Calculating Relative Velocity for Safety in Autonomous Vehicles

Jonathan Wyatt Pilling, Student, IEEE

***Abstract*—To make autonomous vehicles reliable, we need to make them as safe as possible. We need a full proof way to detect the relative velocity of other vehicles on the roadway. Using computer algorithms is one of the key methods for solving this relative velocity problem. Here, we will look at some of the algorithms used in detecting relative velocity and other methods to make autonomous vehicles safe. We will also see the results and issues with the current safety features that autonomous vehicles use.**

***Index Terms*—automobiles, learning (artificial intelligence), predictive control, learning-based framework, autonomous driving, model predictive control, time-to-contact, self-driving car, relative velocity, Android smartphone, temporal aliasing**

I. Introduction

The prospect of self-driving cars has been a hopeful prediction for years to come. Imagine long road trips simplified with the autonomous vehicle driving for you through the night. This is one of the benefits of autonomous vehicles. Other benefits include safer roadways and better traffic flow. If the autonomous vehicle is perfected, it could aid future generations in efficiency and safety. Autonomous vehicles are vehicles that are driven by a computer. Autonomous vehicles have been a science fiction phenomenon ever since the first automobile was made in the late 1800s.

One of the first autonomous cars were hypothesized in the early 1990s by Carnegie Mellon researcher Dean Pomerleau. This researcher wrote a thesis on using neural networks to have autonomous vehicles take in images from roadways and output steering controls. The use of neural nets proved way more efficient than most other attempts around this time. Autonomous vehicles became more popular once Google became involved in the research and started their first self-driving car project, Waymo, in 2009. By the end of 2017, Google had more than 2 million miles driven by its autonomous vehicles [1].

Autonomous vehicles, as they are now, are insufficient for human roadways. There are a couple present issues in self-driving cars now. One of the most pressing matters with autonomous vehicles is how they detect objects and keep people inside and outside the car safe. For safety and reliability of autonomous vehicles, their systems must use the relative velocity of other vehicles on the road [2]. The relative velocity is the difference between two vectors, where the two vectors represent the velocity of two different objects. This is important to understand in autonomous driving because when cars on the road change speeds, whether they're autonomous or not, your vehicle has to adapt and adjust speed accordingly.

In the current methods for calculating relative velocity, there are some issues. Relative velocity is commonly calculated by taking the time derivative of the distance between two cars. However, any extra data in the measurement will be amplified by taking the time derivative. This can lead to inaccurate values for calculated relative velocity [2].

Some methods are presented to help solve the issues with calculating relative velocity. One method to make self-driving cars safer is to use sensor data and estimate relative velocity. To estimate the relative velocity of a moving object, you can calculate the TTC (time to contact) first. Calculating the TTC is vital in seeing how many seconds you have before a potential collision. Calculating the TTC quickly and accurately can lead to safer and collision-free roadways.

For example, Wang et al. assisted in the programming and deployment of an android phone camera atop a robot-controlled car. This Android smartphone was able to detect and resolve issues by calculating the TTC with the assistance of a Java program [2]. Other versions of this TTC calculation were used in different ways as well. In [3], the authors used a TTC calculation on image data and simulated the data in MATLAB. This TTC calculation was acquired differently than the one in Wang's work; we will discuss these differences later on. The TTC calculation is not the only way to handle calculations of relative velocity though. In [4], Lefevre et al. used autonomous vehicles with radar for relative velocity calculations. Their driving simulations used a learning-based framework to have autonomous vehicles use models to learn from human drivers. Learning from these drivers, the cars were able to use predictive controllers to ensure safety constraints. Then they were able to make correct decisions based on other vehicles relative velocity.

There are a lot of different methodologies in current research to help make autonomous vehicles as safe as possible. One of the most prominent issues in autonomous vehicles is the detection of objects with velocity. Below, will be given experiments and proposed solutions to this dilemma. Highway vehicles move very quickly. Planning on vehicles that are fully

Submitted for review for ECE 3030:Technical Communication & Writing for Engineers on Oct 15, 2018.

Submitted for final review for ECE 3030-Technical Communication & Writing for Engineers on Dec 5, 2018.

