Z. Zhang, Y. Zhang, *et al.*, "A survey on automated driving system testing: Landscapes and trends," *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 5, Jul. 2023, ISSN: 1049-331X. DOI: 10.1145/3579642. [Online]. Available: https://doi.org/10.1145/3579642.

[21] K. Miller, C. Fan, and S. Mitra, "Planning in dynamic and partially unknown environments," *IFAC-PapersOnLine*, vol. 54, no. 5, pp. 169–174, 2021, 7th IFAC Conference on Analysis and Design of Hybrid Systems ADHS 2021, ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2021.08.493. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896321012684.

[22] C. Fan, K. Miller, and S. Mitra, "Fast and guaranteed safe controller synthesis for nonlinear vehicle models," in Jul. 2020, pp. 629–652, ISBN: 978-3-030-53287-1. DOI: 10.1007/978-3-030-53288-8_31.

[23] Y.-S. Wang, N. Matni, and J. C. Doyle, "A system-level approach to controller synthesis," *IEEE Transactions on Automatic Control*, vol. 64, no. 10, pp. 4079–4093, 2019. DOI: 10.1109/TAC.2018.2890753.

[24] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis, "Controller synthesis for timed automata1," *IFAC Proceedings Volumes*, vol. 31, no. 18, pp. 447–452, 1998, 5th IFAC Conference on System Structure and Control 1998 (SSC'98), Nantes, France, 8-10 July, ISSN: 1474-6670. DOI: https://doi.org/10.1016/S1474-6670(17)42032-5. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667017420325.

[25] C. Baier, M. Größer, M. Leucker, B. Bollig, and F. Ciesinski, "Controller synthesis for probabilistic systems (extended abstract)," in Sep. 2010, vol. 155, pp. 493–506, ISBN: 1-4020-8140-5. DOI: 10.1007/1-4020-8141-3_38.

[26] M. Althoff, M. Koschi, and S. Manzinger, "Commonroad: Composable benchmarks for motion planning on roads," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 719–726. DOI: 10.1109/IVS.2017.7995802.

[27] H. J. Cho and M. Behl, "Towards automated safety coverage and testing for autonomous vehicles with reinforcement learning," *ArXiv*, vol. abs/2005.13976, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:218971564.

[28] S. Jha, S. Banerjee, T. Tsai, *et al.*, "Ml-based fault injection for autonomous vehicles: A case for bayesian fault injection," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019, pp. 112–124. DOI: 10.1109/DSN.2019.00025.

[29] A. Corso, P. Du, K. Driggs-Campbell, and M. J. Kochenderfer, "Adaptive stress testing with reward augmentation for autonomous vehicle validatio," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, Auckland, New Zealand: IEEE Press, 2019, pp. 163–168. DOI: 10.1109/ITSC.2019.8917242. [Online]. Available: https://doi.org/10.1109/ITSC.2019.8917242.

[30] S. Feng, Y. Feng, X. Yan, S. Shen, S. Xu, and H. X. Liu, "Safety assessment of highly automated driving systems in test tracks: A new framework," *Accident Analysis & Prevention*, vol. 144, p. 105 664, 2020, ISSN: 0001-4575. DOI: https://doi.org/10.

1016/j.aap.2020.105664. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0001457520302621.

[31] R. Zhang, D. Meng, S. Shen, *et al.*, *Evaluating roadside perception for autonomous vehicles: Insights from field testing*, 2024. arXiv: 2401.12392 [cs.RO].

[32] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "Carla: An open urban driving simulator," Nov. 2017. [Online]. Available: http://arxiv.org/abs/1711.03938.

[33] D. J. Fremont, T. Dreossi, S. Ghosh, X. Yue, A. L. Sangiovanni-Vincentelli, and S. A. Seshia, "Scenic: A language for scenario specification and scene generation," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2019, Phoenix, AZ, USA: Association for Computing Machinery, 2019, pp. 63–78, ISBN: 9781450367127. DOI: 10.1145/3314221.3314633. [Online]. Available: https://doi.org/10.1145/3314221.3314633.

[34] V. S. Babu and M. Behl, "F1tenth. dev-an open-source ros based f1/10 autonomous racing simulator," in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2020, pp. 1614–1620.

[35] B. M. Bill Hoffman, *Continuous testing drives innovation with drake robotics software*, 2024. [Online]. Available: https://www.kitware.com/continuous-testing-drives-innovation-with-drake-robotics-software/.

[36] *How to build a ci/cd pipeline with github actions in four simple steps*. [Online]. Available: https://github.blog/2022-02-02-build-ci-cd-pipeline-github-actions-four-steps/.

[37] M. Jiang, K. Miller, D. Sun, *et al.*, "Continuous integration and testing for autonomous racing software: An experience report from graic," May 2021.

