"Certified Control" Safety Architecture for Autonomous Vehicles: Applications with LiDAR

by

Valerie Richmond

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

May 2020

Author .....
Department of Electrical Engineering and Computer Science
May 12, 2020

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

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

"Certified Control" Safety Architecture for Autonomous Vehicles: Applications with LiDAR

by

Valerie Richmond

Submitted to the Department of Electrical Engineering and Computer Science
on May 12, 2020, 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 a safety monitor checks actions proposed by the main controller before they may be executed by the actuators. Unlike conventional runtime monitors, the certified control monitor receives an argument for the safety of the proposed action from the controller (rather than receiving data from the vehicle sensors directly). In this architecture, the monitor has the potential to do all of the following to a reasonable degree: intervene when safety is compromised, *not* intervene when safety is *not* compromised, and remain simple enough to be verifiable. First, this work describes the certified control architecture in detail, including how it achieves those three desiderata, which we argue are otherwise difficult to achieve simultaneously. Second, we present two novel applications of certified control: an implementation of LiDAR-based obstacle detection, and a LiDAR-augmented implementation of visual lane following. Finally, we evaluate those two systems using simulation and a physical robot car, and demonstrate that they indeed achieve the three desiderata.

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

Acknowledgments

I would like to thank my advisor Daniel Jackson for his outstanding mentorship and continued support, both academic and personal. His flexibility with my changing interests, dedication to my well-being, and confidence in my abilities (even when I lacked it), made this thesis possible.

I also owe thanks to all those on CSAIL's Software Design Group Interlock team, who helped in completing this work. Thanks especially to Mike Wang for his ROS expertise and assistance with experiments, to Jeff Chow for debugging my "trivial" trigonometric computations and co-presenting this work with me, and to Geoffrey Litt for his consistent help in clarifying my thoughts.

Finally, I so appreciate my friends and family for keeping me going through the years. Thank you, Nathaniel, Dad, Melinda, Alexandra, Anlong, Hope, and others.

This project was supported in part by the Toyota Research Institute in a collaboration between TRI and MIT CSAIL.

Contents

1 Introduction **9**

1.1 Motivation 9

1.1.1 The Problem of Safety 10

1.1.2 An Alternative to Testing 10

1.1.3 The Cost of Verification 11

1.1.4 Small Trusted Bases 12

1.1.5 Runtime Monitors and Safety Controllers 13

1.1.6 The Problem of Perception in Autonomous Vehicles 13

1.1.7 Desiderata for a Runtime Monitor: Choose Two 14

1.2 Certified Control 15

1.2.1 LiDAR Application 16

1.3 Related Work 18

1.4 Contributions 20

2 Certified Control for LiDAR **21**

2.1 Design 21

2.1.1 Monitor 21

2.1.2 Controller 26

2.1.3 Implementation Notes 30

2.2 Experiments 31

2.2.1 Setup 31

2.2.2 Well-tuned Controller 31

2.2.3 Over-filtering Controller 32

2.3 Evaluation . . . 33
2.3.1 Experiments . . . 33
2.3.2 Minimum Certificate Algorithm . . . 33
2.3.3 Complexity Gap . . . 35
2.3.4 Speed . . . 36
2.4 Limitations . . . 36
2.4.1 Blindness of LiDAR . . . 36
2.4.2 Lack of Time Domain Awareness . . . 37
2.4.3 Data Non-uniformity . . . 38

3 LiDAR Sensor Signatures 39
3.1 Assumptions . . . 39
3.2 Simulation Setup . . . 41
3.3 Signature Schemes . . . 41
3.3.1 Simple Hashing . . . 41
3.3.2 Hashed RSA . . . 42
3.3.3 Ed25519 . . . 43
3.3.4 Digital Signature Standard (DSS) . . . 43
3.4 Evaluation . . . 43

4 Certified Control for Vision 45
4.1 Background . . . 45
4.2 Combined Vision and LiDAR Certificate . . . 47
4.2.1 Ground Plane Subcertificate . . . 48
4.3 Controller . . . 49
4.3.1 Lane Line LiDAR Points . . . 49
4.3.2 Ground Plane Detection . . . 50
4.3.3 Subcertificate Generation . . . 53
4.4 Monitor . . . 53
4.5 Experiment . . . 54
4.6 Evaluation . . . 54

4.7 Limitations 56
4.7.1 Ground Plane Subcertificate 57

5 Conclusions and Future Work **59**

Chapter 1

Introduction

1.1 Motivation

To begin, we present the following motivating narrative piece-by-piece: autonomous vehicle (AV) systems require not just safety, but a highly demonstrable *confidence* in that safety. Statistical testing is often the standard for establishing such confidence, but it cannot possibly cover the range of inputs to cyber-physical systems like AVs. Instead, AV designers can adopt "safety cases" to demonstrate confidence. However, a safety case requires, among other things, that the software components of the system be verified, which is exorbitantly expensive. This motivates the idea of a small trusted base, often embodied in a safety controller. Such a safety controller must include a convincing argument that the perception system is accurate, since that perception system is in the trusted base. This thesis includes work to create such convincing on-the-fly arguments for the perception system's accuracy, not through a safety controller, but through the "certified control" architecture's monitor, described later.

1.1.1 The Problem of Safety[1]

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 [42]—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 [40], 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 [5], 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) [19].

1.1.2 An Alternative to Testing

Statistical testing is the gold standard for quality control for many products because it is independent of the design and development process. This independence is also a weakness, though, because it denies the designer the opportunity to use the structure of the system to bolster the safety claim, and at the same time fails to focus testing on

[1]The text in this section and the following sections, up to and including Section 1.3, are taken from the coauthored paper [8]. Other material is attributed in Section 1.4

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 [43]. The quality of the argument and the extent to which experts are convinced is then the measure of confidence. This approach lacks the scientific basis of statistical testing, but is widely accepted in 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 expensive testing, but on analytical arguments for stability and resilience in the presence of anticipated forces.

In software too, there is growing interest in safety cases (or, more generally, assurance or dependability cases) [18]. 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 must cover the specification of the software that controls the machine, the physical properties of the environment, and the behavior of human users and operators. Each part of this argument 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. Some parts, of course, are not amenable to mathematical reasoning. Properties of the environment, for example, must be justified by expert inspection.

1.1.3 The Cost of Verification

For software-intensive systems, the software itself can become problematic to justify. Complex systems require complex software, which 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. The cost tends to be orders of magnitude higher than for conventional software development. NASA's flight soft-

ware, for example, has cost over $1,000 per line of code, where conventional software might cost $10 to $50 per line [14]. Verifying a large codebase 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 [23].

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" [36]). As long as encryption and decryption algorithms that execute at the endpoints of the transmission 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. 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 smaller 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, for example, 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 [41].

1.1.6 The Problem of Perception in Autonomous Vehicles

The safety controller approach relies on the assumption that the controller itself is the most complex part of the system. But in the context of autonomous vehicles, 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 [11, 37]), 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 scale and their use of non-linear activation functions (such as ReLU [30]) which confound automated reasoning algorithms like SMT and linear programming [33].

One possible solution to the difficuly of verifying the percetion system is to exclude perception functions from the trusted base, and to use a runtime monitor that incorporates both a safety controller and a simplified perception subsystem. Initially, this approach seemed viable to us, but we came to the conclusion that it is in fact not. We explain why in the next subsection.

1.1.7 Desiderata for a Runtime Monitor: Choose Two

We enumerate three critical properties that a monitor should obey, and argue that they are mutually inconsistent.

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 [37]).

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 [17].

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, 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 com-

plexity. 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 [38], 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

This thesis presents work related to a new architecture that is designed to achieve a reasonable compromise of the above desiderata, without a safety controller. The architecture is a variant on the traditional concept of the runtime monitor. Like a conventional monitor, our monitor checks actions proposed by the main controller before passing them on to the actuators; if an action is found to be unsafe, it is blocked or replaced by a safer action. If the action is found to be safe, the monitor has essentially ratified a runtime safety case.

Unlike a conventional monitor, however, the monitor does not make this decision based on its own evaluation of the environment. Instead, it checks a certificate generated by the main controller, as shown in Fig. 1-1. The 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 is an image of the road ahead, the interpretive data might be the purported lane lines; if the sensor data is various low-lying LiDAR points, the interpretive data might be an

approximation of the ground plane.

The generation of the certificate is the responsibility of the main controller, and is a byproduct of its normal analysis. The certificate is unforgeable due to being a subset of the signed sensor unit outputs: even a malicious agent could not convince the monitor that an unsafe situation is safe. The form of the certificate, and the argument that a successful check ensures safety, are developed at design time. This development, while possibly time-intensive, enables the concise articulation (shared with the entire team) of the rationale for the controller's decisions, which is helpful for implementation.

This architecture exploits four powerful computer science ideas in a novel combination and context: (1) the gap between the complexities of **finding vs. checking**; (2) the idea of a **small trusted base** (here, the monitor, sensors, and actuators) allowing a guarantee of safety without dependence on unreliable components (in particular the perception system) outside the base; (3) the use of **authentication** (e.g., of sensor data) to guarantee safe transmission through an untrusted channel (namely the main perception/controller subsystems); and (4) the idea of **end-to-end safety cases** whose form is justified at design time, but which are applied on the fly.

The next section describes an example to illustrate the architecture.