J. Wyatt Pilling is currently enrolled at the University of Utah College of Engineering studying Computer Engineering. 201 Presidents Cir, Salt Lake City, UT 84112

Peer reviewed by Ian Hilton, Kyle Price.

autonomous requires fast, efficient algorithms. A processor making a correct decision in a self-driving car could be the difference between life and death. Being able to reliably and accurately calculate relative velocity is imperative. Relative velocity is an important calculation because we need autonomous vehicles to follow safely behind other moving cars. Autonomous vehicles also need to be able to stop, evade or accelerate quickly to avoid catastrophic situations. Self-driving cars must be more than safe and efficient, they must also interact socially with other vehicles on the road [5]. This is an important nuance of the autonomous vehicle.

Although these solutions have assisted tremendously, none of them are perfect. An assortment of these methods suffers from vibration issues when cameras are on top of moving vehicles. These vibrations can distort the images in unexpected ways making it difficult for the computers to compute the data correctly. This can lead to the TTC value being calculated incorrectly and should signal safety concerns. This paper will focus on the methods and results of calculating relative velocity to make autonomous vehicles safe and reliable.

II. Methods

In the TTC Java and MATLAB experiments, the authors both implemented the time to contact algorithm. They used this value to estimate a value for relative velocity. Using this estimated value for the relative velocity they were able to safely determine following distances. The TTC calculation is a simple algebraic expression. Relative velocity is equal to distance multiplied by a constant *C*. This constant is the inverse of the ratio of depth and speed.

$$d_n = r_n C \tag{1}$$

Lefevre et al. used a radar for relative velocity calculations in their experiments. The testing process for all researchers (except for Wang) included simulations and field tests. Wang's tests were performed with a robot-controlled car.

A. Android TTC Java Implementation

The TTC is a numerical value for the time it takes to come in contact with an object. This value can be used, given visual information, to determine when two objects will collide with each other. The following experiment was performed by Wang and his team of authors to implement the TTC algorithm on an Android smartphone with a camera atop a robot-controlled car [2]. Wang et al used a camera coordinate system to help implement the TTC algorithm. Using this camera coordinate system on an x, y, and z-axis, the camera was able to capture motion and depth and track differences between frames. These changing frames help generate movements from image to image. This generated movement is called optical flow. Optical flow is the noticeable movement of objects. Using this movement in between picture frames, the authors were able to figure out how fast objects were moving relative to the robot-controlled car. Given below are some of their experiments.

In [2], Wang et al. were able to implement the TTC algorithm in Java. It's important to note they used a recursive filter to suppress noise. The Java code contained three classes. One class to calculate the TTC using image data, one class to display the image from the smartphone camera. The final Java class was used to show transitional results for the TTC. Samples were taken at 10 frames per second. This means the time interval was 0.1. The number of pixels was decreased from a 720 x 1080 image taken with the smartphone display to a 180 x 270 image. This down-scaled image was utilized as input data. Wang was able to use a low-pass filter to suppress noise and get a more accurate TTC output. Wang used a controllable car to experiment with their implementation of the TTC algorithm. The car was able to move 40 cm/sec, and the screen recording was performed by the Android IDE.

B. TTC Implementation Modeled with MATLAB

The previous methodology used object detection in an attempt to calculate how the velocity of objects was changing. Horn et al. had a similar idea of using the TTC estimate but used accumulated sums of suitable products of image brightness derivatives. These brightness derivatives are the directional change in brightness across an image. This method has no need for object detection, or any other intense processing capabilities [3]. The difference between these methodologies is in these experiments, pictures were taken and then processed later in MATLAB. This meant there was no real-time image processing.

The authors implemented their TTC algorithm and used MATLAB to plot actual values for the TTC and estimated values for the TTC. Experiments were performed with synthetic sequences, stop-motion sequences, and video from a camera mounted atop an automobile. Horn et al. showed that synthetic images can be generated, based on a single image which can be manipulated and changed to represent motion. This was all artificial motion. The main drawback of this method was the motion was underestimated. Sometimes the bias would be upwards of 20%. Bias is the difference between an expected value and an actual value. Using stop-motion image sequences, the authors experimented with these against their TTC algorithm. The authors used pictures of a toy car along a track to represent stop motion. Moving the car back and forth within 1 mm increments, the authors would snap pictures and use these different images to test their algorithm. Finally, the authors were able to test the TTC algorithm atop a moving vehicle driving around Massachusetts. The file recorded from the video contained 600 x 800 images taken at 57 fps. Video sequences were then taken from that file and put into RGB color format [3].

