AN ADVANCED DRIVER-ASSISTANCE SYSTEM: USING MONO-CAMERA VISION-BASED DETECTION AND TRACKING FOR CONTROL OF AN AUTONOMOUS VEHICLE

A Thesis

by

WILLIAM JAMES TAYLOR VI

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Sivakumar Rathinam
Co-Chair of Committee,	Swaminathan Gopalswamy
Committee Member,	Alireza Talebpour
Head of Department,	Andreas Polycarpou

December 2018

Major Subject: Mechanical Engineering

ABSTRACT

Advanced driver assistance systems used to increase the safety of everyday transportation are increasingly common in today's automotive industry. In conducting research for the Texas Department of Transportation's Commercial Truck Platooning Project, an autonomous truck capable of communicating with a manually driven lead vehicle can follow at close distances to increase the fuel economy and safety of both vehicles. In the event of a communications failure, however, the following vehicle requires immediate human intervention to regain control of the vehicle. To increase the safety of these situations, a backup vision-based advanced driver assistance system comprised of a separate camera and computer was developed and successfully implemented in an autonomous vehicle.

The system is comprised of a controller that takes inputs from a truck detection algorithm and a tracking algorithm that run in parallel at all times. The detection algorithm passes new detection information to the tracking algorithm when available. In each frame of incoming video, the tracking algorithm communicates the position of the lead vehicle to the controller. Based upon the size of the detection and the intrinsic properties of the camera, the longitudinal distance to the lead vehicle and the lateral error are calculated.

The detection, tracking, and control algorithms were implemented in an autonomous car. A variety of lane changes and emergency stop maneuvers were successfully completed. GPS position information was gathered during testing to validate the accuracy of the algorithms. The results of the GPS information show that the tracking algorithm can encounter some error in estimating the separation at large following distances of more than 30 meters. Overall, the single camera system is capable of providing smooth and safe control in a variety of test scenarios.

DEDICATION

I'd like to dedicate this thesis to my family and friends who have supported me in my educational aspirations. I especially thank my parents, William and Jan Taylor for their continuous assistance throughout my life. I hope that this culmination of my education thus far reflects the encouragement and unfailing trust in me that you so graciously provided.

To my siblings, thank you for sharing in my educational aspirations. Robby, your pursuit of a Ph.D. so early into your graduate career inspired me to follow in your footsteps. Michael, I hope that your ability to so gracefully balance work, fun, and your education while pursuing medical school can shine as an example to your many peers. Daniel, I hope that your first year of college is fruitful. Your determination to shine as an individual while following in the footsteps of your three older brothers is uplifting. Ella and Ben, thank you for sharing in your brothers' love for education. I look forward to helping you in the future when you eventually pursue your engineering degrees.

I would also like to thank my extended family and friends. Uncle Tom, thank you for consistently pressuring me to better my understanding and use of the English language. To my grandparents, thank you for granting our families the opportunity to pursue education. Your hard work and continuing desire to learn has fruitfully overflown both directly and indirectly to your many grandchildren. To Dr. Foust, thank you for believing in me and providing me the educational tools and instruction that launched me into my career at Texas A&M.

Finally, I'd like to thank my beautiful bride-to-be, Allie. If it weren't for your constant support, attention, and advice, I would have never made it to this day. I cannot wait to spend the rest of my life with you, explaining how things work all along the way.

ACKNOWLEDGEMENTS

I would like to thank my committee members who guided me along my educational journey. Dr Rathinam, thank you for showing me the excitement with which you approach new problems. Your continuing desire to learn in working with new technology is inspiring. Dr Swami, thank you for your guidance and support of our many projects. The structure and viewpoint that you bring to most situations had a profound impact on the way that I interpret the design and implementation of new and current processes. Dr. Talebpour, thank you for agreeing to serve as my committee member. Your passion for developing connected vehicle technology has a profound impact on many students at Texas A&M. I'd also like to thank the many professors, advisors, and teacher's assistants that have helped me in my educational journey. Without your instruction and support, I don't know where I'd be today.

CONTRIBUTORS AND FUNDING SOURCES

This work was made possible through the support of my advisory committee consisting of Co-Chairs Dr. Sivakumar Rathinam and Dr. Swaminathan Gopalswamy of the Department of Mechanical Engineering at Texas A&M along with Dr. Alireza Talebpour of the Department of Civil Engineering at Texas A&M. The work was conducted in conjunction with my peers Andy Hwang, Vamsi Vegamoor, Yuntao Zhang, and Mengke Liu on the TxDOT Commercial Truck Platooning Project. The research was funded by the Texas Department of Transportation project number 0-6836 and implemented through the Texas A&M Transportation Institute (TTI). A special thanks to Mike Lukuc, the program manager at TTI who allowed our group to work on the project.

The development and implementation of the lateral and longitudinal controllers was made possible through the help of Kenny Chour. The use of portable GPS units for algorithm validation was made possible through the help of Abhishek Nayak. All other research was completed independently by myself.

NOMENCLATURE

Platooning	Vehicles following at close distance on a highway to reduce drag, improve fuel economy, and improve highway throughput
Lead Vehicle	(LV) The front vehicle in a platoon, driven manually by a driver
Following Vehicle	(FV) The semi-autonomous vehicle following the lead vehicle. In this research, the following vehicle refers to both a truck and a car
Truck(s)	The two commercial 18-wheeler trucks used for truck platooning research. A manual lead vehicle and a semi-autonomous following vehicle
Car	The drive-by-wire enabled Lincoln MKZ sedan used for the implementation of our tracking, detection, and control algorithms
Frame	An image in a sequence of images being passed to a vision algorithm
Template	An image containing only the last detected truck, used in the tracking algorithm
Tracking	Searching for likeness of a pre-defined image in the current frame
Detection	Independent re-definition of the template
TxDOT	Texas Department of Transportation
TTI	Texas A&M Transportation Institute
ROI	Region of interest: The cropped region of an image used for processing
KCF	Kernelized Correlation Filter: A high-speed, accurate tracking algorithm investigated for use in our research

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

1. INTRODUCTION AND BACKGROUND

This paper introduces a truck detection and tracking algorithm developed for use with the Commercial Truck Platooning Project that is currently in development at Texas A&M. The research is funded by the Texas Department of Transportation and is intended to aid in the commercialization of truck platooning capabilities. The aim of this research is to provide a means for commercial trucks to follow each other at close distances on long stretches of Texas highways. This reduced following distance has been shown to increase fuel efficiency for both vehicles by up to 6.4%.[1]

Autonomous capabilities for the following vehicle(s) can improve the overall safety of this idea, as it removes the inherent added danger of driver reaction time in emergency situations. By adding autonomous capabilities to existing commercial trucks, the drivers will have the ability to communicate with each other and join a platoon. Upon joining the platoon, the lead vehicle will still be driven manually, while the following vehicle(s) will switch to autonomous mode and begin following at close distances. In the event of sudden braking, the following vehicle(s) can brake along with the lead vehicle, due to the real time communication and autonomous control that this technology makes available. This process thereby increases safety for the drivers and other cars on the road.

In this area of research, a multitude of sensors can be utilized to improve the safety and overall reliability of the autonomous platooning system. Currently, the Commercial Truck Platooning Project utilizes two 18-wheeler trucks for the research, one of which is equipped with fully

[1] *Assessing the Fuel-Saving Potential of Semi-automated Truck Platooning*, NREL/FS-5400-64133, June 2015

autonomous capabilities. The trucks utilize GPS, RADAR, and other hardware components that allow them to communicate valuable information to each other via Designated Short-Range Communication (DSRC). Without GPS or DSRC capabilities, the following vehicle is unable to successfully follow the lead truck, as it has lost all the required position and path planning data necessary to operate.

Although there are many safety protocols in place, including halting all autonomous functionality in the event of a loss of GPS or DSRC connection, there is no backup plan for the autonomous vehicle in the event that the driver is not fully attentive and capable of instantly reclaiming manual control. To improve the safety of this situation, it was proposed that a single-camera vision-based detection and tracking algorithm be developed for use as an advanced driver assistance system (ADAS) in these emergency situations.

