Certified Control in Autonomous Vehicles with Visual Lane Finding and LiDAR

by

Jeff Chow

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2021

Author . . . .
Department of Electrical Engineering and Computer Science
Jan 15, 2021

Certified by . . . .
Daniel Jackson
Professor, Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . .
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Certified Control in Autonomous Vehicles with Visual Lane Finding and LiDAR

by

Jeff Chow

Submitted to the Department of Electrical Engineering and Computer Science
on Jan 15, 2021, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Certified Control is a safety architecture for autonomous vehicles in which the controller must provide evidence to a runtime monitor that the actions it takes are safe. If the monitor deems the current state of the vehicle as unsafe, it intervenes by signalling the vehicle actuators to brake. In this work, we demonstrate how Certified Control can be used to increase safety through two implementations: one involving visual lane-finding and one involving LiDAR. Through experiments utilizing real driving data, a robot racecar, and simulation software, we show examples in which these runtime monitors detect and mitigate unsafe scenarios.

Thesis Supervisor: Daniel Jackson
Title: Professor, Electrical Engineering and Computer Science

Acknowledgments

I would like to thank my advisor Daniel Jackson for his guidance and support throughout the past two years. I'm thankful for participating in a project so closely aligned with my interests, for being given the opportunity to present our group's work, and for learning so much under his mentorship.

It was a pleasure working with members of the Software Design Group for this project. Special thanks to Valerie Richmond for her encouragement and help with experiments (even after graduating), Mike Wang for help with the racecar and CARLA, and Uriel Guajardo for his assistance in designing and executing experiments.

Finally, I would not have gotten here without the support of my friends and family. Thank you Roy, Jeana, Martin, Kristen, and Andy T. for your friendship, especially in the past several months. Thank you Mom, Jesse, and Andy M. for your love and for always being there for me.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

If autonomous vehicles are to become widespread, it will be necessary not only to ensure a high level of safety but also to justify our confidence that such a level has been achieved. More traditional methods of ensuring confidence such as statistical testing and formal verification of software are either not comprehensive enough to clearly establish safety or would require a vast amount of resources to become comprehensive. This motivates the idea of a small trusted base, a piece of software that can make run-time checks to provide run-time safety assurances. This trusted base must be simple enough to be verifiable, but must also have sufficient information to ensure safety even in a system as complicated as an autonomous vehicle.

1.1.1 The problem of safety

The problem of safety for self-driving cars has two distinct aspects. First is the reality of numerous accidents, many fatal, either involving fully autonomous cars—such as the Uber that killed a pedestrian in Tempe, Arizona [1]—or cars with autonomous modes—such as the Tesla models, which have spawned a rash of social media postings in which owners have demonstrated the propensity of their own cars to repeat mistakes that had resulted in fatal accidents. The metric of "miles between disengagements," made public for many companies by the California DMV [2], has revealed the troublingly small distance that autonomous cars are apparently able to travel without human intervention. Even if the disengagement metric is crude and includes disengagements that are not safety-related [3], the evidence suggests that the technology still has far to go.

Second, and distinct from the actual level of safety achieved, is the question of confidence. Our society's willingness to adopt any new technology relies on our confidence that catastrophic failures are unlikely. But, even for the designs with the best records of safety to date, the number of miles traveled falls far short of the distance that would be required to provide statistical confidence of a failure rate that matches (or improves on) the failure rate of an unimpaired human driver. Even though Waymo, for example, claims to have covered 20 million miles—a truly impressive achievement—this still pales in comparison to the 275 million miles that would have to be driven for a 95% confidence that fully autonomous vehicles have a fatality rate lower than a human-driven car (one in 100 million miles) [4].

1.1.2 An alternative to testing

Statistical testing is the gold standard for quality control for many products (such as pharmaceuticals) because it is independent of the process of design and development. This independence is also its greatest weakness, because it denies the designer the opportunity to use the structure of the artifact to bolster the safety claim, and at the same time fails to focus testing on the weakest points of the design, thus reducing the potency of testing for establishing near-zero likelihood of catastrophic outcomes.

One alternative to statistical testing is to construct a "safety case:" an argument for safety based on the structure of the design [5]. The quality of the argument and the extent to which experts are convinced then becomes the measure of confidence. This approach lacks the scientific basis of statistical testing, but is widely accepted in all areas of engineering, especially when the goal is to prevent catastrophe rather than a wider range of routine failures. For example, confidence that a new skyscraper will not fall down relies not on testing (since each design is unique, and non-destructive tests reveal little) but on analytical arguments for stability and resilience in the presence of anticipated forces. In the UK, the use of safety cases is mandated by a government standard [6] for critical systems such as nuclear power plants.

In software too, there is growing interest in safety cases (or, more generally, assurance or dependability cases) [7]. For a cyber-physical system, the safety case is an argument that a machine, in the context of its environment, meets certain critical requirements. This argument is a chain of many links, including: the specification of the software that controls the machine, the physical properties of the environment (including peripheral devices such as sensors and actuators that mediate between the machine and the environment), and assumptions about the behavior of human users and operators. Each link in the chain needs its own justification, and together they must imply the requirements. Ideally, the justification takes the form of a mathematical proof: in the case of software, for example, a verification proof that the code meets the specification. But some links will not be amenable to mathematical reasoning: properties of the environment, and of physical peripherals, for example, must be formulated and justified by expert inspection.

1.1.3 The Cost of Verification

For software-intensive systems, the software itself can become a problematic link in the chain. Complex systems require complex software, and that inevitably leads to subtle bugs. Because the state space of a software system is so large, statistical testing can only cover a tiny portion of the space, and thus cannot provide confidence in its correctness. So for high confidence, verification seems to be the only option.

Unfortunately, verification is prohibitively expensive. Even for software produced under a very rigorous process that does not involve verification, the cost tends to be orders of magnitude higher than for conventional software development. NASA's flight software, for example, has cost over $1,000 per line of code, where conventional software might cost $10 to $50 per line [8]. Verifying a large codebase is a Herculean task. It may not be impossible, as demonstrated by the success of recent projects to verify an entire operating system kernel or file system stack. But it typically requires enormous manual effort. SEL4, a verified microkernel, for example, comprised about 10,000 lines of code, but required about 200,000 lines of hand-authored proof, whose production took about 25-30 person years of work [9].

1.1.4 Small trusted bases

One way to alleviate the cost of verification is to design the software system so that it has a small trusted base. The trusted base is the portion of the code on which the critical safety properties depend; any part of the system outside the trusted base can fail without compromising safety. This idea is exploited, for example, in secure transmission protocols that employ encryption (and is generalized in the "end-to-end principle" [10]). So long as the encryption and decryption algorithms that execute at the endpoints are correct, one can be sure that message contents are not corrupted or leaked; the network components that handle the actual transmission, in particular, need not be secure, because any component that lacks access to the appropriate cryptographic keys cannot expose the contents of messages or modify them without the alteration being detectable.

Of course, the claim that some subset of the components of a system form a trusted base—really that the other components fall outside the trusted base—must itself be justified in the safety case. It must be shown not only that the properties established by the trusted base are sufficient to ensure the desired end-to-end safety properties, but also that the trusted base is immune to external interference that might cause it to fail (a property often achieved by using separation mechanisms to isolate the trusted base).

1.1.5 Runtime monitors and safety controllers

One widely-used approach is to augment the system with a runtime monitor that checks (and enforces) a critical safety property. If isolated appropriately, and if the check is sufficient to ensure safety, the monitor serves as a trusted base.

For safety-critical systems, the runtime monitor might be an entire controller in its own right. This "safety controller" oversees the behavior of the main controller, and takes over when it fails. If the safety controller is simpler than the main controller, it serves as a small trusted base (along with whatever arbiter is used to ensure that it can veto the main controller's outputs). This scheme is used in the Boeing 777, which runs a complex controller that can deliver highly optimized behavior over a wide range of conditions, but at the same time runs a secondary controller based on the control laws of the 747, ensuring that the aircraft flies within the envelope of the earlier (and simpler) design [11].

The Simplex architecture ([12, 13]) embodies this idea in a general form. Two control subsystems are run in parallel. The high assurance subsystem is meticulously developed with conservative technologies; the high performance subsystem may be more complex, and can use technologies that are hard to verify (such as neural nets). The designer identifies a safe region of states that are within the operating constraints of the system and which exclude unsafe outcomes (such as collisions). A smaller subset of these states, known as the *recovery* region is then defined as those states from which the high assurance subsystem can always recover control and remain within the safe region. The boundary of the recovery region is then used as the switching condition between the two subsystems.

1.1.6 The problem of perception in autonomous cars

The safety controller approach relies on the assumption that the controller itself is the most complex part of the system—that from the safety case point of view, the correctness of the controller is the weakest link in the argument chain. But in the context of autonomous cars, perception—the interpretation of sensor data—is more complicated and error-prone than control. In particular, determining the layout of the road and the presence of obstacles typically uses vision systems that employ large and unverified neural nets.

In standard safety controller architectures (such as Simplex [12, 13]), only the controller itself has a safety counterpart; even if sensors are replicated to exploit some hardware redundancy, the conversion of raw sensor data into controller inputs is performed externally to the safety controller, and thus belongs to the trusted base.

This means that the safety case must include a convincing argument that this conversion, performed by the perception subsystem, is performed correctly. This is a formidable task for at least two reasons. First, there is no clear specification against which to verify the implementation. Machine learning is used for perception precisely because no succinct, explicit articulation of the expected input/output relationship is readily available. Second, state of the art verification technology cannot handle the particular complications of deep neural networks—especially their their scale and their use of non-linear activation functions (such as ReLU [14]) which confound automated reasoning algorithms such as SMT and linear programming [15].

An alternative possibility is to not include perception functions in the trusted base. Instead, one could perhaps use a runtime monitor that incorporates both a safety controller and a simplified perception subsystem. Initially, this approach seemed attractive to us, but we came to the conclusion that it was not in fact viable. In the next section we explain why.

1.1.7 Desiderata for a runtime monitor: choose two

To see why a runtime monitor that employs simplified perception is not a solution to the safety problem for autonomous cars, we shall enumerate three critical properties that a monitor should obey, and argue that they are mutually inconsistent (at least for the conventional design).

The first property is that the monitor should be **verifiable**. That is, it should be small and simple enough to be amenable to formal verification (or perhaps to fully exhaustive testing). If not, the monitor brings no significant benefit in terms of confidence in the overall system safety (beyond the diversity of an additional implementation, which brings less confidence than is often assumed [13]).

The second property is that the monitor should be **honest**. It should intervene only when necessary, namely when proceeding with the action proposed by the main controller would be a safety risk. Applying emergency braking on a highway when there is no obstacle, for example, is clearly unacceptable. Even handing over control to a human driver is problematic, due to vigilance decrement [16].

The third property is that the monitor should be **sound**. This means that it should ensure the safety of the vehicle within an envelope that covers a wide range of typical conditions. It is not sufficient, for example, for the monitor to merely reduce the severity of a collision when it might have been able to avert the collision entirely.

Unfortunately, it seems that these three properties cannot be achieved simultaneously in a classic monitor design. The problem, in short, is that the combination of honesty and soundness requires a sophisticated perception system, leading to unverifiable complexity. It is easy to make a monitor that is sound but not honest simply by not allowing the vehicle to move; and conversely it is easy to make one that is honest but not sound by not preventing any collisions at all.

This does not mean that monitors that fail to satisfy all three properties are not useful—only that such monitors are not sufficient to ensure safety. Automatic emergency braking (AEB) systems are deployed in many cars now, and use radar to determine when a car in front is so close that braking is essential. But because AEB might brake too late to prevent a collision, it is not generally sound. Responsibility-Sensitive Safety systems [17], on the other hand, use the car's full sensory perception systems to identify and locate other cars and pedestrians, and can therefore be both honest and sound, but due to the complexity of the perception are not verifiable.

1.2 Certified Control: A New Approach

Certified control (Fig. 1-1) is a new architecture that centers on a different kind of monitor. As with a conventional safety architecture, a monitor vets proposed actions emanating from the main controller. But the certified control monitor does not check actions against its *own* perception of the environment.

Instead, it relies on the main perception and control subsystems to provide a *certificate* that embodies evidence that the situation is safe for the action at hand. The certificate is designed to be unforgeable, so that even a malicious agent could not convince the monitor that an unsafe situation is safe. The evidence comprises sensor readings that have been selected to support a safety case in favor of the proposed action; because these sensor readings are only *selected* by the main perception and control subsystems (and are signed by the sensor units that produced them), they cannot be faked.

A certificate contains the following elements: (1) the proposed action (for example, driving ahead at the current speed); (2) some signed sensor data (for example, a set of LiDAR points or a camera image); (3) optionally, some interpretive data. This data indicates what inference should be drawn from the sensor readings. For example, if the sensor data comprises LiDAR points intended as evidence that the nearest obstacle is at least some distance away, the interpretive data might be that distance; if the sensor data is an image of the road ahead, the interpretive data might be the purported lane lines.

The evidence and interpretive data are evaluated by the certificate checker using a predefined runtime safety case. As an example, consider a certificate that proposes the action to continue to drive ahead using LiDAR data. In this case, the LiDAR points argue that there is no obstacle along the path; each LiDAR reading provides direct physical evidence of an uninterrupted line from the LiDAR unit to the point of reflection. Together, a collection of such readings, covering the cross section of the path ahead with appropriate density, indicates absence of an obstacle larger than a certain size.

Compare this with a classic monitor that interprets the LiDAR unit's output itself. The LiDAR point cloud is likely to include points that should be filtered out. In snow, for example, there will be reflections from snowflakes. Performing snow filtering would introduce complexity and likely render the monitor unverifiable. On the other hand, a simple monitor would not attempt to identify snow, and could set a low or a high bar on intervention—requiring, say, that 10% or 90% of points in the LiDAR point cloud show reflections within some critical distance. The low bar would result in a monitor that violates soundness, failing, for example, to prevent collision with a motorcycle whose cross section occludes less than the 10% of points. The high bar would result in a monitor that violates honesty, since it would likely cause an

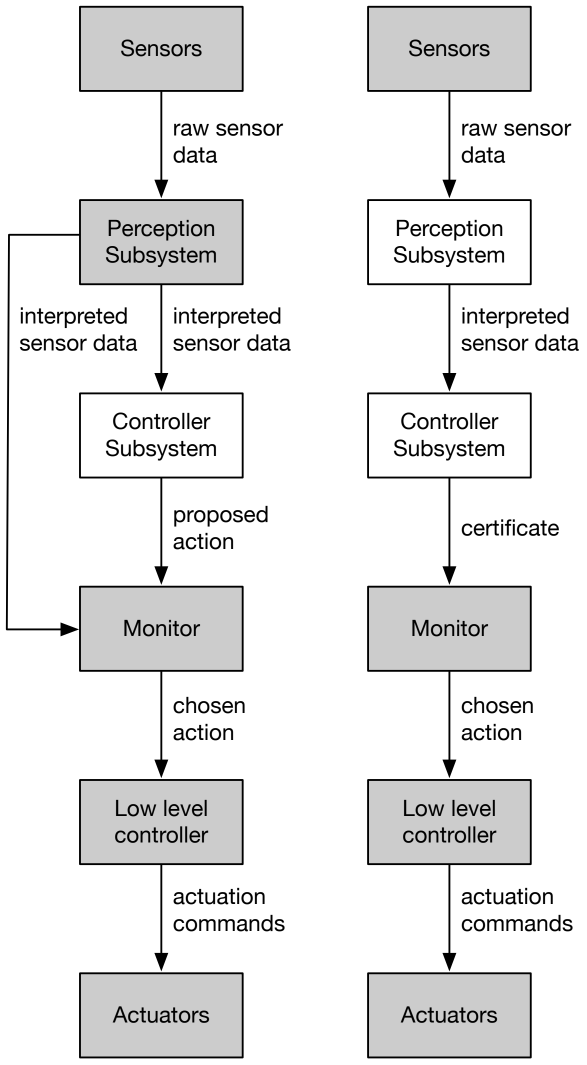


Figure 1-1: A conventional runtime monitor (left) and certified control monitor (right). The trusted base is shown in gray.

intervention due to snow even when the road ahead is empty of traffic.

