IMPLEMENTATION OF A SCALE SEMI-AUTONOMOUS PLATOON TO TEST CONTROL THEORY ATTACKS

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Erik Miller

July 2019

COMMITTEE MEMBERSHIP

TITLE:	Implementation of a Scale Semi-Autonomous Platoon to Test Control Theory Attacks
AUTHOR:	Erik Miller
DATE SUBMITTED:	July 2019
COMMITTEE CHAIR:	Bruce Debruhl, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	John Bellardo, Ph.D. Professor of Computer Science
COMMITTEE MEMBER:	John Seng, Ph.D. Professor of Computer Science

ABSTRACT

Implementation of a Scale Semi-Autonomous Platoon to Test Control Theory Attacks

Erik Miller

With all the advancements in autonomous and connected cars, there is a developing body of research around the security and robustness of driving automation systems. Attacks and mitigations for said attacks have been explored, but almost always solely in software simulations.

For this thesis, I led a team to build the foundation for an open source platoon of scale semi-autonomous vehicles. This work will enable future research into implementing theoretical attacks and mitigations. Our 1/10 scale car leverages an Nvidia Jetson, embedded microcontroller, and sensors. The Jetson manages the computer vision, networking, control logic, and overall system control; the embedded microcontroller directly controls the car. A lidar module is responsible for recording distance to the preceding car, and an inertial measurement unit records the velocity of the car itself. I wrote the software for the networking, interprocess, and serial communications, as well as the control logic and system control.

ACKNOWLEDGMENTS

Thanks to:

- Uncle Steve for his editing expertise
- Mom and Dad for their support, patience, and feedback
- My sister Annika for her editing assistance

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

Autonomous vehicles have captured the interest of researchers and businesses alike with the promise of safer, faster, and more efficient transportation. Numerous papers on the safety, security, and efficiency of such vehicles [13, 28, 35] have been written, and many businesses are looking to produce their own autonomous vehicles. Notable examples include Tesla's autopilot program [37] and the Waymo One autonomous taxi [43], which are some of the first commercially available self driving vehicles. Many other automakers either have research groups working on developing autonomous vehicles, such as GM [7], Mercedes-Benz [25], Toyota [39], Ford [22], and Baidu [2], or have announced partnerships to do so, such as Renault-Nissan and Microsoft [29], Volvo and Uber [42], Waymo and Chrysler [19], Waymo and Honda [15], and BMW, Intel, and MobileEye [4].

Despite this research and interest, as of 2018 [16] there are still no autonomous vehicles capable of handling every driving situation that may arise. While many people are working to solve the technical, legal, and ethical challenges of creating a fully autonomous vehicle, there are a number of more limited systems that have the potential to provide many of the same efficiency and accessibility benefits under more constrained circumstances. One current solution is a platoon of semi-autonomous vehicles that follow behind a human driven vehicle [41]. This is entirely feasible with current, widely implemented technologies such as Adaptive Cruise Control and Lane Keeping, and communications protocols such as Dedicated Short-Range Communications (DSRC) and 5G cellular.

Since this application involves humans and heavy, fast-moving machinery, there are many projects [8, 9, 11, 32, 34, 38] that explore the robustness and security of

these systems. However, most of this research is being performed in simulation [9, 11, 32, 34, 38], without any verification on a physical implementation. The one research group that did design their own scale platoon [8] does not appear to have released more than a high-level overview of its construction as of March 2019, and their platoon was fixed to a guide wire, circumventing the challenges posed by steering. I speculate that the lack of physical implementation is largely due to the lack of open source scale platoons. While simulation work is valuable to evaluate different options much more quickly, physical implementations may have factors that simulations don't fully account for, such as steering, friction, and real world communications.

In this paper, I describe the process of designing components for an open source, 1/10th scale platoon of semi-autonomous vehicles. In particular, we design control logic, interprocess communications, and vehicle-to-vehicle communication. The foundation we create will allow other researchers to build a scale, string-stable platoon of semi-autonomous vehicles for the purpose of researching attacks and mitigations on computer controlled vehicles.

Chapter 2

BACKGROUND AND RELATED WORKS

With the exploding popularity of autonomous vehicles, many different systems with varying capabilities have been called autonomous vehicles.

2.1 Levels of Autonomous Vehicles

In order to distinguish between the capabilities of autonomous vehicles, the Society of Automotive Engineers (SAE) created a scale with 6 levels of autonomy along with supporting vocabulary [31]. To paraphrase:

Driving Automation System A catch all term that the SAE uses to describe a system that automates some portion of the complex task of navigating and controlling a car, referred to as the Dynamic Driving Task (DDT). This could be as simple as a fixed speed cruise control or as complex as a fully autonomous car.

Automated Driving System (ADS) A system capable of handling the entire DDT without human input, though it may be constrained to specific Operational Design Domains (ODD), like highway or city driving, and it may request human intervention in cases it is not equipped to deal with. Thus, ADSes must be a level 3, 4, or 5 autonomous system.

Level 0 – No Driving automation The driver is responsible for all driving functions, though they may be assisted by active safety systems like lane and blind spot warnings, as well as advanced features like automatic braking systems.

Level 1 – Driver Assistance In this case, the driving automation system is responsible for controlling either the steering (lateral movement) **or** the acceleration and braking (longitudinal movement) of the vehicle. Such systems can be simple, like cruise control that holds a static speed, or sense the world around the car to handle more of the DDT, like Adaptive Cruise Control or Lane-Keeping. However, the driver still carries the burden of Object and Event Detection and Response (OEDR), meaning the human driver is responsible to avoid objects the system does not detect, such as animals, pedestrians, stopped vehicles, and debris, and they must react accordingly to events like rain, ice, and emergency services.

Level 2 – Partial Driving Automation The driving automation system controls both the lateral **and** longitudinal motion of the vehicle, but again leaves OEDR to the driver. This is the point where the average consumer might perceive that the car is "self driving", so it is imperative that the driver understand that they must prepared to take control of the vehicle back without notice or warning.

Level 3 – Conditional Driving Automation The system performs the DDT and OEDR in a sustained manner for a specific ODD with the expectation that the driver will take over if the system requests driver intervention or fails. Perhaps the most widely deployed implementation of such a system is Tesla's Autopilot [37], which supports freeway and limited city driving in mild weather.

Level 4 – High Driving Automation The system performs the DDT and OEDR for a specific ODD in a sustained manner, and is able to handle fallback or failures without the user responding to an intervention request. Compared to a level 5 autonomous system, a level 4 system may be limited to

certain areas, speeds, or weather. For example, a company may limit its cars to a specific geographical areas to prevent cars getting stuck or lost in areas where the car lack sufficient information to navigate successfully, or cars may be prevented from operating in conditions that significantly degrade the capabilities of their sensors, like snow or heavy rain. While such systems may be achievable with current technologies, we may see their deployment delayed for legislative or liability reasons [5].

Level 5 – Full Driving Automation The system performs the DDT and OEDR in any ODD in a sustained manner without any user intervention. This is unlikely to be achieved any time soon due to poor sensor performance in extreme weather like snow and heavy rain [16], as well the challenge of responding to pedestrians and other human controlled vehicles [16, 20].

2.2 Adaptive Cruise Control

With the advancements in developing higher-level ADSes, there have also been many improvements to level 0 and 1 driving automation systems, like Lane Keeping, Adaptive Cruise Control (ACC), and blind-spot warning systems. ACC is of particular interest due to its already widespread deployment and increasing availability in less expensive vehicles [40, 44]. Adaptive Cruise Control uses sensors to observe the vehicle in front of the ACC equipped vehicle and maintain a safe headway to the preceeding vehicle, or a set speed in the absence of one.

An ACC implementation can be as simple as a forward-facing lidar or radar [27] that is fed into a control loop to balance the desired speed with the desired headway. There has also been work with radar facing both forwards and backwards [21] to balance the distance between the preceding and following vehicle. The first situation is more common, likely because it requires fewer sensors, less

computation, and there is no real need to react to the following vehicle, as existing law places the responsibility to not collide on the following vehicle.

ACC coupled with a simple latitudinal control system like Lane Keeping or vehicle following creates a simple autonomous vehicle autonomous vehicle capable of staying in its lane and keeping a safe headway behind the vehicles in front of it.While that vehicle is only capable of operating in a narrow set of optimal conditions, by following a vehicle capable of performing the full DDT and OEDR it can piggyback off the preceding vehicle's decisions to handle situations that it is normally not equpped for. The preceding vehicle could be human-driven or a level 4 or 5 fully autonomous vehicle, and handle the important speed, safety, and navigation duties for the simpler vehicles, expanding the number of situations in which level 2 vehicles may be used.

2.3 Platooning Autonomous Vehicles

Additionally, a level 2 autonomous vehicle is able to follow another level 2 autonomous vehicle. Combined with the previous section, we can chain together a large number of level 2 autonomous vehicles following a better-equipped vehicle in what we will refer to as a platoon of semi-autonomous vehicles. A vehicle in a platoon typically falls into one of two roles: the leader, which sets the speed and direction of the platoon, and the followers, which are all governed by the same control algorithm to follow the preceding vehicle and maintain a safe but efficient spacing in the platoon.

