Safer Adaptive Cruise Control for Traffic Wave Dampening

1st Emily Baschab
Department of Physics
University of Alabama
Tucson, United States
efbaschab@crimson.ua.edu

2nd Savannah Ball
Department of Math
Monmouth College
Tucson, United States
sball@monmouthcollege.edu

3rd Audrey Vazzana
Department of Computer Science
Rose-Hulman Institute of Technology
Tucson, United States
avazzana@email.arizona.edu

4th Jonathan Sprinkle
Department of Electrical and Computer Engineering
University of Arizona
Tucson, United States
sprinkjm@arizona.edu

***Abstract*—Our goal is to develop an adaptive cruise controller for vehicles at low speeds in stop-and-go traffic. Current adaptive cruise controllers can use radar sensors to follow a vehicle at high speeds (greater than 18 mph), but reach their limits if the lead vehicle's velocity dips below threshold, requiring the driver of the host vehicle to resume control over the car's speed. Some cruise controllers adapt to stop-and-go traffic, but these are mostly experimental and have yet to see widespread commercial implementation. These experimental models often have issues because of their limited data; consequently, the acceleration and deceleration can be jarring and uncomfortable to passengers. In contrast, because of our reliable sensor data, and the sensor configuration unique to the CAT Vehicle, our cruise controller will be capable of following cars at low speeds and functioning continuously, even when the car is stopped.**

This project has the potential to interest automobile companies who could implement this technology in future automobiles. If our technology were to be implemented in future automobiles, it would make driving considerably more convenient for drivers. This technology could also potentially reduce the number of traffic accidents, as well as making drivers feel safer when navigating traffic. However, if errors were to occur, they could potentially put the car's passengers at risk, as well as the passengers in nearby vehicles.

Our project had a time frame of ten weeks during which we were able to model an adaptive cruise controller and test it in a simulation.

***Index Terms*—Adaptive Cruise Control, Vehicle Autonomy, stop-and-go**

I. INTRODUCTION

Adaptive Cruise Control (ACC) works by taking in data from sensors and running that sensor data through an algorithm that calculates the optimal distance, velocity, and acceleration. Based on these optimal calculations, the host vehicle's computer can determine how much to increase the throttle, and what action the engine of the vehicle should take, simulating that same decision a driver makes in pressing either the throttle or the brake. The sensor data is usually gathered from either a RADAR or a LIDAR sensor that can determine the relative distance of the lead vehicle. From this relative distance, the computer can calculate relative and absolute velocity.

The current state of commercially available cruise control, though far more advanced than in the past, is still surprisingly limited. Adaptive cruise control in modern cars either does not operate under a certain speed threshold (less than 18 mph) or, if it does, operates poorly, causing a jarring experience for the driver and allowing cut-ins from other cars. This behavior stems from the fact that ACC at low speeds is hard to implement safely. In order to mitigate risks, cars brake more abruptly and speed up more conservatively, making for an unpleasant experience in stop-and-go traffic. For this reason, there is much improvement yet to be made in the field of low speed and stop and go cruise control for commercial vehicles.

II. BACKGROUND

A. RADAR and LIDAR Sensor Capabilities

Successful implementation of ACC systems depends critically on automotive components capable of detecting the potential target vehicles ahead of the host vehicle and determining the kinematic characteristics of each target, such as the distance and relative velocity. Detection of targets in most ACC vehicles is done by either Light Detection and Ranging sensor (LIDAR) or the older but still reliable Radio Detection and Ranging sensor (RADAR). While Radar uses radio waves to detect objects and determine range, angle and velocity, Lidar does the same with pulsed laser light. When tested in real-world driving scenarios, no significant differences exist in the capabilities of either detection sensor technology, while operating with the exact same environmental conditions (e.g., traffic patterns, roadway geometry, weather conditions, etc.)