It should be noted that certified control does not remove the sensor and actuator units from the trusted base. What is removed is the main perception and controller subsystems, crucially including complex algorithms that process and interpret sensor readings.

1.2.1 A LiDAR-based Implementation of Certified Control

An implementation of Certified Control was developed to handle this point cloud filtration problem. In this implementation, the certificate proposes to continue straight ahead, using LiDAR points as evidence that there is no impending obstacle. To ensure the safety of a controller's decision to move forward, the monitor checks that the points in the certificate have sufficient *spread* and sufficient *density*. To achieve sufficient spread, certificate points must horizontally span the size of one lane in front of the car and vertically span the height of a car. Additionally, the monitor checks whether there are any points closer than some minimum forward distance from the car.

The monitor considers the certificate sufficiently vertically dense if it includes points from each of the LiDAR scan rows which fall within the lane. The points are sufficiently horizontally dense if there are no horizontal gaps of a size greater than some pre-specified parameter.

The controller uses radius outlier removal (ROR) filtering, an improvement upon statistical outlier removal (SOR), to identify LiDAR points reflecting off snow [18]. The controller preprocesses the entire LiDAR point cloud into a k-dimensional tree, which it then queries for nearest neighbor information. Points with few neighbors, relative to the average neighborhoods in the point cloud and based on tunable parameters, are labeled as snow. The controller then constructs the certificate by selecting points from the remaining subset (namely, those not identified as snow) to meet the spread and density criteria.

This monitor's performance was tested in simulated snowy conditions by dropping confetti-like paper in front of a robot car and confirming that the monitor accepts a certificate when there is sufficient space between the car and an obstacle ahead, despite the presence of simulated snow. When the controller's filtering parameters were set appropriately, it properly identified points as snow and passed a certificate excluding them to the monitor. Assuming snow was distributed fairly evenly across the lane ahead (and there was not a total "white out") the remaining points were still sufficiently dense to establish absence of an obstacle in the lane ahead.

The converse case was also tested. Tuning the filtering parameters, or adapting them to the perceived environment, can be difficult. One can imagine even a non-adversarial controller failing to set appropriate parameters, and therefore miscategorizing true obstacles as snow. To demonstrate that the monitor would detect such obstacles even when the controller fails to do so, the controller was modified so that it would erroneously filter out some sparse but significant obstacles, such as fallen tree branches (simulated, for the robot car, with plastic cables). As expected, since the controller omitted points on the obstacle from its certificate, the monitor's horizontal density check failed, rejecting the certificate, and preventing an unsafe action.

1.3 Related Work

The literature on safety assurance of vehicle dynamics splits roughly into two camps, both of which focus on assurance of the planning/control system, and assume perception is assured by some other mechanism.

One focuses on numerical analysis of reachable states. For example, reachable set computations can justify conflict resolution algorithms that safely allocate disjoint road areas to traffic participants [19]. The other focuses on deductive proofs of safety. The work of [20] models cars with double integrator dynamics, and uses the theorem prover KeYmaera to prove safety of a highway scenario, including lane changing, with arbitrarily many lanes and arbitrarily many vehicles. The work of [21, 22] demonstrates how these safety constraints can be used for verification and synthesis of control policies, including control policies with switching. In [23], the authors develop safety contracts that include intersections, and provide KeYmaera proofs to demonstrate safety.

Responsibility-Sensitive Safety (RSS) [17] is a framework that assigns responsibility for safety maneuvers, and ensures that if every traffic participant meets its responsibilities, no accident will occur. The monitoring of these conditions is conducted as the last phase of planning, and is not separated out as a trusted base. The RSS safety criteria have been explored more formally to boost confidence in their validity [24].

All of these works focus on control, and assume that the perception system is reliable. In contrast, certified control takes the perception subsystem out of the trusted base. Nevertheless, these approaches are synergistic with ours. In our design, the low-level controller is still within the trusted base and would benefit from verification. RSS provides more sophisticated runtime criteria than those we have considered that could be incorporated into certificates (e.g., for avoiding the risk of collisions with traffic crossing at an intersection).

Several approaches aim, like ours, to establish safety using some kind of monitor. The Simplex Architecture [13, 25], described above, uses two controllers: a verified safety controller and a performance controller. When safety-critical situations are detected, the system switches to the verified controller, but otherwise operates under the performance controller. [26] extends Simplex to neural network-based controllers. As we noted, this approach does not address flaws in perception. In contrast, reasonableness monitors [27, 28] defend against flawed perception by translating the output of a perception system into relational properties drawn from an ontology that can be checked against *reasonableness constraints*. This ensures that the perception system does not make nonsensical inferences, such as mailboxes crossing the street, but aims for a less complete safety case than certified control. Similarly, the work of [29] seeks to explain perception results by analyzing regions of an image that influence the perception result. These techniques may be useful to enable the perception system to present pixel regions to the safety monitor as evidence of a correct prediction.

Other techniques seek to use the safety specifications to automatically stress test the implementation [30, 31, 32]. A different approach checks runtime scenarios dynamically against previously executed test suites, generating warnings when the car strays beyond the envelope implicitly defined by those tests [33]. Certified control is similar in that the certificate criteria represent the operational envelope considered by the designers, and outside that envelope, it will likely not be possible to generate a valid certificate, leading to a safety intervention.

Assurance of perception systems needs to grapple with two key challenges. The first is the difficulty of determining appropriate specifications for perception systems, and the second is with developing scalable reasoning techniques to ensure that the implementation satisfies its specifications. Reasonableness monitors provide a partial answer to the first. The second is an active area of research: [34], [35], and [36], for example, develop efficient techniques to prove that a deep neural network satisfies a logical specification. While these technologies are promising, their applicability to industrial-scale applications has not yet been demonstrated. Perception is a particularly thorny problem, since it is not clear what properties of a perception system one would want to formally verify.

1.4 Contributions

The majority of the sections above were taken from a co-authored paper submission on Certified Control[37]. The contributions of this thesis include:

1. an implementation of certified control for visual lane detection;
2. an augmentation of the visual lane detection monitor with LiDAR;
3. a new LiDAR-based monitor incorporating both object position and velocity data;
4. a full visual lane detection system focusing on improving monitor security;
5. the evaluation of these methods with a physical racecar, real driving data, and in simulation.

The code discussed in the following sections is publicly available at: `https://github.com/jefftienchow/Thesis_Code`.

Chapter 2

Certified Control with Vision

To explore the application of certified control in the domain of vision, we focused on the verification of lane-line detection. When an autonomous vehicle proposes an action (such as "continue in lane") within the context of the current lane, it must know the current locations of the lane boundaries for the action to be safe. To maintain safety, the controller must provide evidence that the current lane is where it says it is. We designed an application of Certified Control that confirms whether the controller is correctly perceiving the locations of lane boundaries.

2.1 Design

The goal of the monitor is to be able to confirm with high confidence that the controller is correctly perceiving the road with relatively low latency. To do so, the monitor must be presented with enough information to make these deductions without having to resort to more complicated methods (i.e. machine learning). Thus, the controller is tasked with assembling a certificate containing a set of proposed lane line locations and sufficient metadata that serves as evidence that the proposed lane lines are correct.

This Certified Control implementation is specifically designed to only be applicable to highway-driving situations. Therefore, this may not be robust to residential or less-conventionally marked roads; these conditions may need a different design to work.

2.1.1 Controller

In this scheme, the certificate assembled by the controller contains the following elements:

- the signed image frame from which the lane lines were deduced;
- the left and right lane boundaries given, as second-degree polynomials in the bird's-eye/top-down view ($L(d)$ and $R(d)$, respectively), both lines giving the distance from the left side of the top-down view as a function of distance from the front of the car;
- a transformation matrix T used to transform the lane points from the bird's-eye view to the camera view;
- a series of color filtering thresholds used to process the image and highlight the presence of the lane lines.

Depending on performance constraints, the controller may instead incorporate a compressed/downsampled image in the certificate. This can greatly speed up the monitor's latency. However, if the image is downsampled too much, this can of course come at the cost of accuracy.

2.1.2 Monitor

Using the provided certificate, the monitor verifies performs two tests:

1. a **geometric** test which checks that the proposed lane lines are geometrically consistent and plausible
2. a **conformance** test which checks that there are light-colored strips on the road present at the specified locations

In the geometric test, the monitor evaluates the proposed lane lines as a pair; if they conform to specific geometric bounds (such as parallelism) then the lane lines pass this test. The conformance test is a computer vision check. Computer vision techniques are used to determine whether the proposed lane lines correspond to lane line markings in the image. If they do, the lane lines pass. The proposed lane lines must pass both checks in order for the certificate to be accepted.

Geometric Test

The geometric test ensures that the purported lane lines are parallel and spaced according to local regulations. In the US, for example, the width of a freeway lane is 12 feet [38]. The geometric check is done as follows:

1. The polynomial $L(d)$ corresponding to the left proposed lane line is sampled at intervals to form a discrete set of points along the line $l = l_0, l_1, l_2, ..., l_n$;

2. For each l_i, we compute the slope of the normal line n_i to the lane polynomial. We then find the intersection of the n_i with the right polynomial $R(d)$ and compute the distance x_i between l_i and that intersection point;

3. We calculate the standard deviation of the distances $x_1, ...x_n$ and check that it is below some threshold;

4. We calculate the mean of the distances $x_1, ...x_n$ and check that it is within some delta of the expected distance between lane lines.

In our initial design, we compared the distances between left and right lane lines at various distances from the front of the car to get the average width of the lane and deviation between the lines. However, this does not work well with curved lane lines, as the distance between the left and right lines increase as the lines curve. We fixed this by instead comparing the length of the *normals* between the lines. Using the normals allows us to evaluate the distance between the lane lines as constant, even when the road curves (Fig. A-1).

The problem of checking the lane width and comparing lane boundaries for parallelism is greatly simplified by working with a bird's-eye perspective of the lane boundaries. The transformation and fitting of the lane-lines is done by the main controller, so the checks performed by the monitor remain simple.

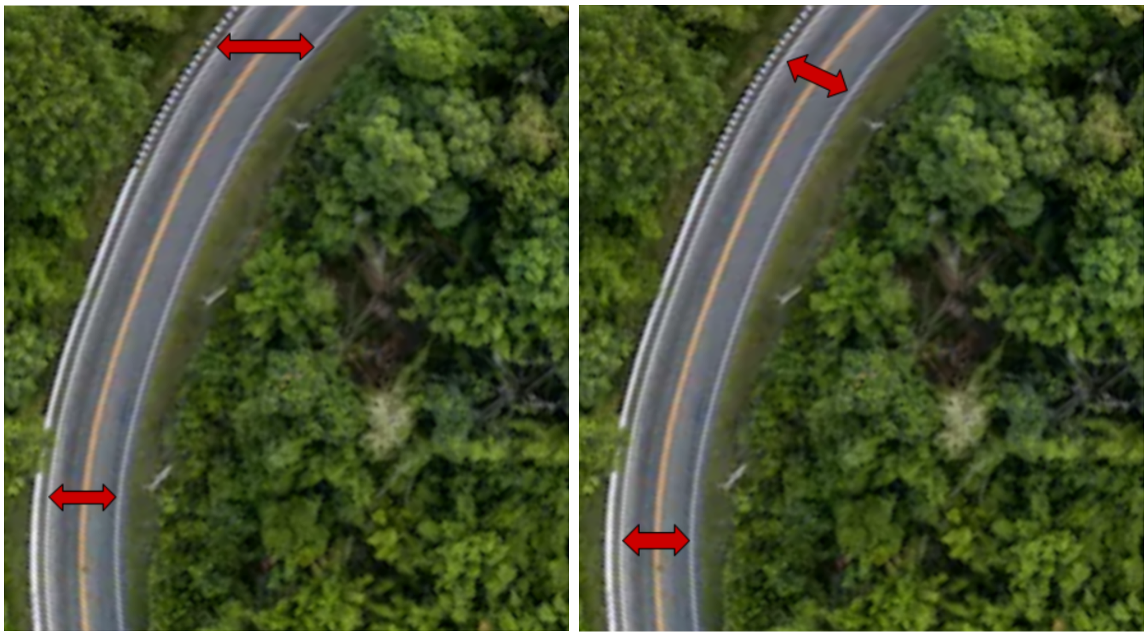

(a) Distance measured using horizontal lines (b) Distance measured using normals

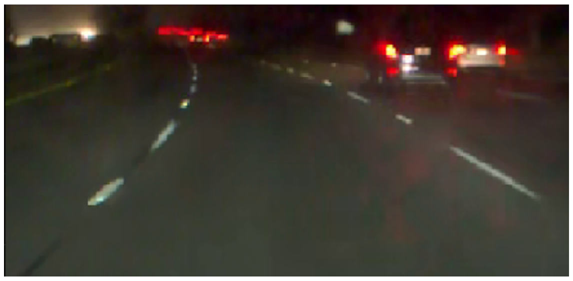
Figure 2-1: Measuring distance between the lane lines is more accurate when measuring across the normals of the lines.

In the case that lane lines happen to not adhere to local standards, we argue that it is fine for the monitor to intervene. Since it cannot reliably guarantee that rules of a typical road apply in that scenario, it cannot guarantee safety.