It is important that as the platoon grows longer the control algorithm does not allow any oscillations or errors to amplify; this design criteria is defined as string stability [41].

Platooning autonomous vehicles together offers a number of benefits. The primary benefit is that it can reduce the complexity of the hardware needed for vehicle to drive under computer control. The leader of the platoon still needs to be able to perform the full DDT and OEDR, whether that is a level 4 or 5 autonomous system or a human driver, but the vehicles that follow the leader only need to be level 2 autonomous vehicles capable of following the vehicle in front of them. Autonomous platoons also can react much faster than a human driver is physically able to, so the vehicles can drive much closer together [28]. The decreased distance between vehicles can bring significant fuel savings by allowing the following vehicle to draft in the wake of the preceding vehicle [33], and the decreased distance increases the density of vehicles on the road, allowing more vehicle to use the same road in a given time [28].

2.4 Collaborative Vehicles

Another area of significant interest is the use of wireless signals to enable cooperation or collaboration between vehicles. The US Department of Transportation's AV 3.0 initiative [28] discusses a number of applications for Vehicle to Vehicle (V2V) and Vehicle to Environment/Infrastructure (V2X) communication, including "coordination of signalized intersection approach and departure...", "cooperative lane change[s] and merge[s]...", "speed harmonization ... to reduce bottleneck conditions", and "vehicle platooning to enable safe close following between vehicles and improve highway capacity".

The last point is of particular interest for this thesis. Connecting the members of a semi-autonomous platoon together allows the vehicles to provide feed-forward information to the other vehicles to let them react even faster than a non-collaborative platoon can solely with sensor-based feedback. Current

implementations with networked pairs of tractor-trailers found a combined 7% fuel savings and increased driver awareness and responsiveness [26].

2.5 Security Research

With the potential that autonomous vehicles and platoons offer, there are also risks that must be addressed. Due to the combination of heavy, expensive vehicles carrying humans traveling at high speeds and low following distances, autonomous platoons require the underlying systems to react before a human can perceive the change in conditions.

Most platoons are currently designed with the assumption that all parties involved will act in the best interest of the platoon, but that is not a safe assumption. Security researchers have introduced another role to the platooning model: the attacker, which also follows the leader, but varies their velocity and uses a different set of control parameters to disrupt the stability of the platoon. Through manipulation of parameters that control the platoon, an attacker can catastrophically destabilize the platoon, resulting in collisions [9]. With more carefully-chosen-values, an attacker can cause crashes that involve vehicle behind them in the platoon while the attacker's vehicle drives away unscathed [11, 12]. If platooning becomes commonplace and a large stream of vehicles on a highway are platooning together, researchers have proposed methods by which attackers can induce behavior that causes delay, discomfort, excess resource consumption, and an increased rate of accidents [8].

In response, researchers have suggested methods to mitigate such attacks, such as verifying the trustworthiness of participants in a platoon through both local observations [38] and collaboration with other vehicles [34], as well as falling back on safer control schemes when bad behavior is identified [32].

Most of the works referenced are conducted in simulation, as it is significantly faster and cheaper, and there are currently no inexpensive, open source platoon implementations on which to test. The one work that did implement a physical platoon [8] fixed their vehicles to a guide wire, simplifying their implementation at the cost of some realism. This thesis seeks to address that by providing a platform on which a semi-autonomous collaborative platoon can be built.

Chapter 3

PLATOON DESIGN

In this thesis, we design components that can be used for a human-led, string-stable platoon of 1/10th scale semi-autonomous vehicles in the future. A human-led semi-autonomous platoon lets security researchers focus on the level 2 autonomous vehicle implementation without needing to develop a level 4 or 5 ADS to lead the platoon. The platoon must be string-stable [41] to prevent the error between vehicles from growing as we increase the number of cars in the platoon, as that could result in instability or collisions. To acheive string stability, we designed a PID controller for the throttle based on previous platooning work, as well as a PID controller for the steering due to instability that occurred during testing

3.1 Velocity Control

To decrease the complexity of the implementation, our scale platoon makes four assumptions that are commonly found in string stability literature [8, 10, 11, 41]; the vehicles in the platoon:

- Will follow a single path
- Will not reorder themselves
- No vehicle will join or leave the platoon while it is in progress
- The platoon is homogeneous (all the vehicles perform similarly).

Restricting the platoon to a single path means the followers only need to follow the human-driven vehicle and thus do not require additional sensors and logic to follow

other paths or avoid obstacles that the lead car did not detect. By keeping the order and size of the platoon fixed, the headway can be fixed and the platoon does not need to have logic to expand or contract the headway between vehicles or modify the sources and destinations of the feed-forward logic in the control loops. A homogeneous platoon decreases the effort required to source, configure, and assemble the platoon, as we only need to build one model. While a heterogeneous collection of vehicles may more accurately reflect real world conditions, the same control logic applies to either platoon composition, so it should not affect the results.

Our platoon consists of K Cars numbered C_0 through C_{K-1}, which want to keep a reference headway $d_{r,i}$ from the preceding car. Car C_0 is the human-driven leader while Cars C_1 thorough C_{K-1} are the semi-autonomous followers. Each following Car C_i is equipped with a lidar sensor to measure the distance to the preceding Car, d_i, and an inertial measurement unit (IMU) to measure the velocity, v_i. The reference headway for car C_i is defined such that $d_{r,i} = h_{d,i}v_i + L_i$, where $h_{d,i}$ is a headway in seconds to increase the space between cars at higher speeds, and L_i is a constant offset distance to prevent the cars from coming into contact at low speeds. From that, the error is defined as $e_i = d_i - d_{r,i}$, which expands to

$$e_{i,vel} = d_i - h_{d,i}v_i - L_i \tag{3.1}$$

Because we are using a double integrator model, we also need the derivative of the error, which is approximated on-the-fly by dividing the difference between the current error at time τ and previous error at time $\tau - 1$ by the period Δt

$$\dot{e}_{i,vel}[\tau] = \frac{e_{i,vel}[\tau] - e_{i,vel}[\tau - 1]}{\Delta t} \tag{3.2}$$

$U_{ff,i-1}$ is the feed-forward throttle input from Car C_{i-1}, and is the portion that, as discussed in Section 2.4, allows the platoon to react more quickly and smoothly. The throttle input U_i is calculated by adding together the error, derivative error, and feed-forward value, adjusted by control constants $k_{p(roportional),vel}$, $k_{d(erivative),vel}$, and $k_{f(eed)f(orward),vel}$.

$$U_i = k_{p,vel}e_{i,vel} + k_{d,vel}\dot{e}_{i,vel} + k_{ff,vel}U_{ff,i-1} \tag{3.3}$$

3.2 Steering Control

We do make one modification to the common assumption [8, 10, 11, 41] that the platoon will follow a fixed path. To the best of the author's knowledge, most other research platoons fix the steering, either by only calculating forwards velocity [8, 10, 11, 41], or by physically attaching the platoon to a guide wire to limit motion to one dimension [8]. Since real world platoons, such as Peloton's [26], need to be able to steer, we chose to implement a system to allow our cars to steer towards the marked car in front of it. Our implementation does limit the steering to gentle curves like one would experience in highway driving due to sensor based limitations we will discuss in Section 3.3.2.

We also found that just feeding the raw error between the expected and desired heading caused our cars to overshoot the steering and oscillate, as discussed in section 3.3.1. To mitigate this, we filtered the steering input through another PID controller.

For the steering PID controller, Car C_i measures a_{meas} to the center of the preceding car $C_i - 1$, which is a combination of the angular distance from the center of the following car's view and angular skew of the back of the vehicle. Due to the implementation of our vehicle, an a_{meas} of 90° is straight forward. From there, the error is defined as

$$e_{i,ang} = a_{i,meas} - a_{desired} \tag{3.4}$$

$a_{desired}$ is defined to be 90° for our platoon, as we would always like to be moving straight towards the preceding car. Like the velocity PID controller, the derivative angular error is computed taking the difference of the error at time τ and $\tau - 1$. Unlike the velocity PID controller, the derivative error is not divided by the period, as we found that it destabilized our results in testing. Since our target period of 8 milliseconds is so small, it ended up magnifying even small changes in proportional error. Instead of compensating by using a tiny $k_{d,ang}$, we essentially factored the time constant into $k_{d,ang}$.

$$\dot{e}_{i,ang}[\tau] = e_{i,ang}[\tau] - e_{i,ang}[\tau - 1] \tag{3.5}$$

Our new dampened angle $A_{dampened}$ is the combination of the proportional and derivative errors, also tempered by control constants $k_{p,ang}$ and $k_{d,ang}$.

$$A_{i,dampened} = k_{p,ang}e_{i,ang} + k_{d,ang}\dot{e}_{i,ang} \tag{3.6}$$

3.3 Discussion

As we will discuss in Section 4, our platoon has been through three generations of implementation. As stability improved with each iteration, we discovered several sources of instability in the steering and sensors that we did not anticipate. A large source of instability was eliminated by filtering the steering through a PID

controller, while others are mitigated by constraining the operation of the platoon until future work can address them.

