Additional Information Sources to Improve the Robustness of Self-Driving Car Algorithms

Brandon Rinderle, Evan Fisher, Elizabeth Hayes
Advisor: Chong Tang
University of Virginia
Charlottesville, VA, United States

Abstract - This paper discusses potential additional information sources of sensor data from smartphones to improve the security of self-driving cars, especially around stop signs and stop lights. The sensor data can be used to create an algorithm to identify stops during a drive. Can the incorporation of information about upcoming stoppages during a drive save a car from a collision if it cannot correctly identify the stop sign or stoplight on the road through its normal image-processing sensors?

I. INTRODUCTION

Autonomous vehicles rely on a variety of different information inputs to plan and execute logistical driving decisions while on the road. They also employ data-fusion algorithms that enable themselves to make coherent decisions based off of a variety of input data. These two integral components of these vehicles must work in constant unison to ensure proper route planning. However, many of the sensor inputs of autonomous vehicles are easily susceptible to even simple disabling and fake imaging attacks.

The goal of this research was to study a number of current security systems for autonomous vehicles, and to present alternative information sources to improve the robustness of autonomous vehicles in the wake of an attack. Researchers developed a database of stops that a car would have access to in the event that critical sensors were disabled or unusable for a period of time. This included collecting data along routes with stop signs and traffic lights. Additionally, researchers developed an algorithm to determine the location of stops based on specific parameters, leading to the creation of a database of stops and their associated geographical locations. This database was then tested using a self-driving car simulator.

This paper describes the research methods used in this project and will discuss the methods, shortcomings, and future work from the results.

II. BACKGROUND RESEARCH

A. Light Detection and Ranging (LiDAR)

LiDAR is a remote sensing mechanism that employs a pulsed laser to emit light and determine distances from objects. It consists of a laser, a scanner, and a GPS receiver to accurately detect natural and manmade objects. In autonomous vehicles, this is extremely useful as LiDAR provides 360 degrees of precise information regarding surrounding and far-away objects. Because of these attributes, LiDAR is an extremely popular sensor, and is used by nearly all autonomous vehicles and simulators. However, one main limitation of LiDAR is its dependence on light for detecting objects, meaning that inclement weather diminishes detection capabilities.

LiDAR sensors are also extremely vulnerable to outside attacks that lower the quality and reliability of the sensor. One such attack is called spoofing, in which light pulses are sent by an attacker towards the LiDAR sensor before or after the light echo from a real object is received, tricking the sensor into perceiving an object as closer or further than it actually is. Another common attack on LiDAR systems is saturation, in which the sensor is repeatedly sent input at the upper range of what it is able to detect, and because the sensors are not reliable at interpreting information at this upper limit they also become unable to interpret regular input. Because LiDAR is so vulnerable, the researchers sought to propose an alternative that would provide valid information to the car if the LiDAR was incapacitated.

B. Random Sample Consensus (RANSAC)

The RANSAC algorithm was developed by M.A. Fischler and R.C. Bolles to estimate a parameter solution from a set of data with a large number of outliers. The algorithm finds the minimum number of points required to determine the model parameters by choosing a random value as the number of points, solving for the parameter, then determining if the number of inlying points based on this parameter is greater than a predefined minimum threshold, and eliminating these points and retesting until the number of inliers is above the threshold. Unlike other estimation techniques, RANSAC was developed within the field of computer vision, and has been used primarily with LiDAR and other sensor types.

C. Convolutional Neural Networks (CNN)

CNNs are the basis for many image processing and machine learning components of self-driving cars. Rather than simply comparing two whole images, CNNs examine images in parts and match each part together. This process increases the capabilities of computer vision systems to recognize convoluted versions of images that regular image processing systems would miss. CNNs are used in many different kinds of image processing, but they are particular

helpful in autonomous vehicles. In a study done by the NVIDIA Corporation, researchers programmed end-to-end learning from images directly to steering wheel commands by providing the CNN with images to generate a steering command. The proposed command is compared to the desired command, and the weights of the CNN are altered to better predict future desired output. This training process improves robustness when analyzing slightly different images, such as different types of cars, because they are more easily identified to be permutations of the same basic image.