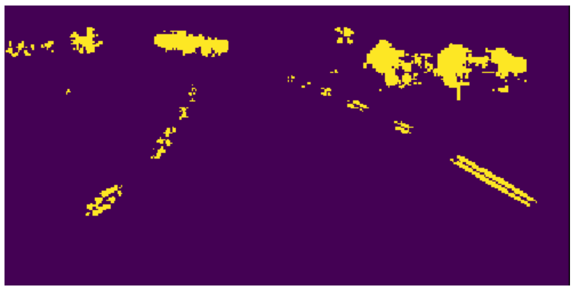
Conformance Test

It is not enough to check whether the proposed lane lines have correct geometry, as this only tells us whether the proposed lane lines could be *possible*. Proposed lane lines might happen to pass the geometric test (or be maliciously tailored to pass it) but not actually correspond to lane lines on the road itself. It is therefore essential to also check that the lane lines conform to a signed image of the road. To do this, the monitor applies simple and well-tested computer vision algorithms to determine if there are corresponding lane markings in the image. The match between the proposed lane lines and the image is checked as follows (Fig. 2-2):

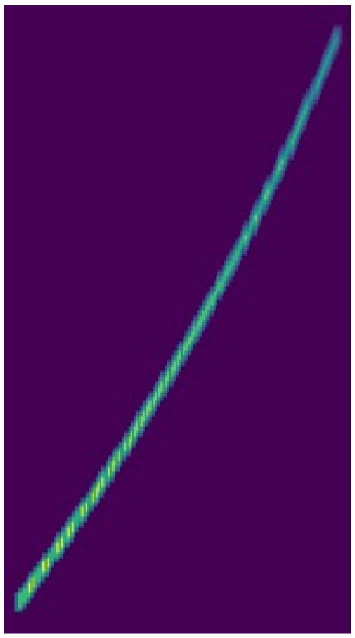
1. The monitor computes the logical OR of the image filtered with an edge detection algorithm on lightness value (such as the Sobel operator) and the image



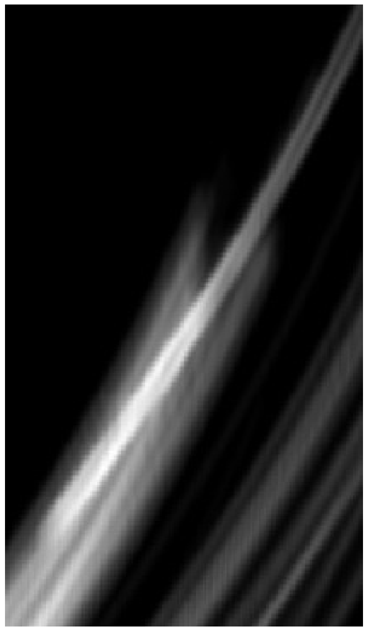
(a) Original image



(b) Edge detection & lightness filtering



(c) Filter for left lane line



(d) Correlating filter and left-half of image

Figure 2-2: The steps of the conformance test.

filtered by lightness above a certain threshold. These thresholds are computed by the controller and passed to the monitor as part of the certificate.

2. Using the bird's-eye transformation matrix passed from the controller, the monitor transforms the lane lines $L(i)$ and $R(i)$ from the bird's-eye view to the camera's view.

3. For the transformed left lane curve $L_T(i)$ (and correspondingly for the right), the monitor creates a filter based on the curve, slightly blurred to allow for some margin of error in the proposed lane lines. The bottom of the filter is weighted more heavily because deviations in the region closest to the car are more important. At points not on the curve, the filter is padded with negative values so that only thin lines that match the filter's shape will produce a high correlation output.

4. The monitor computes the cross-correlation C between the filter and the left side of the processed image (and correspondingly for the right), and finds the point of highest correlation in the image (namely $(i_{\max}, j_{\max})$ such that $C[i_{\max}, j_{\max}] = \max(C)$). It checks that the point of highest correlation lies within some small distance from the transformed lane line $L(i)$, and that the maximum correlation is above a predefined threshold.

Once again, we push the complexity of having to determine the specific bird's-eye transformation to the controller. Since the monitor just accepts the transformation as part of the certificate, it can remain simple while still adapting to the curvature and environment of the road (as the controller does). However, since the monitor does not check the validity of the transformation, the transformation is currently considered as part of the trusted base. A potential solution to this problem would be to once again task the controller with producing a certificate proving the correctness of the suggested transformation and add some check to the monitor to confirm the validity of that transformation. For example, the certificate could also include LiDAR points on the ground, and the monitor (which now has the exact coordinates of those

LiDAR points) could create a two-dimensional projection of those points from both the camera view and the bird's-eye view. The monitor could then check whether the transformation matrix accurately maps points from the bird's-eye view to the camera view.

Originally, the certificate was composed of the lines in the *camera view* and a transformation from the *camera view to the bird's-eye view.* This was beneficial because it allowed us to distinguish between dotted and solid lane lines. If we look at images from the bird's-eye view, the space between dots on a dotted lane line are more consistent (whereas for images in the camera's view, spaces between those dots are larger closer to the vehicle); therefore, we were able to create separate filters for dotted and solid lane lines. However, *openpilot*, our vision controller of choice that is discussed in the next section, already internally computes a camera view to bird's-eye view transformation matrix, so we decided to use that in our certificate to reduce project complexity.

2.2 Experiments

The vision certificate checking scheme was implemented and evaluated against two different lane-detection software systems. While tweaking parameters, our primary goals were to eliminate the false positives (the monitor rejecting the certificate when the suggested lane lines are accurate) and the false negatives (the monitor passing the certificate when the suggested lane lines are inaccurate). In addition, we primarily focused on highway conditions, as this certified control implementation is not designed to work on non-highway conditions.

2.2.1 Openpilot Implementation

The first lane-detection system we tested our design with was *openpilot* [39], an open source production driver assistance system that can be used to control and drive vehicles. We used real replay data from a car driven using the system. In particular, we ran our monitor using the *Comma2k19* dataset [40], which includes

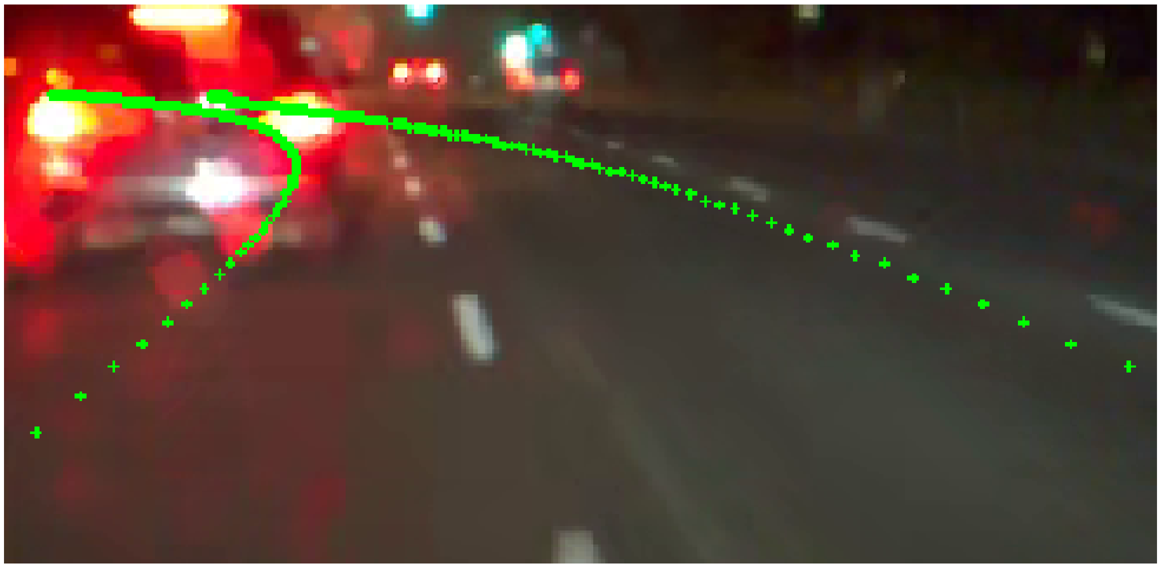


Figure 2-3: An example of a lane-detection failure detected by the monitor. The green lines represent the proposed lane lines.

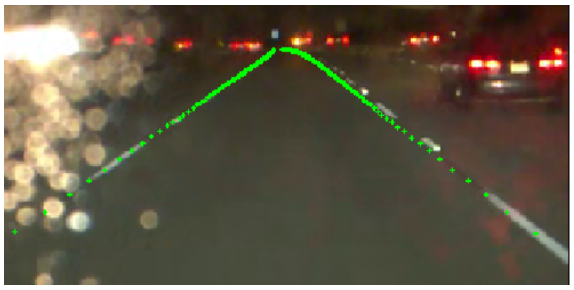
driving segments with non-highway driving and adverse lighting conditions (such as a rainy night).

Openpilot's implementation exposes a few variables that allows for easy integration into our monitor implementation. In particular, it exposes:

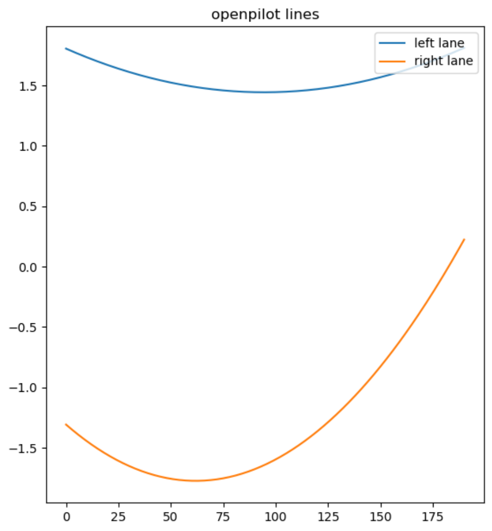
1. Points on the left and right lane lines in the image in the bird's-eye view.
2. The image used to deduce the location of the lane lines.
3. A transformation matrix that can be used to transform the lane lines points to the camera view.

Since *openpilot* uses a complex neural network to do its lane-finding, there were not any color-filtering thresholds readily available to incorporate into the certificate. Instead, we configured openpilot to pass in a predefined set of thresholds to the monitor to use in its check.

In several instances, our monitor caught lane detection failures (Fig. 2-3). In all failure cases, however, *openpilot* was able to successfully correct its lane detection within a few seconds. For some of these cases, it was not obvious from inspection of the image taken from the camera's perspective that the proposed lane lines were not



(a) The dashcam image with the proposed lane lines (green) superimposed on the image.



(b) A graph of the lines

Figure 2-4: The lane-lines looks plausible from the camera's view, but further away, the lines are not parallel; this can more easily been seen from the bird's-eye view.

geometrically correct. However, viewing from the bird's-eye perspective, it was easier to see that they could not correspond to plausible lane lines (Fig. 2-4). See Fig. 2-5 for more examples of passing and failing lane lines.

2.2.2 Racecar Implementation

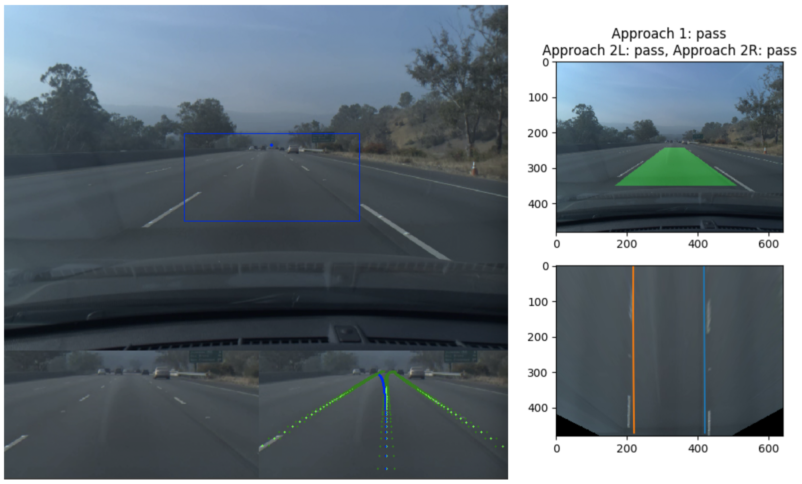
The second experiment was conducted with a physical racecar. We implemented a naive lane-detection scheme to serve as the controller, which deduced lane lines through the following steps:

1. The image from the racecar's camera is converted to a hue, saturation, value (HSV) image to allow for easier segmentation based on lightness values.

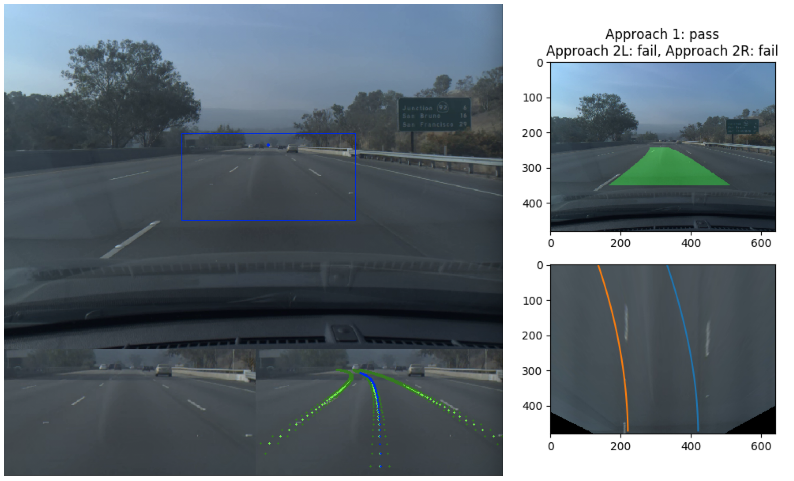
2. Upper and lower HSV color thresholds are used to segment the the image into a mask for the blue portions of the image.

3. Noise is removed using an erosion filter.

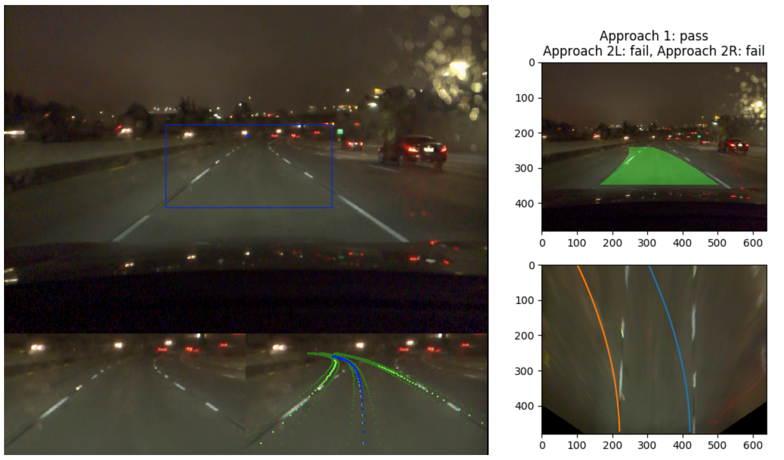
4. A Hough line-detection algorithm is run to isolate the lane lines in the image.



(a) All tests pass.



(b) Lane lines pass the geometric test but correctly fail both conformance tests.



(c) Conformance test also works under darker lighting conditions with rain.