3.3.1 Why Steering is Controlled by a PID Controller

The first two generations of the platoon only implemented the velocity PID controller, as they hit some roadblocks that prevented them from reaching stability at higher speeds, and the raw angles from the camera system were deemed good enough to guide the platoon. As the velocity control of our third-generation platoon matured to the point that we could consistently keep up with the preceding car, we found that raw angles from the camera system would cause the system to steer towards the center of the preceding car so aggressively that it commonly overshot its target. The correction for the overshoot commonly became an oscillation which grew until the car ended up turning so far that it completely lost sight of the preceding car.

This destructive oscillation led to the implementation of the steering PID controller detailed in Section 3.2, and provided the context for why the vast majority of platoon simulations and implementations assume only velocity control.

3.3.2 Steering Limitations

Another behavior that appeared once our third generation platoon matured was the tendency of the following car to engage full throttle and take off if the preceding car moved too far from the center of the following car's view. This was the result of our lidar, the only source of distance to the next car, having a very narrow field-of-view.

The TeraRanger One lidar sensor we use is a hobby-grade distance sensor with a 3° field-of-view [36] that is commonly used for drones and robotics platforms. It works well when the object we would like to identify is directly in front of the car.

The OpenCV portion of our project is good at keeping the following car lined up with the tracking pattern on the back of the preceding car through gentle curves. However, if the preceding car turns sharply, it may move so far laterally that the lidar on the following car registers the next obstacle beyond the preceding car, usually meters away. With that large distance fed into the PID controller, it outputs full throttle and the car takes off.

For the time being, it is sufficient to limit the platoon to straight paths and gentle curves, given that most platooning research assumes only forward motion anyway [8, 10, 12, 41].

In the long term, the team plans to investigate different lidar modules and computer vision based distance tracking to alleviate this limitation. There are lidar modules that offer much wider fields of view through the use of a spinning module or more advanced arrays of lasers. Unfortunately, the wider and more precise the field-of-view is, the more expensive the module; a quote we received in October of 2018 put full-scale automotive modules in excess of $10,000. Computer vision based systems have the advantage of working with our existing 74° field-of-view camera [14], and thus are much cheaper, but will require additional cameras and processing if we want to track the distance of objects other than the tracking pattern on the back of the preceding car.

3.3.3 Camera and Lidar Minimum Distances

There is one more limitation on our third-generation design. Our current mounting positions for the lidar and camera place them at the front of the car, protected from collisions by 8-10 cm of bumper. The lidar's minimum range is 20 cm, so it is impractical to have a following distance lower than 25-30 cm, as the lidar will have difficulty differentiating between a slight slowdown and an impending collision.

Additionally, when following less than 30 cm behind another car, any significant side-to-side movement by the preceding car may cause the tracking pattern to leave the view of the following car's camera, causing OpenCV to return inaccurate values.

Overall, these are low impact issues, as the lateral limitation is within the limitations imposed by the lidar discussed in Section 3.3.2, and our typical headway values of 30 to 50 cm are comparable with current full-scale platoon implementations [26] when we account for the 10 cm the bumper takes up.

Chapter 4

IMPLEMENTATION

The scale platoon has gone through at least three different generations at Cal Poly: the initial project before I was involved, the generation that I worked on for my senior project, and the generation that I am leading for this thesis.

4.1 Goals for the Semi-autonomous Platoon

The overarching goal of building a scale semi-autonomous platoon is to test a number of control-theory-based attacks and attack mitigations that had previously only been tested in simulation [9, 11, 32, 34, 38].

The specific focus of this thesis was to build the string-stable semi-autonomous platoon which could be used to test these attacks. By implementing our platoon at 1/10th scale, we decrease the danger of collisions that inevitably happen in development, as well as the collisions we plan to induce in future security research. Additionally, building in scale allows us to significantly decrease the cost through the use of an off-the-shelf RC car chassis, sensors, and microcontrollers.

4.2 The First Generation

My first exposure to the project was with the generation created by David Xenakis and Willy Okpobua. This version used an off-the-shelf 1/10th scale RC car chassis controlled by a Pixhawk drone controller and a Raspberry Pi 2B. The Pixhawk used a sonar sensor to measure the distance to the next car in the platoon while the Raspberry Pi ran an OpenCV program to communicate to the Pixhawk how to

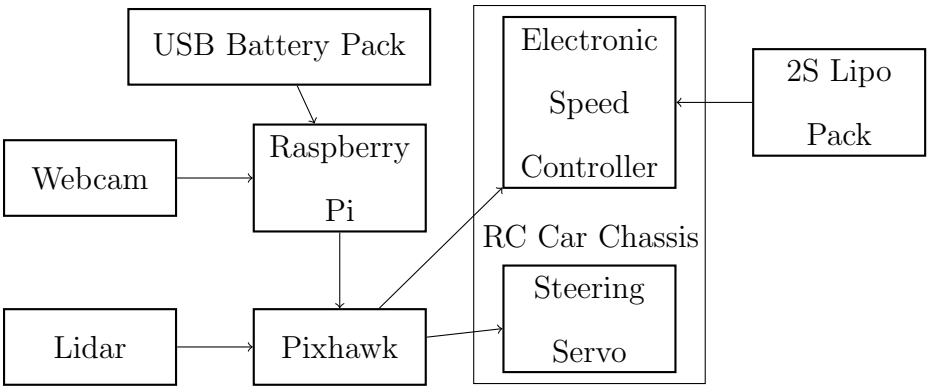

Figure 4.1: A High Level Block Diagram for the First Generation's Design The Raspberry Pi calculated the angle to next car using the webcam. The Pixhawk then took that angle, along with the distance from the lidar, and forwarded that to the RC car's electronic speed controller and steering servo. Power for the Raspberry Pi and camera was supplied by a USB power bank, while the Pixhawk, lidar, and car were powered by a 2S lipo pack. The arrows indicate the high level data and power flow.

steer towards the preceding car. The sonar sensor did not perform well in even moderately noisy environments, so it was swapped out for a fixed lidar sensor.

While I did not personally see the platoon driving, it reportedly reacted slowly due a combination of the Raspberry Pi's low compute power and the lack of hardware or GPU optimizations for OpenCV on a Raspberry Pi 2B. Thus, the system had difficulty performing real-time tracking of an object in OpenCV with low enough latency to direct an RC car. This was the main factor in the upgrade that led to the second generation.

4.3 The Second Generation

The second generation was undertaken as my Senior Project at Cal Poly, and was done in conjunction with Eva Chen (also for her senior project [6]), Willy Okpobua, Sebastian Seibert von Fock, Justin Nguyen, Greg Chu, Alexandria Adams, and Cassidy Elwell. This iteration was largely predicated on the upgrade from a Raspberry Pi to an Nvidia Jetson TX 1, mostly due to the Jetson's 256 CUDA

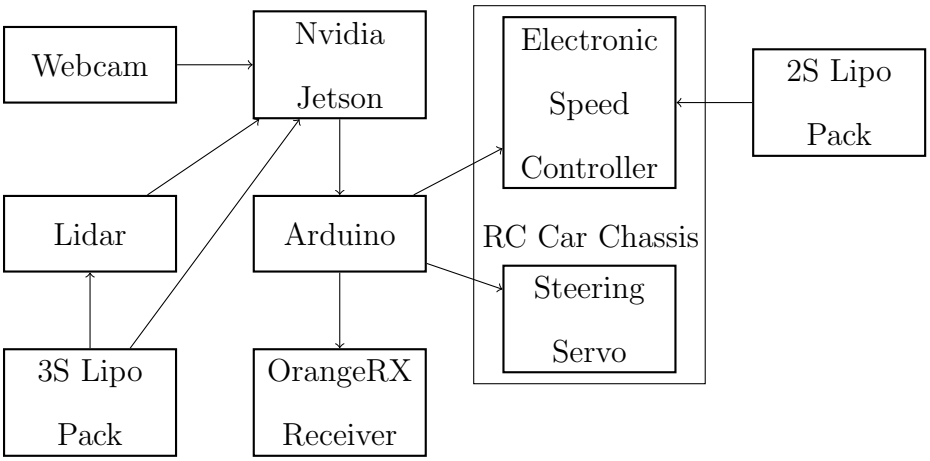

Figure 4.2: A High Level Block Diagram for the Second Generation's Design

The Jetson is responsible for calculating the angular distance to the next car using the webcam, then sending that angle to the Arduino. The Arduino is responsible for forwarding that angle to the steering servo, as well as retrieving the distance to the next car from the lidar, then calculating the next throttle and send that to the ESC. The Jetson and lidar are directly powered by the 3S lipo pack, while the ESC is powered by the 2S lipo and shares that power with the servo and Arduino. The OrangeRX Receiver is used to communicate a stop signal to the platoon as a fail safe mechanism. The arrows indicate the high level data and power flow.

cores that could significantly accelerate the OpenCV work. While recent hardware optimizations allow the Raspberry Pi to perform SIMD [24] and improve the Floating Point operation performance [23] for very noticeable gains in OpenCV performance [30], the Jetson's much more powerful GPU still provides a significant advantage.