The biggest contrast between these two TTC algorithms presented is that one requires object detection and one does not. Horns method used brightness derivatives to get a value for the TTC, this gives a very direct estimation utilizing only image data. A key benefit of Horn's method is low latency. Downsides of this methodology are that sometimes brightness-gradients will change in regions of non-interest on the images. This results in wasted computing power. A negative of Wang's algorithm is that the estimation of the TTC will not be very accurate unless considerable filtering is used [2][3].

C. Calculating Relative Velocity with Learning-Based Framework

The final experiments we will discuss for calculating relative velocity is to use a learning-based framework from human drivers. This utilized artificial intelligence and used predicted values for the car's acceleration to react appropriately. One of the benefits of this technique is that different end-users have different interpretations of how their autonomous vehicle should handle situations. Every individual has their own interpretation of a "normal" reaction. Because this framework was able to learn from the driver, it can react similarly to how the user might react in similar situations [4].

In their actual implementation as opposed to simulation, relative velocity was calculated with a Delphi ESR radar. Radar uses reflected frequencies to calculate relative velocity. Waveforms are sent out from the radar and reflected back. The time in between the emitting and receiving of the waves is how the radar determines how far away an object is, and how fast it's moving. It's important to note that this radar was only used in the researcher's actual implementation with a car.

Lefevre et al. were able to use a predictive controller which enforced comfort and safety constraints. This controller utilizes the learning-based framework from the model to understand how humans drive. Using the data, the controller was able to output acceleration sequences that they deemed appropriate for human driving. The authors used a combination of Hidden Markov Model and Gaussian Mixture Regression. This method is a non-parametric regression method which surpasses standard acceleration prediction methods [4].

Each driving model had to go through a training phase. During this phase, the data was utilized to assist in controlling the motor vehicle. This driver model created a normal, or Gaussian distribution from the training data. It then used an Expectation-Maximization algorithm to find parameters, number of hidden modes, the initial distribution, the mean and covariance matrix, and the transition probabilities between hidden modes. Using the Gaussian Regression, the model was able to compute the current acceleration of the car. The model then provided the controller with a reference acceleration sequence. This looks ahead to the vehicle in front of it and tells itself how fast it can go [4].

D. Framework Experiments

The controller was used to ensure safety constraints. This meant to keep safe distances behind to allow the vehicle to stop in case of emergency situations. The authors used a Hyundai Azera with a DELPHI ESR radar. This radar provided the distance and velocity of the preceding vehicle. Computations were performed on a MicroAutoBox II, which consisted of a processor clocked at 900 MHz. Acceleration inputs were utilized in the adaptive cruise control system. This autonomous driving test was performed under two main conditions. One condition was evaluating the performance of the driver modeling. This condition helped determine how well the model could adapt to different driving styles. The other condition the authors tested were for when the data was insufficient for reliable driver training. This led to testing how the controller was able to take over and handle the situation, keeping the driver safe [4].

The authors collected driving data from five human drivers. Each driver drove for 60 min in different levels of traffic. Each of these drivers was assigned two models. A personalized model, (which used the data from the individual driver), and also an average model, (which utilized the data from the four other drivers.) Using the data from the training, the vehicle may experience some issues if it deals with a situation that it hasn't experienced before. The authors address this by having a confidence value, this value indicates how well the model will perform at time *t*. The authors used the personalized model for the first driver in a heavily congested traffic situation. Because the model has data on traffic situations it's able to handle the traffic naturally. In another example, the authors used the personal model again but for a driving situation that had never been encountered in the data set. The scenario was as follows: The autonomous vehicle was following behind a car keeping a safe following distance, then the authors had the car disappear abruptly. This created an unnatural situation that the training had no idea how to deal with [4].