1.2.1 LiDAR Application

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

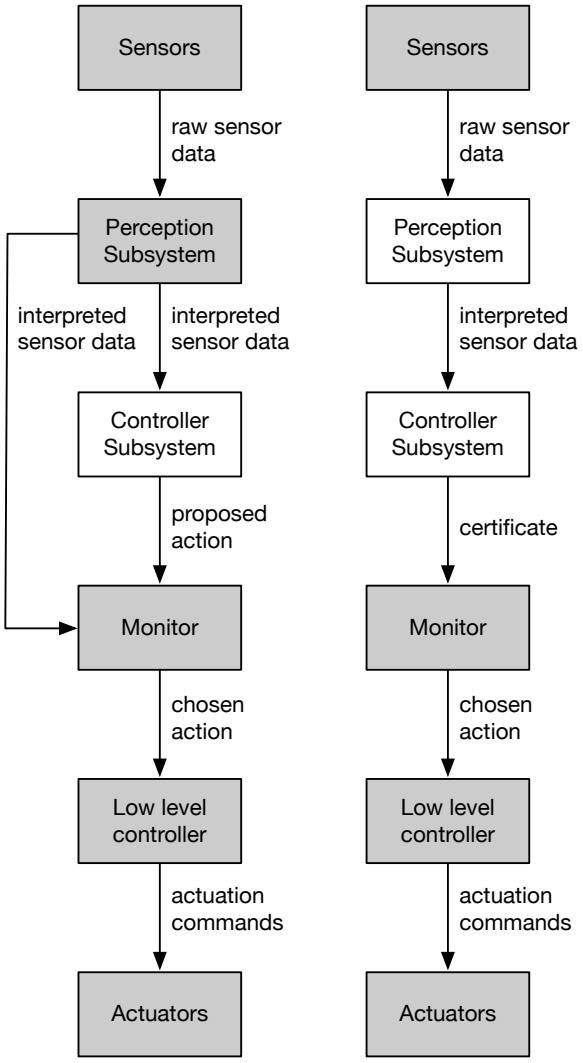


Figure 1-1: A conventional runtime monitor (left) and certified control monitor (right). The trusted base is shown in gray.

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

1.3 Related Work

The literature on safety assurance of vehicle dynamics generally focuses on assurance of the planning/control system, and assumes perception is assured by some other mechanism.

Some research 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 [28]. Other work focuses on deductive proofs of safety. For example, the work of [26] 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 [1, 2] demonstrates how these safety constraints can be used for verification and synthesis of control policies, including control policies with switching. And in [26], the authors develop safety contracts that include intersections, and provide KeYmaera proofs to demonstrate safety.

Responsibility-Sensitive Safety (RSS) [38] is a common 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 reli-

able. 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 [31, 37] uses two controllers: a high-assurance controller and a performance controller, running in parallel. The high-assurance subsystem is meticulously developed with conservative technologies; the 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. [31] extends Simplex to neural network-based controllers. As noted, this approach does not address flaws in perception. In contrast, reasonableness monitors [15, 16] 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 [22] 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 [10, 25, 34]. 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 [29]. 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.

1.4 Contributions

The introductory sections above were taken and lightly modified from the recent coauthored paper [8]. Parts of Sections 2.2, 2.3.1, and 4.1, 4.5, and 4.7 are also taken, and more heavily modified, from the paper.

The contributions of this thesis include (1) an implementation of certified control for LiDAR-based obstacle detection (including design of the certificate, creation of the monitor and controller subsystems, and signing of sensor data); (2) the augmentation of LiDAR data, and a ground plane certificate, to a certified control certificate for visual lane following; (3) evaluation of each of those implementations using a physical racecar; (4) analysis of the potential and limitations of the certified control approach in these domains.

Selections from the discussed code are publicly available at `https://github.com/interlock-mit/interlock` and `https://github.com/jefftienchow/interlock`.

Chapter 2

Certified Control for LiDAR

2.1 Design

Although certified control can be implemented in a variety of contexts and systems, we implemented the architecture on a robot car. We designed a LiDAR certificate that confirms the perception system correctly identifies the lack of obstacles ahead of the car. The controller produces this certificate, proof of the absence of obstacles ahead within a certain distance, in an attempt to justify the decision to move forward at the same speed. Other controller decisions are of course possible, and would require different proof in the certificate, but we focused only on moving forward. Here we describe the details of each of the system's components.

2.1.1 Monitor

To ensure the safety of a controller's decision to move forward, the monitor checks that the points in the certificate given to it satisfy four conditions: *authenticity*, sufficient *distance*, sufficient *spread*, and sufficient *density*. Authenticity of the points is determined by the signature verification described in Chapter 3. Sufficient distance would be determined by the car's speed: the LiDAR points must be beyond the stopping distance at the current speed, or else they indicate obstacles which would impact the car. For our robot car implementation, whose speed remained in a very predictable

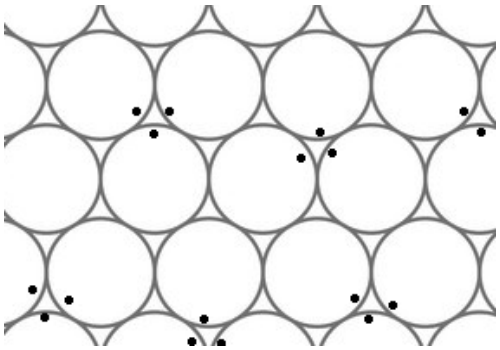


Figure 2-1: Visualization of one considered notion of density: requiring at least one projected LiDAR point from the certificate to be within each circle on a grid of every-row-offset circles. The black points show a worst-case choice of LiDAR points by the controller, which results in circle-sized holes in the certificate of radius greater than that of the grid circles.

range, we hardcoded an allowable distance. The monitor ignores any points which are closer than that distance. To achieve sufficient spread, certificate points must span the size of one lane in front of the car, both vertically and horizontally. The size of the lane is hardcoded based on the dimensions of the robot car; for a real car, the width would be based on US highway standards, and the height on standard vehicle heights.

Density Criteria

We explored many notions of density which the monitor could require of the certificate points. We considered requiring the certificate to contain one point inside every circle within a grid of circles projected onto the cross-section of the lane (Fig. 2-1); one point inside every cell of a matrix imposed on the cross-section of the lane; one point inside every cell of such a matrix in which every other row is horizontally offset; etc.[1]

However, because of the symmetry of these density requirements (i.e. their requirements being the same vertically and horizontally), these could not be meaningfully applied as the monitor's density check, due to the non-uniform nature of the LiDAR data. Our car's LiDAR unit provides only sixteen rows of rotating sensors, each of

[1]We also considered allowing the controller to give an interactive proof of the density of points, through a protocol in which the monitor can query the controller about specific suspicious regions of points. However, this wasn't obviously superior to a classical proof, and would cost time for the back-and-forth.

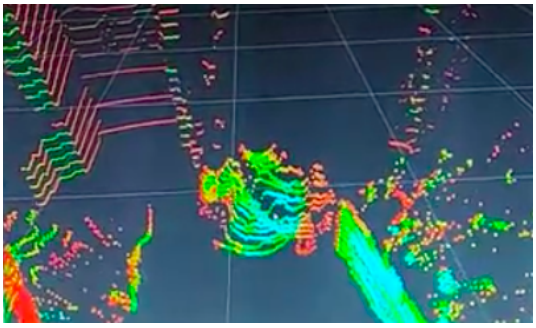


Figure 2-2: Visualization of a partial LiDAR scan. Light gray grid is a remnant of the visualization tool. Colored points are LiDAR readings. Notice, as is especially visible in the top left corner, they form "stripes" across the space, since there are many more point per scan row than there are scan rows.

which produces thousands of points per scan, so the data is much more dense horizontally than vertically. A visualization of a partial LiDAR scan in Figure 2-2 illustrates this property. The density check we employ therefore differs in the two directions. 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 `max_horizontal_separation`.

Since the LiDAR points may fall spatially anywhere in the 3D region, we must precisely define the notion of horizontal distance. In particular, we want it to be the case that two pairs of points, each of which contain points collected from two different LiDAR sensor rays at an angle α apart, have the same "horizontal" distance from one point to the other, regardless of the data points' distance from the sensor. That is, the notion of horizontal distance we care about is that in the angular domain: we want that the LiDAR sensor did not need to rotate very far further from one point in the certificate, to hit the next point in the certificate. We do not, on the other hand, care about the distance from the sensor to the point on the certificate, nor the distance in the 3D region's "x" dimension, between the two points. To achieve this definition, the horizontal density check is performed after a projection of the certificate points onto a plane oriented like the cross-section of the road ahead, at a set distance away from the LiDAR unit. This projection enables the use of horizontal distance on the plane

(the "x" dimension distance) between two points as a proxy for the angular distance traveled by the LiDAR unit to read those two points.

A Note on Accuracy

A perfectly accurate scheme would project the certificate points instead onto a slightly curved plane, whose curvature is dependent on specifications of the LiDAR unit. The flat projection has the undesirable property that a pair of points on the edge of the plane with the same "x" dimension distance between it as a pair of points in the center of the plane, actually has slightly greater "angular" horizontal distance between it than the pair on the center of the plane. However, this discrepancy is slight given the relatively small size of the lane's cross section, and an appropriate (slightly larger) choice of `max_horizontal_separation` can mask the issue.

Sufficiency of the Criteria

One possible choice of `max_horizontal_separation` ensures no obstacles of width 3.5cm or greater at a distance of 3.7m ahead, or of width 90cm or greater at 96m ahead (the 70mph stopping distance)[2]. This parameter value is fairly conservative; it would establish the absence of a car but not a motorcycle in the lane ahead. For greater safety, one could decrease `max_horizontal_separation` to require more LiDAR points, increasing the certificate size and the time to check it proportionately. In a production implementation, the required density of LiDAR points in the certificate should depend on the current velocity, since greater resolution is required to prevent collision with the same size obstacle at a greater stopping distance.

Together with the spread and distance criteria, an appropriate choice of `max_horizontal_separation` guarantees the absence of obstacles of certain sizes and shapes. To formalize this, consider an oblong obstacle, such as a mailbox, with approximate height 1 meter and width 0.25 meters, at a 10 meter distance directly in from of the vehicle. Assume that the vehicle is traveling at a speed which makes that

[2]As discussed in the note in Section 2.1.1, these allowable obstacle widths would change slightly based on whether the obstacle lies within the center of the lane cross section, or near the edge. Our analysis here assumes obstacles are in the center of the plane.

distance insufficient for safety. A malicious or faulty controller wishing to justify the unsafe action to move forward at the same speed has three options for its certificate creation[3]:

1. Include all LiDAR points on the obstacle in the certificate

2. Include some LiDAR points on the obstacle in the certificate, but exclude some others

3. Exclude all LiDAR points on the obstacle from the certificate

The monitor, in each of these scenarios, detects the insufficient evidence of safety, and rejects the controller's unsafe proposal to move the vehicle forward:

1. In this case, since the certificate contains points within the lane which fall closer than the allowable distance (based on the vehicle's speed), the distance criterion is not met.