D. Adaptive Cruise Control (ACC) and Cooperative Adaptive Cruise Control (CACC)

ACC has become commonplace in many recent car designs and allows a car to use radar to detect the speed of a preceding car. For example, an ACC vehicle might detect that a preceding vehicle is slowing down. The ACC car registers this and slows itself down accordingly. This feature minimizes the likelihood and damage of a crash between two vehicles. However, ACC is reactive in that it responds to stimuli as they occurring, which may present problems in scenarios where there is not much time to react to a situation.

CACC, however, provides the capability for ACC to be proactive, rather than reactive. CACC cars communicate with one another through beacons on each car. In theory, a preceding CACC car can send messages to other CACC cars behind it to warn that the preceding car will be slowing down, which will cause all other cars behind the first to slow down as well. This provides for an accurate anticipation process, resulting in a smoother ride for passengers. However, CACC is vulnerable to attacks on the beacons of the vehicles they are used on, causing the propagation of incorrect messages to other vehicles, as discussed by M. Amoozadeh et al. in the IEEE Communications Magazine.

Despite the efficacy of many of these different aspects of autonomous vehicles, all suffer from vulnerabilities in some capacity, to the extent that more information is required to enhance autonomous vehicle security.

III SENSORLOG

Sensorlog is an application available for download on the iPhone which collects information from the phone's sensors relating to both the external state of the phone such as pedometer, motion, and location data as well as internal information such as battery life and the IP address of the connected WiFi network. The app records all of this data approximately 16 times per second, and displays it in a comma-separated values (CSV) file to make it easy to access and manipulate.

The researchers were most interested in using the data relating to latitude, longitude, speed, and acceleration in the X, Y, and Z directions collected while the phone was in the car during periods of driving for the purpose of this project. This data was the key source of information in observing the motion of a car while driving, and was used to identify stops along driving routes.

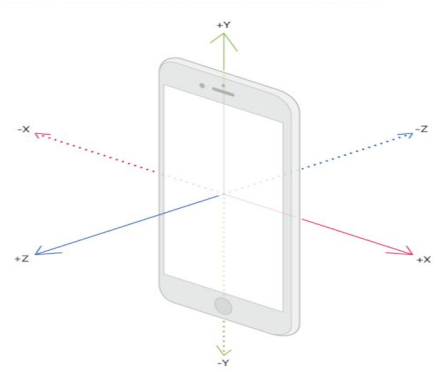

Figure 1: Depiction of the collection of directions from a phone in the SensorLog application

Data was collected from 13 separate drives in the Charlottesville, Virginia area, with some of the routes overlapping each other, while others were unique. The phone was set in the same fixed position during each drive, having the phone laying flat on its back with the top of the phone pointing towards the front of the car. It was important to control for this variable because the acceleration in various directions recorded in SensorLog depends on the orientation of the phone. With the orientation used in this project, the X acceleration corresponded to side to side movement of the car, the Y acceleration corresponded to forward and backward movement of the car, and the Z acceleration corresponded to up and down movement of the car. All the data was then uploaded onto computers in the default CSV format to be analyzed.

IV. APOLLO BY BAIDU

Apollo is an open-source, self-driving car platform available on Github. The system is built on an overall kernel, which either replaces the Linux kernel found embedded in cars, or can be used in the Docker Virtual Machine for simulation purposes. The functionality of the whole car is provided within C++ files which have a wide variety of purposes, from navigation to image sensing. The system allows for the creation of real-world scenarios that an autonomous vehicle would experience while driving.

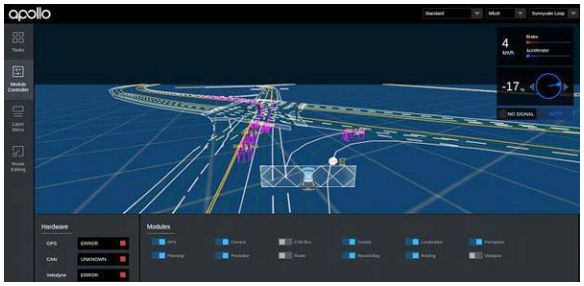

Figure 2: The Apollo Simulation

Multiple versions of Apollo have been released for download on GitHub. In this project, both version 1.0 and 2.0 were experimented with, although they were essentially the same in all of the files that were of significance for the incorporation of the database of stops. The version the researchers focused on was version 1.0 because it had more

demo maps and was slightly simpler to work with. Version 2.5 has been uploaded as the master branch very recently, days before this paper was written, so it was not used during the project.