RADAR is unique among sensors in that it is able to calculate relative velocity using the doppler effect, rather than differentiation. Consequently, RADAR calculations of relative

This opportunity is supported by the National Science Foundation under awards IIS-1950359, CNS-1659428. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

velocity are both more efficient and more reliable than those of other types of sensors, including LIDAR. However, the size of the wavelengths required for RADAR technology mean that the maps it produces are very low resolution. RADAR, while more efficient computationally, produces less data and is generally less accurate than other sensor methods such as LIDAR, especially in optimal weather conditions. [1]

LIDAR, on the other hand, while limited in situations with inclement weather and more expensive than most alternatives, is considered among the most high resolution and accurate methods for three dimensional mapping in vehicle autonomy. As this technology has become cheaper, it has become increasingly prevalent in commercial cars. Because of its accuracy, the CAT Vehicle uses LIDAR for terrain mapping and sensing. [2]

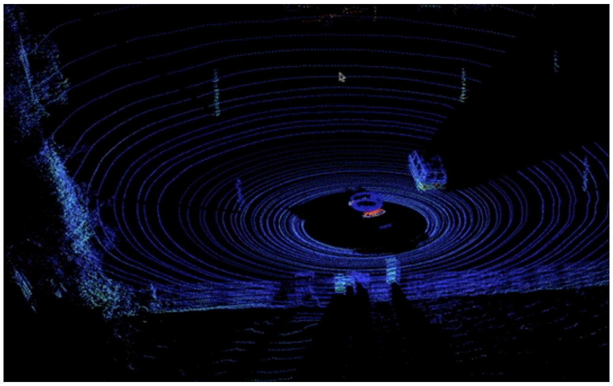


Fig. 1. An image of the velodyne LIDAR sensor map from the CAT Vehicle in a parking lot.

III. THE CAT VEHICLE

Further advances in the field of autonomous driving technology necessitate complete and comprehensive testing of autonomous features. The CAT vehicle testbed at University of Arizona provides a unique integrated simulation and physical platform for design, intensive testing, and real vehicle simulation to effectively verify and validate design ideas in a seamless workflow. It features transfer of controller design from simulated environment to physical platform without rewriting any component of the controller. The CAT Vehicle test bed comprises the Robot Operating System (ROS) based simulator that runs on the ODE physics engine and the virtual environment using the Gazebo simulator to simulate vehicle to vehicle interaction and traffic like situations.

Because the test bed supports code generation using open-source C++ and Python APIs, distributed teams can transfer MATLAB or Simulink generated designs and implement and validate a proof of concept prior to accessing the physical platform. Researchers can then demonstrate results on the physical platform within 2 days. A typical autonomous vehicle setup consists of a vehicle controller and sensors mounted strategically on different substrates on the vehicle. The CAT vehicle testbed contains three types of simulated sensors: Front Laser Rangefinder, Velodyne LIDAR, and two side cameras mounted on the left and right side of the vehicle. The physical platform for the CAT Vehicle Hardware in Loop (HIL) is a Ford Hybrid Escape vehicle mounted with a Front Laser Rangefinder, Velodyne LIDAR, two side cameras and a GPS. [3]

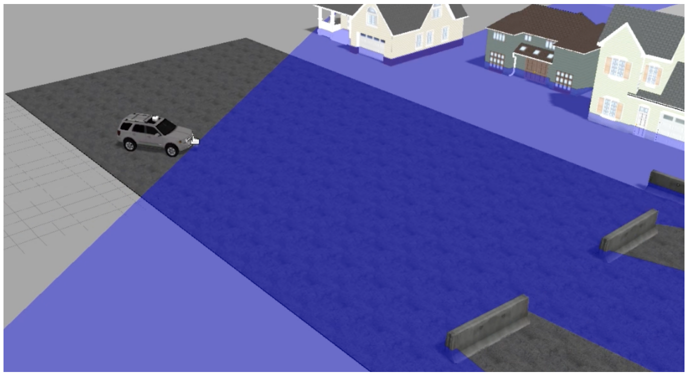


Fig. 2. A simulation of the scope and perception of the CAT Vehicle in the ROS.

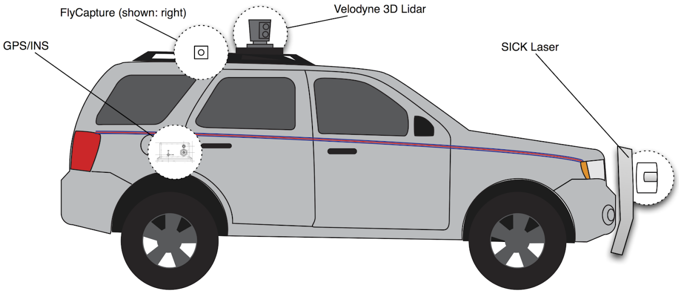


Fig. 3. Diagram of the CAT Vehicle sensor configuration.