2. Just as above, the distance criterion is not met.

3. If the certificate does not contain enough points to span the entire lane ahead of it (e.g. it is a degenerate certificate with no points in it, or one which spans only the obstacle-free portion of the lane), then the spread criterion is not met.

   If the spread criterion is met, meaning that the certificate points do extend to the edges of the lane, then the density criterion must not be met, since the certificate contains a horizontal gap where the obstacle is. Assuming the parameter is not well outside of the reasonable conservative range, this gap will be larger than the allowable `max_horizontal_separation`.

The controller is thus unable to hide the obstacle from the monitor.

[3]The only other option is for the controller to include in the certificate points which aren't in the LiDAR data from the sensor. Inclusion of those fake points, however, would cause the monitor to reject the certificate as inauthentic.

2.1.2 Controller

The controller of a production AV system would house complex algorithms, often using machine learning. We did only some basic processing, but even this proved sufficient to demonstrate the benefits of certified control and illustrate a complexity gap between the monitor and the controller. We describe two of the main implementation components, snow filtering and the algorithm for choice of minimum certificate.

Snow Filtering

The key example of perception we implemented in our controller was the filtering out of LiDAR points hitting snow.

To do this filtering, we at first implemented statistical outlier removal (SOR). This technique constructs a k-d tree containing all points in the LiDAR cloud. A k-d tree is a spatial data structure that allows for efficient querying of nearest neighbors, among other things. The tree is used to compute the mean and standard deviation of the distance of each LiDAR point to its nearest k neighbors. A threshold T is then set to be some β_{SOR} number of standard deviations above the mean. Any points whose mean distance to its k neighbors is above T are filtered out [6].

However, we transitioned to a slightly more sophisticated technique, dynamic radius outlier removal (DROR), in order to account for the nonuniformity of the LiDAR data. In particular, LiDAR points become more sparse as the distance from the sensor increases, since the laser rays emanate from the single sensor point. This means that a SOR filter would be overly likely to filter out distant points, since they cannot have as many close neighbors.

DROR takes this nonuniformity into account[4]. After creating the k-d tree like in SOR, it computes, for each point, the number of neighbors found within some radius. The radius is dynamic: it depends on the distance of the particular point from the sensor, the angular resolution of the LiDAR sensor (a constant for the sensor, which describes the the extent to which the LiDAR rays spread out with distance), and

[4] [6] gives a more thorough comparison of the SOR and DROR approaches, and lends more evidence to the superiority of the latter for LiDAR data.

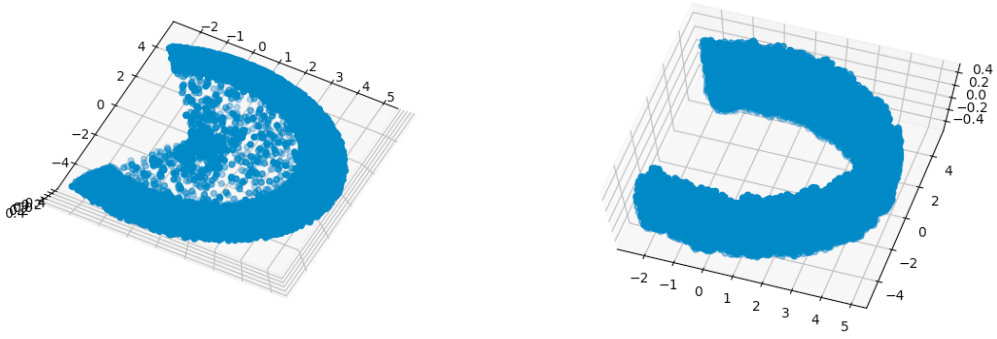
a multiplicative parameter β_{DROR} [5]. Any points which have fewer than some set number n of neighbors within this radius are identified as snow and removed.

Visualizations of a fake LiDAR dataset before and after DROR filtering are seen in Fig. 2-3. The dataset was generated to mimic a real snow environment, with most of the LiDAR points falling on a semicircular obstacle ahead, but some random subset of points hitting snow points closer to the car. Notice that the density of snow points is greater closer to the sensor (single points are drawn with partial opacity to make this more clear), a phenomenon which one would observe with real data as well, and which motivates use of the DROR filter.

Of course, the correctness of the described filtering depends on reasonable setting of the parameters. In DROR, the parameters are β_{DROR} and n. We chose a suitable pair of values for those based on simulated experiments. For example, we generated fake data containing obstacles of various sizes and shapes, and ensured that the obstacle was not filtered out as snow. One such dataset is shown, before and after filtering, in Fig. 2-4. Notice that the filtering was not perfect on this particular example: it failed to filter out one snow point. But, importantly, it maintained the contours of the obstacle.

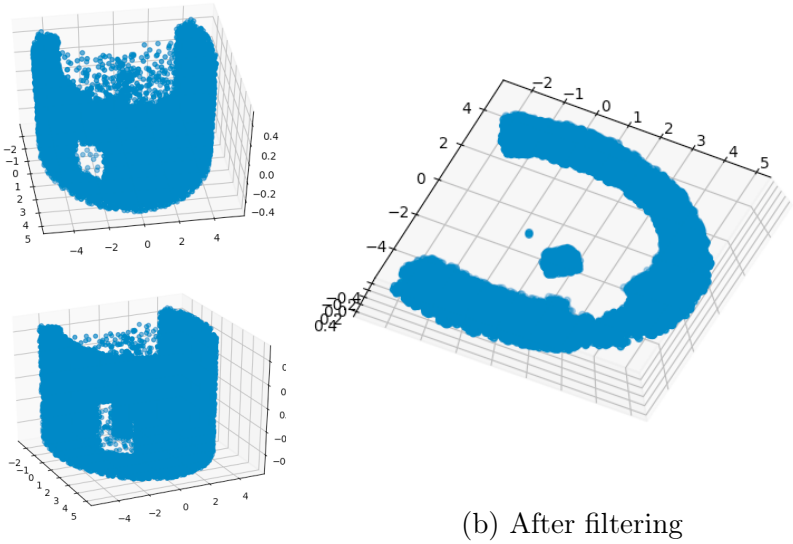
If this experimental tuning method seems somewhat insufficient or unsatisfying to ensure well-set parameters, we agree. Tuning the algorithm parameters is a tricky task, yet one upon which the correctness of the algorithm relies entirely! In a production system, the parameter setting would be cast in terms of the safety case, i.e. it would be based on a definition of snow independent of the implementation. However, one can imagine that such a definition is difficult to formulate, and even complex and intelligent controllers may not achieve perfectly tuned parameters, making their snow filtering too (in)sensitive. Our experiment in Section 2.2.3 addresses just this.

[5] We also require, for practical purposes, the search radius to be at least some small minimum value.



(a) Before filtering (b) After filtering

Figure 2-3: Results of DROR snow filter on generated dataset.



(a) Before filtering (b) After filtering

Figure 2-4: Results of DROR snow filter on data with fake obstacle.

Minimum Certificate Generation

After snow filtering, the controller has a set of candidate LiDAR points for its certificate: all those which are not snow. Under most circumstances, this will still be the vast majority of the LiDAR points—many more than might be strictly necessary to prove safety to the monitor. The controller therefore attempts to reduce the set further. It implements an algorithm to transform the large candidate LiDAR set into a final, smaller set for the certificate. We describe the algorithm and argue that it upholds two guarantees: first, that if the candidate LiDAR set was convincing to the monitor, then the resulting smaller set should also be convincing, and vice versa; and second, that it does indeed reduce the number of LiDAR points to close to minimal.

The first part of the algorithm is to eliminate any points which the controller thinks are on obstacles. In our implementation, this simply includes points which are too close to the sensor (given the car's assumed stopping distance). This necessarily maintains the acceptability of the certificate points, since the monitor checks only apply to points which are not on obstacles, i.e. are far enough away from the car. A large and close obstacle would be interpreted as an unacceptable certificate hole by the monitor whether or not the too-close points are in or out of the certificate.

The second part[6] of the algorithm is to use knowledge of the monitor's density and spread checks to eliminate LiDAR point candidates. To take advantage of the spread check, the controller may eliminate all candidate points which lie outside the considered lane-sized cross section ahead.[7] To do this, it considers, like the monitor would, the projection of the points onto the plane of the lane's cross-section. It eliminates candidate points outside of the monitor's desired rectangular spread.

To take advantage of the density check, the controller eliminates candidate points

[6]Unlike the description here, the actual algorithm implementation is not incremental in this way—it performs many of the checks on the candidate LiDAR points at the same time, in order to reduce the number of loops through all the points.

[7]This suggests that it may have been computationally wasteful to perform snow filtering on the entire point cloud, if the controller need only create a certificate covering the lane ahead. However, we expect that performing snow filtering on the whole point cloud yields much better results, since it allows the thresholds to reflect entire surrounding weather environment. And the controller as is is fast enough, as seen in Section 2.3.4. The exact tradeoff of speed and accuracy of filtering with different number of LiDAR points could be explored in future work.

which are redundant in satisfying the density. For horizontal density, it ensures that there is at most one point within each gap of size `max_horizontal_separation`. For this, it goes through the remaining candidate points in each row of LiDAR data in sorted order, from left to right. It adds the leftmost candidate point to the certificate. Then, it examines the points which fall within `max_horizontal_separation` distance rightward of that point. It adds the rightmost of those points to the certificate, and rejects the others. It then repeats this process, advancing by steps of at most `max_horizontal_separation` rightward, adding at most one point to the certificate per step. This algorithm is greedy, and could theoretically be improved by employing dynamic programming. Specifically, one can imagine adversarial LiDAR candidate sets in which taking the rightmost candidate point within `max_horizontal_separation` distance is unideal, and causes one to have to select more points later down the line. A more sophisticated algorithm which "thinks ahead" about that possibility, and explores all possible choices of points, would be superior. However, the density of the candidate LiDAR points is so high that the advantages of such an algorithm would be minimal. Additionally, the greedy approach already gives us significant reduction in the certificate size, with relatively little additional computation by the controller, as we show in Section 2.3.