V. DEVELOPING STOP IDENTIFICATION ALGORITHM

A. Finding Stop Parameters

The stop identification algorithm was developed in Python using data from multiple SensorLog collections. After reading in the SensorLog data, different parameters including speed, acceleration in multiple directions, and rotation were tested to see which coordinates they identified as stops. During development, the data points given as output, meaning they were classified as stops, were pasted into a coordinate visualization application online at a website developed by Darrin Ward which maps coordinates from a CSV file. The results of this mapping process with both the original unrefined stop detection algorithm and the final algorithm are shown in Figure 3.

Figure 3: The initial map of points classified as stops (left) vs. the final map (right).

Using this tool, the stops could be seen on a map and compared to where the researchers knew stops existed in the area. The parameters were refined until all stops along the routes were identified and there were no erroneous stops at locations where no stop existed. .

The parameters that defined a stop were finally classified as points when the car was moving less than 2 m/s and its acceleration in the Y direction (forward and backward) was greater than 0.1 m/s². The speed must be above zero, even though a stop is typically defined as when the car must stop moving, but in reality, drivers do not actually fully stop at all the stop signs they encounter. After testing many values and examining the values in the datasets, greater than 0.1 m/s was found to be a reasonably accurate value for detecting possible stop signs when it was paired with the acceleration value.

B. Filtering Stop Output

Identifying stops using the speed and acceleration introduced a new problem of differentiating between multiple points corresponding to the same stop. These duplicate points would occur when the car stopped at a slightly different point in front of a stop sign, for example, or further back in the line in front of a traffic light. To remove the repeated stops, the output was then filtered by distance from the other points along with order in the SensorLog dataset to only allow for the last of those coordinates in each stopping process to remain in the output. The pattern in the data showed there was a sequence of points leading up to the intersection that were identified as stops. Only the last of those points in the sequence was kept to represent the coordinates of that stop. The filtering left behind an accurate set of stops for each individual drive. However, a similar issue arose when combining all of the stops from each drive into one single database. There were multiple coordinates corresponding to the same stop sign from different drives, differing by only very small points of precision by the GPS system. To filter these out, distance was compared between all stops as they went into the database, and if the coordinates were too close to each other, the new point was not added because it was essentially a duplicate of another stop already represented in the database.

C. Stop Direction Component

After identifying all of the unique stops across all of the routes from the SensorLog data, the researchers decided to add direction into the database, paired with each individual stopping point, so that it would be possible to tell the self-driving car there was a stop approaching only when it was heading in the correct direction that the stop sign would appear. For example, an intersection could contain stop signs on only one of the roads while the cars on the other road could continue driving without worry of having to stop. With the database incorporated into a self-driving car algorithm, it should only be alerted there is a stop ahead at that intersection when it is driving on the road that has the stop signs at the intersection. The direction component added to the stop identification algorithm was calculated in degrees from a SensorLog data point approximately one second before the detected stop. Therefore, the output of the Python program produced a point with latitude and longitude, along with the direction the car was traveling as it encountered the stop.

VI. INCORPORATING STOP DATA

A. MySQL Database

MySQL is an open source, relational database management system that uses the SQL language to store and process data. MySQL also uses a server system which allows the user to access their databases on multiple different machines using a single server account, which made it ideal for a group research project like this one.

The output from the combination of stop information across all drives taken was loaded into a MySQL database. This was done so information will be easily accessible with some commands and new data can be incorporated in the future. If this project were continued, the car in Apollo could add information on stop signs to the database by itself, without having to have someone else load the information into the system. Screenshots of the final MySQL database are included in Figure 4.

```
mysql> select * from stops;
+--------+----------+-----------+-----------+
| number | latitude | longitude | direction |
+--------+----------+-----------+-----------+
|      1 |  38.0357 |  -78.4986 |   69.5196 |
|      2 |  38.0391 |  -78.4969 |    185.46 |
|      3 |  38.0401 |  -78.4963 |   206.058 |
|      4 |  38.0415 |  -78.4975 |         0 |
|      5 |  38.0414 |  -78.4974 |         0 |
|      6 |  38.0417 |  -78.5004 |   131.366 |
|      7 |  38.0455 |  -78.4937 |   244.747 |
|      8 |  38.0527 |  -78.4989 |   141.117 |
|      9 |  38.0544 |  -78.4973 |   223.974 |
|     10 |  38.0357 |  -78.4986 |   106.612 |
```