Figure 2-5: Additional examples of the vision monitor working with *openpilot*

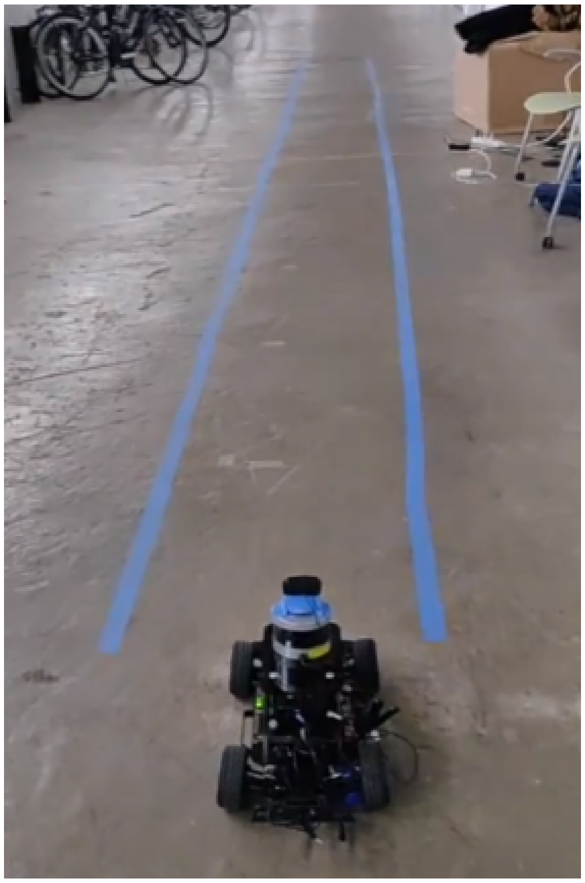
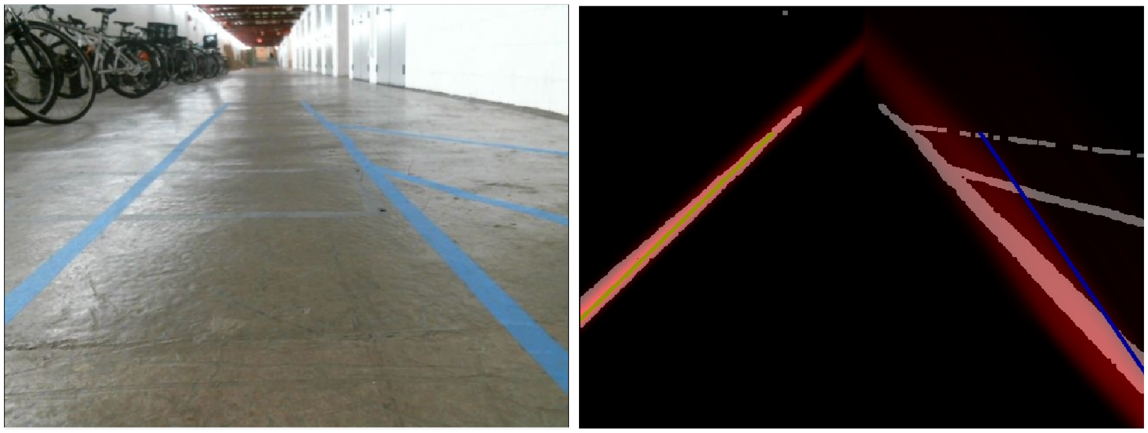


Figure 2-6: The setup of the racecar with blue tape representing lane lines.

5. The lines are separated into two groups based on slope. Due to the camera's perspective, a positive slope corresponds to a left lane line and a negative slope corresponds to a right lane line.

6. The lines in each group are averaged to produce an estimate of the locations of the left and right lane lines. These lines are represented as first-degree polynomials.

To simulate potential lane lines, we placed blue tape on the floor in front of the racecar (Fig. 2-6). When we tested the monitor with only straight and parallel lane lines, the monitor never rejected the certificate given from the controller. However, when we placed additional tape segments in inappropriate positions, we were able to get the lane detector to report bad lane lines (Fig. 2-7); in all cases, the monitor correctly rejected them.


(a) Additional blue tape is placed on the right side to cause the naive controller to propose bad lines.

(b) The proposed right lane line (blue) fails because the correlation (red) at points along the proposed line is not strong enough.

Figure 2-7: The monitor correctly rejects lane lines when the controller proposes incorrect lane lines due to the additional blue tape.

Component	Lines of Code
Vision Monitors library calls	90
Vision Monitors self-written code	235
Vision Monitors Total	**325**
Controller library calls	460
Controller self-written driver code	612
Vision Controller Total	**1072**

Figure 2-8: Comparison of code sizes for the monitor vs. certificate generation within the controller. Only the racecar code is included; the *openpilot* version uses a complex neural network.

2.3 Evaluation

Both implementations of the vision monitor were able to correctly detect errors in lane-finding controllers. Neither controller, however, had the capability of generating variable color filtering thresholds (depending on lighting conditions) to incorporate in the certificate. The motivation for including these thresholds in the certificate was to allow the monitor to adapt to different lighting conditions since lighting can potentially affect whether the monitor verifies a certificate. See Appendix A for an experiment demonstrating how varied color thresholds can improve the monitor's accuracy under adverse lighting conditions.

Overall, even though the racecar used only a naive lane-finding algorithm with no machine learning, the algorithms still used three times as many lines of code as the monitor's three vision checks combined (Fig. 2-8). Production controllers are of course much more complex—*openpilot*'s deep learning container for lane-finding on github has around 26 convolutional and 7 fully connected layers.

The certificate used for *openpilot* required about 45KB of storage, dominated by the image. On average, checking of the certificate took about 0.08 seconds on a Dell XPS 9570 Intel Core i7-8750H CPU with 16GB RAM. We achieved this latency and certificate size by downsampling the image by a factor of 2. This may not be adequate performance in a production system, but it is within an order of magnitude of what would be required. With proper optimization of the checking algorithm, and some additional compression of the image, performance seems unlikely to be a problem.

2.4 Limitations

2.4.1 Using Visual Data

This certificate is inherently less trustworthy than the LiDAR certificate. This seems unavoidable, since unlike the physical obstacles detected by the LiDAR scheme, a single pixel does not tell much about what is present at a particular location. Visual data generally must be interpreted at a larger scale (than just a single pixel), introducing more complexity. In addition, following lane lines is a social convention, with the lane lines acting as signs whose interpretation is not a matter of straightforward physical properties. Therefore, it may be necessary to use more sophisticated methods to confirm the locations of lane lines when the lines are only implied by the presence of these conventions as opposed to the presence of light-colored bands on the road.

2.4.2 Obstructed/Unclear Lane Markings

This implementation assumes that both the left and right lane lines are fully visible and therefore does not work with partially (or fully) occluded lane lines. There are

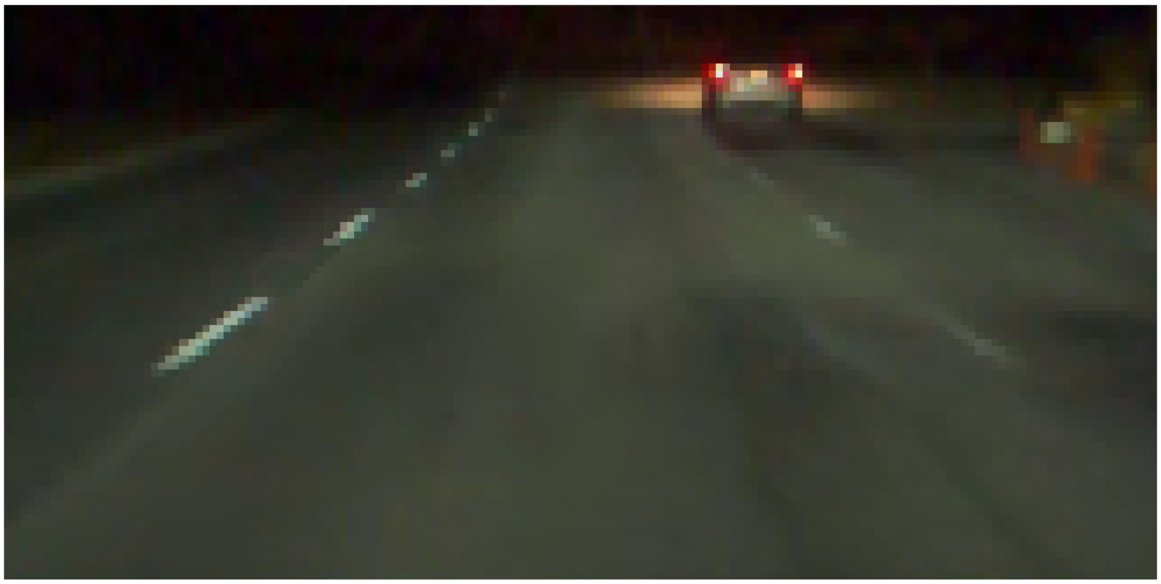


Figure 2-9: A frame that caused the conformance test to fail due to the faded right-hand lane line.

two failure cases that we encountered during testing. The first involves obstructed lane lines. Our design assumes that the lane boundaries are always at least partially visible. Since the *Comma2k19* dataset includes segments with non-highway driving, there were cases in which multiple cars were lined up at a stoplight and obscuring the lane lines. On the highway, the lane lines could be similarly obstructed during traffic or when a car in front is changing lanes. It is possible that more sophisticated methods would be needed to address these cases. The second failure case involves poorly painted lane lines. In one case, a lane line was so faded it caused the correlation output of the conformance test to not reach the desired threshold and therefore fail the check (Fig. 2-9). This situation could be mitigated if color filter thresholds were dynamically computed and passed by the controller to the monitor. However, since such thresholds were not exposed in *openpilot*, we passed constant values to the monitor. This failure case also reflects the fact that compelling evidence of the presence of a lane will require well-drawn lane lines; a successful deployment of autonomous cars might simply require higher standards of road markings. It is essential to realize that certified control does not create or even exacerbate this problem but merely exposes it. If the designer of an autonomous vehicle were willing to rely on inferring lanes from poorly drawn lines, the certificate requirements could be reduced accordingly, so that the level of confidence granted by the certificate reflect the less risk-averse choice of the designer.

Another concern with the current implementation is the use of conventional edge detection algorithms. The current implementation fails to confirm lane lines when the road is poorly lit (ex: trees casting shadows on the lane lines). This is a known problem with the Sobel operator, but can potentially be fixed using better edge detection algorithms [41].

2.4.3 Lack of Time-Domain Awareness

This implementation does not store any metadata or results from previous timesteps. The previous limitations discussed could potentially be resolved if the monitor observed how lane lines and images change over time. For example, if a lane line is present in one frame and occluded in the next, one might infer that the lane line is still there but just concealed in that frame. This could potentially eliminate some incorrect interventions from the monitor, as the monitor would be more robust to sudden changes in lane visibility.

On the other hand, utilizing temporal data would increase the complexity of the monitor and would lead to a larger trusted base. A potential solution would be to only have the controller include temporal data when necessary (e.g., when a lane line is occluded) and incorporate past frames as evidence that there is likely still a lane line present at the specified location.

Chapter 3

Combining Vision with LiDAR

3.1 Motivation

Image data lacks the inherent physical properties of LiDAR data: a single pixel, unlike a single LiDAR point, says nothing about the car's environment, absent other context. As a result, one can only perform limited checks on the lane lines using vision alone. In a particular case involving a Tesla driving with autopilot, the vehicle almost crashed into a concrete barrier. The problem was that bright lines that look similar to lane lines appeared on the barrier in the camera's image [42]. These lines were actually bands of direct or reflected sunlight. In most cases, we believe that the vision tests described above would catch such anomalies; we took one particular image from an online video (Fig. 3-1) illustrating this problem and confirmed that the inferred lane lines would indeed fail the geometric test and be correctly rejected.

However, it is conceivable that these spurious lane lines would have passed the geometry and conformance tests. Indeed, in the Tesla example, the yellow ray of sun almost looked, in the image, like a plausible lane line. Apparently a more basic property is being violated: the detected lane line is on the barrier and not on the ground.

This motivates a certificate that integrates LiDAR and vision data for verifying lane line detection. In addition to providing the lane line polynomials and the camera image (as above), the certificate also provides a set of LiDAR points that lie on the

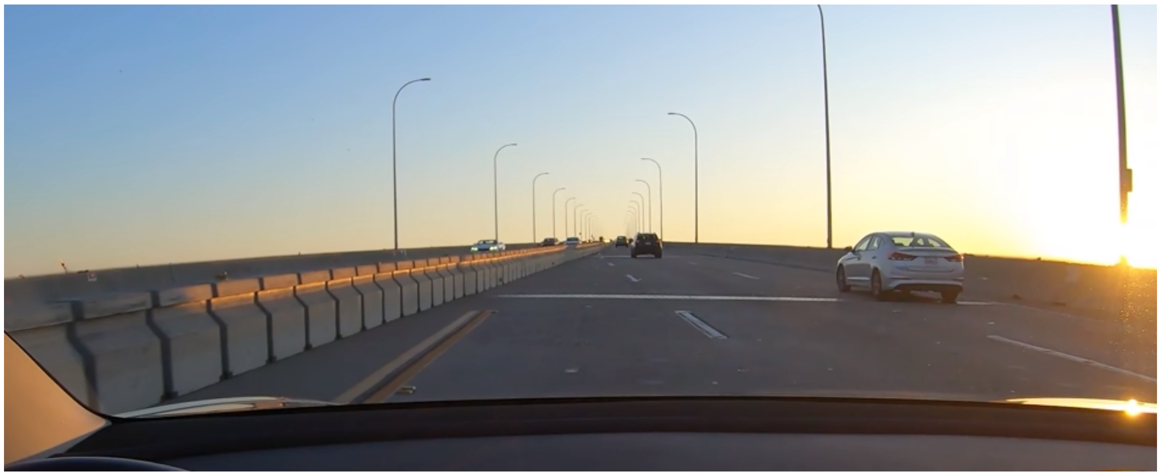


Figure 3-1: An image taken from dashcam footage in which a car using Tesla's autopilot almost ran into a concrete barrier. The malfunction was presumably caused by the perception system mistaking the reflection along the barrier for a lane line.

purported lane lines. The monitor checks that these LiDAR points indeed correspond to the lane lines and that they reside on the ground plane. The controller, as usual, is given the more complex task: finding that ground plane and selecting the LiDAR points.

3.2 Design

After the controller computes the location of the lane lines on the image, it is responsible for determining the LiDAR points that correspond to those lane lines. The controller does this by sampling points along the lane line polynomial and transforming them into 3D space. We assume that the camera and the LiDAR scanner are both at the same location, which is approximately true given our setup (Fig. 3-2). We also assume we are working with a pinhole camera, allowing us to use the pinhole camera model to relate the 3D location of objects to their 2D projection. To determine which pixels along the lane lines to sample, we utilize the vertical angle of the LiDAR rows given by the LiDAR scanner specifications. This is to prevent us from choosing pixels on the image that lie in between LiDAR rows (which would naturally not correspond to any scanned LiDAR points). For each row with a vertical angle corresponding to a pixel below 1/3 the height of the image, we sample one point for each of the lane lines.