In summary, the methodologies behind all these experiments differed. Wang used the TTC algorithm on top of a robot-controlled car. This gave real-time TTC calculations that would display on the smartphone screen. Horn's methodology also used a camera atop a car, but this was only used to snap pictures and later process the images through MATLAB. Using the images, Horn then compared algorithm outputs of the TTC and actual values of the TTC. This gave no real-time processing or safety constraints for the vehicle attached with the camera. Lefevre's method was to implement a learning framework to learn from human drivers and then use predictive modeling to control autonomous vehicle speeds relative to other cars on the road, and the speed limit. Relative velocity was accurately calculated with a Delphi ESR radar.

III. Results/Discussion

A. Android TTC Car

In [2], the car with the Android phone running the TTC algorithm was able to accurately determine dangerous and safe conditions. These results were available in real-time with little latency. If the TTC was calculated to be a positive number, then the HUD would show red bars on the right-hand side of the screen to indicate danger. This meant that the car was coming close to the object and the time to contact was getting smaller. If the TTC was calculated to be negative, then the HUD would display green bars to indicate safety, this meant the car was moving away from the object, or the distance between the two objects was getting larger. Pictured in Figure 1 and 2, you can see the display from the smart-phone camera. In this display notice the boxes with the numbers. This is the object that the car is trying to be wary of. In the top left-hand corner on both figures, you can see how the camera interprets the image in different colors for object detection. On the right side of the image, there are some bars, green in Fig. 1, indicates a safe following distance/condition. Red bars on Fig. 2, indicates a dangerous condition.

Using the inverse of the TTC, the authors were able to calculate relative velocity by multiplying distance acquired by the inverse of the estimated TTC. This was done in the phones

Java program. It’s important to note that this TTC value can be very erratic unless filtering is used appropriately. One issue with these calculated TTC values was when the camera would vibrate during the recording process. These vibrations caused distorted images between frames, causing inaccurate calculations of the TTC [2]. In a real situation with a real vehicle instead of a robot car, a miscalculation on a crucial value like the TTC could lead to a catastrophic situation.