While in the process of migrating, we found the Pixhawk models we had on hand required too much development effort for what was a relatively simple PWM control of our RC car chassis' electronic speed controller and steering servo, so the Pixhawk was dropped in favor of an Arduino Mega.

Sebastian and Justin were responsible for porting the python-based OpenCV code from the Pi to C++ on the Jetson for the significantly better performance from CUDA support in C++. The rest of the functionality was implemented on the Arduino by the rest of the team, including lidar-based distance following with ESC control, passing the steering value from the Jetson to the steering servo, and a radio-based "kill button" to allow us to halt the entire platoon for testing safety.

We did successfully test a two-car platoon, with one computer-controlled car following a human-driven one. This implementation lacked a network to send feed-forward information, and it was a bit jerky and responded slowly, which motivated the creation of the third generation.

4.4 The Third Generation

The two main goals of the third generation were to implement a network connection for the feed-forward mechanism and to smooth out and improve the performance of the platoon. This portion of the project was completed in conjunction with Justin Nguyen, Matt Lewis, and Willy Okpobua.

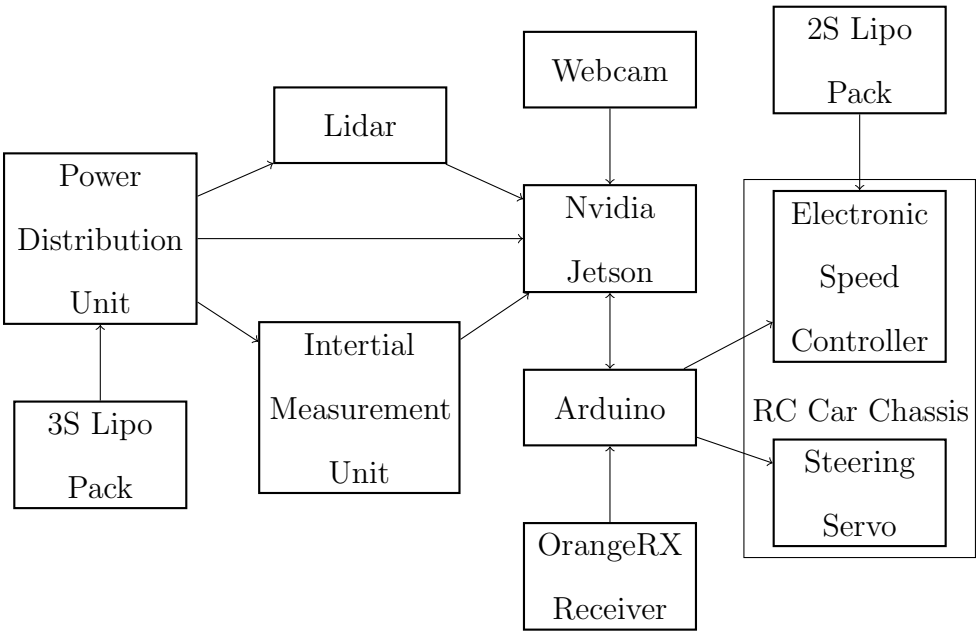

Figure 4.3: A High Level Block Diagram for the Third Generation's Design This generation builds off much of the architecture of the second generation, with some components and roles shifted around. The Jetson is still responsible for using the webcam to meausure the angle to the preceeding car, but it also is now responsible for retreiving the sensor data from the lidar (previously done by the Arduino) and the IMU (a new addition to capture the velocity of the vehicle). This change was made to shift the PID logic onto the much more capable 4 or 6 core processor of the Jetson. The Arduino was also upgraded to communicate back to the Jetson, so that it can keep the Jetson appraised of the throttle of the lead, human-driven car and any stop signals that come through. The Arduino's connction to the Jetson was also switched to USB as that simplified the logic level shifting between the 5 V Arduino and 3.3 V Jetson, as well as the power distribution. For this generation, the 3S lipo powers a Power Distribution Unit (PDU), which in turn powers the lidar, Jetson, and IMU. The PDU is used to more cleanly step the voltage down to levels more appropriate for the individual components, as well as provide a sufficient number of connectors for all the compoents that need power. The arrows indicate the high level data and power flow.

4.4.1 Overview

After analyzing the code to diagnose the jerky performance of the second generation, I discovered that several portions of the code on the Arduino would halt execution while waiting for results. Since the Arduino only has a single core and only executes a single code loop at a time, the sleeps would halt the execution of everyone's code.

After a quick test with FreeRTOS on the Arduino to enable multiprocessing support, we decided to re-architect the project to move most of the processing to the Jetson, as the Jetson's four cores and use of Ubuntu allow for multiple concurrent processes.

The Jetson communicates with the lidar and IMU over I2C, the Arduino over UART via USB, and the camera over USB. The reader processes for the lidar and IMU, as well as the OpenCV code that makes use of the CUDA cores and USB camera, all run as independent processes written in C++, as most of our collective developmental experience is in writing code for Linux. There was some discussion of using a real-time operating system like FreeRTOS or ROS, but the non-standard packages used by our OpenCV implementation and our general inexperience with RTOSes led us to choose Linux.

All of the sensor processes are connected through named pipes back to a central piece of code, dubbed "Grand Central", that is responsible for collecting data input from the reader processes asynchronously. Grand Central is also responsible for receiving a throttle value over a wifi network from the preceding car in the platoon for the feed-forward portion of the control algorithm. Periodically, Grand Central calculates the next throttle value based on the control parameters and inputs it has through the process described in Chapter 3. Once Grand Central has calculated the next throttle value, it sends the updated velocity and turning angle to the Arduino, as well as the next car in the platoon for its feed-forward value.

We were able to do some preliminary testing with a leader and one following car that proved to be much more stable than the previous generation, but we did not achieve string stability or test platoons with 3 or more cars.

4.4.2 My Contributions

For the third generation, I was responsible for the code for the control logic, interprocess, serial, and networking communications, and main process, named Grand Central, that combined all those.

The control logic is separated out into a set of functions to allow for simple unit tests, and the control constants exist in a separate header file. The unit tests were very helpful to double check that the logic implemented produced sane results and prevent simple slip ups like sign errors. Separating the control constants into a header file allows for quick changes that make calibration of the control systems very simple.

All of the communications for the platoon are serialized by NanoPb, a C-based implementation of Google's Protobuf that fits well in the memory constraints of the Arduino. This allows for a simple centralized declaration of the data that is being transported around, as well as a consistent serialization library that has already been extensively tested. The interprocess communications are backed by Posix named pipes, the serial communications are backed by the Posix Serial implementation, and the network communications are a publisher-subscriber scheme backed by NNG. The publisher-subscriber model of network communications was chosen because it allows any number of vehicles to subscribe to the feed-forward signal of any car in the platoon, enabling other schemes than the current model that only listens to the preceding car. Each of the communication methods lives in its own class designed to abstract away many of the specifics and make integration into

other module compact and easy to understand.

Grand Central collects all those parts into one central place. It opens up a serial connection to the Arduino and a network connection to any sources it is programmed to subscribe to. Each of the sensor processes open a named pipe to Grand Central and will send their data as they retrieve it. Grand Central uses Posix's Select function to wait and asynchronously consume the data from the sensor processes. On a timeout that is defined in the control constants, Grand Central will calculate the next throttle and angle, send those values to the Arduino over serial, and publish the throttle to any network subscribers. Once Grand Central has sent out the throttle and angle, it will return to waiting with Select. As of the current implementation, it is the responsibility of the sensor processes to update in a timely manner; if a process does not send an update, then Grand Central will simply re-use the previous value. That said, in our testing, we found that the sensors often update as many as 10 times faster than Grand Central did even with an 8 ms tick, so that is not a large concern for us.

4.5 Part Selection for Cost and Simplicity

One of the main goals of this project is to develop a platoon that is inexpensive and easy for others to replicate. The components chosen for this project were purchased for $750 and are available off-the-shelf. All prices and availability are as of March 2019.

For the chassis, we purchased a Redcat Volcano 1/10th scale RC car for $150. It has handled the various collisions in our testing very well, though we are looking for a model that trades some of the speed for torque, as discussed in Section 4.6.1.

Our latest generation of platoon makes use of Nvidia Jetsons and Arduinos to control the cars. Specifically, we use a mix of Jetson TX1s and TX2s in the default

carrier dev board and Arduino Megas. The TX2 offers a noticeable improvement in processing speed for OpenCV, the two extra ARM cores offer some extra headroom for additional processing, and the extra flash storage on board make installing large packages like OpenCV much easier, but the TX1's $100 price difference may make it more attractive. Nvidia's education discount of 50% brings them down to a much more affordable $300 for the TX2 and $250 for the TX1.

Our selection of the $40 Arduino Mega was influenced by our second generation design, as we intended to do much more computation and communicate with multiple sensors. For the third generation, our microcontroller requires a serial communication channel to talk to the Jetson and three to five PWM channels to control the ESC and steering, and read the stop, throttle, and angle signals from the remote control. There are a number of smaller, less expensive microcontrollers that should still be sufficient, like the AtTiny microcontroller family, the Adafruit Trinket or Feather series, or Arduino Pro Micro clones.