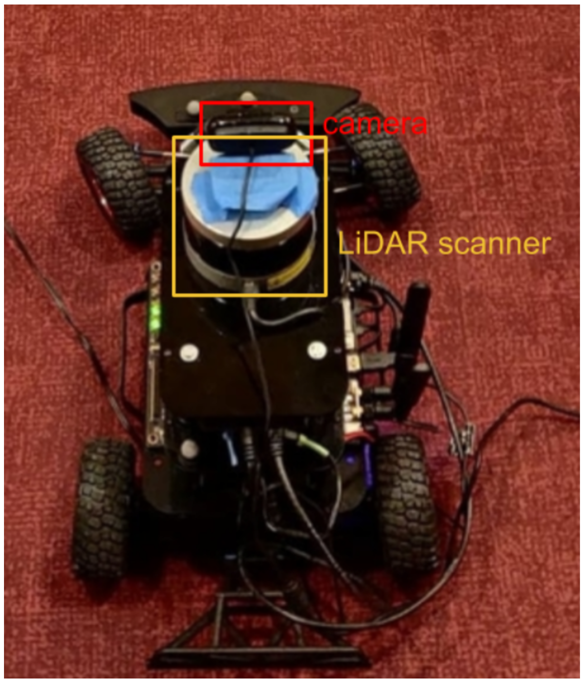


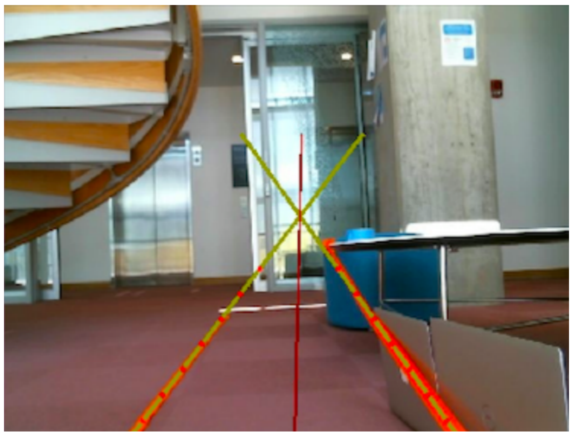
Figure 3-2: Our racecar setup, with the camera mounted directly above the LiDAR scanner.

Given a point (x, y) in our 2D image sampled in this way, we can simply search that corresponding LiDAR row to find the point that most closely matches the horizontal angle of the pixel, telling us approximately which LiDAR point corresponds with that pixel.

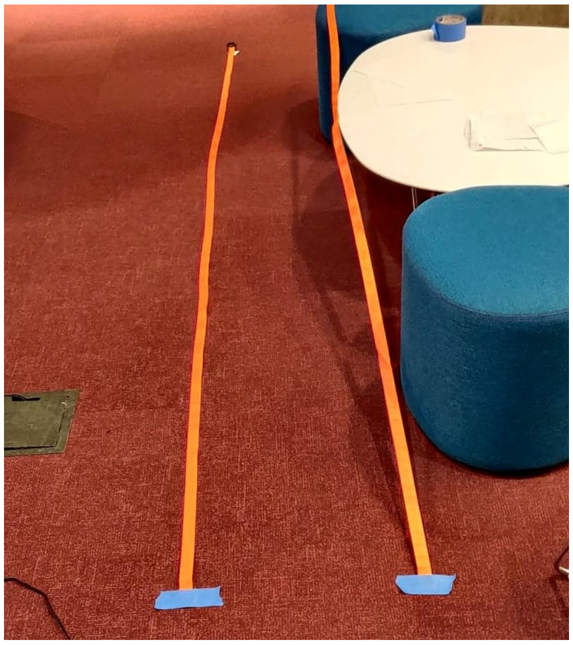
We additionally task the controller with identifying a ground plane. Our controller implementation runs a random sample consensus (RANSAC) algorithm [43], augmented with some constraints. More specifically, the controller uses additional heuristics to make sure that the proposed ground plane is plausible. For example, the controller requires that the ground plane lies below the camera and that the plane is sufficiently flat.

Given the LiDAR lane line points and the ground plane, the monitor checks that:

- the proposed LiDAR points indeed match the proposed lane lines. This is again done using trigonometric methods;
- the proposed LiDAR points are sufficiently close to the ground plane;
- the ground plane is sufficiently low-lying (below the camera) and flat;



(a) Image taken from the camera on the racecar showing the sampled points (red) corresponding to the LiDAR rows.



(b) The same setup from a higher perspective.

Figure 3-3: The lines look plausible from a 2D perspective, but the right line is not on the ground. The red dots on the lines in (a) are the points sampled along the line that lie at the same angles as the LiDAR scanner rows.

- there is a high enough density of points lying close to the ground plane.

3.3 Experiment

To simulate the Tesla barrier malfunction, we lined up two rows of tape to look like lane lines (Fig. 4-1). The tape for the left line was placed on the ground while the tape for the right lane was elevated on a platform; from the camera's perspective, the pair of lane lines looked geometrically plausible. When we ran our combined vision-LiDAR implementation on this scenario, the monitor correctly rejected the certificate because the points from the right lane line were not on the ground plane. This check would theoretically also detect false lane lines which lie above or below the ground plane for any other reason.

3.4 Evaluation

The integration of vision with LiDAR presents a few unique challenges. First, the ground plane detector we implemented is not robust to steeply sloped roads or to very uneven road surfaces. Our RANSAC algorithm only finds flat planes, so the controller cannot yet propose curved planes. However, improving the system to work with those conditions would mean only a more complex controller–the monitor need not become more complex. Second, since the monitor's checks to determine the validity of the ground plane are not exhaustive, it is possible the monitor would not detect a false ground plane. This is partly because the controller does not include in its certificate any proof of validity of the ground plane it detected, and thus, the ground plane algorithm must be regarded as within the trusted base. The obvious remedy—namely including the ground plane detection in the monitor—is not straightforward for two reasons: the algorithm is too complex to be easily verifiable, and the monitor only has access to the points in the certificate, not the entire LiDAR point cloud. Nevertheless, this could potentially be solved with our usual approach: by tasking the controller with producing a certificate which proves to the monitor the validity of its suggested ground plane, e.g. by inclusion of certain low-lying points.

Chapter 4

Enhancing the LiDAR Monitor

The original LiDAR-based implementation of certified control establishes that the LiDAR points have sufficient spread and density and verifies that there are no points within some minimum forward distance from the ego vehicle. In other words, the monitor determines whether or not the controller is filtering out too many points and intervenes if there are obstacles closer than some set distance from the front of the car. In reality, some objects can be closer to the ego vehicle while maintaining safety and others must be further away. More specifically, danger of collision also depends on the velocity of objects relative to the ego vehicle. For example, if the ego vehicle is driving at 20m/s and an object is 5m in front of the ego vehicle, the safety of the situation depends on how fast that object is moving. If it is stationary, then the ego vehicle might crash into it; if it's also moving forward at 20m/s, then the situation is probably not dangerous.

4.1 Design

Similar to the original LiDAR certificate, the controller is tasked with taking the LiDAR point cloud and filtering out noise (such as snowflakes) while keeping the points dense and spread out enough to ensure no large objects are filtered out. As of now, the controller is only tasked with passing points in front of the ego vehicle to the monitor. In addition, the LiDAR points now include a velocity in addition to a position; this velocity corresponds to the speed of the particle that reflected the LiDAR point relative to the ego vehicle. We assume there is some method of obtaining the relative velocity of a LiDAR point. For example, LiDAR technology has the ability to discern the speed of objects using the Doppler Effect with high accuracy [44]. These position-velocity pairs are passed to the monitor to determine whether the proposed velocity of the ego vehicle is safe.

Similar to the previous design, the monitor checks for sufficient density and spread within the LiDAR points, confirming that there are no large areas in the ego vehicle's lane that do not contain LiDAR points. The monitor also evaluates each point in the certificate by checking whether it's possible for the ego vehicle to collide with the point, assuming that the ego vehicle continues straight at the suggested velocity. To perform this calculation, we will work within a one-dimensional framework where we only care about object velocities in the forward direction.

4.1.1 Case 1: Objects Don't Move Backwards

We first handle the case where objects in front of the car will not move backwards toward the ego vehicle. We assume that the maximum deceleration β_{max} an object can have is $9m/s^2$, an estimate from a study on emergency braking capabilities of vehicles [45]. We argue that this is acceptable because objects on the road are not likely to have a higher maximum deceleration than a car. Given these assumptions, the worst case for the ego vehicle would be if all other objects were to suddenly decelerate at β_{max}, as this would be the situation that brings objects as close to the colliding with the ego vehicle as possible. Since we are assuming objects can have a maximum deceleration of β_{max}, objects that have lower deceleration rates would end up further in front of the ego vehicle; using β_{max} for all points gives us a conservative worst-case estimate of where points could end up in the near future.

If the ego vehicle were to approach an unsafe scenario where it might crash into an upcoming obstacle, we also want to give the controller time to circumvent the danger. Thus, we can determine whether we need to intervene by calculating whether the ego vehicle will crash into any objects decelerating at β_{max} given that the ego vehicle is also decelerating at its maximum deceleration rate β_{egomax}. In other words, we assume that the controller will drive safely until the last possible moment: when braking at the ego vehicle's maximum deceleration will just barely cause a collision with a particle decelerating at β_{max}.

We compute the position of the ego vehicle if it were to fully brake and compare that position to the location of objects if they were to decelerate at β_{max} toward the ego vehicle. If the ego vehicle is in front of an object, there must have been a collision. See Appendix B for our reasoning.

To calculate the braking distance of the ego vehicle, we also need to consider the latency from when the monitor sends the command to when the actuator brakes. Taking this into account, we get a formula for determining when to intervene:

$$\frac{v_{ego}^2}{2\beta_{egomax}} + \rho \cdot v_{ego} > d + \frac{v_f^2}{2\beta_{max}}$$

where

- v_{ego} is the forward velocity of the ego vehicle;
- β_{egomax} is the maximum deceleration capability of the ego vehicle;
- ρ is the latency between messages going from the monitor to the actuators;
- d is the forward distance between the front of the ego vehicle and the object in question;
- v_f is the forward velocity of the object in question;
- β_{max} is the maximum deceleration capability of the object, assumed to be $9m/s^2$.

4.1.2 Case 2: Objects Move Backwards

To handle cases where objects are moving *backwards* towards the ego vehicle (ex: a tire that comes off a car), using the assumption that objects are going to have a deceleration of β_{max} no longer leaves us with a worse-case analysis. On the other hand, assuming objects will *accelerate* towards the ego vehicle is also not a good assumption; if we were to assume objects on the road were to accelerate toward the ego vehicle, it would be impossible to share the road with other objects/vehicles. Thus, we assume that objects moving backwards will *continue at their current velocity*, giving us a more reasonable worst-case analysis.

This poses the question, how long do we assume the object will move backwards? Since the monitor only has the power to stop the ego vehicle, and putting the ego vehicle in reverse would complicate matters much more, we will again assume that the best course of action in a dangerous situation is to maximally decelerate the ego vehicle. Therefore, we only need to analyze how far objects will move backwards in the time it takes the ego vehicle to fully stop. If the object continues to move toward the ego vehicle even after the ego vehicle has completely stopped, then the monitor could not have prevented the collision (at least with only the power to actuate the brakes).

Therefore, we can summarize whether or not the monitor should intervene with the following formula that considers the worst result of the two above cases:

$$\frac{v_{ego}^2}{2\beta_{egomax}} + \rho \cdot v_{ego} > \min(d + \frac{v_f^2}{2\beta_{max}}, d + v_f \cdot t_{stop})$$

where

- $d + \frac{v_f^2}{2\beta_{max}}$ is the distance between the ego vehicle and an object if it decelerates at β_{max};
- $d + v_f \cdot t_{stop}$ is the distance between the ego vehicle and an object if it continues at v_f;
- t_{stop} is the time needed to fully stop the ego vehicle, given by: $\frac{v_{ego}}{\beta_{egomax}} + \rho$.

Similar to the previous LiDAR monitor implementation, noisy points that do not represent large obstacles that would otherwise cause the certificate to fail should be filtered out by the controller. This greatly simplifies the monitor, as it therefore does not need to do any object segmentation to figure out whether objects in the point cloud are small enough to be ignored.

The monitor evaluates every point in the certificate (using their respective position and velocity) to determine if the vehicle is continuing at a safe speed. If all points are safe, and there is sufficient spread and density within the ego vehicle's lane (as in the previous monitor implementation), the certificate passes.

4.2 Implementation

We tested this certified control implementation in CARLA, an open-source vehicle simulation program [46]. CARLA has the ability to spawn and simulate objects in a 3D environment and was developed for the purpose of testing autonomous vehicles. The program comes with a variety of preset maps, environmental conditions, and actors (anything that plays a role in the simulation or can be moved around) which greatly facilitated development. We ran CARLA on Dell XPS15 9500 laptops with 16GB of RAM, an NVIDIA GTX 1650Ti dedicated graphics card, and an Intel i7-10750H CPU.

To implement our certificate design and monitor in CARLA, we took the following steps:

1. Create a world in synchronous mode to allow for full control of the speed of timesteps in simulation. We set the simulated time between ticks to be .05 seconds.

2. Spawn the ego vehicle on the map.

3. Mount a LiDAR object on the ego vehicle set to rotate 10 times per second. Since the LiDAR only completes a full rotation every two timesteps, the controller only passes a certificate to the monitor every two timesteps.

4. When a full rotation is completed, filter the LiDAR points for points in front of the LiDAR scanner, above a certain height (1.2m below LiDAR scanner), and within a given left/right boundary (2m, or half the width of a lane).

5. For each of the remaining points, if they are within the bounding box of an actor (person, vehicle, etc), attach the velocity for that actor to that point. If a point is not within the bounding box for an actor, assign it a velocity of 0. This

is a workaround to incorporate velocity into the LiDAR scans, since CARLA does not have a built-in method for detecting velocity with LiDAR.

6. Pass a certificate containing remaining LiDAR points with both position and velocity to the monitor, which verifies the safety of each point using the above scheme. For our implementation, we assume the ego vehicle has a maximum deceleration rate β_{egomax} of $9m/s^2$ (the same as for other objects) and a latency ρ from the monitor to the actuators of two timesteps (.1 seconds) as specified in the CARLA documentation.

7. If any point does not pass the check, the monitor sends a command to CARLA, braking the car at its maximum possible deceleration.

4.3 Experiments

We ran experiments in CARLA to examine the behavior of this monitor implementation. Some examples include:

- The ego vehicle follows a lead vehicle going at the same speed. No unsafe scenarios occurred, and the monitor did not intervene.

- The ego vehicle travelling at a constant velocity approaches a stopped vehicle. The monitor intervenes and prevents a collision.

- The ego vehicle travelling at a constant velocity approaches a pedestrian walking in the same direction as the ego vehicle. The monitor intervenes and prevents a collision.

- The ego vehicle travels at a constant velocity and passes by a stopped vehicle in a neighboring lane. The points corresponding to the stopped vehicle are not included in the certificate, so the monitor does not intervene.

- The ego vehicle follows a lead vehicle going at the same speed. The lead vehicle then suddenly brakes as hard as it can. The monitor intervenes and prevents a collision.

- The ego vehicle travels at a constant velocity and approaches a vehicle going backwards. The monitor intervenes and stops the ego vehicle; however, the backwards-moving lead vehicle continues and eventually hits the ego vehicle. This is expected since the monitor's only method of intervention is braking.

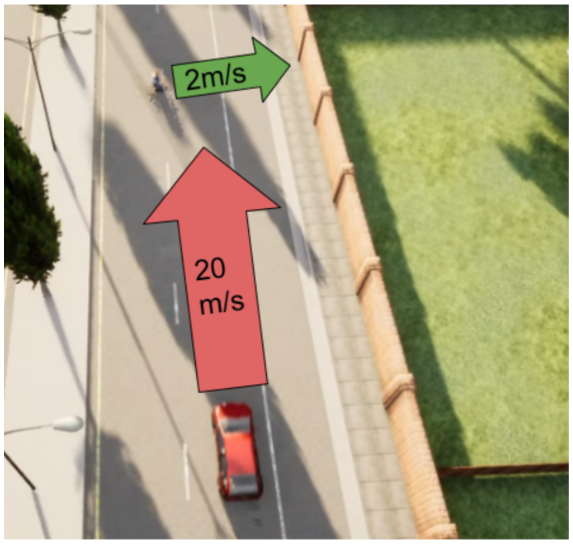
- The ego vehicle travelling at a constant velocity approaches a bicyclist crossing the street. The paths of the ego vehicle and the cyclist cross. The monitor intervenes and prevents a collision.