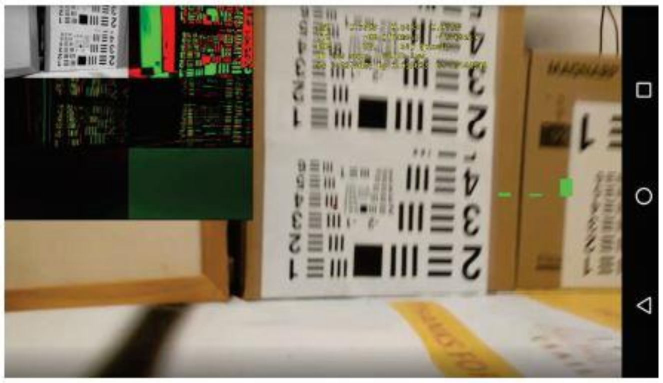

Figure 1. Green bars, negative TTC

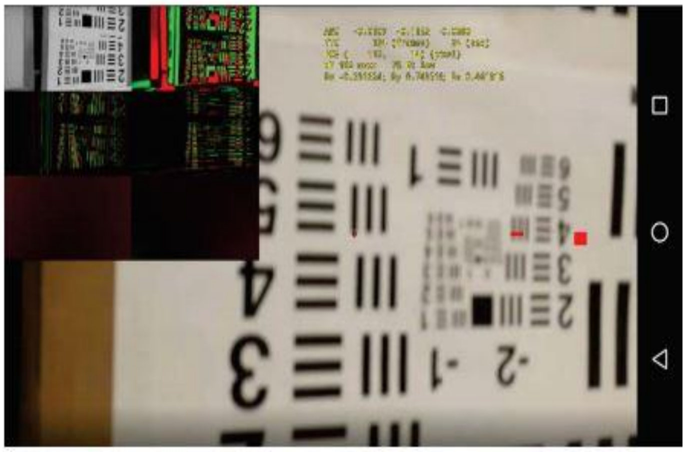

Figure 2. Red bar, positive TTC

B. TTC MATLAB Results

In [3], Horn et al. used three different experiments to test their TTC algorithm. Image manipulation, stop images, and video feed from a vehicle. In the image manipulation experiment, where they changed an image to simulate motion, the calculated values of the TTC vs. the real values of the TTC followed similarly and linearly. When the TTC value was very large, there was a bias in the estimate, upwards of 20%. Near the end of the simulation, the results were unreliable. One of the reasons for these unreliable results was temporal aliasing. This meant that the frame to frame change was too large. When the images were changing drastically in-between frames, this resulted in some confusion from the computer. For the stop motion sequences, the values of the estimated TTCs were larger than actual values for the TTC. When the toy car was far away, it was in the background of the image. This accuracy was less than that of the synthetic image. The authors state this was because it was difficult to position the toy car by hand [3]. The final experiments using the video feed from the vehicle showed that the TTC was very accurate in the author’s manual estimations. The authors also state that the estimations could be better if masking or segmentation is used on the image. The outstanding feature of these results is that they are accurate and you don’t need any motion tracking or optical flow detection equipment. The authors state you could use spatial averaging and sub-sampling to get better values when the TTC is really small [3].

Refer to fig. 3 for the MATLAB plot of the video feed data. The red dots represent the value of the TTC from the algorithm. The green dots represent the manually estimated TTC. This figure only corresponds to the actual car implementation. Along the x-axis of the graph, the frame number is given. Along the y-axis, the value of the TTC is given.

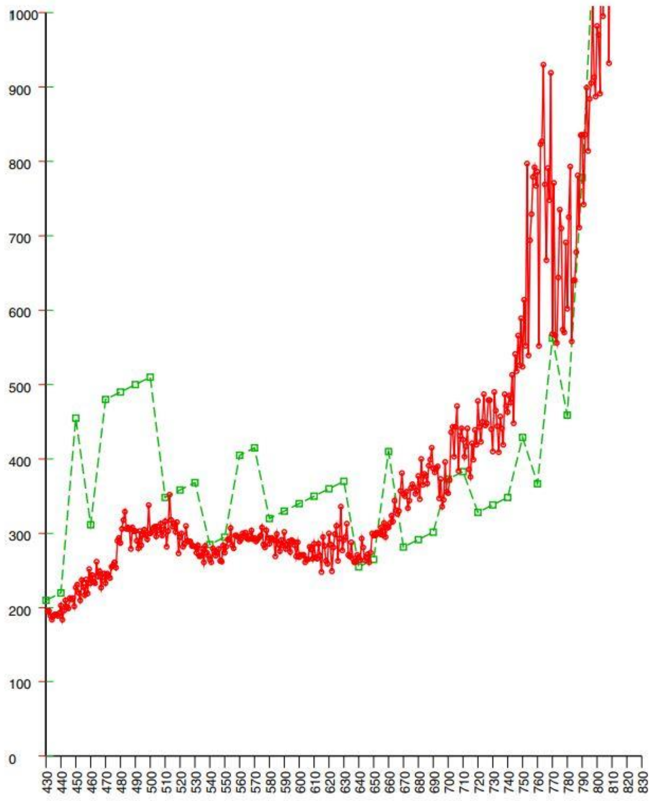

Figure 3. Plot of TTC values for video feed experiment

C. Framework Results

In [4], when the personalized model of the first driver was used, the authors created a situation that the model did not know how to handle. The initial conditions were set as the preceding vehicle was 10 m ahead of the autonomous vehicle. This car was traveling at 5 m/s. Acceleration references were generated normally and the autonomous vehicle followed behind at a safe distance. The preceding vehicle then sped up a little bit, the autonomous vehicle acted appropriately and followed behind at a safe distance. The autonomous vehicle always went as fast as it could while maintaining a safe following distance. This is where the unfamiliar situation was created. The virtual vehicle was removed from the scenario. When this happened, it took five-time units for the controller to figure out how to handle this situation. The situation being it was going well under the speed limit for no apparent reason. After the five-time units, the vehicle accelerated back up to its appropriate speed. Then the vehicle was reintroduced back in front of the vehicle. This created a similar situation again that the autonomous vehicle knew how to deal with. The accelerations were then generated and the car was a safe following distance from the vehicle again [4].

The behavior of this system shows that it can handle all kinds of different situations. The relative velocity of the preceding vehicles was calculated accurately in all simulations. The downsides to this system are that some human drivers may have a driving style that they do not want the autonomous vehicle to replicate. If an individual was an angry driver, then they may not want their vehicle behaving similarly in infuriating situations on the roadway [4].