1.1 MONO-CAMERA VISION BASED SAFETY SYSTEM

Our algorithm was developed to be a backup system that constantly runs in the background and does not interfere with the existing baseline functionality of the truck platooning system. As the cost of cameras and small computers has declined in recent years, the feasibility of introducing camera technology into autonomous systems has correspondingly increased. This secondary vision-based safety system can be implemented using relatively low-cost components. In addition to the cost, the developed system is stand-alone, made up of a computer and a single camera. This allows it to be easily added to the existing platooning system by passing commands directly to the controller.

The decision to use a secondary, stand-alone system to develop the detection and tracking algorithm was made for a number of reasons. By developing the system separate of the existing platform, it was guaranteed that there would be no interference with the current platooning system; all that was required for the ADAS was a live video feed. As a result, a great deal of video was taken while platooning to aid in the development of the algorithm.

Additionally, it was assumed that previously developed, currently available tracking algorithms could be implemented and tested on the pre-recorded video. Based upon the performance of these tracking algorithms, we would be able to implement them directly, eliminating the need to develop a tracking algorithm. In any case, however, there was a need to develop a truck detection algorithm.

1.2 VISION-BASED TRUCK TRACKING AND DETECTION LITERATURE

Advanced driver assistance systems have been in development for quite some time and are the topic of a great deal of research. As a result, there are many vision-based approaches to properly detect and track cars and trucks. In most cases, these methods begin by converting the image to greyscale and applying some form of edge detection and/or lane recognition.

Schwarzinger et al. [1] developed an algorithm that most closely resembles our intended use case. In general, they identified four main tasks for the vision algorithm to accomplish. These tasks included the detection of cars and objects, the classification of detected objects, tracking an object once detected, and accurate measurement of the car's size in the image.

With these goals in mind, the group developed a detection and tracking algorithm capable of classifying a car and measuring its size. Based upon the detected size, the time to collision can be calculated from the change in area of the detection along with the speed of the host vehicle. Using this parameter, adaptive cruise control and collision avoidance capabilities can be implemented in a car.

To detect the size of the vehicle, they proposed that a horizontal histogram be applied to the edge-filtered image. In the histogram image, they determined that symmetry in the peaks of the histogram can be reasonably assumed to localize the position of the car. This technique therefore does not provide accurate position information; it only presents a region of the image that may contain the car.

To increase the accuracy of the detection, a "symmetry enhanced edge detection" (SEED) algorithm is applied to the portion of the image outlined by the histogram. The result of this operation allows for accurate estimation of the size of the leading vehicle in the image. The

result of the SEED algorithm is then compared to a database of example vehicles and objects. Eventually, the algorithm is able to determine the type of object detected, and the size is used for further control. Overall, this algorithm provided a basic framework for the development of a vision-based control algorithm. However, the detection time was stated to be roughly 1-2 seconds per detection, and the overall cycle time for tracking was found to be 300 milliseconds. Even assuming complete accuracy in the detection and tracking process, these speeds are not sufficient for control in an emergency scenario.

Huang et al. [2] expanded upon the histogram-based symmetry detection method from [1] with the addition of lane detection using Hough lines. Since the detection in [1] is highly dependent upon an accurate and symmetrical edge detection result, it may be difficult to identify the horizontal position of the leading vehicle. To supplement the histogram, a Hough line transform is applied to the image to identify the lanes. This combination therefore accurately segments the horizontal region of the image containing the lead vehicle.

Once the region containing the lead vehicle has been identified, the new area is searched from the bottom up with a variety of edge detection procedures. Each procedure is meant to extract different data based upon the image resolution, color, and lighting, among other variables.

A symmetrical result from the histogram can provide the position of the leading vehicle in the horizontal direction, but knowledge of the vehicle's vertical position is still unknown. Instead of using a process similar to the SEED algorithm from [1], the new area is searched from the bottom up with a variety of edge detection procedures. Each procedure is meant to extract different data based upon the image resolution, color, and lighting, among other variables.

In the bottom-up search, the different edge detection results discussed above to identify the ground underneath the lead vehicle. This is done by identifying the lighting change from the shadow of the vehicle. Based upon the size and location of the shadow, the vehicle can be detected in the image. This process was very similar to [1] but was more robust in handling changes in lighting and asymmetrical histogram results. Tracking and control was not explored, and the cycle time was still 70 milliseconds per detection.

Kuo et al. [3] show a noticeable improvement upon the methods in [1] and [2] through the adoption of research-based assumptions that allow for much more accurate and robust vehicle localization algorithm. As opposed to previous methods, the group used a camera with fixed known position. This assumption of the camera's height and orientation allowed them to calculate the location of a region five meters in front of the car. In addition to this assumption, lane recognition using an edge detection filter allowed for precise assumption of the lead vehicle's horizontal position in the image.

Along with the five-meter estimation from the camera's position, the group identified lanes in the image using edge filtering. Hartley and Zisserman [5] propose in their research that any two parallel lines converge to a single point in an image. This point of convergence lies at the horizon, commonly referred to as the line at infinity. After identifying the lane markers in the image, the point of convergence for the lanes, and thus a point on the horizon, can be calculated.

After combining the results of the above assumptions, Kuo et al. were able to locate a precise region of the image containing the lead vehicle. This region is bounded horizontally by the detected lanes on either side. The vertical segmentation is then achieved by taking the region above the calculated line five meters in front of the car, and below the calculated horizon line.

After the lead vehicle is detected, a template matching algorithm is used to track the lead vehicle in successive frames. With a detection and tracking time of 160 and 58 milliseconds, respectively, this algorithm is capable of running in real time. Based upon the size of the detection, the distance to the lead vehicle can be estimated. Control of a vehicle based upon these methods was not attempted.

Ponsa et al. [4] expand upon the previously described methods by estimating the trajectory of the leading vehicles. Similar to the approach in [3], lane markings are used to identify the horizontal regions of the image that may contain a leading vehicle. Additionally, the road surface extending to the horizon is used to segment the image vertically.

A combination of Haar features and edge detection is applied to the reduced area of the image. Rectangular regions identified in the result of this detection are assumed to be the rear windows of the car. Based upon the size and position of these detections, a relative 3D position and trajectory can be constructed for the lead vehicle. Additionally, RANSAC was used to increase the effectiveness of lane detection. Due to the added computational burden of pose estimation, trajectory mapping, and RANSAC, the detection and tracking cycle time was found to be between 150 and 700 milliseconds. These results were not tested for control of a vehicle.

2. OUR VISION-BASED TRUCK TRACKING AND DETECTION ALGORITHM

2.1 OVERVIEW OF THE DETECTION AND TRACKING ALGORITHM

Our algorithm was developed in two parts: the detection algorithm, and the tracking algorithm. It was designed to be able to run both algorithms in parallel at all times. There are some notable differences between our detection and tracking algorithm when comparing it to the examples discussed above. To begin, our algorithm is capable of detecting and tracking the leading vehicle in real time, using relatively large images. Currently, the algorithm is capable of processing 1280x960 images at a frame rate of 30 fps, corresponding to a cycle time of roughly 30 milliseconds. In addition to a fast cycle time, our algorithm is robust in its ability to detect and track in multiple lighting scenarios. Due to its design, it is capable of seamlessly updating the tracked object in real time without any pause for additional initialization of the tracker when the tracked object is updated.

As mentioned above, the detection and tracking algorithms run in parallel. The detection algorithm is constantly attempting to find the lead truck. If it succeeds, it passes the newly found template to the tracking algorithm for that frame. If it fails to find a new template, it passes the old template to the tracking algorithm. Due to the reduced search area and constantly updated template, the tracking algorithm can accurately track the lead truck in every frame. It then saves the location of the truck and uses it for the region of interest (ROI) definition for the next frame. A flowchart showing each of the steps is shown in Figure 1.