The final case discussed above is of particular interest because it was modelled after an Uber accident in Tempe, Arizona that killed a pedestrian [47]. In the accident, there was a variety of problems that ultimately caused the autonomous Uber vehicle to crash into a pedestrian that was walking their bike across the street. According to the report, the Uber was travelling at approximately 20m/s when approaching the crash site, and only brakes .7s before the impact. We closely simulated this Uber crash, and the monitor intervened in time to stop the collision.

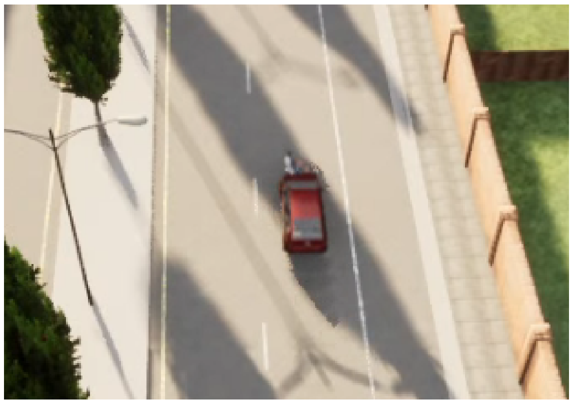
4.4 Limitations and Future Work

The current design only considers objects directly in front of the ego vehicle and reduces points to only having a velocity in the forward direction relative to the ego vehicle. Obviously, there are many situations when we might want to consider objects outside one-dimensional range (ex: curving roads, intersections). When simulating the Uber crash and attempting to get the crash parameters correct, there was a scenario that caused a crash. If the bicycle's starting position is 1m further to the left, this causes the bicycle to move into the ego vehicle's direct line of sight much later and therefore the monitor intervenes much later. Although the monitor intervenes 1.4s before the crash, there is still a slight collision between the ego vehicle and the bicyclist.

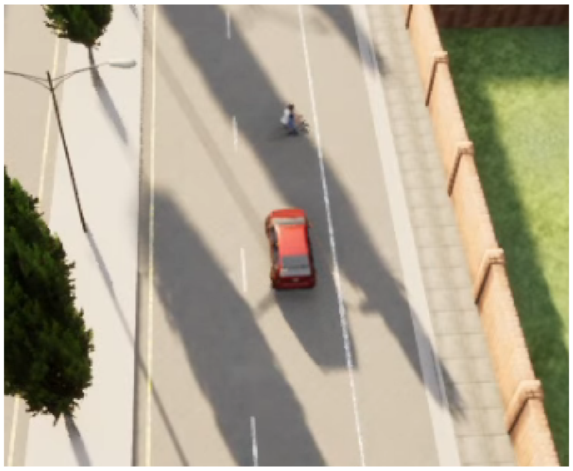
If the certificate considered points not directly in front of the ego vehicle and considered 2-Dimensional velocities, this collision could potentially have been avoided.



(a) The ego vehicle is on a collision path with the bicycle.



(b) Without the monitor, the ego vehicle continues and crashes.



(c) The monitor intervenes and prevents a collision.

Figure 4-1: A simulation of the Uber crash in Tempe.

However, the problem of opening up the certificate to consider 2D velocities increases the monitor's complexity. Objects on the road (such as vehicles) have less predictable lateral movements, so we cannot make the same assumptions. For example, for the 1-D implementation, we assumed that objects will not accelerate towards the ego vehicle unless braking. However, this happens in the lateral direction when vehicles are changing lanes (a vehicle changing into the ego vehicle's lane will accelerate toward the ego vehicle). Therefore, it may be necessary to use some sort of path prediction to determine if an object not directly in front of the ego vehicle will collide with the ego vehicle.

Chapter 5

Strengthening Monitor Security

For the monitor to be effective, it must also be secure. Naturally, we do not want the monitor deciding whether actions are safe to be incorrect or tampered with. Moreover, the monitor is part of our trusted base, so it must be protected from outside attacks and vulnerabilities. We designed a system using containers to strengthen the security of the visual lane-detection monitor and created adversarial controllers to test this new system.

This implementation has four main components: the sensor, the controller, the monitor, and the actuator. The sensor fetches a dashcam image, signs it with a timestamp, and passes this along to the controller. The controller detects lane lines in the image, and then sends a certificate (the signed sensor data and the location of the lane lines) to the monitor. The monitor makes checks to ensure that the proposed lane lines are consistent with the image and that the controller did not tamper with the sensor data. If the checks do not pass, the monitor intervenes by sending a signal to the actuator, presumably to prevent an unsafe scenario.

5.1 Threat Model

Our main goal is to prevent an attacker from convincing a self-driving car to continue when it should stop. To analyze possible attack vectors better, we need to examine each component. The sensor needs to be trusted because otherwise there would be no way to have any assurances about the car's state. The monitor is also trusted, since it is designed to be a small piece of software that makes simple checks on the certificate. However, the controller is untrusted because it is made of many more lines of code that may use unverifiable methods (such as machine learning) or untrusted/less verifiable software. This makes it more prone to bugs and potential exploitation by adversaries. Finally, the actuator must be trusted, because it is our only way of utilizing our decisions to command how the car will move.

We will assume that the attacker has a method of gaining full control of the controller. For example, the controller could have some internet-connected component in which the attacker could exploit a buffer overflow to gain control. Similar attacks have been performed before [48]. After gaining control, the adversary has the power to:

- Access image/timestamp output from sensor;
- Send any data to the monitor;
- Attempt to exploit some bug in its environment (ex: a known buffer overflow in Linux).

Recall that in the intended function of the controller, the image/timestamp is passed along to the monitor in addition to the proposed lane lines. Assuming full control of the controller, the attacker can perform all of, but not limited to, the following:

- Propose incorrect lane lines;
- Send a modified version of image and/or timestamp;
- Pass along stale images from the sensor;
- Not send anything to the monitor.

All of these could result in a car crash in an insecure model. In addition, while controller access does allow an attacker to spy using dashcam footage, our model is focused on the (arguably) more important issue of preventing car crashes.

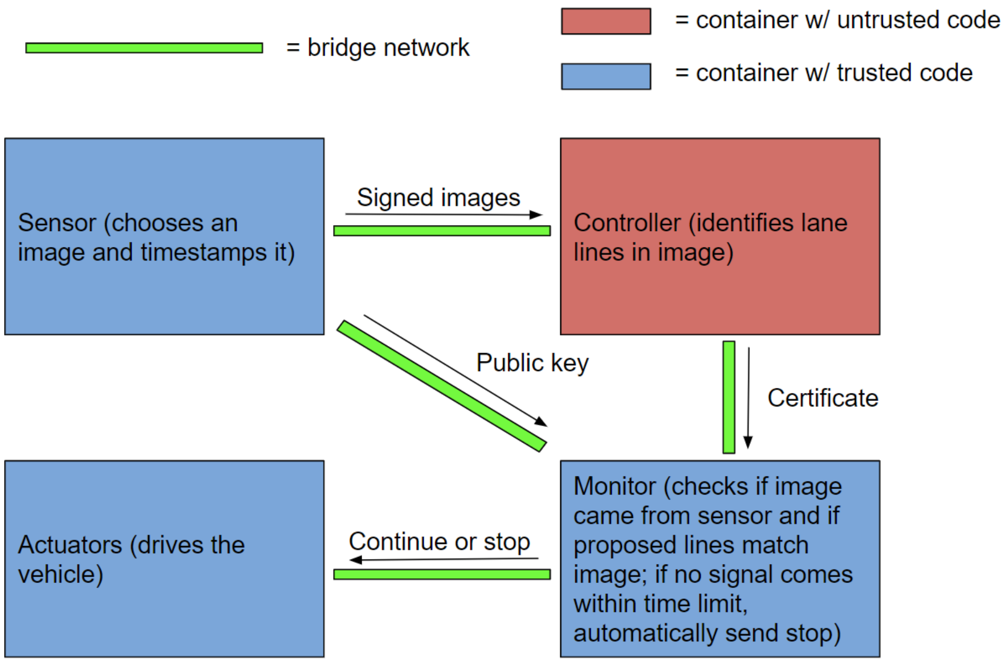

= bridge network
= container w/ untrusted code
= container w/ trusted code
Sensor (chooses an image and timestamps it)
Signed images
Controller (identifies lane lines in image)
Public key
Certificate
Monitor (checks if image came from sensor and if proposed lines match image; if no signal comes within time limit, automatically send stop)
Continue or stop
Actuators (drives the vehicle)

Figure 5-1: Our system design using Docker containers for isolation and bridge networks for communication between components

5.2 Design

To prevent the attacker from compromising the trusted components (sensor and monitor), we isolate each component into its own Docker container. We also create separate bridge networks with separate subnets between each possible connection (Fig. 5-1). Benefits of using containers include [49]:

- isolation in the form of separate, distinct namespaces between containers;
- isolation of network stacks between containers;
- control groups to limit resource allocation
- the ability to restrict; capabilities/privilege levels of specific containers;
- easier deployability and higher performance compared to virtual machines.

The goal is to have the monitor detect when the controller is trying to perform the aforementioned attacks. In addition, we would like the performance of the system to be able to handle the sensor generating images at .25-.5 seconds/image. The actual computer vision checks performed by the monitor are not a major focus of this section.

5.2.1 Sensor

The goal of the sensor is to produce dashcam image/timestamp data and securely pass it along to the controller. For performance reasons, the sensor also produces a low resolution version of the dashcam image. The sensor sends new data to the controller at fixed intervals (once every .4 seconds in our implementation).

Using RSA encryption, the sensor also produces a hashed signature by first concatenating the low quality image and timestamp data, and then computing the sha512 hash of the resulting string, finally using the private key to encrypt the resultant hash. Additionally, during the one-time startup of the system, the sensor generates a key pair using a 1024-bit RSA modulus, passes the public key directly to the monitor, and stores the private key within the sensor.

5.2.2 Controller

The responsibility of the controller is to perform perception and planning and produce a certificate that convinces the monitor that it is correctly performing these tasks. For our project, we are only having the controller perform perception and thus only verifying that the perception is correct.

The controller receives both the high and low quality images from the sensor as well as the signature produced from the images and a timestamp. It identifies the location of the lane lines on the image and passes along the data from the sensor, the proposed lane line locations (given as 2nd degree polynomials from the bird's-eye view), and additional metadata about the image to the monitor. The collection of data the controller sends to the monitor comprises the certificate. Note that the controller only sends the low-res version of the image for the monitor to check, to improve performance.

5.2.3 Monitor

The monitor takes the certificate given from the controller and does the computer vision checks described in Chapter 2 to see if the proposed lane lines are reasonable. Whenever a certificate comes in, the monitor checks that its timestamp comes after the last certificate (so the controller cannot send out of order certificates) and that the timestamp is not too old. In addition, the monitor intermittently checks whether or not the last certificate's timestamp is within a fixed delta away from the current time, ensuring that the controller is still sending certificates.

The monitor also determines if the low-res image and timestamp come unmodified from the sensor. The monitor first decrypts the sha512 hash from the sensor using the public key. Then the monitor takes the image/timestamp from the controller and produces the hash of that. The image is legitimate if the hashes are equal, otherwise it's been tampered with. With this model, a malicious controller cannot modify the image or timestamp because it would result in an unpredictably different hash, and cannot forge an entirely new signature because the sensor keeps the private key.

If any of the given tests do not pass, the monitor sends a signal to the actuator telling it to stop the car.

5.2.4 Actuator

The actuator takes a single boolean from the monitor; a false boolean signifies an intervention. In a physical car, this would cause the actuators to apply the emergency brakes.

5.3 Implementation

We used Python 3.7 to implement the system. To ensure isolation between components, we used Docker to create separate containers for each component (4 in total). Each container has a different IP address for each network it is a part of (ex: the sensor has a different IP for its communication network with the monitor than its communication network with the controller). For container communication, we used Python's socketserver library, and used the `socketserver.TCPServer` and `socketserver.StreamRequestHandler` classes. To coordinate the clocks, both the sensor and monitor share the same clocks as the host machine. The system is deployed through docker-compose for convenience.

For image processing and detection of road lines, we used OpenCV. More specifically, we used it in the monitor to check if the white bands were at proposed locations with the proposed shapes.

For signing and hashing of sensor data, we use the pycryptodome library, and their built-in RSA key generation and sha512 hashing functions.

5.4 Experiments and Results

To test the security of our implementation, we created three malicious controllers that:

1. Send a different image than the one given from the sensor.
2. Send along stale images from the sensor.
3. Stop sending certificates after a while.

The monitor correctly intervened for each malicious controller, and correctly did not intervene for the normal controller. We also found an image that causes the controller to incorrectly identify lane lines in the image; the monitor correctly catches this mistake.

Meeting our performance goals was challenging during implementation. Originally, when not run on docker, the system could handle images at $\sim .1$ seconds/image; however, performance deteriorated to$\sim$.4 seconds/image when run through docker-compose. This was likely due to the use of virtual bridge networks, as programs run on Docker have shown to run with near-native performance [50]. By decreasing the resolution of the signed image by a factor of 5, we were able to get the system to run consistently at $<.1$ seconds/image, meeting our goal.

5.5 Future Work

As self-driving cars become more popular, it becomes increasingly important that safety monitors are not only correct, but secure. A few possible methods of increasing security even further include:

- Reducing the size of the monitor: this would include modifying the code so that it does not depend on networking/security packages.
- Using stronger forms of isolation: Docker containers were used in this project for their performance and simplicity; however, stronger forms of isolation such as virtual machines would enforce better security; for example, since containers running on a machine operate within the same OS, an attacker could potentially exploit a bug in the OS to break out of the container and control the monitor.
- Formally verifying the monitor: the monitor is small and trusted, so it could potentially be feasible to verify the entirety of its code.

In addition, there are a few ways we could improve the performance of this system. One method is to not sign every image being passed to the controller. Although this might make the system more susceptible to vehicle hijacking while driving, the monitor only needs one frame to determine if the situation is unsafe. If images are still being signed several times per second, this would only give an attacker a split-second to cause an unsafe scenario. Another performance improvement would be to randomly hash a fraction of the image's pixels. This may potentially allow an attacker to change a few pixels, but there won't be a reliable way for an attacker to change the image so much that a different set of lane lines is produced.

Chapter 6

Conclusions and Future Work

6.1 Results and Conclusions

This research aimed to demonstrate how Certified Control can be used to increase safety in autonomous vehicles. We demonstrated through the visual lane-finding implementation that a monitor using simple checks can reveal both subtle and dramatic errors in production visual perception systems. We found that by having the controller perform most of the computation and provide a certificate acting as evidence of safety, the checks made by the monitor are able to remain simple, verifiable and relatively low latency. Although the current implementation can only be used for highway driving and even then still has limitations, these problems can potentially be solved by enriching the certificate to provide further evidence of safety in more complicated situations. For example, we augmented the vision certificate to include LiDAR data, which allowed us to check for an important feature of lane lines: that they must be on the ground.