[38] Y. Li, H. Zhu, K. Braught, K. Shen, and S. Mitra, "Verse: A python library for reasoning about multi-agent hybrid system scenarios," in *Computer Aided Verification*, C. Enea and A. Lal, Eds., Cham: Springer Nature Switzerland, 2023, pp. 351–364, ISBN: 978-3-031-37706-8.

[39] R. R. Wiyatno, A. Xu, O. Dia, and A. de Berker, *Adversarial examples in modern machine learning: A review*, 2019. arXiv: 1911.05268 [cs.LG].

[40] S. Mitra and D. Liberzon, *Stability of hybrid automata with average dwell time: An invariant approach*, http://theory.lcs.mit.edu/~mitras/research/cdc04-full.ps.gz.

[41] T. Dreossi, D. J. Fremont, S. Ghosh, *et al.*, "VerifAI: A toolkit for the formal design and analysis of artificial intelligence-based systems," in *31st International Conference on Computer Aided Verification (CAV)*, Jul. 2019.

[42] A. Anta, R. Majumdar, I. Saha, and P. Tabuada, "Automatic verification of control system implementations," in *Proceedings of the Tenth ACM International Conference on Embedded Software*, ser. EMSOFT '10, Scottsdale, Arizona, USA: ACM, 2010, pp. 9–18, ISBN: 978-1-60558-904-6. DOI: 10.1145/1879021.1879024. [Online]. Available: http://doi.acm.org/10.1145/1879021.1879024.

[43] N. Chan and S. Mitra, "Verifying safety of an autonomous spacecraft rendezvous mission," in *ARCH17. 4th International Workshop on Applied Verification of Continuous and Hybrid Systems, collocated with Cyber-Physical Systems Week (CPSWeek) on April 17, 2017 in Pittsburgh, PA, USA*, 2017, pp. 20–32. [Online]. Available: http://www.easychair.org/publications/paper/342723.

[44] D. Reis, J. Kupec, J. Hong, and A. Daoudi, *Real-time flying object detection with yolov8*, 2023. arXiv: 2305.09972 [cs.CV].

[45] D. Neven, B. D. Brabandere, S. Georgoulis, M. Proesmans, and L. V. Gool, "Towards end-to-end lane detection: An instance segmentation approach," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, Changshu, Suzhou, China: IEEE Press, 2018, pp. 286–291. DOI: 10.1109/IVS.2018.8500547. [Online]. Available: https://doi.org/10.1109/IVS.2018.8500547.

[46] C. Hsieh, Y. Li, D. Sun, K. Joshi, S. Misailovic, and S. Mitra, "Verifying controllers with vision-based perception using safe approximate abstractions," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 11, pp. 4205–4216, 2022.

[47] A. Angello, C. Hsieh, P. Madhusudan, and S. Mitra, "Perception contracts for safety of ml-enabled systems," in *Proc. of the ACM on Programming Languages (PACMPL), OOPSLA*, 2023.

[48] D. Sun, B. C. Yang, and S. Mitra, *Learning-based perception contracts and applications*, 2023. arXiv: 2309.13515 [cs.RO].

[49] C. Hsieh, Y. Koh, Y. Li, and S. Mitra, *Assuring safety of vision-based swarm formation control*, 2023. arXiv: 2210.00982 [cs.MA].

[50] C. S. Păsăreanu, R. Mangal, D. Gopinath, *et al.*, "Closed-loop analysis of vision-based autonomous systems: A case study," in *Computer Aided Verification*, C. Enea and A. Lal, Eds., Cham: Springer Nature Switzerland, 2023, pp. 289–303, ISBN: 978-3-031-37706-8.

[51] Z. Xu, Y. Sun, and M. Liu, "Icurb: Imitation learning-based detection of road curbs using aerial images for autonomous driving," *CoRR*, vol. abs/2103.17118, 2021. arXiv: 2103.17118. [Online]. Available: https://arxiv.org/abs/2103.17118.

[52] B. Tan, N. Xu, and B. Kong, "Autonomous driving in reality with reinforcement learning and image translation," *CoRR*, vol. abs/1801.05299, 2018. arXiv: 1801.05299. [Online]. Available: http://arxiv.org/abs/1801.05299.

[53] S. Dean, N. Matni, B. Recht, and V. Ye, "Robust guarantees for perception-based control," in *Learning for Dynamics and Control*, PMLR, 2020, pp. 350–360.

[54] S. Dean, A. Taylor, R. Cosner, B. Recht, and A. Ames, "Guaranteeing safety of learned perception modules via measurement-robust control barrier functions," in *Conference on Robot Learning*, PMLR, 2021, pp. 654–670.

[55] C. Dawson, B. Lowenkamp, D. Goff, and C. Fan, "Learning safe, generalizable perception-based hybrid control with certificates," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 1904–1911, 2022.