2.1.3 Implementation Notes

The certificate itself is implemented as an instance of a `Certificate` class in Python2. Each instance stores the LiDAR points as a NumPy array, and a representation of the action of whose safety it is meant to prove (e.g. move the car forward).

Another team member's work to securely isolate the monitor from the controller is ongoing. The proposal is to run each component in its own Docker container, with bridge networks between components allowed to communicate.

An architectural diagram of the certified control implementation is shown in Figure 2-5.

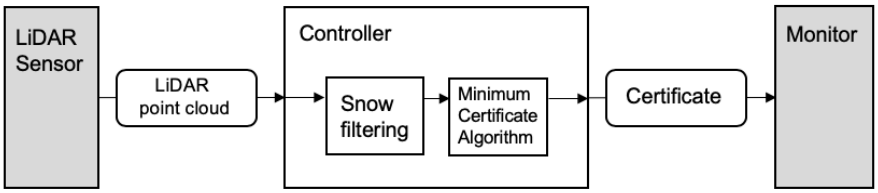


Figure 2-5: High-level architecture of the LiDAR certified control system. The trusted base is again shown in gray. Pathways for authentication of LiDAR points are omitted for simplicity. More context shown in Figure 1-1.

2.2 Experiments

Here we describe two experiments which together show that the robot car implementation achieves the safety benefits of certified control.

2.2.1 Setup

The experimentation platform was our 1/10-scale remote-controlled robot car (Fig. 2-6), outfitted with a Velodyne Puck VLP16 LiDAR scanner and a CPU running Linux with ROS. The above-described Python2 implementations of a controller and monitor were loaded onto the car and executed in real time. We physically manipulated the car's environment to reflect various road conditions.

2.2.2 Well-tuned Controller

When the controller's filtering parameters are set appropriately, it properly identifies points as snow and passes a certificate excluding them to the monitor. Assuming snow is distributed fairly evenly across the lane ahead (and there is not a total "white out") the remaining points are still sufficiently dense to establish absence of an obstacle in the lane ahead, so the monitor should approve the certificate.

We simulated snowy conditions by dropping confetti-like paper in front of the robot car (Fig. 2-7a). We confirmed that the monitor accepts a certificate when there is sufficient space between the car and an obstacle ahead, despite the presence

Figure 2-6: Remote-controlled robot car outfitted with LiDAR unit.

of simulated snow. Fig. 2-7b shows a visualization of the LiDAR point cloud associated with the experiment. The green-highlighted points are those in the accepted certificate, and the purple-highlighted points are those eliminated from the certificate by the controller, based on snow filtering. The gray grid lines are a remnant of the LiDAR visualizer, irrelevant here.

This experiment provides evidence that the certified control implementation upholds *honesty*, as defined in Section 1.1.7: it does not intervene when the controller has proposed a truly safe action.

2.2.3 Over-filtering Controller

As mentioned above, 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, we modified the controller so that it would erroneously filter out some sparse but significant obstacles, such as sparse fallen tree branches (simulated, for our robot car, with plastic cables in Fig 2-7c). As expected, since the

controller omits points on the obstacle from its certificate, the monitor's horizontal density check fails, as there is a gap of size greater than `max_horizontal_separation`. The monitor rejects the certificate, and the unsafe action is prevented. Fig 2-7d shows the certificate points in red to indicate the monitor's rejection.

This experiment provides evidence that the certified control implementation upholds *soundness*, as defined in Section 1.1.7: it does intervene when the controller proposes an unsafe action.

2.3 Evaluation

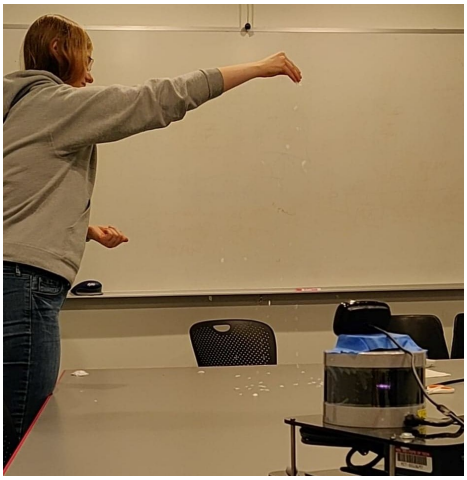
2.3.1 Experiments

The two snow experiments above show that our monitor is agnostic to the complexity of selecting points that meet the density criteria but do not represent reflections off snow particles. This is exactly the benefit certified control brings, since errors in the algorithm that selects the points are immaterial so long as the final point selection comprises adequate evidence. As demonstrated by the second experiment, errors related to overzealous filtering are caught by the monitor.

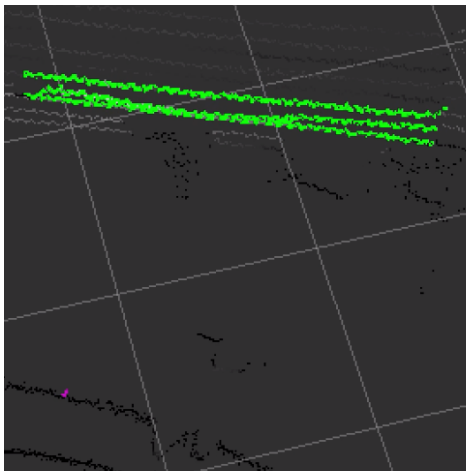
In addition, as noted, these experiments provide two data points which lie on either side of the system's honesty/soundness boundary, indicating that both reasonable soundness and reasonable honesty are achieved, despite the apparent difficulty of doing so.

2.3.2 Minimum Certificate Algorithm

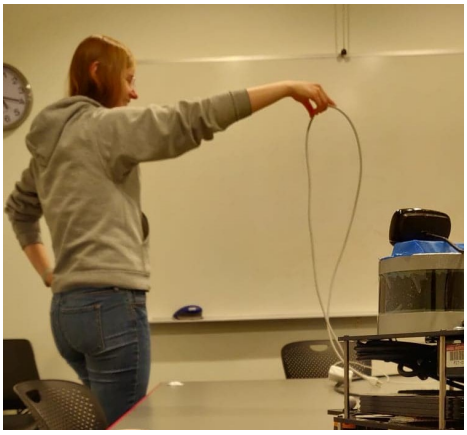
We ran the monitor with two versions of the controller—with and without the described algorithm for minimizing the certificate—on one timestep of saved LiDAR sensor output. The timing differences between these are shown in Fig. 2-8. With the algorithm, the controller did take longer to run, as expected, but the monitor took wildly less time, due to the reduction of the number of points in the certificate from almost the entire original LiDAR scan's 25,000+, to about 70. Since we expect pro-



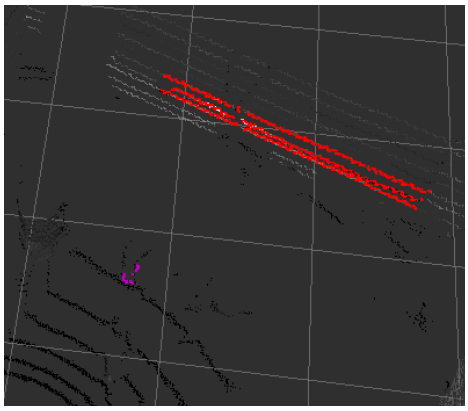
(a) Simulating snow with paper confetti



(b) Certificate points (green) with excluded snow points (purple, bottom left)



(c) Simulating thin obstacles



(d) Certificate points (red) with excluded obstacle points (purple)

Figure 2-7: Experiments to test LiDAR certificates against simulated snow (top) and thin obstacles (below) that may be mistakenly identified as snow by the main controller

	Time w/o Algorithm (ms)	Time w/ Algorithm (ms)
Controller	6.969	9.646
Monitor	3.264	0.586
Combined	10.233	10.232

Figure 2-8: Comparison of runtimes with and without minimum certificate algorithm. All times measured on a 2017 Macbook Air with a 1.8 GHz Intel Core i5 Processor and 8 GB RAM.

duction controllers to have high-performance hardware in order to perform complex processing, a negligible increase in the controller's runtime is not concerning, whereas the significant drop in the monitor runtime is a significant win for keeping the monitor simple. And, as expected, the monitor's decision about the safety of the controller's action was the same for both runs: the reduction of the certificate size did not affect its accuracy in proving (un)safety of the action.

These results embody the principle of certified control: by tasking the controller with making a minimum certificate, which it can do with complex and difficult-to-verify algorithms, we ensure that the monitor remains fast, simple, and verifiable.

The controller with the minimum certificate algorithm produces a certificate with size of only 3 kilobytes, on average (without compression). When the controller and monitor components of the architecture are separated, we can therefore expect transmission of the certificate between them to be trivially fast.

2.3.3 Complexity Gap

The behavior in all experiments was achieved with a system that demonstrates the expected complexity gap between the controller and monitor. Even with a controller performing only basic snow filtering, without machine learning algorithms, the controller implementation requires a factor of almost 30 times more lines of code than the monitor (Fig. 2-9).

Component	Lines of Code (Python2 or Pyrex)
LiDAR monitor Total	**82**
Controller K-d tree (scipy.spatial)	739
Controller Numpy calls	1325
Controller self-written driver code	167
LiDAR Controller Total	**2231**

Figure 2-9: Comparison of code sizes for monitor and controller.

2.3.4 Speed

As noted in Fig. 2-8, the final versions of the controller and monitor take approximately 9.646ms and 0.586ms to run, respectively. This gap would only grow in a production system, as the complexity of the monitor would not need to increase, but the controller would need much more sophisticated processing to handle perception and control.