We also showed that by including both position and velocity in a LiDAR monitor, we can ensure safety over a wider range of cases. More specifically, instead of not allowing objects to be closer than a set distance away, we can take into account each LiDAR point's position and velocity to determine whether that point poses a safety risk with respect to the ego vehicle's velocity. We simulated a real autonomous vehicle accident and demonstrated that this enhanced monitor was able to intervene and completely stop a collision.

Finally, we experimented with increasing the security of the monitor by isolating components into separate containers. The primarily goal was to enforce isolation between the trusted monitor and untrusted controller so we can be confident in the integrity of the monitor despite potential bugs in the controller. We showed that a Certified Control system can maintain high performance despite this isolation.

In our experiments, our Certified Control systems achieved reasonable levels of the three runtime monitor desiderata: our monitor for the most part did not intervene when the situation was safe (honesty), it correctly intervened in unsafe cases (soundness), and was much more verifiable than a typical controller as it used more traditional methods and contained much less code (verifiability). However, the tests we ran on these monitor implementations were relatively simple; it remains to show that these systems can maintain the three desiderata even in more complicated scenarios. This is particularly the case for the vision implementation which confirms the locations of lane-lines; our monitor makes certain assumptions that may not be realistic in all highway driving conditions. For example, it assumes that the lane lines are always fully visible. In cases where these assumptions do not apply, conventional perception methods may not prove to be enough; machine learning or other less verifiable methods may be necessary.

6.2 Future Work

6.2.1 Increased Testing

Extensions of this work would include further testing of both the vision and LiDAR monitors in simulation. CARLA provides near-limitless possibilities for simulating driving scenarios, and it presents an interface for quick development of these scenarios. The LiDAR monitor was only tested under a set of specific cases, so it definitely needs to be exposed to a wider variety of cases to be confident it improves safety. This process could potentially be expedited through random generation of test cases.

Moving forward, it is necessary that we develop some sort of benchmarking test suite so we can determine whether changes made to the monitor are actually beneficial. This would include both simple and more complicated cases where the monitor either should or should not intervene. This will allow us to maintain confidence that we are maintaining both *honesty* and *soundness*.

6.2.2 Addressing Complicated Scenarios

To handle more complicated driving scenarios, it may be necessary to use more complicated methods of verifying safety. For example, including temporal information in the monitor seems crucial to determining how objects will move and inferring where occluded objects are; we may need to incorporate path prediction-algorithms to determine how objects will move in a two-dimensional plane. However, as more complicated methods are introduced to the monitor, it is important that it remains verifiable because it is part of the trusted base.

Through our testing, Certified Control appears to be a good method for catching obvious failures in perception, such as when the ego vehicle is about to slam into a pedestrian. Moving forward, we should also focus on how these monitors act in less-obvious dangerous scenarios (e.g., small debris falling down a hill).

It is also very important that safety monitors maintain a high degree of honesty and do not intervene in safe scenarios. In particular, we need to address complicated scenarios where the situation is safe but may seem unsafe to a more naive monitor (e.g., a paper bag floating in the air). Safety monitors that intervene under safe circumstances often end up being turned off by users, as was the case in the Uber crash in Tempe. Therefore, our monitors need to be extended to accommodate a wider range of cases and make less assumptions.

6.2.3 Keeping Up-to-date with CARLA

CARLA is constantly being improved and updated, so it seems promising as a long-term method of simulating and testing our Certified Control designs. As more features are updated, it is important that we update how our monitor operates in CARLA. For example, the velocity data we are using in the enhanced LiDAR monitor does not come from CARLA's LiDAR sensor object; instead, we are mocking the velocity readings by directly reading object velocities from CARLA. If CARLA adds velocity readings to their LiDAR sensors, the certificate should be updated to use those velocities for more realistic results.

6.3 Final Thoughts

As the adoption of autonomous vehicles continues to increase, it is imperative that there are reliable safety mechanisms in place to assure the safety of passengers. Autonomous driving is a very difficult problem, and very complicated methods are being used to tackle it. Failures in driving software have resulted in death, and it is inevitable that further casualties are to come. As new technologies emerge to improve vehicle perception and control, we need to ensure that there remain methods of verifying passenger safety regardless of the complexity of these new technologies.

Creators of driving software must be held accountable to ensure that their safety technologies being used are verifiable and robust. Technology has the power to greatly improve quality of life as well as safety; it is important that creators of autonomous vehicles do not take shortcuts and respect the fact that human lives are in the hands of their technology.

Appendix A

Experiment with Adverse Lighting

We experimented with different color filters and the possibility of allowing the controller to select from a variety of filters for the monitor to use. This would allow the monitor to adapt to different lighting conditions, allowing it to work under a wider range of scenarios. In this experiment, we colored the road to make it light-grey so the white lane-lines are less visible (Fig. A-1). The controller proposes the correct ground truth lane lines and a set of generic color thresholds to the monitor, and the original monitor incorrectly rejects the certificate (Fig. A-3). When the controller proposes a set of thresholds adapted to the adverse lighting, the monitor correctly accepts the certificate.

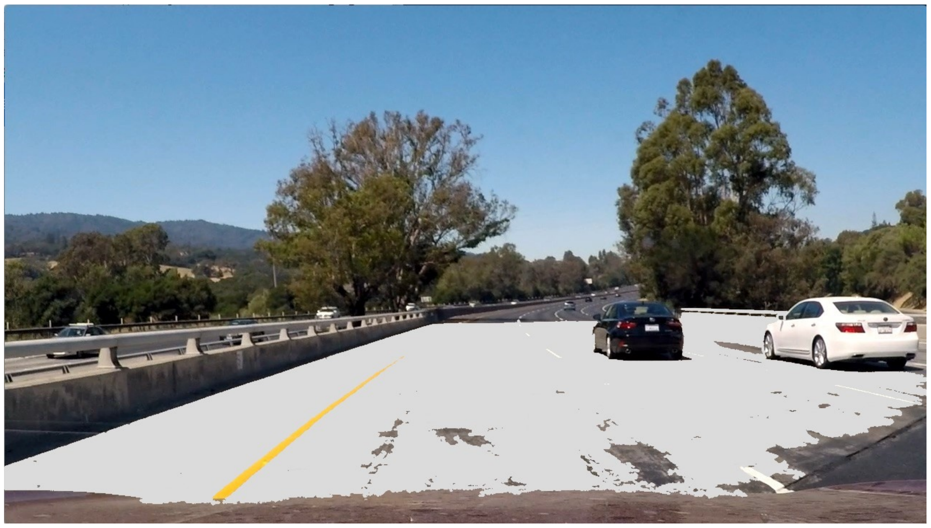
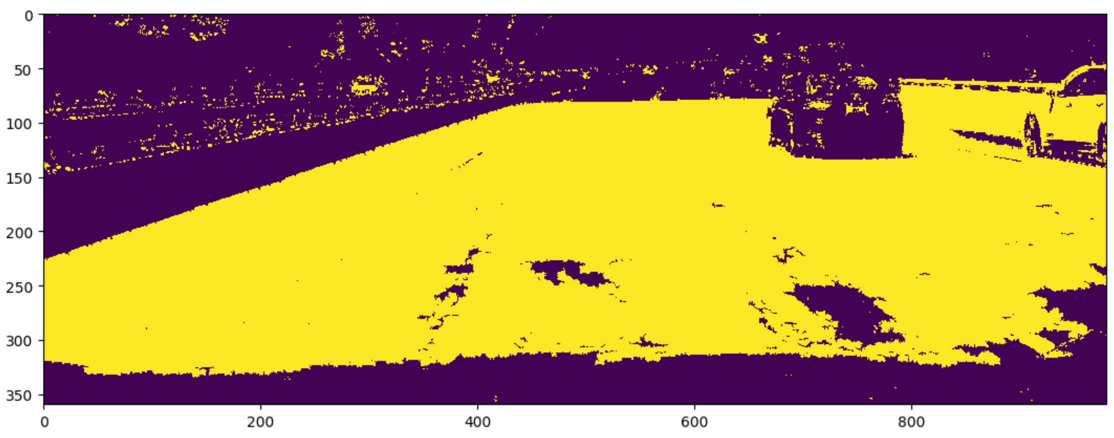
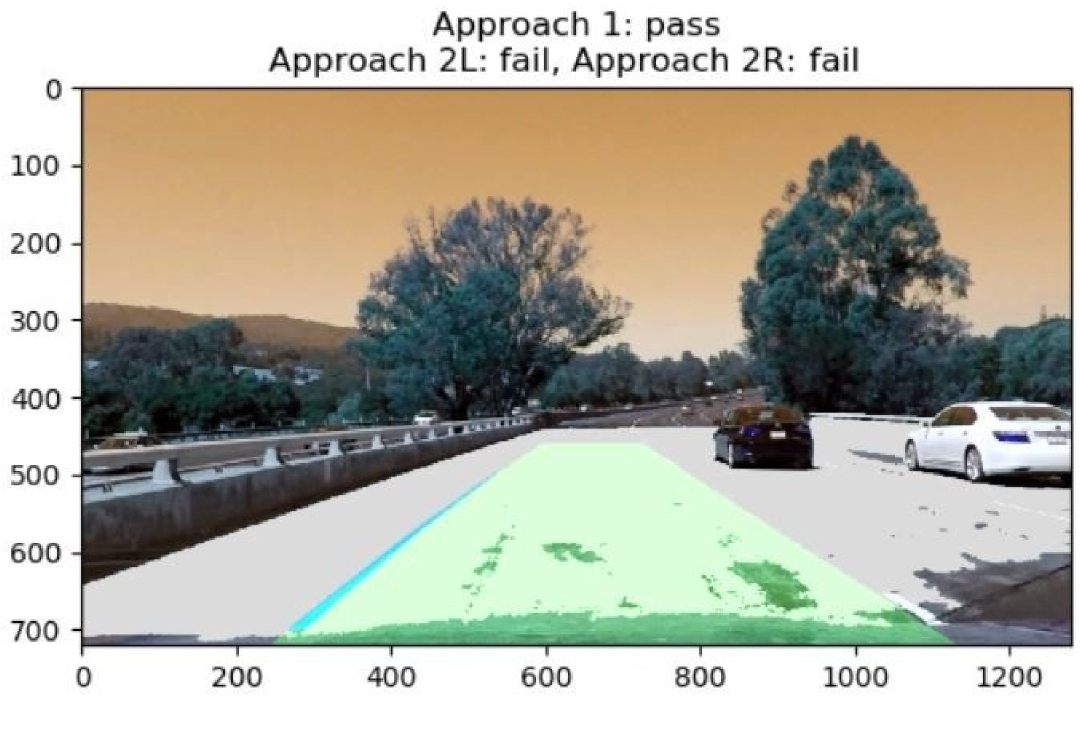


Figure A-1: A dashcam image with the road painted light-grey.

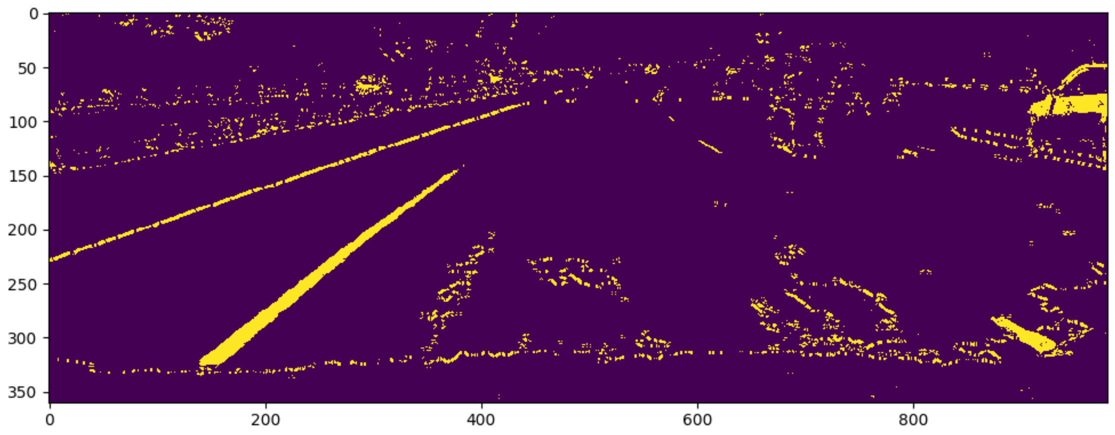


(a) Binary mask using the original color filter. The lane lines are almost completely masked by the grey-colored road.

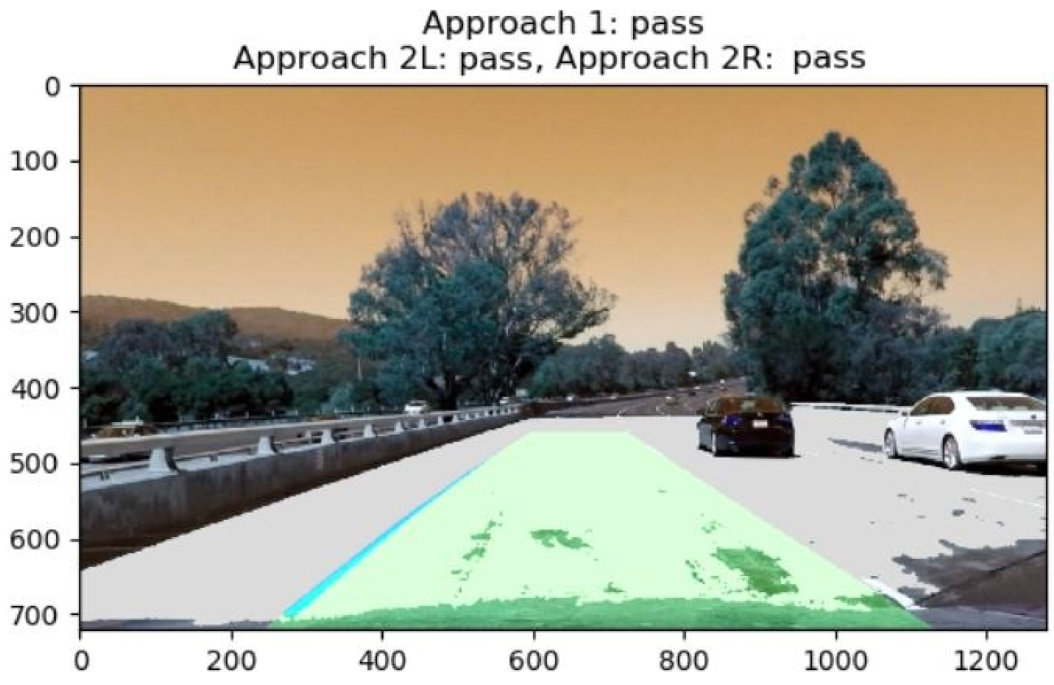


(b) Monitor result

Figure A-2: The monitor fails to confirm the lane lines given the adverse lighting. Both the left and right lane lines do not pass the conformance test.



(a) Binary mask using the adapted color filter. The lane lines are visible.