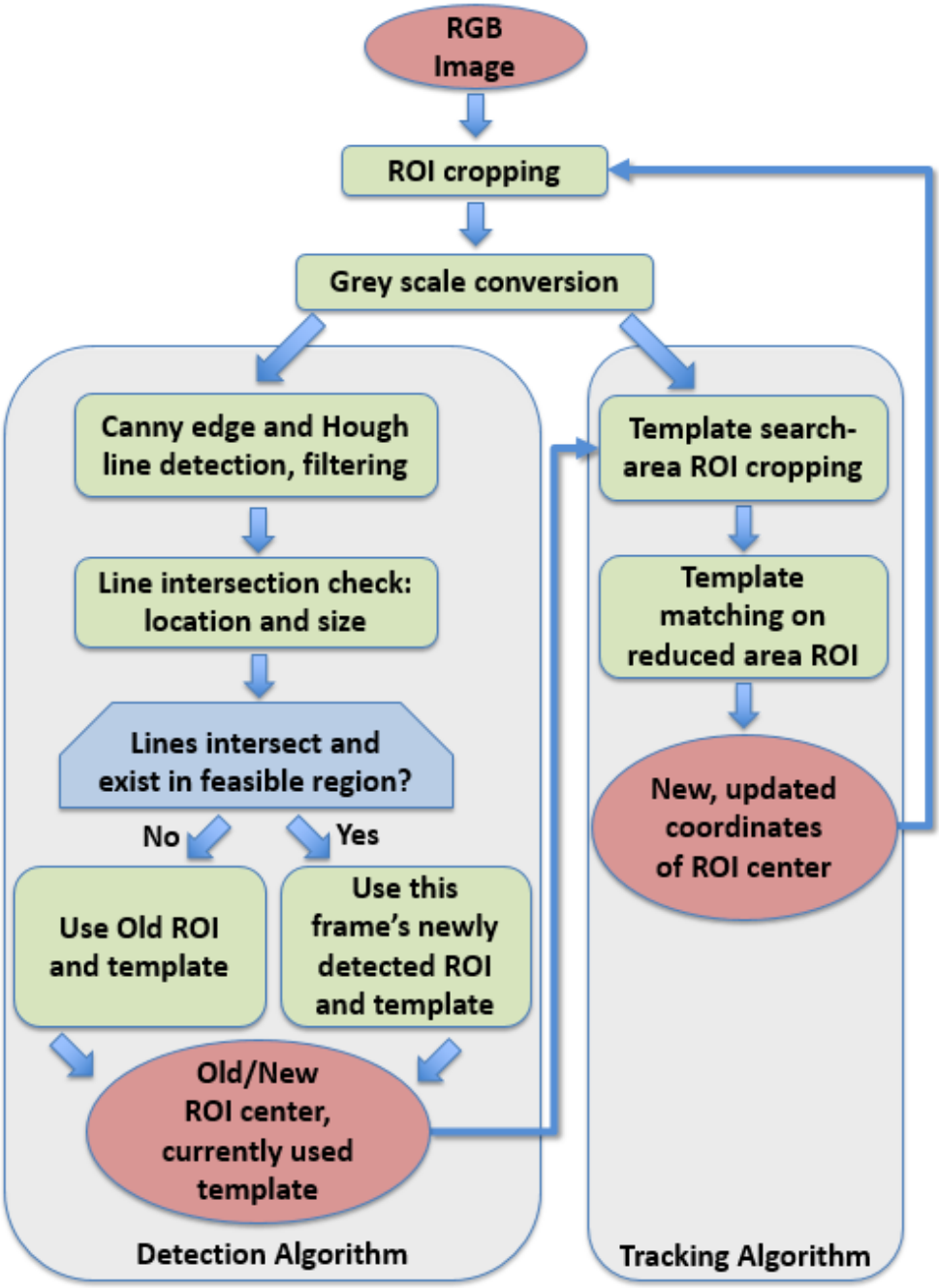
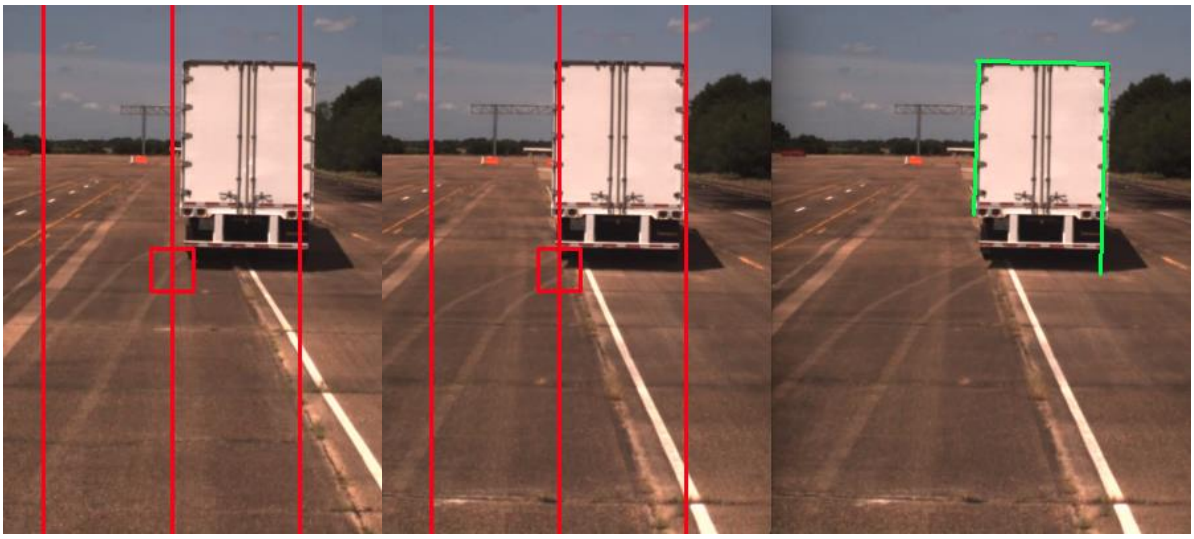


Figure 1: Description of the detection and tracking process

2.2 DEFAULT INITIALIZATION OF THE ALGORITHM

The chart in Figure 1 shows the standard tracking and detection that is performed on every frame once the tracker is initialized. When the algorithm is started, however, there is no template or initial ROI for the tracker to use, thus the algorithm falls into a default initialization mode. In this mode, the detection algorithm assumes that the target vehicle is somewhat directly in front of the following vehicle. Specifically, it looks for three lines to intersect: the first two lines are assumed to be vertical, and the third is assumed to be horizontal. Additionally, these lines are assumed to lie within the middle 3/5ths of the ROI. The vertical lines should lie either entirely to the left or to the right of the middle of the screen. The horizontal line should begin on one side of the screen and extend across to the other side of the screen. An example of the default detection scenario can be seen in Figure 2.

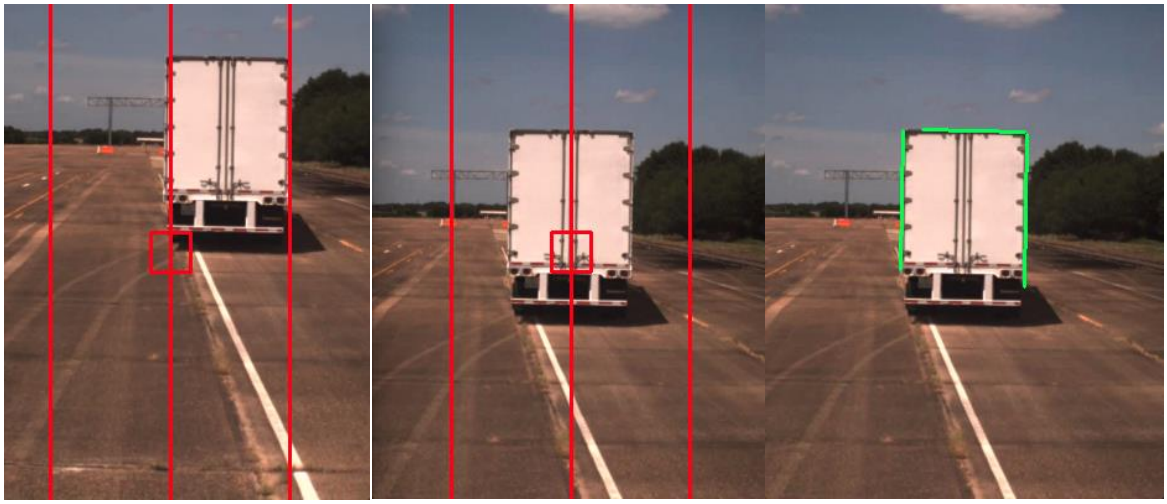


(a) (b) (c)