The Velodyne sensor produces a scan approximately every 50 ms (20Hz). Therefore the current system's speed is entirely sufficient to allow for an monitor check on every LiDAR scan. We expect that even further reduction in the speed would be feasible, if needed, through more extensive use of NumPy processing tools, which performs array operations in C instead of Python.

2.4 Limitations

Implementation and evaluation of this certified control system helped us identify limitations, some of which are particular to this implementation and could be improved in future work, and some of which are inherent to the use of LiDAR scanners like ours. We describe these here.

2.4.1 Blindness of LiDAR

On one hand, LiDAR data is compellingly physical. A single LiDAR point gives evidence of the lack of obstacles on a line from the sensor to that point. This is a

strong physical interpretation which other sensors, like cameras (which we explore in Chapter 4), don't afford, and is part of what makes LiDAR such an essential sensor for AV systems.

However, a LiDAR certificate can provide information *only* about the shape and size of obstacles[8]. But, of course, objects with the same shape and size may present varying degrees of danger. A harmless paper bag, for example, could occlude the same region of points in a LiDAR certificate as a falling rock. Our LiDAR monitor cannot distinguish between certificates with these same-sized holes–it is in this sense "blind" to certain characteristics. Similarly, a shower of small stones may be indistinguishable from snow.

2.4.2 Lack of Time Domain Awareness

The above limitation may be partially overcome if the monitor were able to observe the size and shape of objects as they evolve over time. That is, while a paper bag may occlude the same region as a rock during one particular time step of the LiDAR scan, a monitor with information about multiple LiDAR scans may be able to deduce that the fast-falling and consistently-shaped object is likely a hard obstacle, whereas the slowly fluttering and almost-transient obstacle is less dangerous.

However, in an effort to remain simple and verifiable, the monitor does not store any information from past certificates, and thus cannot make observations about the evolution of certificates. We may in the future add time awareness to the monitor with as little added complexity as possible. Or, since a production controller would have awareness of the time domain in order to do its other processing anyway, we could task the controller with providing an augmented certificate if its chosen action is dependent on data from multiple time steps. In this way we would leverage the certified control approach again.

[8]LiDAR sensors usually also provide reflectivity data; however, this information is not leveraged in our system and is not thought to be critical to obstacle detection.

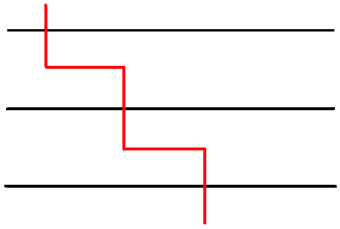


Figure 2-10: Rough sketch of the most difficult-to-detect obstacle shape. View is the projection of LiDAR scans and obstacle onto a section of plane on the cross-section of a lane. Black horizontal lines indicate the level at which the LiDAR scans hit. The red stair-step shape comprises a large continuous obstacle which nonetheless is barely detected by the LiDAR unit.

2.4.3 Data Non-uniformity

Another limitation of the system is the non-uniformity of the LiDAR data itself, mentioned before and illustrated by Figure 2-2. The relative lack of vertical density of points means that the system (and indeed, any AV system based on the same LiDAR sensor) is not well-equipped to detect thin horizontal obstacles, like parking arms. Such obstacles, especially at a great distance, may fall between horizontal scans, and therefore appear totally hidden to the LiDAR unit. The scan non-uniformity also makes it easier for thin vertical objects, like metal poles, to be overlooked, as they appear as narrow groups of relatively disjoint LiDAR points, one group on each LiDAR scan row.

We anticipate that the most difficult obstacle to detect given the form of the data would be a stair-step obstacle aligned perfectly out-of-step with the LiDAR scan rows, sketched in Fig. 2-10. Improvements to LiDAR scanners which allow them to house more row-scanning units within a small vertical hardware space, or which allow for varied angling of the row-scanners on different time steps, would mitigate these problems. Barring those unanticipated hardware improvements, though, these limitations will be inherent to all LiDAR-based AV systems.

Chapter 3

LiDAR Sensor Signatures

This chapter details work in relation to the monitor's *authenticity* check from Section 2.1.1. To ensure that a malicious controller cannot forge a signature, LiDAR points contained in the certificate must be securely signed by the sensor unit.

The implementations in this chapter are done on a simulated system, rather than on the physical robot car with implementations as described in Chapter 2, motivating the separation of its discussion into a different chapter.

We will give some context for this work, briefly introduce four signature schemes, then evaluate them with a focus on performance. Code for the simulation and for each of the schemes can be found at `https://github.com/capoling/857-lidar-security`. Work in this chapter was done in collaboration with MIT students Giulio Capolino, Cynthia Liu, and Carolina Ortega Perez.

3.1 Assumptions

The threat model against which these LiDAR signature schemes are meant to protect is one in which LiDAR points emanate from a trusted source (the sensor), travel through an untrusted channel (the controller), and arrive at the monitor. It is assumed that the controller may be connected to the internet, or is otherwise susceptible to malicious interference. It is assumed that the controller has high computational abilities (but not, of course, the ability to break computationally intractable problems

like the discrete log problem, on which standard cryptographic protocols rely[1]).

It is assumed that secure bridge networks [4] are the mode of communication between components within the system, and that these bridge networks only exist between components expected and allowed to communicate, such that outsiders cannot send messages to components without infiltrating a particular component. That is, the monitor is not itself internet-connected, or otherwise able to receive any messages except for those from the controller. In this way, we mitigate the risk of denial of service attacks in which a non-adversarial controller's communication with the monitor can be blocked by a barrage of outsider messages.

It is assumed that the sensor cannot receive any incoming messages except from the controller. This eliminates some risk to the sensor, but also introduces some complexity: specifically, for our simple hashing scheme (Section 3.3), the sensor and the monitor require a shared secret. Without two-way communication between these components, this shared secret must be achieved via hardcoding into the hardware (done at vehicle assembly time)[2]. If two-way communication between the components were available, however, a secure key exchange protocol, such as Diffie-Hellman, could be leveraged. This would be more fault tolerant, since it would enable easy regeneration of the secret key if a breach were detected.

It is assumed that the monitor and the sensor unit have synchronized internal clocks (or even a shared clock). In each of the signature schemes, a supposed signature includes a plaintext timestamp associated with the point. The monitor should, in addition to the cryptographic checks required for the scheme, confirm that the timestamp is sufficiently close to its own internal clock time. Times which are too far from the current time should be considered stale and rejected. This prevents replay attacks (in which old properly-signed LiDAR points can be sent again at inappropriate times, to construct an out-of-date picture of the landscape). The required closeness of time should be determined based on the speed of transmission from the sensor to the monitor, but we imagine it can be relatively conservative, since successful replay attacks

[1]That is, the threat model does not encompass a controller with quantum computing abilities!

[2]There is precedent for such a scheme: it is analogous to the hardcoding of trusted root certificate authorities on computers at manufacturing time.

require the ability to reuse relatively old data in order to fool the controller—one's environment while driving does not generally change very quickly. For greater protection against replay attacks, however, we recommend the use of Stratum-1 clocks (highly precise clocks often used in AV systems, [44]) which provide guarantees on the delay between synchronized systems.

Our threat model does not include physical attacks which may affect the sensor readings. For example, the signatures do not protect against an adversary placing reflective surfaces on the road, against which LiDAR rays may bounce erratically.

The schemes are meant to achieve verifiable authenticity of the LiDAR points. We do not require confidentiality, or any other security property, of the points (although they are achieved by most of the schemes anyway).

3.2 Simulation Setup

We created two simulators for the LiDAR sensor, each mimicking the behaviour of our Velodyne Puck VLP16 unit. One generates pseudo-random points, within some preset ranges, and yields groups of them at specified time intervals. The other draws from real stored data recorded using the robot car. The real data was taken over a few seconds when the car was in motion. The simulator "replays" the same sensor outputs that the real setup produced.

Our controller simulator simply chooses a random fixed-sized subset of the generated LiDAR points to put in the certificate. The simulated monitor accepts the certificate only if all the points therein are authentic[3].

3.3 Signature Schemes

3.3.1 Simple Hashing

The first scheme is a simple hashing scheme in which the sensor and monitor share a hardcoded secret, s. The signature of a LiDAR point p at a timestamp t is a triple

[3]In particular, it does not perform the other checks from Section 2.1.1

of the time, the point itself, and a hash of the concatenation of the point, time, and secret key[4,5]:

$$Sig(p,t) = t, p, H(p||t||s)$$

To verify a supposed signature $(t, p, hash)$, the monitor itself computes the hash of the concatenation of the pieces with its secret key. If the hash matches that in the triple, it accepts the signed point and time; otherwise, it rejects.

3.3.2 Hashed RSA

We also implemented a signature scheme leveraging the RSA public encryption protocol [35]. Rather than implementing the "textbook" RSA protocol, we leveraged the much faster "hash-and-sign" version, in which the message is hashed before encryption.

More details are in the source, [35], but we briefly recount the scheme as it applies to our context. The sensor randomly generates a public and secret key, both tuples (where n is a large prime):

$$\text{public key} \leftarrow (n, e)$$
$$\text{secret key} \leftarrow (n, d)$$

The sensor uses its secret key to encrypt the message, which is the hash[6] of the concatenation of a lidar point p with the timestamp t, into a ciphertext c. The signature is the triple as below:

$$m \leftarrow H(p||t)$$
$$c \leftarrow (m^d \mod n)$$
$$Sig(p,t) = (t, p, c)$$

To validate a triple (t, p, c), the monitor uses the public key for decryption:

$$d \leftarrow s^e \mod n$$

It accepts if d matches $H(p||t)$ (which it also computes).

[4]Our implementation uses SHA256, but any cryptographically secure hash function is acceptable.
[5]|| here represents concatenation.
[6]for this scheme, we again used SHA256

Scheme	Time to sign 100 points (ms)	Time to sign and verify 100 points (ms)
Simple Hashing	0.648	1.360
Hashed RSA	501.819	506.064
Ed25519	5.590	18.107
DSS	296.887	805.532

Figure 3-1: Comparison of signature scheme runtimes. Each LiDAR point was represented as a triple of three coordinates, each with 17 significant figures. All times measured on a 2017 Macbook Air with a 1.8 GHz Intel Core i5 Processor and 8 GB RAM. Times do not include simulation time required to generate or load in the points.