As for sensors, we use the $170 Teraranger One lidar sensor, $24 Adafruit LSM9DS0 IMU, $3 Sparkfun Bi-directional Logic Level Converter, and $30 Logitech C615 webcam. As discussed in Section 3.3.2, we are exploring other options for the lidar sensor to provide a wider field of view.

We power our cars with a 2S Lithium Polymer (lipo) pack for the motor and servo, and a 3S lipo pack for the Jetson and sensors. Both packs were found on Amazon for around $25, though there are many different options that can be found for cheaper elsewhere, including using the NiMH packs that come with the cars.

4.6 Discussion

As we built the the platoons, there were some limitations that cropped up, as well as a number of lessons that we learned.

4.6.1 Challenges with an Off-the-Shelf RC Chassis

One of the main design choices we made for this project was to use an off-the-shelf 1/10th scale hobby RC car to allow us to focus our budget and development efforts on implementing the control portions of the project. While the cars were reasonably priced and quick to assemble, their pwm control and lack of low-speed torque created additional challenges for the project.

The largest issue was that the chassis was optimized for racing, trading low speed torque for a higher top speed. This made it difficult to drive at or under 6.5 miles per hour, which simulates highway speeds for our 1/10th scale platoon. Increasing the PWM throttle signal 200 hz above the neutral throttle will make the car run at 6.5 mph, but the cars often won't move from rest until we increase the throttle to 100-150 hz above the neutral throttle. The remaining 50 hz do not correspond linearly to any particular speed, leading to very coarse speed control. Additionally, the Jetson is not designed to output PWM signals [17], so we need another microcontroller to translate the Jetson's calculated values to a PWM signal the car can understand. We are looking into a different RC car or a robotics platform that is better suited for speed control up to 7 mph and ideally controlled by a protocol that is better supported by the Jetson, such as I2C or UART.

4.6.2 Why we chose 802.11n for V2V Communication

Our decision to send the feed-forward data over a standard 802.11n wifi network was made with cost and simplicity in mind. The Nvidia Jetson already supports wifi, and there are a plethora of mature, stable APIs to transfer data over it. Our initial tests with traditional wifi access points performed well, and the Jetson's support for creating an ad-hoc wireless network [18] will allow one of the vehicles to create a network in the absence of traditional infrastructure.

This does not affect our testing, but it arguably means our implementation does not use true V2V communications. However, since the proposed V2V standards of Digital Short Range Communications (DSRC) and cellular 5G are both still in development [1, 3], the modules to implement them are large, expensive, and not guaranteed to work with the final standard. As such, we will wait for a clearer standard to emerge before trying to implement it on our vehicles.

4.6.3 Write Design Documents before Coding

As touched on in Sections 4.3 and 4.4, starting with design documents could have saved us a lot of time and helped speed development along, potentially saving us an entire generation of development.

For the second generation, our initial planning only went so far as assigning high level functions to each person, such as reading from the lidar, setting the speed and steering, or transferring the data between the Jetson and Arduino. Everyone developed and tested their function in relative isolation, and some of those pieces were reliant on halting the processor's execution while that task completed. When all the pieces were merged together, one halted part blocked everything else, resulting in very jerky acceleration that hampered the stability of the platoon.

In hindsight, the second generation had too many resources contending for processor time on the Arduino, so for the third generation we backtracked and made sure to talk about not only the individual parts, but how they fit together before we started development. This planning led to our decision to move most of the processing to the more capable Jetson.

4.6.4 Choose a Build System before Starting Development

Since the features of this project were developed concurrently by different developers without prior agreement on a build system, each section of code used a slightly different build setup. The simpler sections could get away with comments with the g++ command to invoke or a simple Makefile, but the codebase grew and we started making simple libraries to share among features. Maintaining Makefiles and individual commands to point to the libraries became a headache that required updating a number of files across the project for even simple changes in the libraries.

That prompted us to switch to Bazel for a more unified build system and nicer syntax than alternatives like cmake. With that move, our code is much better organized, less redundant, and more reliable, so I am much more confident that we can make changes without breaking something elsewhere.

4.6.5 Implement Tests

Testing the intersection of software and hardware can be challenging, especially if we want to automate it. As is quite common on student projects, implementing features on our project took precedence over unit and integration tests.

Later in development, we encountered several bugs whose symptoms were mistaken for or masked by other bugs in our limited manual testing. One such case was a jitter in the steering that made it difficult to get the cars to stay in line while driving. We assumed the jitter originated from the OpenCV data and spend time attempting to smooth out the results from OpenCV. When that did not resolve the bug, our further testing found that a bug in the Jetson to Arduino communication was actually introducing the jitter. With that jitter addressed, we found that our vehicles overshot the expected headway we set, regardless of how we set the control

variables. That led us to discover a missed unit conversion that left the derivative error orders of magnitude smaller than it should be. A simple unit test could have caught the control logic bug, and integration tests could have caught the Arduino-Jetson miscommunication bug and the shown that OpenCV was not the issue.

For some of the more recently developed features, I started with a set of unit and integration tests, and they proved invaluable to confirm that all the parts of my code worked. Development went much faster when I could confirm that the changes I made did not break other functionality. When the module didn't work when dropped into the other code, the tests reassured me that the functionality of the code worked and allowed me to find the bug another portion of the code.

4.6.6 Start with Interprocess Communication

In hindsight, we should have prioritized code that connected the parts that different people worked on individually, then moved on to the parts that were individually accomplishable. In our case, this was mostly the code that transferred data between processes and microcontrollers. The individual development went relatively smoothly, but getting multiple people together to coordinate communication between their code took time and schedule coordination. Front loading that work and filling in gaps with individual development would have been a more efficient approach.

The three practices described above compliment and reinforce each other. Discussing and documenting the connections between the components can help define a number of test cases. Good test cases help development of the connecting portions of the code, and allow developers to confirm that their portion of the code will likely work without needing to wait for the rest of the code to be completed.

Chapter 5

EXPERIMENTAL DATA

5.1 Lidar Validation

This section validates the performance of our lidar sensor. These tests came later in the development; in hindsight, they probably should have been done much sooner, as the results might have encouraged us to choose a different sensor. While we were aware that our measured results were offset from the actual results, we were not aware that it was not constant.

The accuracy of the sensor left a bit to be desired. Figure 5.1 shows the distribution of approximately 3000 readings of the lidar 300 mm away from a cardboard box. The lidar module measured an average of 434.5 mm (134.5 mm farther than the actual distance), with a standard deviation of 5.2 mm and variance of 27.4 mm.

Figure 5.2 repeats the same test with approximately 2,500 readings 400 mm away from the cardboard box. The module measured an average of 525.7 mm (125.7 farther than the actual distance), with a standard deviation of 4.4 mm and variance of 19.4 mm.

Figure 5.3 shows the distribution of approximately 4,900 readings 1000 mm away from the tracking pattern on a metal filing cabinet. The module measured an average of 820.1 mm (179.9 closer than the actual distance), with a standard deviation of 6.4 mm and variance of 41.4 mm.

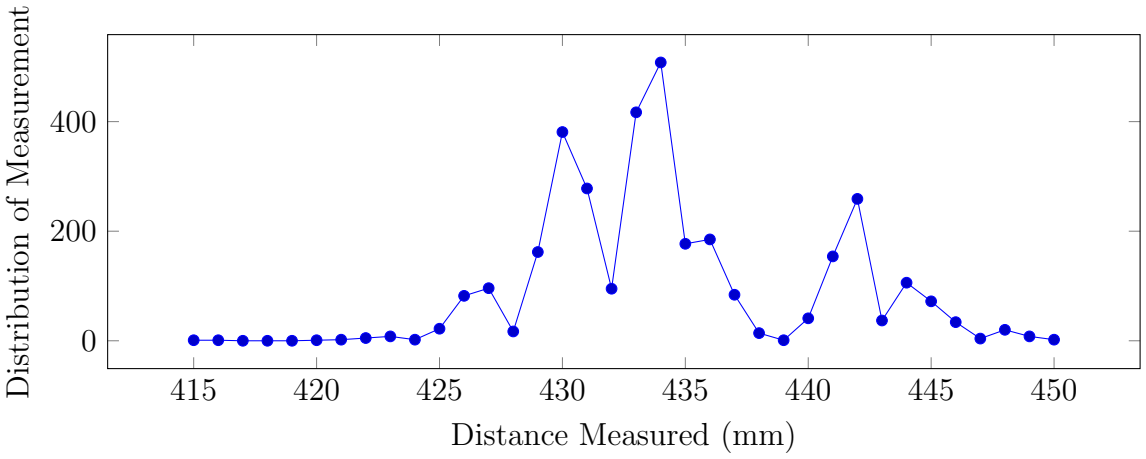

Figure 5.1: 300 mm Lidar Measurement Distribution
The distribution of measurements when the lidar is placed 300 mm from a cardboard box.

During those tests, the lidar module would not read lower than 200 mm, despite having the surface nearly touching the sensor, and did not read higher than approximately 1100 mm away, significantly limiting its useful range.