Because of our unique sensor configuration on the CAT Vehicle, we are able to overcome the typical sensory limitations encountered in most commercial vehicles. As such, we can test our potential models in real time traffic situations without the danger normally inherent in such situations. This has allowed us to develop a model which can afford to behave less conservatively and which is more conducive to overall passenger satisfaction and safety.

IV. SPACING POLICIES FOR ACC

There are two general categories of spacing policies used in ACC. These are Constant Spacing Policies (CSP) and Variable Spacing Policies (VSP).

Constant Spacing Policies are relatively simple: the desired distance is assigned as a designated constant.

Variable Spacing Policies encompass any algorithm under which desired vehicle distance varies based on external and/or internal inputs. In most VSPs, a quantity known as Time Headway is utilized. Time Headway is defined as the time elapsed before the front bumper of the following car will pass the current position of the front bumper of the lead car at its current speed. [4]

In many models, the idea of using Constant Time Headway (CTH), which means that the velocity is calculated to ensure the Time Headway remains at a constant value, is used.

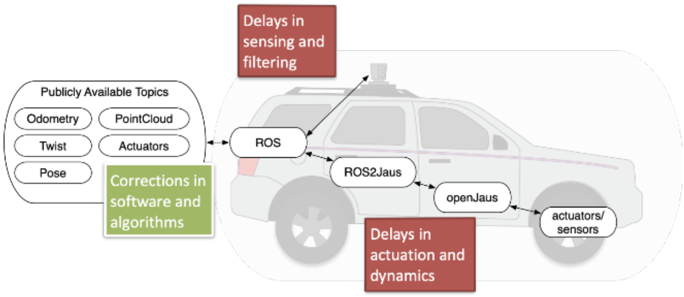
Typical ACC spacing policies tend to work very well at high speeds where margins between cars are high and car behavior is predictable. However, at lower speeds, these techniques tend to result in a degradation in driving quality and experience for passengers.

V. DESIGN DEVELOPMENT

To overcome the limitations of typical ACC, we created a model specifically designed to function at low speeds.

A. *Interpreting and Filtering Inputs*

Upon receiving LIDAR data from the CAT Vehicle sensors, the relative distance, velocity, and acceleration are calculated by a distance estimator program and published to a ROS node



Delays in sensing and filtering
Publicly Available Topics
Odometry
PointCloud
Twist
Actuators
Pose
Corrections in software and algorithms
ROS
ROS2Jaus
openJaus
actuators/ sensors
Delays in actuation and dynamics

Fig. 4. A flowchart showing the progression of data after being processed by the sensors.

The distance estimator iterates through every point received by the laser sensors, determines the angle of the shortest point received, and publishes this distance and angle as the relative distance. It calculates relative velocity by taking the difference in distances over time and acceleration by the difference in velocities. All of these are published to a ROS node.

However, with such a large field of vision, the shortest point does not always reflect the exact position of the lead car. For instance, it might be a car in another lane. Additionally, it fails to distinguish between separate entities in its line of sight. This failure can cause problems when a car from an adjacent lane changes lanes to be in front of the host vehicle. Because the distance estimator registers this car and the previous leader as a single entity, it will return a negative relative velocity. The opposite problem occurs when the current leader switches lanes and the relative distance and velocity both register a drastic increase. Since both relative distance and velocity are crucial to calculating a command velocity for the host vehicle, it is imperative that these values be accurate and that the extraneous values be filtered out.

When a relative velocity of +-10m/s occurs, this value is filtered out and a replacement value is calculated by comparing the current distance value with the previous one and computing a relative velocity based on those two values.

$$d_{rel}(t-s) = \frac{d_{rel}(t) - d_{rel}(t-s)}{s} \tag{1}$$

Where s represents the step size (1/75)

The results of this program are shown in the following figures. The program and code can be found in the Appendix.