Figure 4: Stop data in MySQL
*The decimal precision on coordinates and direction are much greater than displayed

B. Understanding Apollo

The first step in trying to integrate the database in Apollo was to understand how Apollo worked internally. Many hours were spent researching the system, looking into the files, and piecing together information about how the files were interconnected to complete the self-driving car algorithm. This research was done until there was a fundamental understanding of which files were used for which general uses. For the most part, the contents of the files were evident from the names of the files and modules, along with some possible documentation. After gaining familiarity with the system, the searching was narrowed to focus on how stop signs and stop lights were being detected and used within Apollo.

There are hundreds of files in Apollo and the researchers were unable to make an initial conclusion of which files to incorporate the data into, so the most efficient way of finding the relevant files was to recursively search through the Apollo files with the “grep” command in the terminal. Searching keywords such as “StopSign” returned all of the files that contained the keywords, so it was possible to focus attention on those files specifically. When examining those files, useful methods and object types were identified, allowing for more searches with the new information as keywords.

C. Integrating the Database in Apollo

After searching through all the files that appeared to be relevant and establishing as many connections as possible between the files, an attempt was made to integrate the database in Apollo. The researchers explored three main possibilities for where to insert the stop data, including the map, reference line, and traffic decider. Each of these modules worked with different types of objects in C++, but the integration relied on the creation of a new stop sign object that could be placed somewhere in the code. Identifying how to do this was a difficult task because of the many variable types relating to stop signs present in Apollo. The types identified included StopSign, StopSignInfo, stop_sign_table, StopSignConfig, and StopSignInfoConstPtr, with multiple types being defined by other types, including types that did not appear to be directly related to stop signs..

Overall, the structure was difficult to understand, especially because there was not much documentation provided. In the map files, the method of most interest was GetStopByID, in which Apollo accessed a stop sign by its unique ID number. Parts of methods in these files also appeared to allow for the addition of stop signs in the map. The second possibility for implementation involved incorporating the database into the reference line, a planning line which contains points of interest along the route. Adding stops on the reference line would give the car foresight that the stop was coming up on the route. This option relates to the third option of adding to the traffic decider. The traffic decider contains many files that are used to handle traffic situations such as a stop sign, stop light, crosswalk, yield sign, or speed limit sign. Within traffic decider, there is a FindNextStopSign method which takes an object of type ReferenceLineInfo as a parameter, meaning it relies on the reference line to determine which stop signs are approaching during the drive. There were also possibilities of adding the database without using that method, but FindNextStopSign showed it was a viable option to add additional stops during this step.

Rather than using traditional constructors to initialize a variable in C++, Apollo uses .proto files to create object definitions. A StopSign variable holds information of two separate ID’s, a Curve, and a StopType. It was challenging to understand how to translate the stops held in the database to a StopSign in Apollo as a StopSign does not hold coordinate information of latitude and longitude. In searching through the files, the researchers could not establish an explicit connection for how a stop sign could be represented by and hold coordinates in Apollo, preventing them from being able to successfully implement the database in any of the options described above. Between the many confusing object types and the lack of a clear method where all the stop signs on the map were inserted into the Apollo system, it was too difficult to find a reasonable solution.

VII. TESTING OPTIONS

To be able to understand the effectiveness of the additional information source in the algorithm, it is necessary to inhibit the car’s ability to perceive obstacles correctly in the Apollo environment. Without the additional information, the Apollo algorithm runs smoothly when it encounters nearly anything. However, a situation in which the database of stops would be needed would be when not everything is functioning properly, or there is a problem in perceiving the stop sign or stop light. With this in mind, a scenario in which the LiDAR sensor is disabled or inhibited would cause the car to function incorrectly and possibly miss detecting stops along the route. Thus, incorporating the database of stops into the Apollo algorithm should improve the performance of the car along the route, as it will know where stops are even if it is unable to detect them. In the sections below, the options for the testing environment are outlined to show how the car could be visualized along its testing route.