(b) Monitor result

Figure A-3: The monitor correctly confirms the location of the lane lines despite the adverse lighting.

Appendix B

Collision Analysis

We will prove the soundness of our stopping condition under a particular collection of scenarios. We will analyze the behavior of the ego vehicle, with the capability of decelerating at up to β_{egomax}, with respect to an object, with the capability of decelerating at up to β_{max} (also defined to be the maximum possible deceleration for a road vehicle). The object in question will henceforth be referred to as OIQ.

We will assume the following scenario:

- At time 0, the OIQ, with initial forward velocity v_f, is a distance d in front of the ego vehicle, with initial forward velocity v_{ego};
- At time 0, the OIQ decelerates at constant deceleration β_{max} until it fully stops;
- The ego vehicle continues at v_{ego} until time ρ, when it decelerates at constant deceleration β_{egomax} until it fully stops;
- Both the ego vehicle and the OIQ are moving only in the forward direction;
- The OIQ is directly in front of the ego vehicle.

Let x_o be the position of the OIQ after it has fully decelerated, and let x_e be the position of the ego vehicle after it has fully decelerated, assuming it waits ρ time before beginning its deceleration. We will analyze x_o and x_e independently of each other, ignoring potential collisions between the objects.

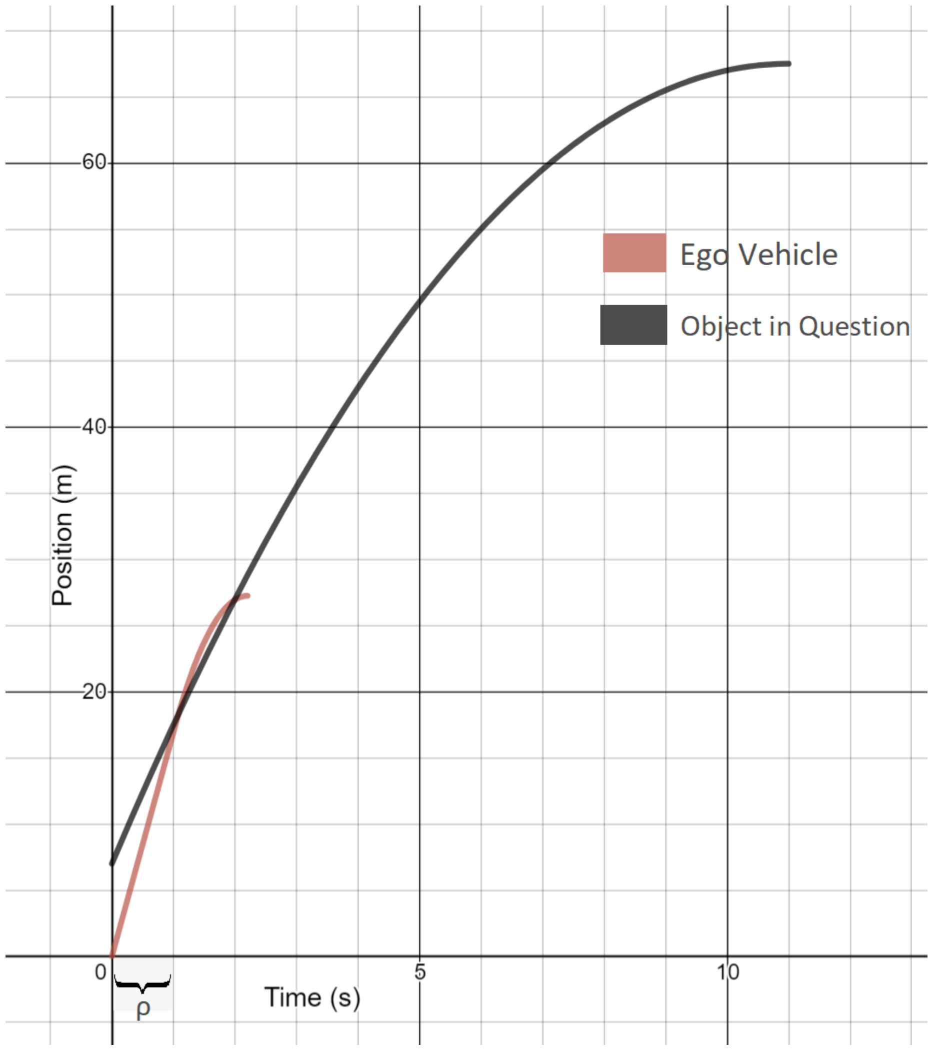


Figure B-1: A graph of a configuration where the ego vehicle and OIQ collide. The delay in the braking for the ego vehicle is shown as ρ. Analyzing both functions separately could lead one to believe there was no collision since the final position of the OIQ is in front of the ego vehicle. However, the ego vehicle has a higher deceleration, causing it to overtake the OIQ, and later fall behind it.

Given this scenario, our goal is to show that $x_e \geq x_o$ if and only if there will be a collision. We will model the movements of both the ego vehicle and the OIQ with second-order polynomial lines with position as a function of time. Since we assume that both objects will not continue accelerating after fully braking, we are only interested in the increasing portions of both functions. If the lines intersect at any point, then that signifies a collision.

If $x_e \geq x_o$, there must be a point where the lines intersect, so a collision must happen.

We will show that $x_e < x_o$ implies that there will not be a collision. First, we will show that the ego vehicle's deceleration never exceeds β_{max}. Recall that we define β_{max} to be the maximum possible deceleration for a road vehicle, so $|\beta_{max}| \geq |\beta_{egomax}|$. In addition, during the time ρ before the ego vehicle starts decelerating, the ego vehicle has a deceleration of 0. Thus, the deceleration of the OIQ is greater than that of the ego vehicle during this time as well.

Let us assume that $x_e < x_o$ and there was a collision. This implies that there must have been two intersections, one where the ego vehicle overtakes the OIQ, and one where the ego vehicle falls behind it. Fig. B-1 shows an example of how this might happen with two second-order polynomials. For the first intersection to have happened, the ego vehicle must have had a higher velocity than the OIQ. For the second intersection to have happened, the ego vehicle must have had a lower velocity than the OIQ. This implies that the ego vehicle has a higher magnitude of deceleration. However, this contradicts our assertion that the ego vehicle's deceleration never exceeds that of the OIQ. Therefore, this cannot happen given our assumptions.

Since $x_e \geq x_o$ implies there will be a collision, and $x_e < x_o$ implies there will not be a collision, this shows that $x_e \geq x_o$ if and only if there will be a collision.

Bibliography

[1] D. Wakabayashi, "Self-Driving Uber Car Kills Pedestrian in Arizona, Where Robots Roam," *The New York Times*, Mar. 2018.

[2] "Testing of Autonomous Vehicles." https://www.dmv.ca.gov/portal/dmv/detail/vr/autonomous/testing.

[3] O. Cameron, "The Driverless Readiness Score." https://olivercameron.substack.com/p/the-driverless-readiness-score. Library Catalog: olivercameron.substack.com.

[4] N. Kalra and S. M. Paddock, "Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability?," Tech. Rep. RAND RR-1478-RC, RAND Corporation, 2016.

[5] C. B. Weinstock, J. B. Goodenough, and J. J. Hudak, "Dependability Cases," Tech. Rep. CMU/SEI-2004-TN-016, CMU Software Engineering Institute, 2004.

[6] "Safety Management Requirements for Defence Systems: Part 2: Guidance on Establishing a Means of Complying with Part I"," tech. rep., UK Ministry of Defense, 2007.

[7] D. Jackson, M. Thomas, and L. I. Millett, eds., *Software for Dependable Systems: Sufficient Evidence?* National Research Council, 2007.

[8] "They Write the Right Stuff." https://www.fastcompany.com/28121/they-write-right-stuff.

[9] G. Klein and et. al., "seL4: Formal verification of an OS kernel," in *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles - SOSP '09*, (Big Sky, Montana, USA), p. 207, ACM Press, 2009.

[10] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Transactions on Computer Systems (TOCS)*, vol. 2, pp. 277–288, Nov. 1984.

[11] Y. C. Yeh, "Dependability of the 777 Primary Flight Control System," in *Dependable Computing for Critical Applications*, 1998.

[12] T. L. Crenshaw, E. Gunter, C. L. Robinson, L. Sha, and P. R. Kumar, "The simplex reference model: Limiting fault- propagation due to unreliable components in cyber-physical system architectures," in *IEEE International Real-Time Systems Symposium*, 2007.

[13] Lui Sha, "Using simplicity to control complexity," *IEEE Software*, vol. 18, pp. 20–28, July 2001.

[14] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," in *International Conference on Machine Learning*, p. 8, 2010.

[15] L. Pulina and A. Tacchella, "Challenging SMT solvers to verify neural networks," *AI Communications*, vol. 25, no. 2, pp. 117–135, 2012.

[16] E. T. Greenlee, P. DeLucia, and D. C. Newton, "Driver Vigilance in Automated Vehicles: Hazard Detection Failures Are a Matter of Time.," *Human Factors*, 2018.

[17] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a Formal Model of Safe and Scalable Self-driving Cars," *arXiv:1708.06374 [cs, stat]*, Oct. 2018.

[18] N. Charron, S. Phillips, and S. L. Waslander, "De-noising of Lidar Point Clouds Corrupted by Snowfall," in *Computer and Robotic Vision*, 2018.

[19] S. Manzinger and M. Althoff, "Tactical Decision Making for Cooperative Vehicles Using Reachable Sets," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, (Maui, HI), pp. 444–451, IEEE, Nov. 2018.

[20] S. M. Loos, A. Platzer, and L. Nistor, "Adaptive Cruise Control: Hybrid, Distributed, and Now Formally Verified," in *FM 2011: Formal Methods* (M. Butler and W. Schulte, eds.), vol. 6664, pp. 42–56, Berlin, Heidelberg: Springer Berlin Heidelberg, 2011. Series Title: Lecture Notes in Computer Science.

[21] N. Arechiga, S. M. Loos, A. Platzer, and B. H. Krogh, "Using theorem provers to guarantee closed-loop system properties," in *2012 American Control Conference (ACC)*, (Montreal, QC), pp. 3573–3580, IEEE, June 2012.

[22] N. Arechiga and B. H. Krogh, "Using verified control envelopes for safe controller design," in *American Control Conference*, 2014.

[23] S. M. Loos and A. Platzer, "Safe intersections: At the crossing of hybrid systems and verification," in *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, (Washington, DC, USA), pp. 1181–1186, IEEE, Oct. 2011.

[24] P. Koopman, B. Osyk, and J. Weast, "Autonomous Vehicles Meet the Physical World: RSS, Variability, Uncertainty, and Proving Safety," in *Computer Safety, Reliability, and Security* (A. Romanovsky, E. Troubitsyna, and F. Bitsch, eds.), vol. 11698, pp. 245–253, Cham: Springer International Publishing, 2019. Series Title: Lecture Notes in Computer Science.

[25] D. Phan and et. al., "A Component-Based Simplex Architecture for High-Assurance Cyber-Physical Systems," *2017 17th International Conference on Application of Concurrency to System Design (ACSD)*, pp. 49–58, June 2017.

[26] D. T. Phan, R. Grosu, N. Jansen, N. Paoletti, S. A. Smolka, and S. D. Stoller, "Neural Simplex Architecture," *arXiv:1908.00528 [cs, eess]*, Mar. 2020.

[27] L. H. Gilpin and J. C. Macbeth, "Monitoring Scene Understanders with Conceptual Primitive Decomposition and Commonsense Knowledge," *Advances in Cognitive Systems*, p. 20, 2018.

[28] L. H. Gilpin, "Reasonableness Monitors," *AAAI*, 2018.

[29] J. Kim and J. Canny, "Interpretable Learning for Self-Driving Cars by Visualizing Causal Attention," in *2017 IEEE International Conference on Computer Vision (ICCV)*, (Venice), pp. 2961–2969, IEEE, Oct. 2017.

[30] M. Koren, S. Alsaif, R. Lee, and M. J. Kochenderfer, "Adaptive stress testing for autonomous vehicles," in *IEEE Intelligent Vehicles Symposium*, 2018.

[31] X. Qin, N. Aréchiga, A. Best, and J. Deshmukh, "Automatic Testing and Falsification with Dynamically Constrained Reinforcement Learning," *arXiv:1910.13645 [cs, eess]*, Feb. 2020.

[32] A. Corso, P. Du, K. Driggs-Campbell, and M. J. Kochenderfer, "Adaptive Stress Testing with Reward Augmentation for Autonomous Vehicle Validation," in *IEEE Intelligent Transportation Systems Conference*, 2019.

[33] M. Mauritz, F. Howar, and A. Rausch, "Assuring the Safety of Advanced Driver Assistance Systems Through a Combination of Simulation and Runtime Monitoring," in *Leveraging Applications of Formal Methods, Verification and Validation: Discussion, Dissemination, Applications* (T. Margaria and B. Steffen, eds.), vol. 9953, pp. 672–687, Cham: Springer International Publishing, 2016. Series Title: Lecture Notes in Computer Science.

[34] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer, "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks," *arXiv:1702.01135 [cs]*, May 2017.

[35] R. Ehlers, "Formal Verification of Piece-Wise Linear Feed-Forward Neural Networks," *arXiv:1705.01320 [cs]*, Aug. 2017.

[36] G. Katz and et. al., "The Marabou Framework for Verification and Analysis of Deep Neural Networks," in *Computer Aided Verification* (I. Dillig and S. Tasiran, eds.), vol. 11561, pp. 443–452, Cham: Springer International Publishing, 2019. Series Title: Lecture Notes in Computer Science.

[37] J. Chow, V. Richmond, M. Wang, U. Guajardo, D. Jackson, N. Arechiga, G. Litt, S. Kong, and S. Campos, "Certified control, a new safety architecture for autonomous vehicles," *EMSOFT, submitted for publication*, 2020.

[38] W. J. Stein and T. R. Neuman, "Mitigation strategies for design exceptions," tech. rep., United States. Federal Highway Administration. Office of Safety, 2007.

[39] "Comma AI OpenPilot Software." https://github.com/commaai/openpilot, Apr. 2020.

[40] "Comma AI Driving Dataset." https://github.com/commaai/comma2k19, Apr. 2020.

[41] Z.-x. Wang and W. Wang, "The research on edge detection algorithm of lane," *EURASIP Journal on Image and Video Processing*, 2018.

[42] "Tesla Autopilot Drives Straight Towards Concrete Barrier on Highway."

[43] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, 1981.

[44] "Lidar speed-measuring device performance specifications." https://www.nhtsa.gov/sites/nhtsa.dot.gov/files/documents/809811-lidarspeedmeasuringdevice.pdf.

[45] N. Kudarauskas, "Analysis of emergency braking of a vehicle," *Transport*, May 2007.

[46] "Carla." https://github.com/carla-simulator/carla.

[47] "Vehicle automation report; tempe, az; hwy18mh010," tech. rep., National Transportation Safety Board, Office of Highway Safety, Washington D.C., 2018.

[48] "Hackers remotely kill a jeep on a highway." https://www.youtube.com/watch?v=MK0SrxBC1xs.

[49] "Docker security." https://docs.docker.com/engine/security/.

[50] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and linux containers," *IEEE International Symposium on Performance Analysis of Systems and Software*, 2015.