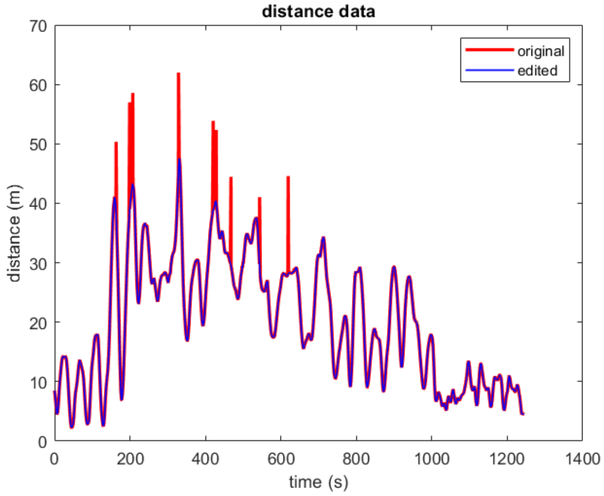


Fig. 5. The difference between the original and modified distance estimator for relative distance.

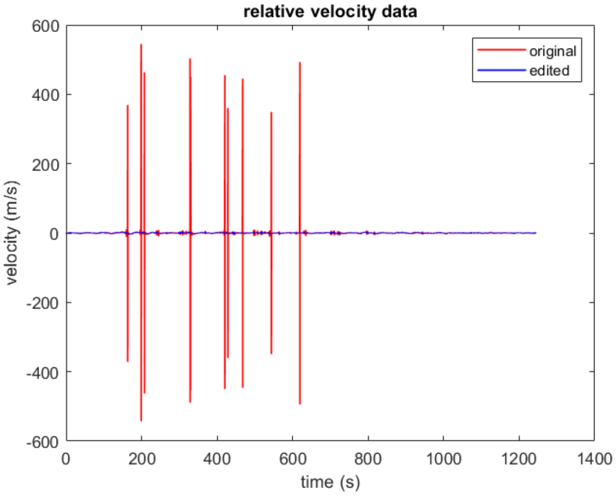


Fig. 6. The difference between the original and modified distance estimator for relative velocity.

VI. CONTROLLER THEORY

In order to follow at an appropriate distance and velocity, we decided to split the domain of our speeds in two. The higher speeds would require the host vehicle to follow at a distance that would increase as the speed of the leading vehicle did. The lower speeds, at which the host vehicle could stop relatively quickly, would require that the host vehicle follow the lead vehicle while maintaining a constant distance far enough away to avoid a crash but close enough not to allow cut-ins. At low speeds, desired distance can be less than one car length (i.e. bumper-to-bumper traffic), but no less than the stopping distance given the velocity of the host vehicle. At higher speeds below the 18mph threshold, where the braking distance at host vehicle velocity was greater than the relative distance between the two vehicles, we used constant time headway to determine our following distance. That is to say, the desired distance varies with velocity, unlike at lower speeds where it remains constant.

Therefore, for speeds below 18mph, there would be two different models, one which governed behavior for traffic where braking distance was less than relative distance and one which governed speeds at which braking distance was negligible (ie less than 5m or one car length). In order to provide a better passenger experience, rather than using the minimum braking distance for this calculation, we wanted to use a realistic braking distance which was calculated as the distance it would take a car to decelerate comfortably to the passenger. However, because of the virtual environment, it was hard to determine exactly what that deceleration would be. In the current model, the minimum braking distance is simply multiplied by a coefficient less than one and greater than zero. However, in the future, we hope to obtain a more exact value for reasonable acceleration.

The conditions to switch between the high speed and low speed model were originally based only on the stopping distance as compared to a constant (5m). However, when relative distance was not taken into consideration to switch, the cars would crash. Therefore, we devised a new idea; the desired distance would be based on constant time headway only when the relative distance was less than the absolute velocity of the host vehicle. When this wasn't the case, which is true for most speeds where braking distance is negligible, the car was instructed to decelerate immediately by setting desired distance to a high constant. This prevented crashes, but left several problems.