5.2 System Sensor Response

Figure 5.4 shows the velocity response when the system is placed two times the resting distance from a wall (with the tracking pattern attached to it to keep the car straight). In this case, the resting distance is 400 mm, k_{prp} is 0.255, and k_{drv} is 0.48.

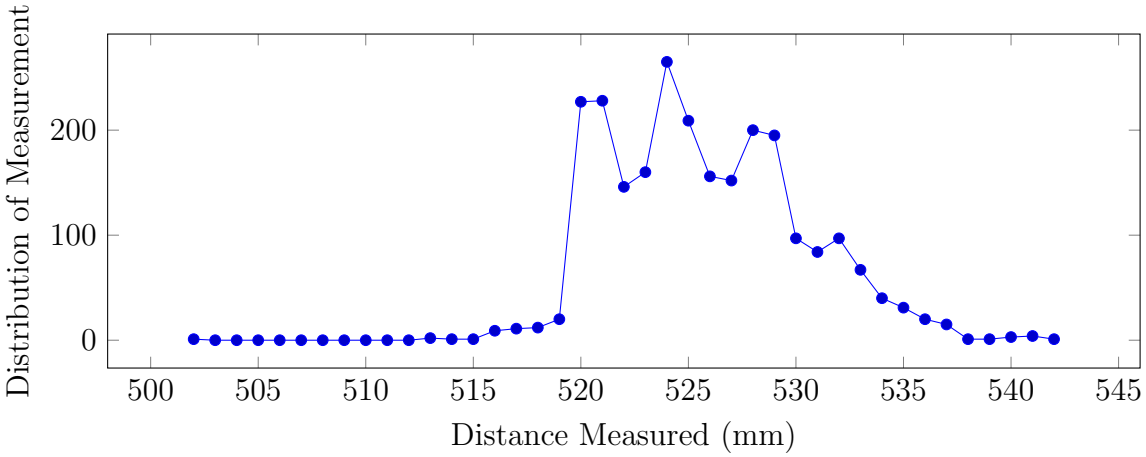

Figure 5.2: 400 mm Lidar Measurement Distribution
The distribution of measurements when the lidar is placed 400 mm from a cardboard box.

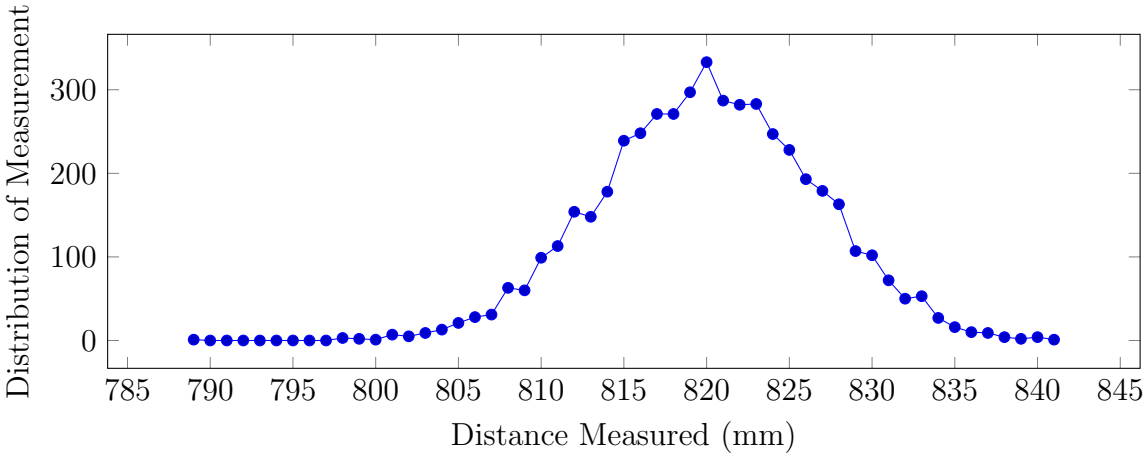

Figure 5.3: 1000 mm Lidar Measurement Distribution
The distribution of measurements when the lidar is placed 1000 mm from a cardboard box.

This response is reasonable, but demonstrates both a bug, and the need for some additional PID tweaking. The bug in this case seems to be that when the velocity controller requests Neutral throttle, the Arduino outputs a full speed backwards signal. This appears around times 5000 ms, 6000 ms, and 14000 ms where the system suddenly jerks away from the resting headway. Of course, the sudden change in velocity creates a large derivative error, jerking the car back towards the resting headway. This seems to cause the system to overshoot the desired resting headway at 14000 ms. The distance does not creep back up to the desired resting distance because the chassis reverses slower than it travels forward and doesn't always follow reverse commands unless they are preceded by a very large reverse signal.

With that bug addressed, it appears that the system would rather smoothly asymptotically approach the desired resting distance. That said, the performance of the PID controller could still likely be improved. In particular, the derivative error rapidly swings between values that greatly exceed the proportional error, and, as mentioned earlier, seems to exacerbate the bug in the throttle to PWM translation. This would appear to be the small time period (40 ms in this case) disproportionately increasing the derivative error. Perhaps we should remove the period like we did for the steering controller

5.3 Two Car Run Without Feed-Forward

Figures 5.5 and 5.6 show the velocity control data from one of our cars following the human driven lead car before the feed-forward from the lead car was implemented. The PID controller is running with $k_{prp} = 0.255$ and $k_{drv} = 0.48$. These results are not stable, as they do not stay bounded.

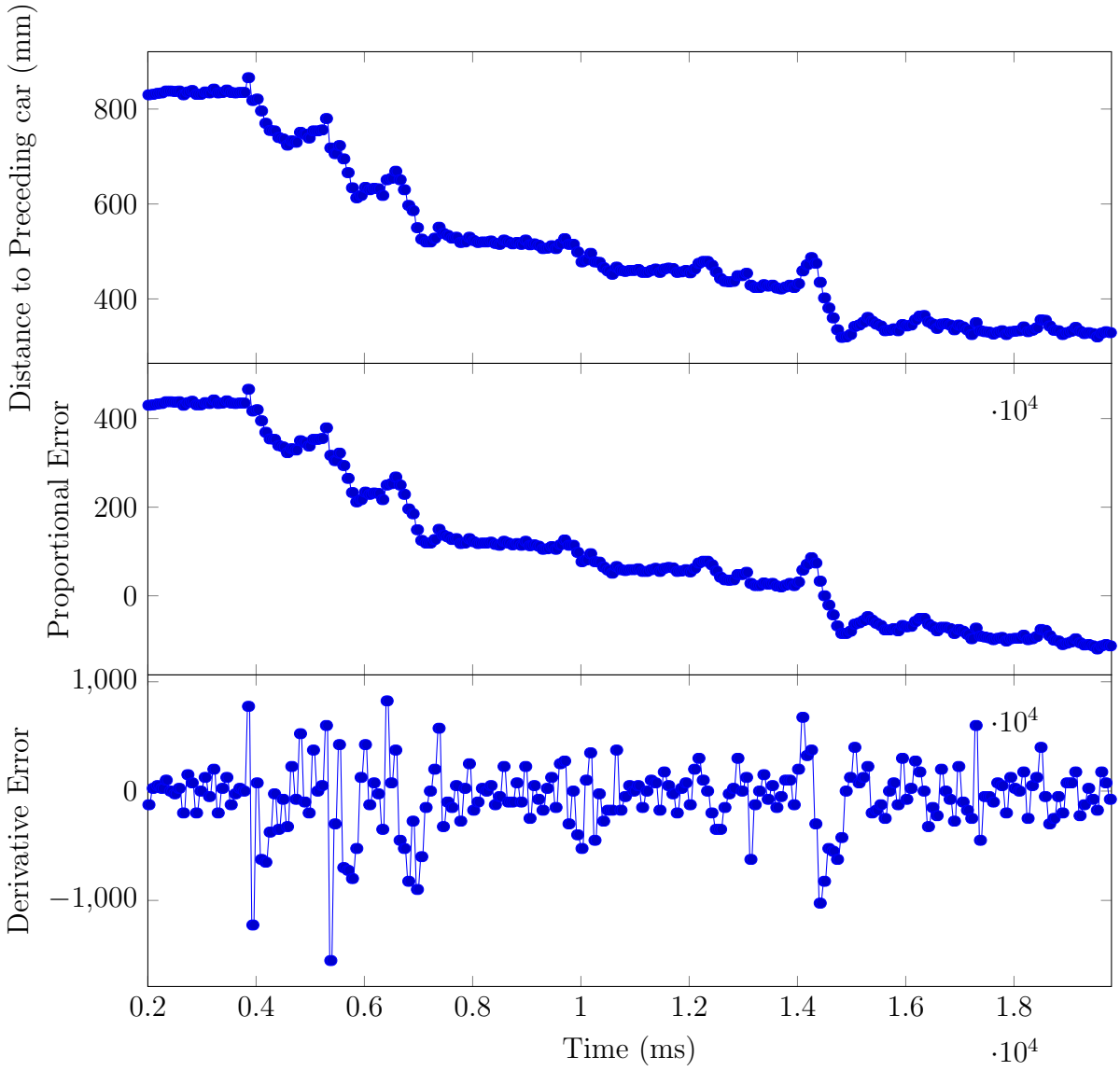

Figure 5.4: Stationary Target 800 mm Drive
These graphs show the system response with only the sensors active while it drives towards a fixed point (in this case, the wall). The system starts at approximately 800 mm away from the wall; twice the headway at rest of 400 mm.