Figure 2: Default initialization: The truck is outside the feasible region (a), the truck enters the feasible region (b), and the lines corresponding to the successful detection of the truck once it's entered the feasible region (c)

Once the truck has been detected Figure 2 (c), the coordinates of the detection are saved and used to update the ROI for the next frame. To make the initial detection more robust, the algorithm was designed to be redundant in its first four detected objects. The first detection sets the ROI to be centered at the detected object. The algorithm then uses these coordinates in subsequent frames to confirm the presence of a truck. In the unlikely event that there is a false detection, the algorithm will continue to look for a truck in the new ROI. After a set amount of time without a subsequent detection, the algorithm resets to the default initialization mode at the center of the image.

Once the detection algorithm has reported four such detections in the same region of the image, the fourth region is taken as the template and the tracking algorithm is enabled. The algorithm then follows the process as shown in Figure 1. The progression of the ROI update and subsequent line detection can be seen in Figure 3.



(a) (b) (c)

Figure 3: ROI from first detection (a) the next frame of the video with the updated ROI after first detection (b) and corresponding detected lines for next frame (c)

2.3 THE TRUCK DETECTION ALGORITHM

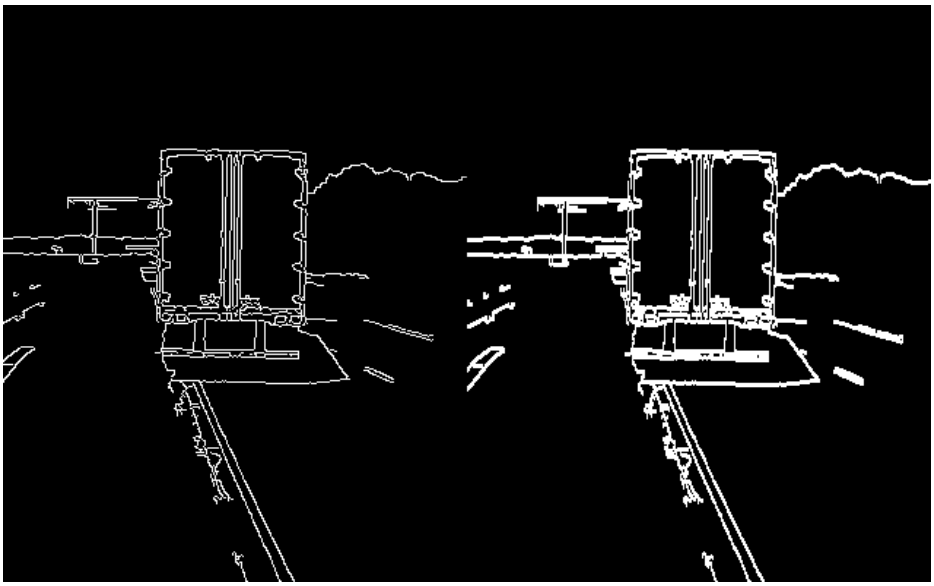
The truck detection algorithm begins by taking in the compressed video feed of a relatively large size (1280 x 960) and resizing it to 640 x 480. Due to the intended use case for the trucks, it was assumed that upon initialization of the algorithm, the target vehicle would be somewhat directly in front of the following vehicle. Using this assumption along with the detected points from the initialization, we can decrease the initial size of the region of interest (ROI). A smaller region of interest is used to reduce the computational burden of the detection and tracking algorithms. Examples of the initial full-size frame, and the corresponding initial region of interest can be seen in Figure 4 below.

(a) (b)

Figure 4: Initial size of video (a) and selected initial region of interest (b)

Edge Detection

The image is then converted to grey-scale for canny edge detection. After edge detection, a dilation operation is then applied.



(a) (b)

Figure 5: Image after canny edge detection (a) and dilation (b)

Dilation

It was found during the research that a dilation operation helped to connect fragmented or relatively short portions of otherwise long, recognizable lines. Dilation is achieved by passing some kernel of size n x n across the entire image. The kernel in our case is of size 3x3, with the central point being in position (2,2) of the kernel. As the central point is passed over each pixel in the image, it takes the maximum intensity value of all the surrounding pixels in the kernel. It then replaces the central point's intensity with that maximum value. In a binary image, white pixels have an intensity value of 1, while black pixels have an intensity value of 0. As a result,

when the kernel is passed over the image in Figure 5 (a), each pixel near a white pixel is also colored white. This operation therefore helps in connecting fragments of lines and increases the robustness of the detection algorithm.

Hough Line Transformation

It can be seen in Figure 5 that horizontal and vertical lines can accurately outline the truck. After dilation, a probabilistic Hough transform is used to find straight lines in the image. The parameters used for the Hough line transform can be specified such that the algorithm discards lines that do not meet certain length criteria. This minimum length can be visualized in Figure 3(a) as the small red square in the middle of the screen. The result of the Hough operation gives the beginning and ending points of the lines in Cartesian x-y coordinates. It is standard in image processing to consider the top left of the image to be the origin, with the positive (x , y) directions extending to the right and downwards, respectively. An example of this convention is shown in Figure 6.

Figure 6: Axis definition with example (x , y) pixel coordinates

Hough Line Filtering

The resulting points from the Hough transform can then be used to calculate the angle of the line connecting these two points according to:

$$\theta = \frac{180}{\pi} \arctan\left(\frac{y_2 - y_1}{x_2 - x_1}\right)$$

The points are then filtered such that only the points corresponding to an angle of $\theta > 85°$ or $\theta < 5°$ are considered. In our case, this constraint on the angle θ leaves us only with horizontal or vertical lines. These lines are then organized corresponding to their position on the screen.

If the points represent a horizontal line, the set is compared to previously saved horizontal points. These points are saved such that only two sets whose average y value is closest to the top of the image are retained. Each of the remaining sets of points is compared to these topmost points, and replacements are made if necessary.

A similar operation is applied to the points representing vertical lines. Contrary to the horizontal lines, the algorithm saves all of the points representing vertical lines. It filters the points into two groups based upon their position in the image. The points that fall to the right and left of the center of the image are considered “right points” and “left points,” respectively.

Geometric Distance Minimization

Once all of the lines have been saved, the algorithm then iterates through the points representing the vertical lines and compares them to the points representing the topmost line. The geometric distance is calculated to find the (x , y) point closest to the beginning or end of the topmost line according to:

$$Geometric\ Distance = \sqrt{(x_1 - x_{top})^2 + (y_1 - y_{top})^2}$$

For lines on the right side of the screen, the comparison is made between the rightmost point of the top line and the topmost point of each line on the right. Likewise for the left side of the image, a comparison is made between the left point of the top line, and the topmost point for each line on the left. The geometric distance is minimized for all sets of points. The points representing the optimal line on either side of the image are then saved.

Transformation to Homogeneous Points in Three Dimensions

At this point, the algorithm has sorted through all of the results from the Hough transform and saved three sets of points with the highest likelihood of representing the truck. These remaining points must be represented as lines. Hartley and Zisserman [5] describe a method to make this transformation. As (x , y) pairs, the points are initially represented in the 2D projective geometrical space. A transformation to 3D space can be made by representing the points using homogeneous coordinates. To do this, the point pairs are represented in three dimensions with the z dimension being of magnitude one: $(x_1 , y_1 , 1)$. Likewise, the second coordinate can be represented as $(x_2 , y_2 , 1)$.

