Calculating Relative Velocity for Safety in Autonomous Vehicles

Jonathan Wyatt Pilling, Student, IEEE

***Abstract*—Most people know now that autonomous vehicles are the motor vehicle of the near future. To make autonomous vehicles fully reliable, we need to make them as safe as possible. We need a reliable way to detect relative velocity of other vehicles on the roadway. Using computer algorithms is one of the key methods to 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 also see the results and the 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

Self-driving cars have been a prediction and hope of the future for years to come. Imagine long road trips simplified with the autonomous vehicle driving for you through the night. This is just one of the many potential 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 cars are simply cars with the ability to drive themselves. They're programmed with an artificial intelligence system that basically acts like a human driver and drives the vehicle without needing user input. Autonomous vehicles have been a science fiction dream 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 issues 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 relative velocity of other vehicles on the road [2]. The relative velocity is the difference between two vectors, where the two vectors represent the force 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.

Some methods are presented to help solve the issues with calculating relative velocity. One method to make self-driving cars safer is to utilize sensor data and make estimates on the relative velocity. To calculate the relative velocity of a moving object you can calculate the TTC (time to contact). 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 collision free roadways.

For example, Wang et al. assisted in the programing and deployment of an android phone camera on a robot-controlled car. This Android smart phone 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], the researchers used a learning-based framework to have autonomous vehicles use models to learn from actual human drivers. Learning from these drivers the cars were able use predictive controllers to ensure safety constraints, and make correct decisions based on other vehicles relative velocity.

However, although these solutions have assisted tremendously. None of these solutions are perfect. An assortment of these methods suffers from vibration issues when cameras are a top moving vehicle. These vibrations can distort the images in unexpected ways making it difficult for the algorithms to interpret the data correctly. This can lead into 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.

Submitted for review for ECE 3030:Technical Communication & Writing for Engineers on Oct 15, 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 Editors information here)

II. METHODS

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 the authors give their experiments and proposed solutions to this hurdle. Highway vehicles move very quickly, and rightly so, you need very fast algorithms to make snap judgement calls and decisions for you if you plan on relying on full autonomy. An algorithm making a correct decision in a self-driving car could be the difference between life or death. Being able to reliably 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. Until all vehicles on the road are autonomous, we need autonomous vehicles to interact well with other human drivers. One of the methods discussed below to calculate relative velocity is to compute the time to contact (TTC). This value can be used, utilizing 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 smart phone with camera a top a robot-controlled car [2].

A. Android TTC Java Implementation

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 variables 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. Given below are some of their experiments using the TTC algorithm.

In [2], Wang and his authors 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 were 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 [2].

This methodology utilized object detection in an attempt to calculate how the velocity of objects were changing. Horn et al. had another idea for calculating TTC using accumulated sums of suitable products of image brightness derivatives. This method has no need for object detection, or any other intense processing capabilities [3].

B. TTC Implementation Modeled with MATLAB

The authors implemented this algorithm and used MATLAB to plot results and estimates. Experiments were performed with synthetic sequences, stop motion sequences, and video from a camera mounted atop an automobile. In [3], Horn et al. showed that synthetic images can be generated, based on a single image which can be manipulated and changed to represent motion. Motion was underestimated using this method. Sometimes the bias would be upwards of 20%. Using stop motion image sequences, the authors experimented with these against their TTC algorithm. The authors used taken 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 against the algorithm. Finally, the authors were able to test the TTC algorithm atop a moving vehicle 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 gradients to get a value for the TTC, this gives a very direct estimation utilizing only image data. Wangs work needed an optical flow sensor, or radar, to reliably compute the TTC for their robot car implementation. One of the key benefits of Horn's method is low latency. One of the downsides of this methodology is that sometimes brightness-gradients will change in regions of non-interest on the image. This can result in a waste of computing power. One of the downsides of Wangs algorithm is that the estimation of the TTC will not be very accurate unless considerable filtering is used [2][3].

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 on 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 how the user might react in similar situations [4].

C. Calculating Relative Velocity with Learning-Based Framework

Lefevre et al. were able to use a predictive controller which enforces sets of 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].

The 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 brake in case of emergency situations. The authors used a Hyundai Azera with a DELPHI ESR radar. This radar provided 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 were 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. The authors used these two models for comparison and contrast. Utilizing 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. The results are discussed below.

The methodologies behind all these experiments were all very different. Wang used the TTC algorithm a top a robot-controlled car. This gave real time TTC calculations that would display on the smartphone screen. Horn's methodology also used a camera a top a car, but this was really only to snap pictures and then later process the images through MATLAB, to compare and contrast estimated values of TTC vs. real values of 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.

III. Results

The different authors above all did different experiments and different research to solve the relative velocity problem in autonomous vehicles. When you have different methodologies, you have different results. Each author came to different conclusions on how well their algorithm or method performed. A lot of the authors also talked about what their future research would entail, how they would make their solution better in the future. Below are the results the authors accumulated thus far.

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. This was able to update real-time with little latency. If the TTC was calculated to be a positive number, then the HUD would show red bars 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. This is pictured in Fig. 1 and 2.

Using the inverse of the TTC in calculation, the authors were able to calculate relative velocity by multiplying distance by inverse of the estimated TTC. 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, 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. The bias was upwards of 20% sometimes. 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. 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, "mostly because of the difficulty of accurately positioning 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 are 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 TTC from algorithm. The green dots represent the manually estimated TTC.

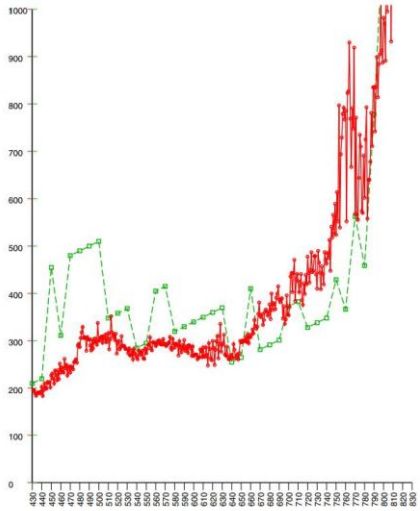

Figure 3. Plot of TTC values for video feed experiment

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 follows 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 staying 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 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 down sides 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 the same as them in infuriating situations on the roadway [4].

IV. Conclusion

In [2], the authors computed the TTC with one camera and one radar. This methodology was accurate in calculating correct values for the TTC as shown in the results section. In [3], the authors calculated the value of the TTC using brightness gradients. This methodology may be better because you only need image data. The results from brightness

gradient calculations were under-estimated a lot of the time but they were still accurate. On the other hand, in [4], they had whole different approach to handle calculations of relative velocity. The authors here used a predictive model to simulate autonomous vehicles.

We cannot explain what the best method for determining relative velocity is in autonomous vehicles. To our surprise, all of these methods work well with each having its own minor issues. Both implementations of the TTC algorithm suffer from vibrations from the camera when positioned on top of a motor vehicle. All of the implementations need to use filtering. The learning-based framework solution only suffers personal driver modeling issues.

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 so many lives. Each year more than 1 million people die in traffic accidents [7]. Solving the relative velocity problem will be the key to the future.

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.