There are a number of factors that likely play into this instability. The control parameters could likely be tweaked further. In particular, the derivative error is very large; that may be a side effect of our very small period amplifying the results. That could be addressed by significantly reducing the derivative constant, or by removing the period, as we did in the steering PID controller. Additionally, adding in the feed-forward will increase the stability of the system by reducing the reaction time of the system. The instability may also be a compounded by the challenges with the chassis, as described in Section 4.6.1.

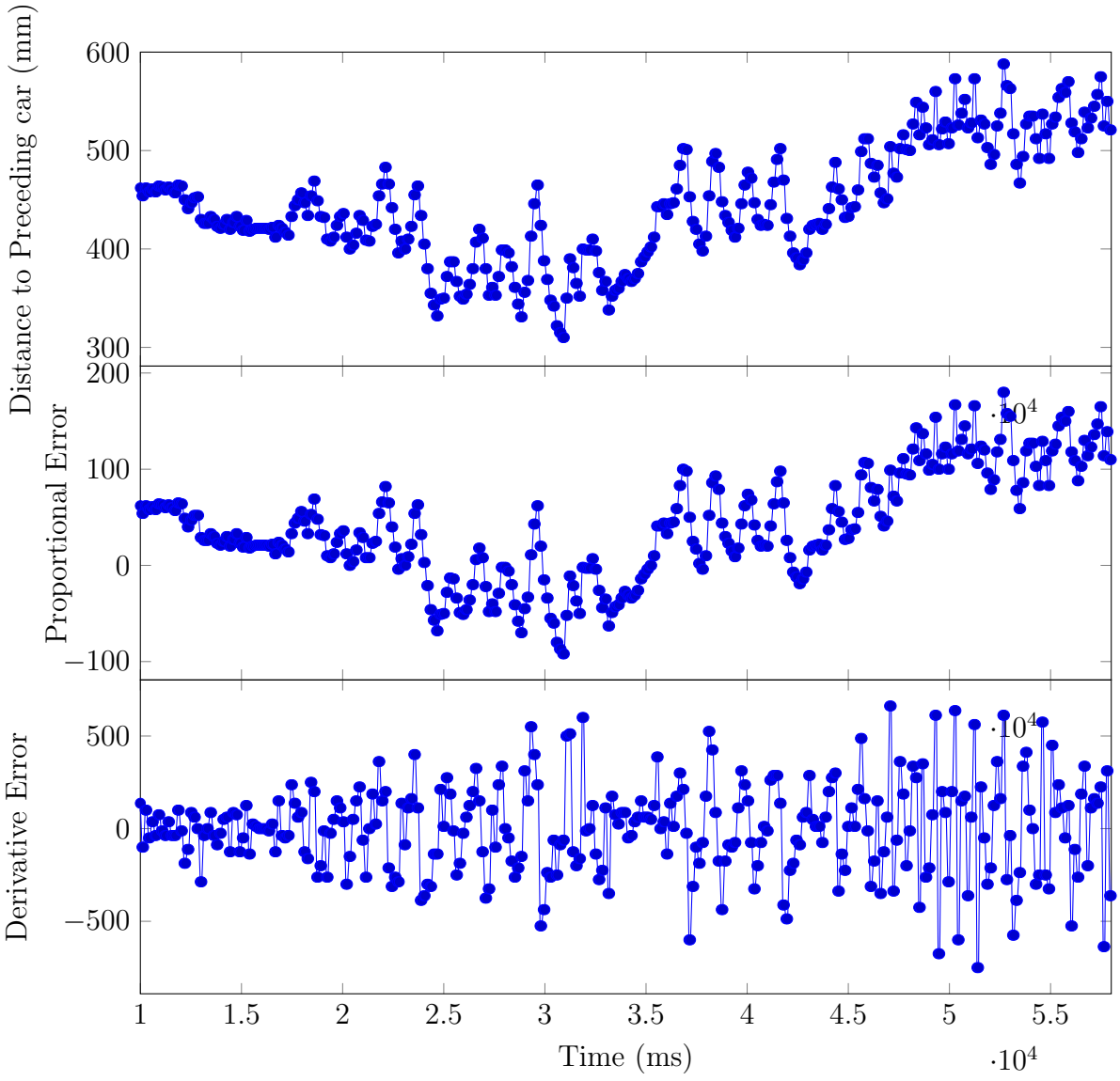

Figure 5.5: Non-cooperative Platooning Run 1
These graphs show the measured distance as well as the proportional and derivative error from the desired distance to the leader of the non-cooperative platoon. The headway at rest was set to 400 mm.

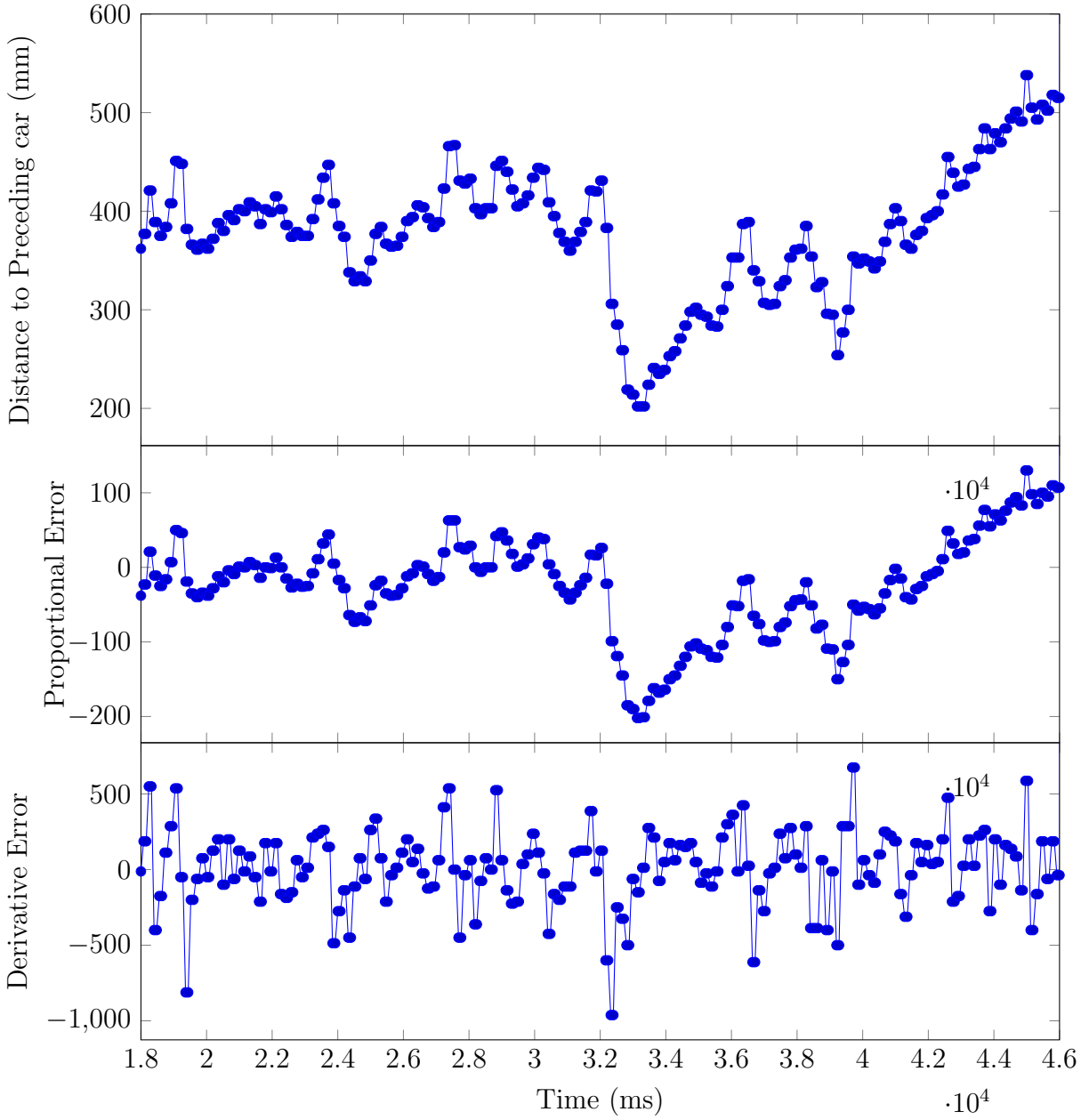

Figure 5.6: Non-cooperative Platooning Run 2
These graphs show another run with the measured distance as well as the proportional and derivative error from the desired distance to the leader of the non-cooperative platoon. The headway at rest was set to 400 mm.