D. Discussion

In the analysis of the methods and results, we have shown that combining sensors will be the way to make autonomous vehicles safe. We have demonstrated this by showing that each of these sensors suffers from at least one weakness. Combining the strengths of these sensors will provide the most accurate relative velocity calculation. In [2], the car suffered from vibration issues. In [3], the image processing would suffer if the changing images were too different. On the contrary, radars do not suffer from vibration issues. Combining the outputs from the TTC camera and a radar would provide a guarantee on estimations. Radar cannot be solely used for relative velocity calculations. Radars are not 100% accurate if there are a lot of objects around. This can cause interference in the reflected waves. In the future, Wang et al. hope to use the smartphones gyroscope to counteract the vibration of the images. Horn et al. hope to subsample regions of interest in their image processing. This would lead to less wasting of computing power.

In this research, we've only covered two main solutions to the relative velocity problem. Using the TTC and calculating relative velocity with radar. The two TTC methods for calculating relative velocity were different in the way they processed the images. Using the brightness-gradient method, if it's refined, could lead to the best image processing capabilities because of its low latency.

Combining all these sensors and more could hypothetically be the definite way to calculate relative velocity. Using multiple sensors gives the autonomous vehicle more information to work with when trying to navigate roadways. Although more sensors are usually better, there comes the issue of autonomous vehicles being able to interpret all of this information synchronously. Especially in commercial autonomous vehicles, if you have multiple radars and multiple cameras, these computations could become taxing on processors.

IV. Conclusion

If we successfully figure out all the safety concerns of self-driving cars, we will have long road trips through the night where we may not have to be behind the wheel. Self-driving cars must be able to react and respond to all different kinds of road conditions [6]. Attaining full autonomy in motor vehicles could also save many lives. Each year more than 1 million people die in traffic accidents [7]. Solving the relative velocity problem will be immensely important for future generations.

References

[1] L. Dormehl. "10 Major Milestones in the History of Self-Driving Cars" Internet: https://www.digitaltrends.com/cars/history-of-self-driving-cars-milestones [Oct. 15, 2018]

[2] L. Wang and B. K. P. Horn, "Time-to-Contact control for safety and reliability of self-driving cars," *2017 International Smart Cities Conference (ISC2)*, Wuxi, 2017, pp. 1-4. doi: 10.1109/ISC2.2017.8090789

[3] B. K. P. Horn, Y. Fang and I. Masaki, "Time to Contact Relative to a Planar Surface," *2007 IEEE Intelligent Vehicles Symposium*, Istanbul, 2007, pp. 68-74. doi: 10.1109/IVS.2007.4290093

[4] S. Lefèvre, A. Carvalho and F. Borrelli, "A Learning-Based Framework for Velocity Control in Autonomous Driving," in *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 1, pp. 32-42, Jan. 2016. doi: 10.1109/TASE.2015.2498192

[5] B. Brown, "The Social Life of Autonomous Cars," in *Computer*, vol. 50, no. 2, pp. 92-96, Feb. 2017. doi: 10.1109/MC.2017.59

[6] Y. Seo, J. Lee, W. Zhang and D. Wettergreen, "Recognition of Highway Workzones for Reliable Autonomous Driving," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 708-718, April 2015. doi: 10.1109/TITS.2014.2335535

[7] C. Urmson and W. ". Whittaker, "Self-Driving Cars and the Urban Challenge," in *IEEE Intelligent Systems*, vol. 23, no. 2, pp. 66-68, March-April 2008. doi: 10.1109/MIS.2008.34

Jonathan Wyatt Pilling was born in Salt Lake City, Utah, in 1995. He's currently working on his B.S. in computer engineering from the University of Utah located in Salt Lake City.

From 2015 to 2016 he worked as a Computer Lab Aide for Salt Lake Community College. Since the beginning of 2016, he's been employed at the Huntsman Cancer Institute as a Patient Transporter/Information Desk Assistant located in Salt Lake City. He's also currently working in the electrical engineering field as an Intern for Colmek located in Murray, Utah.

Mr. Pilling's awards include Hack the U 2016 – 1st Place (Hack to Save Homeless Pets) and Dean's List 2016 awarded from Salt Lake Community College.