3.3.3 Ed25519

Next we implemented Ed25519, also a public-key signature system. Ed25519 is a type of Edwards-curve Digital Signature Algorithm (EdDSA), which is a digital signature scheme using a variant of Schnorr signatures using twisted Edward (elliptic) curves [3]. It is designed for speed. Its implementation is much more technical, so is not described here. Just as before, the message to be signed is taken as the LiDAR point, concatenated with the timestamp.

3.3.4 Digital Signature Standard (DSS)

Finally, we implemented DSS and its associated Digital Signature Algorithm (DSA), a Federal Information Processing Standard[7] for digital signatures. The scheme is a variant on the widely-used Schnorr signatures. Like all aforementioned schemes, we sign a timestamp concatenated with the lidar point.

3.4 Evaluation

An average certificate contains 70 points (as stated in Section 2.3.2). We therefore conservatively tested each signature scheme on the signing and verifying of 100 points (although the times would scale approximately linearly for other numbers of points).

[7]published by the National Institute of Standards and Technology

Figure 3-1 shows the resulting times[8]. Since the Velodyne LiDAR sensor on our physical system produces a scan every 50ms (20Hz), only Ed25519 and the simple hashing schemes were fast enough to comfortably handle the AV performance needs.

In addition to performance, we also desire that the secret keys need not be impractically long in order to make it computationally difficult for an adversary to find. This is certainly achieved. Imagine a malicious controller in the simple hashing scheme. The controller may query the monitor with triples $(t, p, H(p||t||x))$, where x is the controller's guess at the secret key, for multiple values of x. If the secret key is 120 bits, as it is in our implementation, the controller at worst needs to query for all 2^{120} possible keys. With each key taking a few milliseconds, that's 2^{120} queries * 3ms $\approx 10^{35}$ queries * 10^{-13} centuries = 10^{22} centuries. And that is not even to mention the fact that the controller would only be able to make such queries while the vehicle is in operation.

This evaluation shows that multiple signature schemes, including Ed25519 and even a simple hashing scheme, are sufficient to satisfy the security needs of the LiDAR certified control system.

[8]Although irrelevant to the analysis here, it is interesting to note that, while all schemes in the figure take longer to sign *and* verify than just to sign, the discrepancy between those times varies widely. This is reflective of the varying degrees of computational setup required to verify.

Chapter 4

Certified Control for Vision

Here we describe the augmentation of an existing vision certified control system with LiDAR data, and show the resulting improvement in verification of lane line detection.

4.1 Background

A team member designed a vision certified control system, whose certificate includes the signed image frame from which the lane lines were deduced and the position of the left and right lane boundaries as second-degree polynomials from a bird's-eye/top-down view.

The monitor verifies this certificate with two checks. The first check is *geometric*. In this test, the proposed lane lines are evaluated as a pair; if they conform to specific geometric bounds (such as parallelism, and spacing according to local regulations) then the lane lines pass this test. The second check is for *conformance* to the image. Computer vision techniques are used to determine whether the proposed lane lines correspond to lane line markings in the image (more details in [8]). If they do, the lane lines pass this test. The proposed lane lines must pass both checks in order for the certificate to be accepted.

The team member implemented this vision certificate scheme and evaluated it against the *openpilot* [9] lane-detection software system, using real replay data from

Figure 4-1: An example of a lane-detection failure detected by the monitor. The green lines represent the proposed lane lines.

Figure 4-2: Physical setup for testing visual lane-following system. Blue lines of tape on the ground are detected by the simple visual system as lane lines.

a driven car. In several instances, the monitor caught lane detection failures.[1] An example of such a detected failure is seen in Fig. 4-1.

The two-check system was also evaluated with the physical racecar. The controller used a naive lane-detection algorithm, and colored tape was placed on the floor to simulate lane lines, as in Figure 4-2. Placing additional tape segments in inappropriate positions on the ground caused the controller's lane detector to report bad lane lines; in all cases, the monitor correctly rejected them.

Despite these successes, the described two-check monitor was still unable to detect some types of perception failures. 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, the two checks cannot determine if a

[1] In all failure cases, though, *openpilot* was able to successfully correct its lane detection within a few seconds.

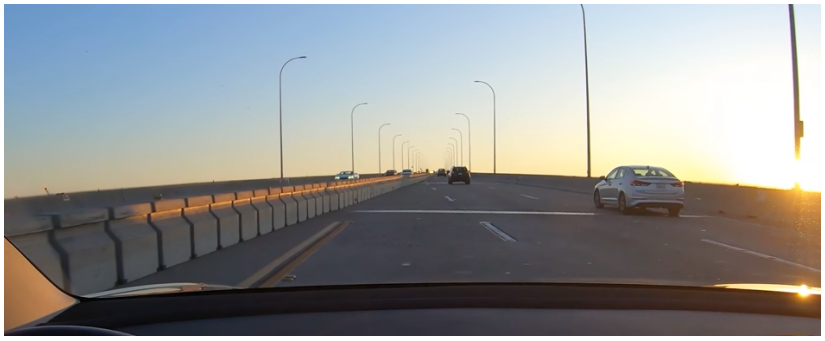


Figure 4-3: An image taken from dashcam footage in which a car using Tesla's autopilot almost ran into a concrete barrier. The malfunction was presumably caused by the perception system mistaking the reflection along the barrier for a lane line.

lane line is not physically situated on the road. That is, if the controller determines that there are geometrically plausible and image-conforming lane lines on top of barriers, say, then the two-check monitor would not be able to catch the mistake.

This scenario is not hypothetical. Erroneous behavior has caused some cars (notably those produced by Tesla, which relies heavily on vision) to swerve toward concrete barriers. In these cases, the problem is often that bright lines, that seem to be lane lines, appear in the camera's image [39]. These lines are actually bands of direct or reflected sunlight, like seen in Fig. 4-3. While these misperceived lane lines may have passed the geometry and conformance tests, a basic property is being violated: the detected lane line is not on the ground. This motivates a new type of certificate, which we now introduce.

4.2 Combined Vision and LiDAR Certificate

In addition to providing the lane line polynomials and the camera image (as before), the new combined certificate also provides the following:

1. the set of LiDAR points that lie on the purported lane lines,

2. a representation of the ground plane, and

3. proof of the legitimacy of that ground plane, in the form of a "ground plane

subcertificate."[2]

The first two components are straightforward. We now describe the motivation and contents of the third component.

4.2.1 Ground Plane Subcertificate

Motivation

Including ground plane detection in the monitor itself is not possible for two reasons: the algorithm is too complex to be easily verifiable (and the monitor should remain plausibly verifiable), and the monitor only has access to the subset of points given to it in the certificate, not the entire LiDAR point cloud (and accurate ground plane detection depends on the presence of most or all points).

Therefore we task the controller with the ground plane detection, and have it, in typical certified control fashion, send a subcertificate with evidence of its proper conclusion, to the monitor for checking. In this way, the monitor can verify the accuracy of the computed ground plane without engaging in complex computation itself.

Contents

The ground plane subcertificate includes (1) some LiDAR points which were collected from the lowest-angled row of the LiDAR unit (the number of which is a parameter), and (2) a dense specified-size patch of points which lie on the ground plane. The patch of points may be anywhere in the space[3], and represent a planar section of the road ahead.

Together, these two components are good identifiers of the plane: the certificate's low-lying points form a line on the ground plane (where that LiDAR scan row hit the

[2]The subcertificate is a certificate of its own right, in the certified control sense, as it gives the monitor proof of accuracy of its ground plane detection algorithm. However, we term it a *sub*certificate to indicate that it is only a part of the vision system's entire certificate.

[3]even near the lowest row, since the required patch size is big enough that it must span multiple rows anyway

ground), and the patch of points defines a 2D region of points on the plane elsewhere. Mathematically, these two pieces suffice to uniquely define a plane.

A controller which successfully detects the ground plane will be able to compute such a certificate: low-lying points are expected to hit the ground plane, and, almost by definition, there must be patches of points that lie on the ground plane elsewhere in the point cloud. On the other hand, it is difficult for a malicious (or faulty) controller to generate a valid certificate for a false ground plane. The false plane would need to contain most of the LiDAR points collected from the low-lying row. This already restricts the possible certifiable planes to those which intersect that low-row line segment. A malicious controller would then have to find a dense, planar patch of points which fall on one of those planes. While this is possible (say, if a truck in front of the car is hauling a billboard slanted at an angle coincident with the low-row points), it is unlikely to happen, especially for more than a brief period. And, in a production implementation, a monitor would not accept a certificate for a ground plane which is radically different from the previously accepted ground planes, such as one for which a malicious controller was only suddenly and briefly able to generate a certificate.

4.3 Controller

We describe how the controller generates the three components of the augmented certificate from Section 4.2.

4.3.1 Lane Line LiDAR Points

First, the controller is tasked with selecting the LiDAR points on the lane lines.

Doing so means, more generally, finding LiDAR points that correspond to some pixels on the camera's image. Transforming between the camera and the LiDAR space, given knowledge of the camera and LiDAR specifications and their mounting locations on the vehicle, is a matter of trigonometric transformations. (In our implementation, we make the assumption that the camera and LiDAR are mounted at the

same location—a physical impossibility, but one which is likely a fair approximation.) Specifically, if ones knows the vertical and horizontal angles spanned by the camera, then, given an (x, y) pixel on an image of known pixel dimensions, one can compute the angle at which that pixel was located, from the perspective of the camera. This angle would also be, given our assumption, the angle of the LiDAR ray which would hit the corresponding point in space. Knowing the number of LiDAR scan rows, the angle that those rows span, and the number of points within each row, one can identify the closest LiDAR ray corresponding to the image pixel.