First, the sudden change in distance gain might create deceleration values that were uncomfortable. However, testing seemed to indicate this was not a major problem although further testing on the real CAT Vehicle is needed to determine this for certain.

Second, there are some speeds for which the CTHW calculator produces a desired distance that is perhaps lower than desirable. To fix this problem, a saturation block will need to be implemented to enforce a minimum value on desired distance from the controller component.

In the future we hope to adjust the final model presented below to incorporate these changes

VII. FINAL MODEL

The most recent simulink model (figure 7), reflects the theory described above.

First, relative distance, relative velocity, and host velocity data are published from their respective ROS nodes, sourced from the filtered distance estimator program. These values are then fed into the controller, shown below (figure 8).

These values are fed into the distance calculator subsystem which determines the desired distance. Subtracting the desired distance from relative distance yields the quantity Distance Gain: the desired change in relative distance. The Distance Gain is added to the relative distance to get the new command velocity for the vehicle.

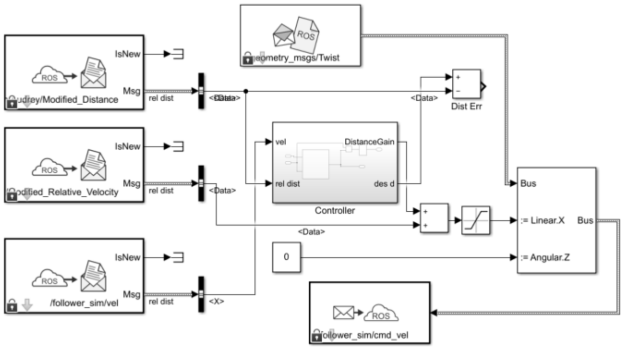


Fig. 7. The final Simulink model.

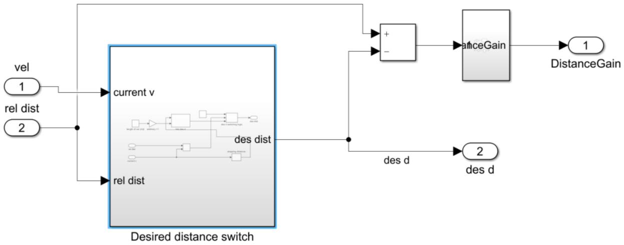


Fig. 8. The controller subsystem.

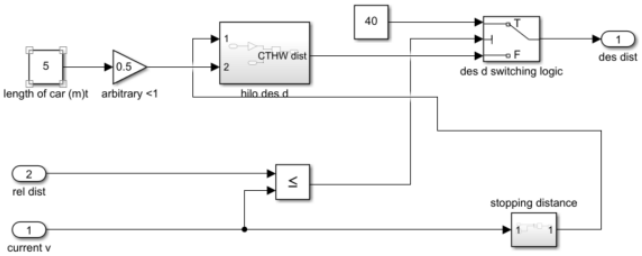


Fig. 9. The desired distance switch subsystem.

The desired distance calculator compares relative distance and host velocity. If host velocity is greater than or equal to relative distance, the desired distance is set to a high constant, causing the vehicle to decelerate. Using trial and error, it was discovered that the value of this constant had an impact on how quickly the vehicle decelerated. Forty meters seems to produce a deceleration which is reasonable for passengers but still safe and high enough to prevent crashes. Further testing is required to determine whether this hypothesis holds. If relative distance is greater than host velocity, a desired distance based on constant time headway is calculated in the following subsystem and output to the system as the desired distance.

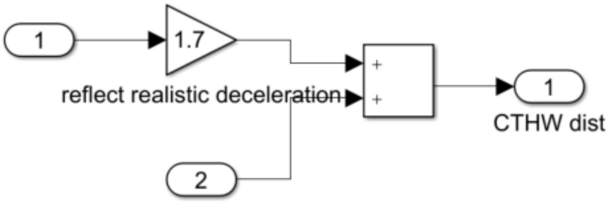


Fig. 10. The CTHW subsystem.

The CTHW distance is calculated here simply by multiplying the minimum braking distance by a gain greater than 1 and adding it to a small constant (half car length). This ensures this distance cannot get any smaller than the specified constant.