Calculation of Line Equations on the Image Plane

Once the two points have been represented in homogeneous coordinates, a line connecting the two points can be calculated. This is done by taking the cross product of the homogeneous point pairs:

$$\begin{vmatrix} X & Y & Z \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix}$$

where X, Y and Z represent the component in each of the x, y, and z directions, respectively. The result of this cross product can then be written as:

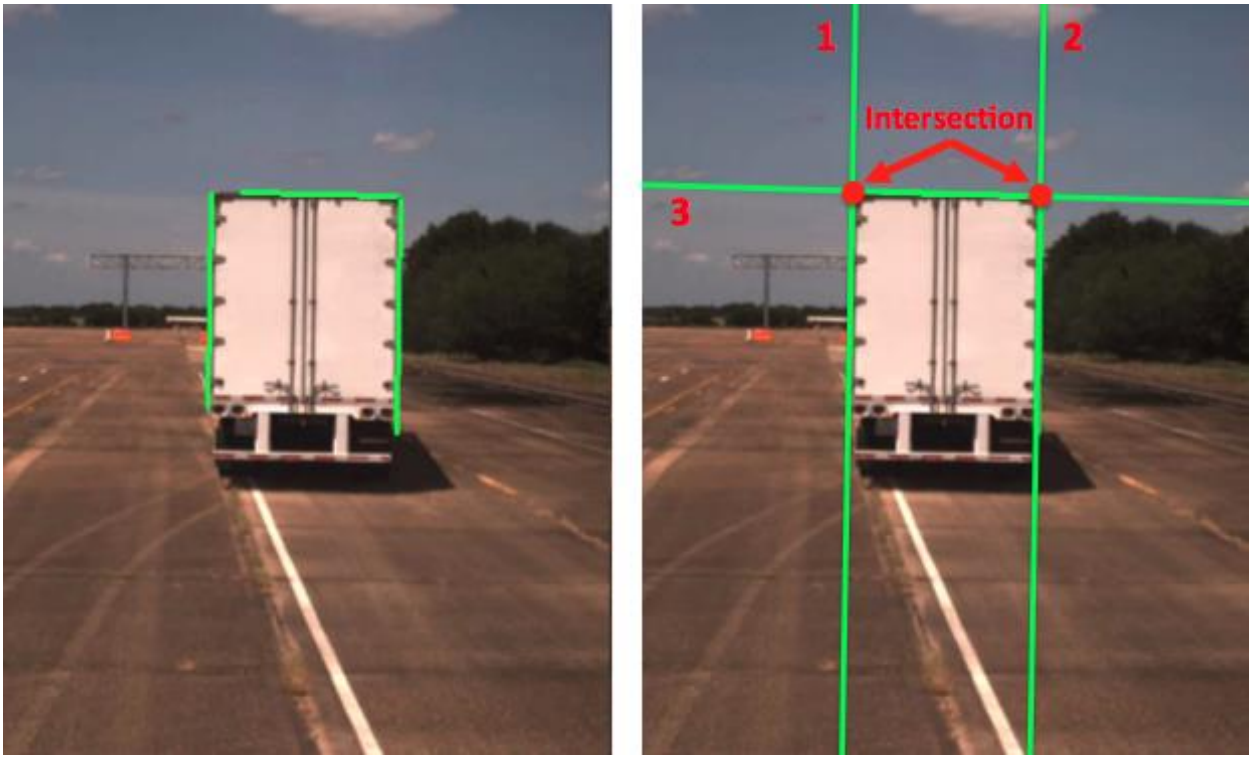
$$(y_1 - y_2)X - (x_1 - x_2)Y + (x_1y_2 - x_2y_1)Z$$

This result, however, is not useful, as it represents a line in three dimensions. To reconcile the line back to the image plane where the Z coordinate is of magnitude one, the x and y coordinates must be normalized by the coefficient of the z term:

$$\left(\frac{X}{Z}, \frac{Y}{Z}, 1\right) \text{ or } \left(\frac{(y_1 - y_2)}{(x_1y_2 - x_2y_1)}, \frac{(x_1 - x_2)}{(x_1y_2 - x_2y_1)}, 1\right)$$

This normalization gives the resulting line in the familiar homogeneous representation. This process is then repeated for each of the point pairs selected for analysis. Once the equation of each line has been calculated in homogeneous coordinates, it is possible to find the intersection points of those lines. It is important to note that the lines seen in Figure 3 (c) are merely drawn by the computer between the selected (x , y) points, and have no mathematical representation. By converting these points to homogeneous coordinates and taking the cross product, the result now

gives the equation of an infinite line on the image plane. This difference can be visualized in Figure 7.



(a) (b)

Figure 7: Lines drawn on the image (a) vs. mathematical representation of infinite lines and their corresponding intersections (b)

Calculation of Points of Intersection on the Image Plane

Once the lines are defined in the normalized homogeneous representation, the point of intersection can be calculated. Like with the point pairs, the point of intersection of two lines can also be calculated using the cross product. Crossing lines 1 and 3 in Figure 7 (b) gives the first point of intersection. Like with the "line from points" method described above, the x and y coordinates must be normalized by the magnitude of the z coordinate to scale the line back to the image plane. Similarly, crossing lines 2 and 3 and normalizing will give the second point of intersection in the image plane.

Cross-Referencing Detected Points with Calculated Points of Intersection

It is now necessary to check whether these points of intersection coincide with the detected lines shown in Figure 7 (a). This is done to confirm that the calculated point of intersection of the lines is very close to the actual detected end of both lines. In the event that the calculated intersection does not coincide with an actual endpoint of the line, the algorithm returns a failure to detect the truck in that frame.

The check for proximity is carried out after the intersection points have been calculated. A box surrounding each of the detected endpoints is taken as the feasible region for intersection. The calculated intersection points are then compared to this box. If the calculated intersection point falls within the feasible region of both detected lines, it is assumed that this point represents one of the upper corners of the truck.

A similar check is applied to the intersection point on the other side of the image. If both calculated intersection points are within the feasible region, these intersection points are used to define the truck for this frame. We know from standard trailer dimensions that the height of a

trailer is approximately 1.15 times longer than the width of the trailer. From this correlation, we can draw a box whose width is the distance between the intersection points, and whose height is 1.15 times longer than its width. This region is taken as a template for use later in the tracking algorithm.

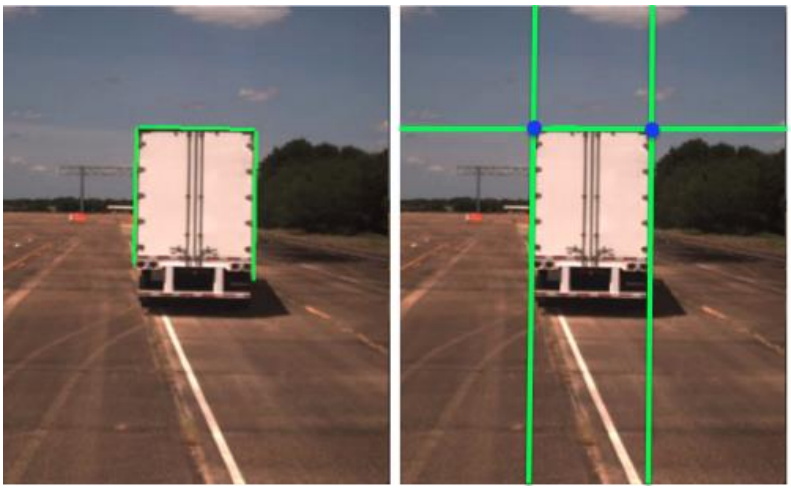
Summary of Detection Algorithm Process

The steps for the detection process are summarized as follows:

1. Assume we are provided (x , y) point pairs for each horizontal or vertical line.
2. The point pair with the smallest average y value (corresponding to the topmost line in the image) is selected to represent the top line.
3. The point pairs are converted to homogeneous coordinates.
4. The points representing the beginning and end of each line are crossed to find the equation of the infinite line connecting points in the image plane.
5. The top line is crossed with one of the vertical lines, providing an intersection point at a location (x , y) in the image plane.
6. A feasible region is drawn around the endpoints of the top line and the endpoints of the vertical line.
7. The calculated point of intersection from step 5 is compared to both feasible regions from step 6.
8. If the calculated point of intersection falls within the feasible regions of both the top and side lines, it is considered a successful line.
9. The geometric distance between endpoints is calculated and minimized for all successful lines on both the left and right side of the image.
10. If there are successful lines on both sides of the image, the region enclosed by these points is taken to be the width of the truck.
11. A length to width ratio is used to outline the truck.
12. The outlined truck is taken as a template, and the coordinates of that template are saved.