A. Use Demo Files

The initial goal for testing was to use the map provided upon download of the Apollo system and simulate a test scenario in the offline version of Apollo. It was assumed that this map was a global map, or at least contained traffic information for many locations across the world, similar to Google Maps, but this was not the case. The provided demo

maps were limited to a small portion of roadways and obstacles, so roadways from outside areas were not supported. It would be necessary to have a separate map or incorporate that information into the current map in order to be able to test on other locations. When looking into the demo files, there are specific points of latitude and longitude that are listed in the document representing start points, stop points, and other points in between along the route. The researchers attempted to modify the coordinates in the file to match up with the relevant Charlottesville coordinates, but were unsuccessful in visualizing this change in the simulator.

B. Creating a Map

After the attempt to modify the demo files did not work as intended, the next option in generating a testing environment was to create an Apollo map. On the Apollo GitHub page, there is a ReadMe document that details specific instructions for creating a new map. It provides commands to enter on the command line, which will allow for new map creation. When the researchers attempted to follow the instructions, they were given an import error. To solve this, they searched keywords from the error in the issues section of the Apollo GitHub page. This issues page was the primary source of information on the Apollo system outside of what was provided in the various ReadMe's throughout the Apollo files.

The resolution for this error was to change the source in the command line so that the imported file could be read. This is not a typical issue that users encounter, but was present in each of the researcher's downloads. After changing the source, the import error was fixed, but another import error emerged which could not be overcome, as it had no documentation and user experimentation could not fix it. The map creation feature was tested on multiple separate downloads of Apollo, but the same errors appeared on each download. This forced an abandonment of of the map creation feature, and a search for a new method of producing a testing environment.

C. Apollo Simulation

The final testing option found was the online Apollo Simulation feature. This is different from the offline, downloadable Apollo system, which was used during this project up to this point. Online simulation was a good testing option for multiple reasons. First, it was much faster than the offline version which ran very slowly on the testing machines, as they did not have the optimal hardware. In addition, it comes with multiple built-in driving scenarios, a pass/fail testing feature, and allows unique GitHub pages containing Apollo files to be used for controlling the car's movement instead of the default Apollo files. he testing feature looked very promising, as it contained nearly 80 tests, which would be classified as passed or failed after the car drove through the scenario environment. It was speculated that the car would fail some of the tests with an inhibited LIDAR feature, given the importance of such sensors as described by M. Jamelska, but would perform measurably better and pass more tests after the incorporation of the database of stops.

Unfortunately, similar to the other testing options, obstacles prevented the use of this option. One problem was that of the over 100 built-in scenarios to Apollo Simulation, none contained stop signs, the main focus of this project. An alternative within the simulation to using the single built-in scenarios is to make your own scenario, which allows for the combination of built-in scenarios using the "Scenario Management" feature. With this feature, it is possible to make more testable scenarios, but there is still no scenario involving a stop sign.

Furthermore, there is no apparent method for incorporating coordinates into the Apollo Simulation, as the platform works differently than the offline version, using specific scenario examples without a normal coordinate layout as opposed to the full maps with more complexity in the offline version. As research was being conducted on Apollo Simulation, it was also determined that in order to use the special testing features, the "Task Management" feature needed to be utilized, and this feature was in private beta, only open for invited partners of certain companies. For all of these reasons, it was concluded that Apollo Simulation was not a viable testing option, either.

Although the testing environment could not be set up properly, the researchers were successful in disabling the LiDAR feature. Thus, if the testing environment worked and the database was implemented correctly, it would have been possible to test the difference in performance with the new algorithm.

VIII. FUTURE WORK

A. Collection of More Stops

Due to practical restraints including the size of the research team and the limited distance that the researchers could realistically travel, data was only collected on a small number of routes, and stops were only recorded in one city. In order to create an accurate database, researchers would need to collect stop data from many different cities and perform these routes multiple times to ensure data is accurate

B. Stop Signs vs. Traffic Lights

At the onset of this project, one of the proposed goals for the database was to determine if stops were simply stop signs or another type of stop; for example, a traffic light. Evaluating stop type would require an extensive quantity of data at each particular stop to collect variety of driving scenarios at that stop. For example, drivers are required to stop at a stop sign under all circumstances, whereas drivers are not required to do this at traffic lights under certain conditions.