VIII. VERIFICATION AND TESTING

To test our data, before implementing it in the CAT Vehicle, we used ROS (Robot Operating System). ROS is ideal for this function because it can take data from a variety of inputs and can output data at any given time in the simulation. To run our ROS simulations, we took data from a bagfile; however, ROS could run data from a simulator like simulink or from real time sensor data, if necessary.

To verify the model, we implemented the distance and relative velocity calculator and the controller for car behavior discussed above into ROS nodes. These two were both originally synthesized in Simulink, making the conversion to ROS node very simple. Running ROS simulations necessitates the creation of several ROS launch files, files which create and run ROS nodes without the need for manual inception. The creation of these files was a crucial step to verify our model's efficacy.

The launch files allow for multiple nodes to be implemented through one command. In these files, topics can be remapped, which allows for different inputs of data without having to regenerate code.

The two main launch files we used were vehicles.launch and 321blastoff.launch. (vehicles.launch) Vehicles.launch placed the vehicles in gazebo, and then 321blastoff.launch allowed for us to test our controller. This launch file contained Audrey's filtereddistanceestimator node and Emily's accmodelros node. The code for each of these files can be found in the Appendices. After executing the launch files, we simulated the behavior of a lead car with data gathered from a previous experiment by executing the file stepvel.launch. This file accessed data from a rosbag file which had been gathered from a previous experiment in stop-and-go traffic and would allow us to test our vehicle under these conditions.

With these files, we were able to determine the areas where our model needed improvement and that our model was behaving safely and could be run on the real CAT vehicle.

IX. RESULTS

The results produced from simulink revealed that the following car would emulate the velocity of the lead car while maintaining an acceptable distance at low speeds. A sample of the car's behavior can be found in the following plot.

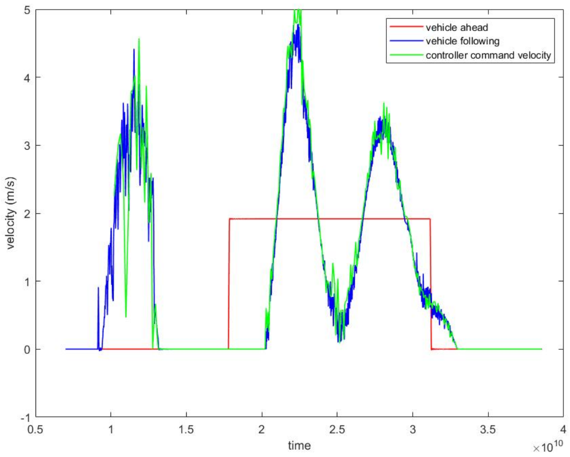


Fig. 11. Plot of the lead and following car behavior from Simulink model data.

These results are corroborated by the results seen in the Gazebo simulation. After placing the vehicles in gazebo, we ran the 321blastoff.launch file and saw the host car move forward to the lead vehicle and stop once it reached the extent of the safest distance to the lead car. We then started the lead vehicle and gave it a constant velocity and once the lead vehicle was far enough away, the host vehicle began accelerating to catch up to the lead vehicle. Once the host vehicle reached the safest distance, it maintained a constant velocity so that it would be following the lead car. They then were both driving the same speed with a constant distance between them for a few minutes. From observation of the simulation in ROS, we noticed that the acceleration of the host vehicle could be improved so that it wasn't as slow and cut-ins were less likely.

X. CHALLENGES AND FUTURE DIRECTIONS

We encountered several unanticipated issues in this project. First, in order to calculate distance gain, we used a PID controller block in Simulink. However, in practice, the numbers being produced by this block were not valid and resulted in the model receiving inputs which were unreasonably high. We temporarily fixed this by adding a saturation block in the controller and making the PID controller from individual components rather than a prescribed block. However, the reason for these high outputs remains unclear, in spite of much debugging. Because we hope that using the PID block will reduce windup error, we will continue looking into why it produces high outputs and how to resolve this issue.