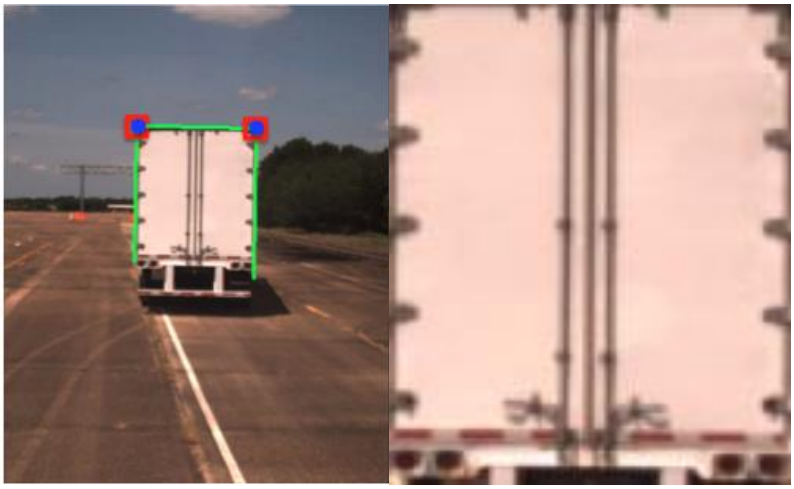
Sample Results from the Detection Algorithm

Some examples of successful and unsuccessful detections can be seen in the figures below.



(a) (b)

Figure 8: Detected lines (a) and calculated intersection points (b)

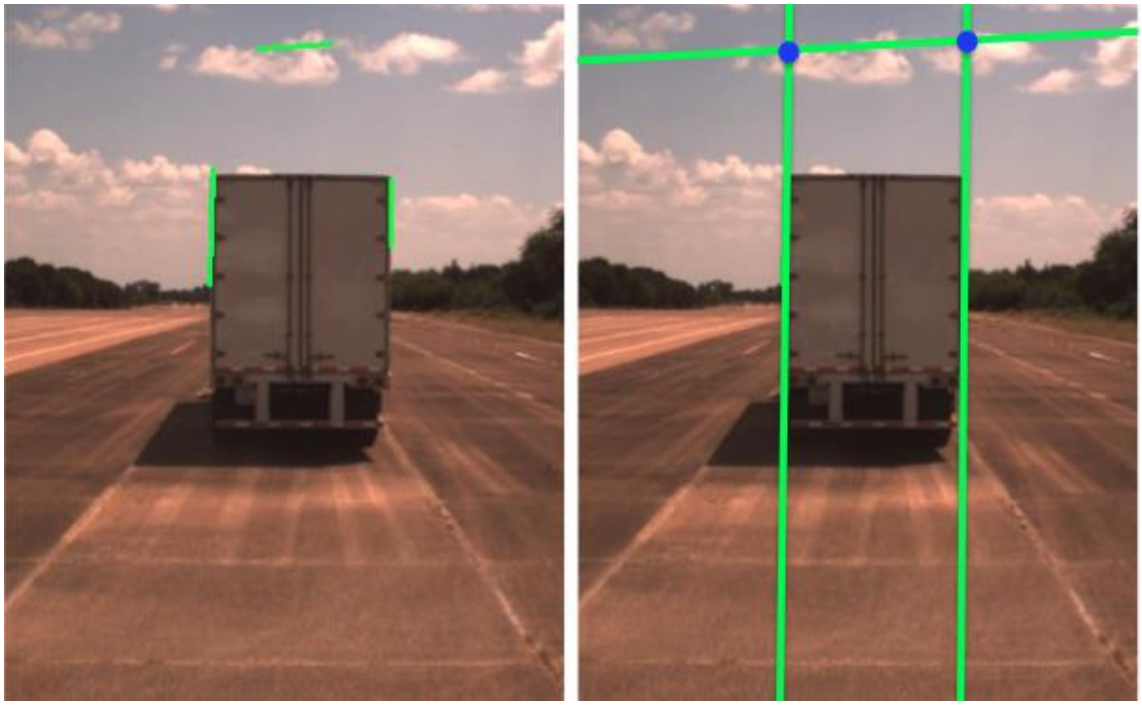


(a) (b)

Figure 9: Successful detection: The calculated intersection points lie within the feasible region (a); the resulting template used for tracking is shown by (b)

Figure 8 and Figure 9 show a successful detection. The feasible region is outlined by a red box. The calculated points of intersection from Figure 8 (b) are represented with blue dots. Since both blue dots lie within the corresponding red boxes for all lines, this detection is considered a success.

In another frame shown in Figure 10, the topmost line is detected above the top of the truck. Three lines are derived from the points, and their intersections are calculated.



(a) (b)

Figure 10: Detected lines (a) and calculated intersection points (b)

The feasible regions are drawn, but none of the intersecting points fall within these regions. In this example, the detection algorithm fails.

Figure 11: Failed detection: The calculated intersection points do not lie within the feasible region

In Figure 12, the selected points lie on the edge of the truck. The resulting lines appropriately outline the top and sides of the truck.

(a) (b)

Figure 12: Detected lines (a) and calculated intersection points (b)

Although the lines appear to represent the truck, the intersection points lie outside of the feasible regions. This again results in a failed detection.

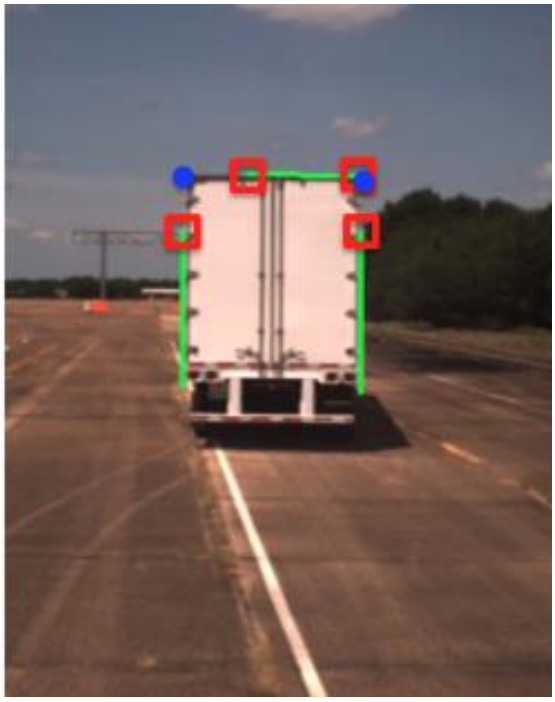


Figure 13: Failed detection: Although the lines can represent the truck, the calculated intersection points do not lie within the feasible regions

The output of the truck detection algorithm is dependent upon whether the tracking algorithm has been enabled. During initialization, the detection algorithm simply returns the pixel coordinates representing the location of the truck. If the algorithm is also tracking, it additionally returns the template as shown in Figure 9 (b). The template is then used in the tracking algorithm.

2.4 THE TRUCK TRACKING ALGORITHM

The working principle behind the truck tracking algorithm is very similar to the detection algorithm. Like the detection algorithm, the tracking algorithm is constantly updating its region of interest. Additionally, the region of interest is much smaller in scale compared to the entire image. This reduction improves both accuracy and computational efficiency.

After the detection algorithm has successfully found the truck in multiple subsequent frames, the tracking algorithm is enabled. To work effectively, the tracking algorithm requires a template, a region of interest from the main image, and the pixel coordinates of the most recently detected truck. It then parses the image appropriately and attempts to search for the truck in the specified region.