C. Incorporation of More Alternative Information

Collecting data on stop signs and traffic lights is one alternative information source that autonomous vehicles can use. There remain many other potential sources of information that could be at a car's disposal. For example, researchers for this project temporarily investigated The Weather Channel API and its potential uses in weather-related driving decisions. Providing additional alternative information sources remains one way to increase autonomous vehicle security.

D. *Usage of Other Autonomous Vehicle Platforms*

While the researchers exhausted many of the options of incorporating their data into the Apollo system, many other systems exist similar to Apollo's that are not open source platforms. For example, NVIDIA DRIVE Developer Program is an alternative platform that has similar capabilities to Apollo in that developers can build additional autonomous vehicle models and test them in the NVIDIA DRIVE platform. More extensive research needs to be done into this system to determine if this platform presents itself as an opportunity to incorporate the researcher's data.

IX. CONCLUSION

After collecting all of the data from phone sensors, detecting stops from the data, and placing the stops into a database, the researchers were unable to determine whether or not the new self-driving car algorithm would be effective in handling stopping situations when sensors cannot correctly identify the road sign ahead of the car. Apollo is a difficult system to work with, especially when accessing it as a normal user because it is lacking good public documentation on many subjects.

Ultimately, the stop database was not successfully incorporated into Apollo. Extensive research was also done on methods for testing if the implementation had passed, although no clear, feasible way to test was identified. Multiple problems arose in various stages throughout the research, preventing the methods for implementation and testing from working as anticipated.

In the future, there is hope that this project can be continued through proper implementation and testing, and improved as more data is gathered on the subject. There is still plenty of room for growth in the stop detection algorithm, as more driving data and more sophisticated analysis tools can be added. This project, despite not being able to produce a concrete answer about the effectiveness of adding the additional information source to self-driving car algorithms, displays the potential and a few ideas for additional information sources to improve the algorithms. The current uses and capabilities of sensors are not powerful enough to keep the cars safe on their own, so this could certainly be a step to enhance future performance.

REFERENCES

[1] Jamelska, M. (2017, August 3). RADAR vs. LIDAR sensors in automotive industry [Web log post]. Retrieved May 3, 2018, from https://mse238blog.stanford.edu/2017/08/mj2017/radar-vs-lidar-sensors-in-automotive-industry/

[2] Petit, J., Stottelaar, B., Feiri, M., & Kargl, F. (2015). Remote Attacks on Automated Vehicles Sensors: Experiments on Camera and LiDAR.

[3] Shin H., Kim D., Kwon Y., Kim Y. (2017) Illusion and Dazzle: Adversarial Optical Channel Exploits Against Lidars for Automotive Applications.

[3] Derpanis, K. G. (2010). *Overview of the RANSAC Algorithm* (Rep.).

[4] Amoozadeh, M. (2015). Security vulnerabilities of connected vehicle streams and their impact on cooperative driving. *IEEE Xplore Digital Library, 53*(6), 126-132. doi:10.1109/MCOM.2015.7120028

[5] Van Arem, B., Van Driel, C., & Visser, R. (2006). The Impact of Cooperative Adaptive Cruise Control on Traffic-Flow Characteristics. *IEEE Xplore Digital Library, 7*(4), 429-436. doi:10.1109/TITS.2006.884615

[6] Bojarski, M. (n.d.). *End to End Learning for Self Driving Cars* [Scholarly project]. Retrieved March 3, 2018, from https://arxiv.org/pdf/1604.07316.pdf

[7] Apple. (n.d.). [Getting Raw Accelerometer Events]. Retrieved March 10, 2018, from https://developer.apple.com/documentation/coremotion/getting_raw_accelerometer_events

[8] Ward, D. (n.d.) Lat/Long Coordinates Map Plotting Tool. Retrieved from https://www.darrinward.com/lat-long/

[9] NVIDIA. (2018, March 26). Software. Retrieved from https://developer.nvidia.com/drive/software