Additionally, the model itself, as discussed in the controller theory section, needs modification to prevent the short distances and hesitancy to accelerate. We hope that real world testing will give more insight into the extent of these issues. Another issue we faced working virtually due to COVID-19 was time constraints. We were unable to test our code on the actual CAT Vehicle. However, before taking that step, it would have been necessary to take additional measures to verify our code. Verification is important because it tests the model in an effort to find errors that may occur so they can then be fixed. Due to lack of time, we didn't get to use S-Taliro, but in the future this tool could help improve our model. To use S-Taliro, a simulink model provides input signals through input ports and is then used as a parameter. The name of the model, along with initial conditions, the constraints, and the MTL specification must be provided to run the tool. Then numerous simulations run, which basically go through every possible outcome that could happen, and it searches for the smallest robustness value. The smaller the value, the closer it is to finding a falsifying trace. If the outcome is a negative value then this means there is a falsification. It will keep searching until it finds an error or until you kill the process. The trace can then be analyzed in the simulink model to find where the problem occurred. For our project, we would want to create parameters for the distance between the vehicles as well as the maximum deceleration of both vehicles. [5] While we did not have the time to write a matlab script, nor did we finalize what our parameters would be, we did investigate what previous researchers had used for their parameters regarding safe distance between the vehicles. Through all of the papers that we examined, we saw that at the core, the distance had to be greater than half the velocity of the first car. [6] We will need to do more research and account for more variables in this parameter, as well as create a parameter for the maximum deceleration of both cars. These two will ensure safety, not only in normal driving conditions, but also for emergency braking situations, as well as comfort. With ACC, a major concern is that passengers will not find it comfortable because it does not mimic a human driver's behavior enough. With these parameters, specifically the one concerning deceleration, we hope to improve this and provide a more enjoyable experience that makes passengers want to use ACC.

With more time, we hope to implement verification, further adjust our model, and test it on the actual CAT Vehicle.

XI. CONCLUSION

In this paper, the potential for a less conservative and more passenger friendly ACC model was explored. In order to create our model, we used two different systems, one dependent on Constant Time Headway to determine desired distance and the other following at a constant predetermined distance. Using this dichotomy, we achieved some successful results. However, we hope to refine the distance setting policy of our model in future works, as well as undergo more thorough testing of our model. In spite of many challenges due to COVID-19, we believe, however, that this model has high potential for even further success in stop and go and low speed traffic.

XII. ACKNOWLEDGMENT

We'd like to thank the National Science Foundation for funding this research project in the summer of 2020. We also want to thank the University of Arizona and the UROC program for their efforts to make this project possible. Finally, we would like to thank the Cognitive Autonomous Textbed REU Program and our faculty mentor Dr. Jonathan Sprinkle for his constant guidance and support on this project. To all involved in these entities, and to Dr. Sprinkle, we'd like to extend our extreme gratitude for making this opportunity possible.

REFERENCES

[1] G. R. Widmann, M. K. Daniels, L. Hamilton, L. Humm, B. Riley, J. K. Schiffmann, D. E. Schnelker, and W. H. Wishon, "Comparison of Lidar-Based and Radar-Based Adaptive Cruise Control Systems Screen reader support enabled," in PSAE TECHNICAL PAPER SERIES, March 2000.

[2] J. Kocic, N. Jovičić, and V. Drndarevic, "Sensors and Sensor Fusion in Autonomous Vehicles," November, 2018.

[3] R. K. Bhadani, J. Sprinkle, and M. Bunting, "The CAT Vehicle Testbed: A Simulator with Hardware in the Loop for Autonomous Vehicle Applications," in 2nd International Workshop on Safe Control of Autonomous Vehicles (SCAV 2018) EPTCS 269, 2018, pp. 32–47.

[4] C. Wu, Z. Xu, Y. Liu, C. Fu, K. Li and M. Hu, "Spacing Policies for Adaptive Cruise Control: A Survey," in IEEE Access, vol. 8, pp. 50149-50162, 2020, doi: 10.1109/ACCESS.2020.2978244.

[5] Y. Annapureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, "S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems,".

[6] M. Hekmatnejad, S. Yaghoubi, A. Dokhanchi, H. B. Amor, A. Shrivastava, L. Karam, and G. Fainekos, "SEncoding and Monitoring Responsibility Sensitive Safety Rules for Automated Vehicles in Signal Temporal Logic," September, 2019, doi: 10.1145/3359986.3361203.