The region of interest for both the tracking and detection algorithms is constantly updated based upon the most recently saved coordinates. The detection algorithm takes priority in re-assigning target coordinates for the next frame because the detection algorithm is quite robust in its ability to detect the truck in multiple lighting scenarios. In the event of a significant lighting change and in the absence of a new detection, the tracking algorithm can fail to accurately track the truck.

The algorithms were developed in parallel precisely for this reason. The detection algorithm is constantly in use to update the template being used by the tracking algorithm. If both the detection and the tracking algorithms successfully find the truck in an image, the coordinates corresponding to the result of the detection algorithm are used for the next frame. If the detection algorithm fails, the result from the tracking algorithm is used to update the next frame.

ROI Parsing from Detection Coordinates

The coordinates from the detection or tracking algorithm are used to update the region of interest for the next frame. As a result, the truck is constantly centered in the region of interest. This is true even if the truck lies towards the edge of the main, full-sized image. As mentioned previously, this simplification is made to reduce computation time and increase the accuracy of the algorithm. An example of this can be seen in Figure 14 below. When the tracking algorithm is running, it draws a green box on the truck in the full-sized image. The coordinates of this box define a region of interest for the detection and tracking algorithms. It is clear that the truck is not in the center of the main image but is still centered in the region of interest.

(a) (b)

Figure 14: Truck not centered in main image (a) but centered in region of interest used by detection and tracking algorithms (b)

In addition to parsing the full-sized image into a smaller region of interest, the tracking algorithm further reduces the size of the image. Based upon the pixel coordinates of the most recently

detected or tracked truck, the tracking algorithm defines a search area for the template. The size of the search area is proportional to the size of the template.

The search area is either 50% or 150% larger than the template, based upon certain criteria. The smaller-sized search area is used if either the tracking algorithm or the detection algorithm correctly identified the truck in the previous frame. If they both failed, however, the search area is expanded to the larger size. This consideration is taken to account for extreme cases where the lead truck's location cannot be precisely localized by either method.

The search area is centered on the coordinates of the most recent truck identification. As mentioned previously, a template from a previous frame is also supplied to the tracking algorithm. These three items constitute the framework for the tracking algorithm. An example of the how the full image is reduced, and how the tracking algorithm is applied can be seen in Figure 15 below.

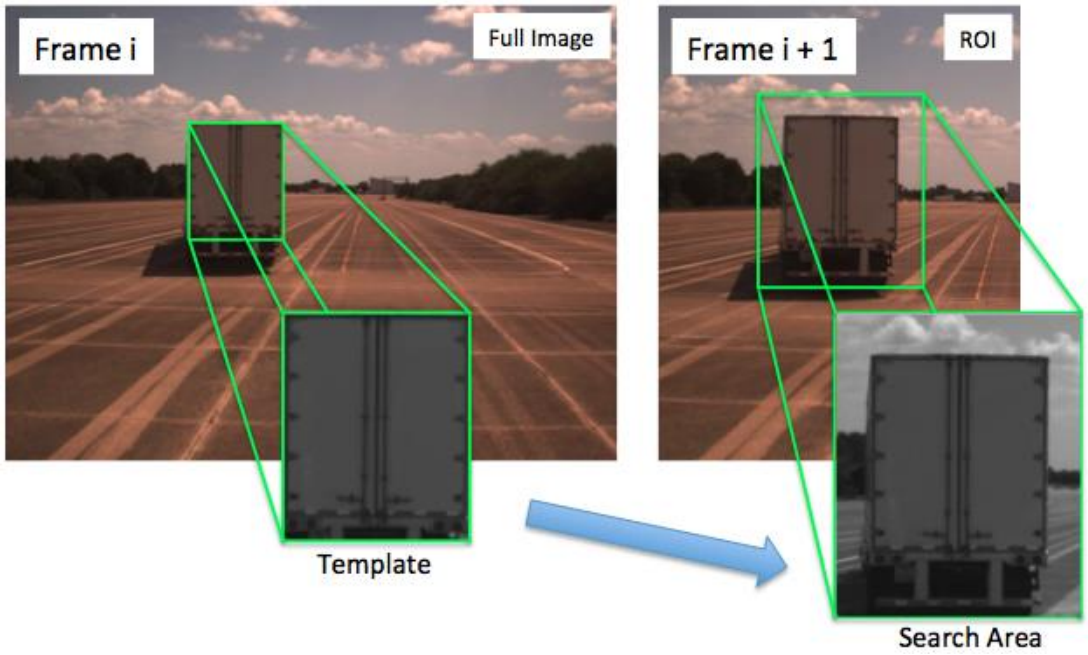


Figure 15: Basic structure of tracking algorithm where the template from frame i is used in the search area for frame i+1

The tracking algorithm is started only after the detection algorithm has been initialized by successfully identifying the truck in multiple frames. The template taken from the last such detection is supplied along with its pixel coordinates to the tracking algorithm. The template and coordinates are then used to define the search area for the next frame.

Grayscale Conversion

The tracking algorithm is based upon OpenCV's template matching algorithm which requires inputs of a template, an image, and a matching method. After some trials, the CV_TM_CCOEFF_NORMED method was selected for performance reasons. Additionally, it works best if the input template and image have previously been converted to grayscale.

Fundamentally, images are stored or represented on the computer as arrays of data. The array representing an image has an entry for each pixel coordinate. In most cases, each of these entries is another array representing the color or intensity of the pixel. In color images, this color array is normally in standard [Blue, Green, Red] (BGR) format, where each value for the blue, green, and red entries can vary between 0 and 255. As an example, a totally blue pixel at location (x , y) in the array would have a value of [255 , 0 , 0].

The BGR convention is very common and can accurately represent the color spectrum due to its variability. A single pixel can take on nearly 17 million different color combinations. For large images like the ones we are receiving from the camera, there are over a million such pixels each taking on one of these 17 million values. As a result, performing operations on color images can become quite computationally burdensome. At frame rate 30 frames per second, such operations become infeasible. Therefore, it is common to convert images to grayscale before processing.

Like color images, grayscale images are represented as arrays of pixels. In a grayscale image, each pixel is represented by a single intensity value, as opposed to a three-dimensional color value. Like with color images, this intensity value ranges from 0 to 255. An intensity value of 0 means that the pixel is completely black. As the intensity value increases, the pixel takes on an

increasingly lighter color of black or grey, eventually resulting in the color white at an intensity value of 255.

Since the intensity values of a grayscale image vary in only one dimension, it is much easier to compare two images if they are both in grayscale. OpenCV can convert an image from BGR to grayscale through a simple transformation. The intensity value is calculated [6] according to:

$$Intensity = 0.114 * Blue + 0.587 * Green + 0.299 * Red$$

It is clear from this equation that pixels with different BGR combinations can provide the same intensity value. Given an image, is quite likely that a pixel on one side matches exactly with a different pixel on the other side of the image. It is unlikely, however, that a whole region of pixels closely matches any other region on the image. This is the working principle behind the template matching algorithm.

The Template Matching Algorithm

As mentioned earlier, the template matching algorithm requires a template and an image as inputs. It works by comparing the given template to different regions of the image. This is accomplished by sweeping the template across the image, starting at the top left (origin) of the search area as defined previously.

The template is placed at the origin of the image, and the intensity value of each pixel correspondence is compared. A “match value” is then associated with the results of the comparisons for this region. The pixel coordinates in the image that correspond to the position of the top left corner of the template are saved along with the match value. The template is moved to the right by one pixel, and the pixel intensity comparisons are made again. The initial

comparison for the first iteration of the template matching algorithm can be visualized in Figure 16.