In reality, the above approach is somewhat flawed. Due to the nonuniformity of the LiDAR data, the LiDAR angle associated with a particular image pixel is likely to fall in the empty space between the LiDAR row scans. Therefore the closest LiDAR point may still be relatively far away, so this method could yield locations that are quite distorted. This is especially problematic since scenarios in which we care about the LiDAR location of lane lines are ones in which precision is key: when, for example, a barrier is just along the edge of a highway's lane lines.

Therefore, our controller in fact does some preprocessing (of the trigonometric sort described above) in order to determine the pixel heights in the image at which the LiDAR rows fall. Then, given proposed lane lines in the image, it finds all intersections of those lines with the LiDAR rows. It is then *these* pixels, which are guaranteed to fall on LiDAR row scans, which are transformed into the corresponding LiDAR points. These LiDAR points form part 1 of the certificate from Section 4.2.

4.3.2 Ground Plane Detection

Next, in order to detect the ground plane from the LiDAR point cloud, our controller runs a random sample consensus (RANSAC) algorithm [13], augmented with some constraints on necessary ground plane features. We parameterize the RANSAC algorithm with the number of trials to run (where each trial is a random selection of three points, and a consideration of the resulting plane), and the closeness required for a

point to be considered on the plane[4]. Our implementation uses a required closeness of 0.1 meters, and runs 20 "viable" RANSAC trials. We define a "viable" trial to be one in which the candidate plane satisfies the aforementioned, and now described, ground plane features.

The features are twofold: the plane must be sufficiently *low-lying* and sufficiently *horizontal* (both with respect to the coordinate system of the LiDAR unit, which is assumed to be mounted flush against the vehicle's top[5]). For example, one very conservative choice of parameters may require that the plane have tilt less than 14 degrees (24% grade) from a perfectly horizontal plane[6], and may require the plane to have height which is below that of the LiDAR unit. While the "height" of an infinite plane is not generally defined, we use the rough proxy of the height of the highest of the three points chosen to represent the plane. The inaccuracy of this proxy is mitigated by the previous requirement on the plane's tilt.

Results from running RANSAC (with less conservative parameter choices) are shown in Figure 4-4. Notice that there are more prominent planes in those point clouds than the ground plane—since the data was collected inside a building, many points fall on the walls—but the ground plane is nonetheless detected due to the ground plane feature checks.

The ground plane found using these methods forms part 2 of the certificate from Section 4.2.

[4]This second parameter is needed because RANSAC, for each candidate plane, determines how many points in the LiDAR point cloud lie "on" the plane. This number of points is the metric for the quality of the candidate plane.

[5]This is typical in AV systems, as it allows the scanner's range to reach the largest possible region of interest.

[6]In the terms of our implementation, this would correspond to a check that the unit normal to the candidate plane has a projection onto the horizontal plane which has magnitude less than 0.25 units. Or, to avoid computing the *unit* normal, this can equivalently be thought of as requiring that, when one projects the candidate plane's normal vector onto the horizontal plane, the projection has magnitude less than one fourth of the normal's magnitude.

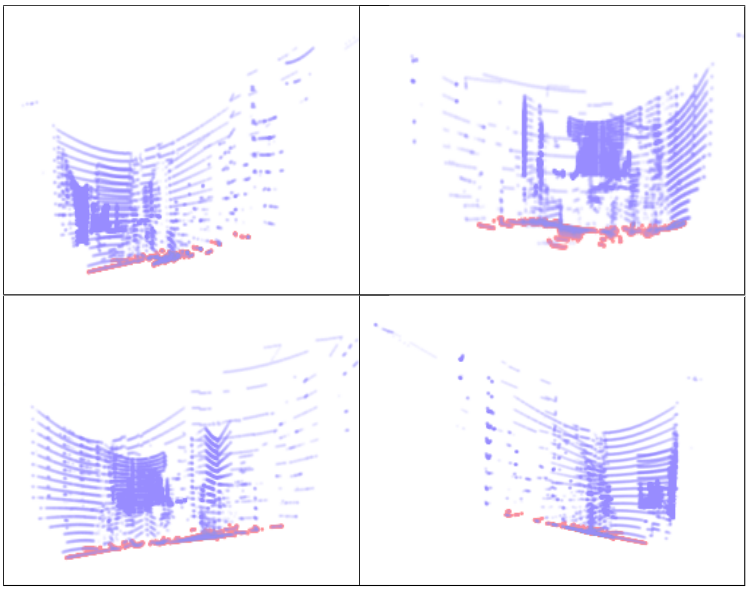


Figure 4-4: Results of four runs of the ground plane detection algorithm on the same LiDAR scan data, taken inside a building. Blue points are all those in the LiDAR point cloud. Points which lie close to the detected ground plane are highlighted red. Each point cloud is shown in a different orientation. The detected plane may also be slightly different in each, due to the nondeterminism of RANSAC. Note that these results are especially accurate since we tuned the parameters (e.g. expected height of the ground plane) based on an understanding of the robot car's location in the environment (in particular, its approximate height off the ground).

4.3.3 Subcertificate Generation

Finally, after detecting the ground plane, the controller must find a dense region of LiDAR points which fall on the ground plane. The controller attempts to find the densest such region. We term this densest region the "patch." The monitor expects a certain-sized patch[7], and this size is known to the controller.

To find the patch, the controller first processes the points that lie on the ground plane into a k-d tree (like for the snow filtering in Section 2.1.2). Then, it considers candidate "patch centers:" points which might be the center of the patch. It considers all possible patch centers in a discretization of a search region. The search region is the part of the ground plane which falls within some hardcoded x and y ranges[8] (i.e. it is some rectangular piece of the plane). For each candidate patch center, the controller queries the k-d tree for the number of points in a radius around the candidate center. The radius is based on the size of the patch which the monitor expects. The candidate patch center which has the highest number of points near it is deemed the center of the ideal patch. The points in that patch, along with the lowest-angled LiDAR row scan points which lie on the ground plane, comprise the ground plane subcertificate, part 3 of the certificate from Section 4.2.

4.4 Monitor

In addition to the two purely-vision checks described in Section 4.1, the monitor checks these conditions when given the LiDAR-augmented certificate:

1. The lane line LiDAR points indeed correspond to the lane lines.

2. Those lane line points lie on the given ground plane.

3. The ground plane is legitimate (as determined by assessment of the ground plane subcertificate):

[7](our testing used about a 0.5 meter squared region)

[8]We found that these ranges can be large—as large as the LiDAR unit's reading range, even—without significant loss of performance.

(a) The ground plane is sufficiently ground-like (horizontal and low-lying).

(b) The low-lying points given in the ground plane subcertificate are truly from the lowest scan row of the LiDAR unit, and also lie on the ground plane.

(c) The points in the patch lie on the ground plane, and are sufficiently dense. For this we use the same horizontal density check as used for the LiDAR certificate, described in Section 2.1.1.

4.5 Experiment

We used the same robot car setup as in the LiDAR certificate experiments (Section 2.2.1) to evaluate the vision certified control system. To simulate the Tesla barrier malfunction, we lined up two rows of tape to look like lane lines. 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 (Fig. 4-5). 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[9]. Fig. 4-6 shows a visualization of the LiDAR points.

4.6 Evaluation

Evaluation of the original two-check vision certified control system was described in Section 4.1 and can be found in more detail in [8]. Here we only evaluate the augmented system, including the ground plane subcertificate.

The experiment showed that the augmented system is sufficient to prevent AV accidents like the Tesla one, in which the controller falsely identifies lane lines on above-ground objects.

[9]While running this experiment, we had not implemented the ground plane subcertificate component of the vision certificate, so we instead considered the ground plane detection algorithm to be part of the trusted base. Independent simulation-based testing of the ground plane subcertificate is described in Section 4.6.

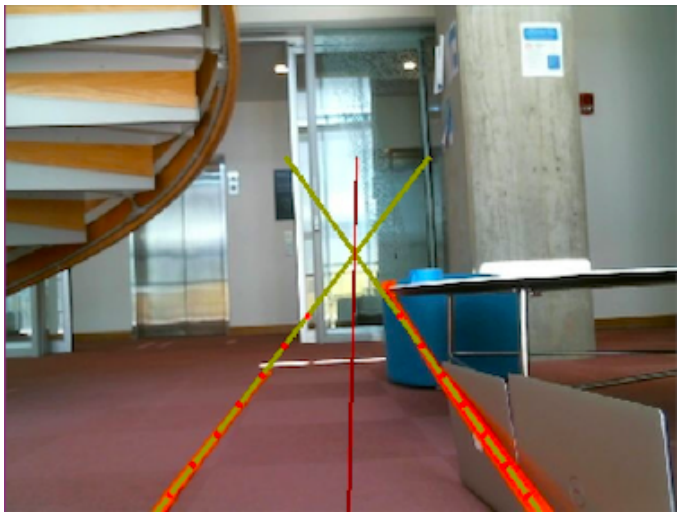


Figure 4-5: Robot car perspective on the lane-line setup. Yellow and red dotted line segments indicate the lane lines indicated by the car's visual detector. They match (and occlude) the physical tape barriers. The red solid line is the middle line of the two lane lines. The right tape line is attached to the table and laptop backs to elevate it.

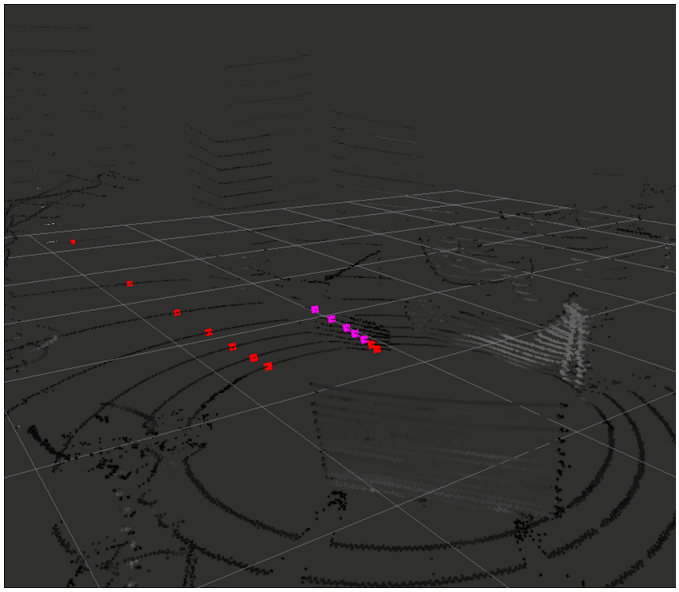


Figure 4-6: LiDAR point cloud visualization. Light gray grid is a remnant of the visualizer; gray and black points are LiDAR points (often ring-like due to the separate LiDAR row scanners). LiDAR points corresponding to the identified lane lines are in red or purple: red for those identified to be on the ground plane; purple for those not on the ground plane.