[56] G. Chou, N. Ozay, and D. Berenson, "Safe output feedback motion planning from images via learned perception modules and contraction theory," in *International Workshop on the Algorithmic Foundations of Robotics*, Springer, 2022, pp. 349–367.

[57] T. Chen, J. Xu, and P. Agrawal, *A system for general in-hand object re-orientation*, 2021. arXiv: 2111.03043 [cs.RO].

[58] S. Prajna and A. Jadbabaie, "Safety verification of hybrid systems using barrier certificates," in *Hybrid Systems: Computation and Control*, R. Alur and G. J. Pappas, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 477–492, ISBN: 978-3-540-24743-2.

[59] S. Prajna and A. Rantzer, "On the necessity of barrier certificates," *IFAC Proceedings Volumes*, vol. 38, no. 1, pp. 526–531, 2005, 16th IFAC World Congress, ISSN: 1474-6670. DOI: https://doi.org/10.3182/20050703-6-CZ-1902.00743. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667016367556.

[60] T. Badings, L. Romao, A. Abate, and N. Jansen, *Probabilities are not enough: Formal controller synthesis for stochastic dynamical models with epistemic uncertainty*, 2022. arXiv: 2210.05989 [eess.SY].

[61] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," *CoRR*, vol. abs/1903.11199, 2019. arXiv: 1903.11199. [Online]. Available: http://arxiv.org/abs/1903.11199.

[62] C. Dawson, S. Gao, and C. Fan, *Safe control with learned certificates: A survey of neural lyapunov, barrier, and contraction methods*, 2022. arXiv: 2202.11762 [cs.RO].

[63] R. McAllister, Y. Gal, A. Kendall, *et al.*, "Concrete problems for autonomous vehicle safety: Advantages of bayesian deep learning," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 2017, pp. 4745–4753. DOI: 10.24963/ijcai.2017/661. [Online]. Available: https://doi.org/10.24963/ijcai.2017/661.

[64] A. Graves, "Practical variational inference for neural networks," in *Advances in Neural Information Processing Systems*, J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, Eds., vol. 24, Curran Associates, Inc., 2011.

[65] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, *Weight uncertainty in neural networks*, 2015. arXiv: 1505.05424 [stat.ML].

[66] Y. Gal and Z. Ghahramani, *Bayesian convolutional neural networks with bernoulli approximate variational inference*, 2016. arXiv: 1506.02158 [stat.ML].

[67] M. Cleaveland, I. Ruchkin, O. Sokolsky, and I. Lee, "Monotonic safety for scalable and data-efficient probabilistic safety analysis," in *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*, Los Alamitos, CA, USA: IEEE Computer Society, May 2022, pp. 92–103. DOI: 10.1109/ICCPS54341.2022.00015. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICCPS54341.2022.00015.

[68] A. Dosovitskiy, G. Ros, F. Codevilla, A. M. López, and V. Koltun, "CARLA: an open urban driving simulator," *CoRR*, vol. abs/1711.03938, 2017. arXiv: 1711.03938. [Online]. Available: http://arxiv.org/abs/1711.03938.

[69] C. Fan, B. Qi, S. Mitra, and M. Viswanathan, "DRYVR: data-driven verification and compositional reasoning for automotive systems," *CoRR*, vol. abs/1702.06902, 2017. arXiv: 1702.06902. [Online]. Available: http://arxiv.org/abs/1702.06902.

[70] I. Gog, S. Kalra, P. Schafhalter, M. A. Wright, J. E. Gonzalez, and I. Stoica, "Pylot: A modular platform for exploring latency-accuracy tradeoffs in autonomous vehicles," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, Xi'an, China: IEEE Press, 2021, pp. 8806–8813. DOI: 10.1109/ICRA48506.2021.9561747. [Online]. Available: https://doi.org/10.1109/ICRA48506.2021.9561747.

[71] M. Jiang, K. Miller, D. Sun, *et al.*, "Continuous integration and testing for autonomous racing software: An experience report from graic," *IEEE ICRA 2021, International Conference on Robotics and Automation, Workshop on OPPORTUNITIES AND CHALLENGES WITH AUTONOMOUS RACING*, DOI: 10.13140/RG.2.2.28270.33605. [Online]. Available: https://par.nsf.gov/biblio/10296575.

[72] V. Sukhil and M. Behl, "Adaptive lookahead pure-pursuit for autonomous racing," *CoRR*, vol. abs/2111.08873, 2021. arXiv: 2111.08873. [Online]. Available: https://arxiv.org/abs/2111.08873.

[73] A. AbdElmoniem, A. Osama, M. Abdelaziz, and S. Maged, "A path-tracking algorithm using predictive stanley lateral controller," *International Journal of Advanced Robotic Systems*, vol. 17, p. 172 988 142 097 485, Nov. 2020. DOI: 10.1177/1729881420974852.