Chapter 6

CONCLUSION

While completely autonomous vehicles are still in development, platoons of semi-autonomous vehicles offer a simple way to achieve most of the same functionality. We laid the foundation for a 1/10th scale platoon of collaborative semi-autonomous vehicles to verify that research conducted in simulation works in scale. This project will enable future research into more complex attacks and mitigation schemes.

Bibliography

[1] A new connected-car battle: Cellular vs. DSRC. en. https://www.autonews.com/mobility-report/new-connected-car-battle-cellular-vs-dsrc, Feb. 2019.

[2] Apollo. http://apollo.auto/.

[3] Backers of V2V aren't waiting for a mandate. en. https://www.autonews.com/article/20180224/MOBILITY/180229894/backers-of-v2v-aren-t-waiting-for-a-mandate, Feb. 2018.

[4] BMW Group, Intel and Mobileye Team Up to Bring Fully Autonomous Driving to Streets by 2021. en-US. https://newsroom.intel.com/news-releases/intel-bmw-group-mobileye-autonomous-driving/.

[5] J. S. Brodsky. Autonomous Vehicle Regulation: How an Uncertain Legal Landscape May Hit the Brakes on Self-Driving Cars Cyberlaw and Venture Law. eng. *Berkeley Technology Law Journal*, 31:851–878, 2016.

[6] E. Chen. Modeling Autonomous Vehicles through Radio Controlled Cars. *Computer Engineering*, June 2017.

[7] Cruise Automation. https://getcruise.com/.

[8] D. D. Dunn, S. Mitchell, I. Sajjad, R. M. Gerdes, R. Sharma, and M. Li. Regular: Attacker-Induced Traffic Flow Instability in a Stream of Semi-Automated Vehicles. In pages 499–510, June 2017. DOI: 10.1109/DSN.2017.61.

[9] S. Dadras, R. M. Gerdes, and R. Sharma. Vehicular Platooning in an Adversarial Environment. en. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security - ASIA CCS '15*, pages 167–178, Singapore, Republic of Singapore. ACM Press, 2015. ISBN: 978-1-4503-3245-3. DOI: 10.1145/2714576.2714619.

[10] S. Dadras, R. M. Gerdes, and R. Sharma. Vehicular Platooning in an Adversarial Environment. en. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security - ASIA CCS '15*, pages 167–178, Singapore, Republic of Singapore. ACM Press, 2015. ISBN: 978-1-4503-3245-3. DOI: 10.1145/2714576.2714619.

[11] B. DeBruhl and P. Tague. Optimizing a MisInformation and MisBehavior (MIB) Attack Targeting Connected Cars. en. *IEEE Connected and Automated Vehicles Symposium (CAVS)*:5, Aug. 2018.

[12] B. DeBruhl, S. Weerakkody, B. Sinopoli, and P. Tague. Is your commute driving you crazy?: a study of misbehavior in vehicular platoons. en. In *Proceedings of the 8th ACM Conference on Security & Privacy in Wireless and Mobile Networks - WiSec '15*, pages 1–11, New York, New York. ACM Press, 2015. ISBN: 978-1-4503-3623-9. DOI: 10.1145/2766498.2766505.

[13] D. J. Fagnant and K. Kockelman. Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations. *Transportation Research Part A: Policy and Practice*, 77:167–181, July 2015. ISSN: 0965-8564. DOI: 10.1016/j.tra.2015.04.003.

[14] HD Webcam C615 - Logitech Support. https://support.logitech.com/en_us/product/hd-webcam-c615/specs.

[15] Honda and Alphabet Inc.'s Waymo Enter Discussions on Technical Collaboration of Fully Self-driving Automobile Technology.

http://hondanews.com/releases/honda-and-alphabet-inc-s-waymo-enter-discussions-on-technical-collaboration-of-fully-self-driving-automobile-technology.

[16] R. Hussain and S. Zeadally. Autonomous Cars: Research Results, Issues and Future Challenges. *IEEE Communications Surveys Tutorials*:1–1, 2018. ISSN: 1553-877X. DOI: 10.1109/COMST.2018.2869360.

[17] Jetson/PWM - eLinux.org. https://elinux.org/Jetson/PWM.

[18] Jetson/TX1 WiFi Access Point - eLinux.org. https://elinux.org/Jetson/TX1_WiFi_Access_Point.

[19] J. Krafcik. A first look at our Waymo fully self-driving Chrysler Pacifica Hybrid minivans, Dec. 2016.

[20] U. Lee, J. Jung, S. Jung, and D. H. Shim. Development of a self-driving car that can handle the adverse weather. en. *International Journal of Automotive Technology*, 19(1):191–197, Feb. 2018. ISSN: 1229-9138, 1976-3832. DOI: 10.1007/s12239-018-0018-z.

[21] F. Lin, M. Fardad, and M. R. Jovanovic. Optimal Control of Vehicular Formations With Nearest Neighbor Interactions. *IEEE Transactions on Automatic Control*, 57(9):2203–2218, Sept. 2012. ISSN: 0018-9286. DOI: 10.1109/TAC.2011.2181790.

[22] Looking Further. http://corporate.ford.com/innovation/autonomous-2021.html.

[23] A. Ltd. Instruction Sets — Floating Point. en. https://developer.arm.com/architectures/instruction-sets/floating-point.

[24] A. Ltd. SIMD ISAs — Neon. en. https://developer.arm.com/architectures/instruction-sets/simd-isas/neon.

[25] Mercedes-Benz Innovation: Autonomous Driving. en-US. https://www.mercedes-benz.com/en/mercedes-benz/next/automation/.

[26] *Peloton Tech Website*.

[27] J. Ploeg, B. T. M. Scheepers, E. van Nunen, N. van de Wouw, and H. Nijmeijer. Design and experimental evaluation of cooperative adaptive cruise control. In *2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC)*, pages 260–265, Oct. 2011. DOI: 10.1109/ITSC.2011.6082981.

[28] Preparing for the Future of Transportation: Automated Vehicle 3.0. en. https://www.transportation.gov/av/3, Text, Sept. 2018.

[29] Renault-Nissan and Microsoft partner to deliver the future of connected driving. en-US. https://news.microsoft.com/2016/09/26/renault-nissan-and-microsoft-partner-to-deliver-the-future-of-connected-driving/, Sept. 2016.

[30] A. Rosebrock. Optimizing OpenCV on the Raspberry Pi. en-US, Oct. 2017.

[31] SAE J 3016-2018 - Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles. https://webstore.ansi.org/Standards/SAE/SAE30162018?source=blog.

[32] I. Sajjad, D. D. Dunn, R. Sharma, and R. Gerdes. Attack Mitigation in Adversarial Platooning Using Detection-Based Sliding Mode Control. en. In *Proceedings of the First ACM Workshop on Cyber-Physical Systems-Security and/or PrivaCy - CPS-SPC '15*, pages 43–53, Denver, Colorado, USA. ACM Press, 2015. ISBN: 978-1-4503-3827-1. DOI: 10.1145/2808705.2808713.

[33] J. Smith, R. Mihelic, B. Gifford, and M. Ellis. Aerodynamic Impact of Tractor-Trailer in Drafting Configuration. en. *SAE International Journal of*

Commercial Vehicles, 7(2):619–625, Sept. 2014. ISSN: 1946-3928. DOI: 10.4271/2014-01-2436.

[34] M. Sun, M. Li, and R. Gerdes. A data trust framework for VANETs enabling false data detection and secure vehicle tracking. In *2017 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9, Oct. 2017. DOI: 10.1109/CNS.2017.8228654.

[35] E. R. Teoh and D. G. Kidd. Rage against the machine? Google's self-driving cars versus human drivers. *Journal of Safety Research*, 63:57–60, Dec. 2017. ISSN: 0022-4375. DOI: 10.1016/j.jsr.2017.08.008.

[36] TeraRanger One - The best distance sensor for drones and robotics. en-US.

[37] Tesla Autopilot. https://www.tesla.com/autopilot.

[38] T. Tithi, C. Winstead, and R. Gerdes. Viability of Using Shadows Cast by Vehicles for Position Verification in Vehicle Platooning. In *2017 IEEE Trustcom/BigDataSE/ICESS*, pages 210–217, Aug. 2017. DOI: 10.1109/Trustcom/BigDataSE/ICESS.2017.239.

[39] Toyota Research Institute Demonstrates Progress in Advanced Technology Research — Toyota USA Newsroom. http://corporatenews.pressroom.toyota.com/releases/toyota+research+institute+demonstrat

[40] Toyota Safety Sense. en. https://www.toyota.com/safety-sense/animation/pcspd.

[41] D. v a h g Swaroop. String Stability Of Interconnected Systems: An Application To Platooning In Automated Highway Systems. en, 1997.

[42] Volvo Cars and Uber join forces to develop autonomous driving cars. en-us. https://www.media.volvocars.com/us/en-us/media/pressreleases/194795/volvo-cars-and-uber-join-forces-to-develop-autonomous-driving-cars.

[43] Waymo. en. https://waymo.com/.

[44] What is Honda Sensing® Suite? Features & More — Honda. en. https://automobiles.honda.com:443/sensing.