In addition to the experiment, we performed some basic testing of the ground plane subcertificate. We ran the system on one scan of LiDAR data and first confirmed that a working controller, implemented as described above, produces a ground plane subcertificate which the monitor accepts. We also confirmed that the monitor does *not* accept an invalid subcertificate. We tested two invalid subcertificates: one whose low-lying LiDAR points are not on the ground plane, and one which does not contain the required number of low-lying LiDAR points. Checking such an invalid certificate took on the order of 10^{-6} seconds for the monitor, and checking a valid certificate took on the order of 10^{-3} seconds, on average. On the other hand, the controller's generation of the certificates took upwards of 20 seconds in some cases[10].

4.7 Limitations

The integration of vision with LiDAR presents a few unique challenges. First, the ground plane detector we implemented is not robust to hilly (non-planar) roads or to uneven road surfaces. Indeed, this is a severe limitation, since the main benefit of a ground plane detector is that it could allow for the detection of non-planar grounds. Otherwise, it is unclear what value the detector has beyond a hard-coded ground plane based on the mounting height of the LiDAR unit and its current orientation. Improving this algorithm is therefore a high priority for future work.

Even for planar ground plane detection, though, RANSAC is a fairly limited algorithm. It is nondeterministic, and may require a large number of trials in order to guarantee with high probability a good ground plane. As we noted, this resulted in some runs taking over 20 seconds. This is an unacceptable delay for the application to AVs (although it is perhaps mitigated by the fact that the controller would not need to entirely recompute the ground plane on every timestep). An improvement on RANSAC would be an adaptive algorithm, which does not choose a completely random candidate plane on each trial, but rather uses information about quality of the previous candidate planes in order to choose a better next candidate.

[10]The time varies quite widely due to the nondeterministic nature of RANSAC.

4.7.1 Ground Plane Subcertificate

There are a number of limitations of the generation and checking of the ground plane subcertificate as well. Some of these take the form of simplifying assumptions which were made for ease of implementation, but which sacrifice accuracy. For example, the candidate patch centers considered by the controller all lie on the *horizontal* plane whose height is the average height of the points on the ground plane. That is, the candidate centers are not projected onto the ground plane. Given our assumption (and enforced requirement) that the ground plane is not very tilted, this may not affect patch detection hugely, although it prevents the easy extension of the ground plane detector to especially non-planar grounds. And, on the monitor's side, the patch density check only includes the horizontal density check, not yet the vertical density check included in the LiDAR certificate.

Other improvements take the form of natural next steps. One such step would be to reduce the size of the ground plane subcertificate by applying the minimizing certificate algorithm from Section 2.1.2 to the patch points. Additionally, as this is the newest addition to our certified control system, more general testing is needed, including testing that the monitor rejects subcertificates which have insufficient patches.

Chapter 5

Conclusions and Future Work

In this work we detailed two novel applications of certified control: the system for AV LiDAR-based obstacle detection, and the augmentation of a visual lane following system with ground plane certificates.

Specifically, we showed that the LiDAR certificate would override a controller which made improper decisions based on such mistakes as improperly identifying visual barrier lines as lane lines, falsely filtering non-snow obstacles away as snow, or otherwise overlooking obstacles of nonnegligible size. Not only did the architecture allow for avoidance of these common and possibly dangerous scenarios, it also contributed to the simplicity, modularity, and interpretability of the system. Enumerating ahead of time the required contents of a certificate informed the implementation work, and in effect provided a clear specification for both the controller and the monitor.

Overall, the implemented systems have demonstrated reasonable levels of each of the three runtime monitor desiderata (Section 1.1.7): the system is honest, as it does not intervene when a controller performs proper perception, or when purported lane lines lie on the ground; it is sound, as it correctly intervenes when the monitor conditions are violated; and it is much more easily verifiable than the alternative safety controller, shown by the gap in complexity and in lines of code of even simple implementations of the controller and monitor components.

Given this positive evaluation, we conclude that certified control is a highly viable

safety architecture for AVs. We can therefore identify promising directions for future work (some of which other team members already plan to pursue).

Natural extensions to this work include integrating the signature schemes (Chapter 3) with the physical LiDAR unit and our actual controller, rather than just their simulations; integrating the current implementations into our team member's proposed Docker container isolation scheme; extending the system to handle controller actions beyond moving forward (and, more generally, expanding the environmental awareness of the entire system in this way); and applying the certified control architecture to other common AV sensors[1].

As mentioned before (Section 2.4.2), work to augment the monitor with time domain awareness (or, rather, to task the controller with providing relevant data from previous timesteps) is expected, and could improve both the soundness and honesty of the system. Also, while certificates are currently interpreted deterministically by the monitor, a probabilistic interpretation would be preferred.

To enable faster iteration on controller and monitor implementations[2], our team plans to transition our development to the CARLA simulator [7]. CARLA is a dramatic improvement on the simple simulators made for LiDAR signature testing (Section 3.2), since it supports very realistic simulation of sensors (including those other than LiDAR units), and allows for specification of environment conditions.

Finally, we expect that broader theoretical work in contextualizing certified control would be of value. The architecture likely has applications outside of AVs, in such diverse areas as airport security scanning, e-mail spam filtering, and medical scanning anomaly detection. Investigating existing systems in those domains (and determining whether expert designers of them already have a mechanism for runtime monitor-like checks) may illuminate the unique contribution of the certified control architecture, beyond the AV-related safety improvements we investigated here.

[1]The time-of-flight (ToF) camera, used for close-range collision detection like during vehicle parking, is of particular interest.

[2]and to replace the experimentation with the physical robot car, which may be indefinitely impossible due to the pandemic

Bibliography

[1] 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: IEEE, Jun. 2012, pp. 3573–3580.

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

[3] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, B. Yang, High-speed high-security signatures. Journal of Cryptographic Engineering 2 (2012), 77–89. 2020. `https://ed25519.cr.yp.to/ed25519-20110926.pdf`

[4] Bridge Networks. `https://docs.docker.com/network/bridge/` May, 2020.

[5] O. Cameron, The Driverless Readiness Score, `https://olivercameron.substack.com/p/the-driverless-rediness-score`.

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

[7] CARLA. Open-source Simulator for Autonomous Driving Research. `https://carla.org/`. 2020.

[8] J. Chow, V. Richmond, M. Wang, U. Guajardo, D. Jackson, N. Arechiga, G. Litt, S. Kong, S. Campos. Certified Control: A New Safety Architecture for Autonomous Vehicles. Submitted for publication, 2020.

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

[10] A. Corso, P. Du, K Driggs-Campbell, and M. J. Kochenderfer, Adpatinve Stress Testing with Reward Augmentation for Autonomous Vehicle Validations, in IEEE Intelligent Transportation Systems Conference, 2019.

[11] 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 systems architectures, in IEEE International Real-Time Systems Symposium, 2007.

[12] R. Ehlers, Formal Verification of Piece-Wise Linear Feed-Forward Neural Newtorks, arXiv:1705.01320 [cs], Aug. 2017.

[13] 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.

[14] C. Fishman. They Write the Right Stuff, `https://www.fastcompany.com/28121/they-write-right-stuff`.

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

[16] L. H. Gilpin, Reasonableness Monitors, AAAI, 2018.

[17] 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.

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

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

[20] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer, Reluples: An Efficient SMT Sovler for Verifying Deep Neural Networks, arXiv:1702.01135 [cs], May 2017.

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

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

[23] G. Klein et. al., sel.4: Formal Verification of an OS Kernel, in Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles - SOSP '09. Big Sky, Montana, USA: ACM Press, 2009, p. 207.

[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. Cham: Springer International Publishing, 2019, vol. 11698, pp. 245–253, Lecture Notes in Computer Science.

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

[26] 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. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, vol. 6664, pp. 42–56, series title: Lecture Notes in Computer Science.

[27] 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: IEEE, Oct. 2011, pp. 1181–1186.

[28] 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: IEEE, Nov. 2018, pp. 444-451.

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

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

[31] D. Phan 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, Jun. 2017.

[32] D. T. Phan, R. Grosu, N. Jansen, N. Paoletti, S. A. Smolka, and S. D. Stoller, Neural Simplex Architecture, arXive: 1908:00528 [cs, eess], Mar. 2020.

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

[34] X. Qin, N. Aréchiga, A. Best, and J. Deshmukh, Automatic Testing and Falsification with Dynamically Constrained Reinforcement Learning, arXiv:1910.136545 [cs, eess], Feb. 2020.

[35] R. Rivest, A. Shamir, A. Adleman. The Original RSA Patent, filed with the U.S. Patent Office. Dec. 1977.

[36] 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, no. 4 pp. 277–288, Nov. 1984.

[37] L. Sha, Using simplicity to control complexity, IEEE Software, vol. 18, no. 4 pp. 20–28, July 2001.

[38] S. Shalev-Shwartz, S. Shammah, and A. Shashua, On a Formal Modelof Safe and Scalable Self-driving Cars, arXiv:1708.06374 [cs, stat], Oct. 2018.

[39] Tesla Autopilot Drives Straight Towards Concrete Barrier on Highway. https://www.reddit.com/r/SelfDrivingCars/comments/du2bnz/tesla_autopilot_drives_straight_towards_concrete/.

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

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

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

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

[44] Wikipedia. Stratum Clocks. https://en.wikipedia.org/wiki/Network_Time_Protocol#Clock_strata May, 2020.