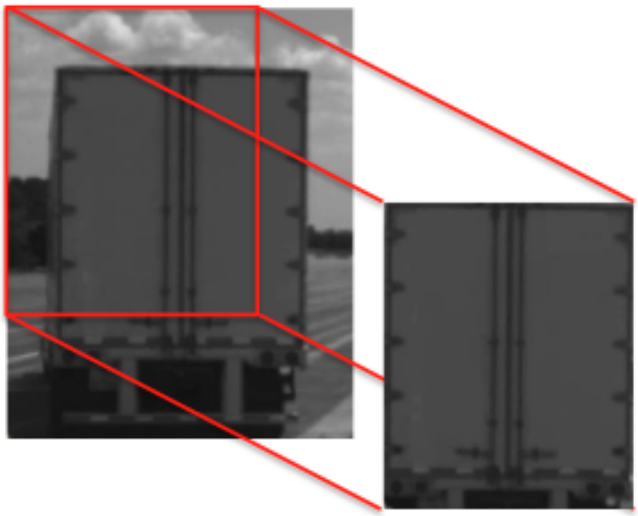
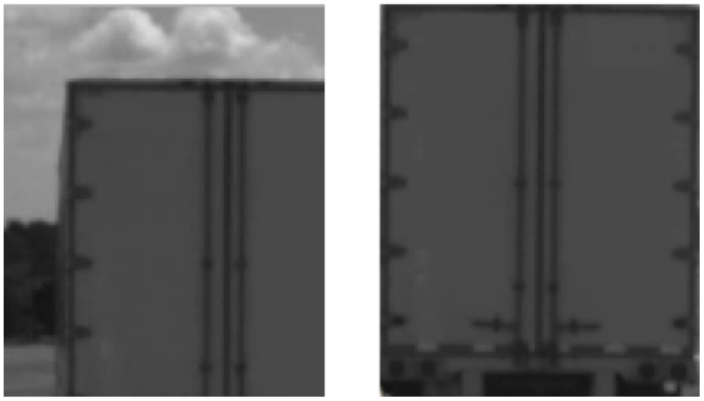


Figure 16: Position of template in first iteration of template matching algorithm

More specifically, the comparison is made between the following two images:



(a) (b)

Figure 17: First region that template is compared to from search area (a) and the template being compared (b)

The intensity value of each pixel correspondence is compared. For example, the intensity value of pixel (0,0) in Figure 17 (a) is compared only to the intensity value of pixel (0,0) in Figure 17 (b). It is clear that the result of this comparison will be poor. One pixel is very bright, with an intensity value closer to 255. The other pixel is very dark, corresponding to an intensity value of closer to 0. This comparison is continued for each pixel in the images. Pixel (100, 100) in (a) is compared to pixel (100, 100) in (b), etc. After each of these pixel comparisons have been made between the template and this portion of the image, a value between 0 and 1 is associated with the results based upon the quality of the match.

The template is moved to the right, and the comparisons are made again for the next region. A few iterations later, the template reaches the right edge of the image. After this comparison, the template moves down to the next row of the image and is compared in similar fashion. An example of another such comparison a few iterations later is shown below in Figure 18.

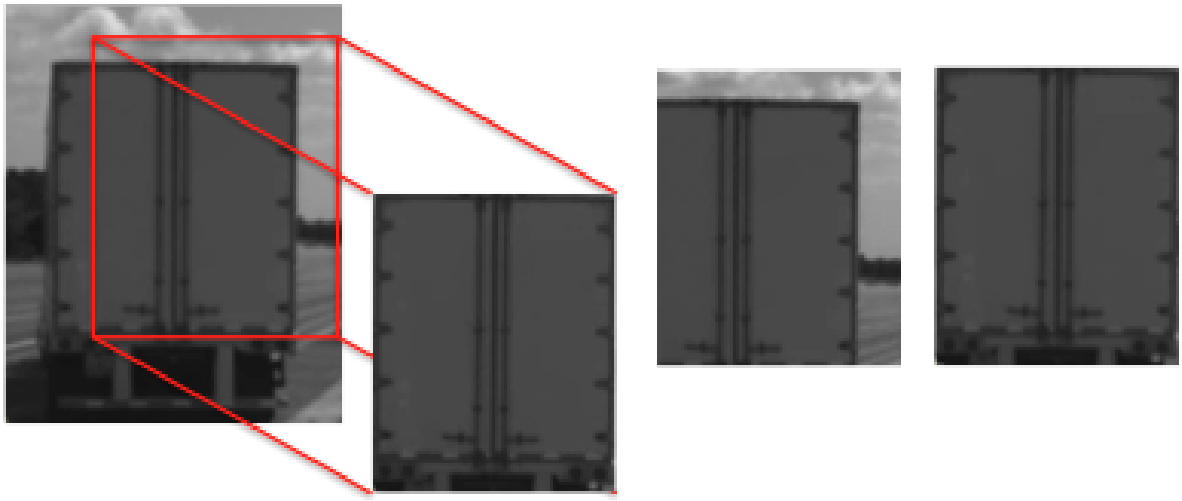


Figure 18: Region of comparison from search area after multiple iterations of template matching algorithm

Once the template has been swept across the entire image, the maximum match value and its associated coordinates are assumed to be the region of the main image that contains the truck. If the template match value corresponds to at least an 80% match, these coordinates are used to update the region of interest for the detection and tracking algorithm in the next frame.

Multiple-Scale Template Matching for Rapid Changes in Following Distance

The template tracking method was found to work extremely well so long as the size of the template and the size of the truck did not vary between frames. As mentioned previously, the template used for the tracking algorithm is only updated once the detection algorithm identifies the truck. If the lead truck accelerates or decelerates rapidly and the template is not updated, the size of the template may not match the size of the truck in the image. This scenario can be seen below in Figure 19 for a truck that decelerates rapidly.

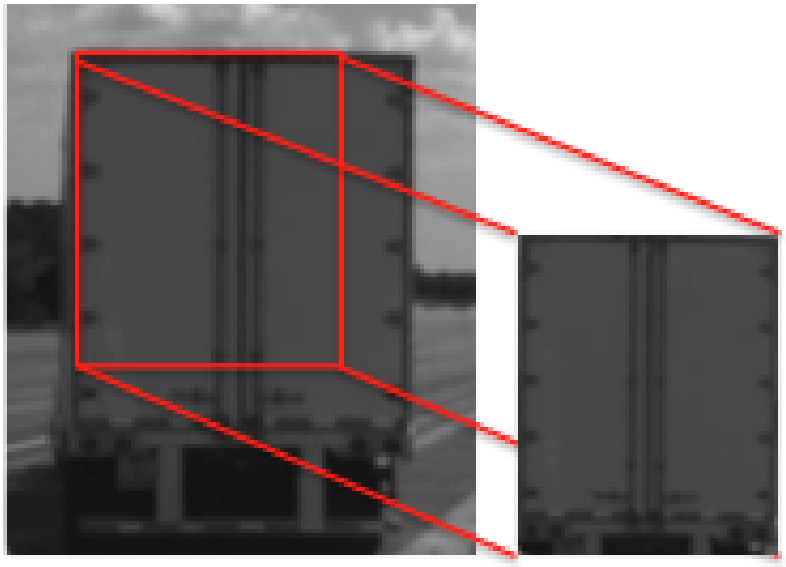


Figure 19: Region of comparison from search area if lead vehicle decelerates and template is not updated

In this case, the template does not accurately represent the lead vehicle. This can lead to an unsafe situation. Since the range is estimated based upon the size of the template, the following vehicle may not slow down unless a new, accurate detection has been made. To mitigate this risk, the tracking algorithm must be capable of efficiently recognizing changes in distance without an updated template from the detection algorithm.

It can be seen in Figure 19 that a change in distance between the lead vehicle and following vehicle causes the size of the truck to change. As the truck moves toward or away from the camera, it gets larger or smaller, respectively. To accurately track the truck in these scenarios, the size of the template in use can be scaled to provide not only position information in the image plane, but depth information as well.

To track the truck across a wide array of sizes in the search area, the template is resized in increments of 1% to between 95% and 105% of the size of the last detected template. The template matching algorithm described previously is implemented for templates of each size in the range. As the algorithm iterates through the different sized templates, the maximum match value and pixel location among templates of all sizes are saved. Additionally, the size of the best-matched template is saved for comparison with the original, detected template. Apart from range calculation, this comparative information can be used very generally to determine if the leading vehicle is getting closer or farther away from the following vehicle in successive frames.

With the addition of the ability track the truck longitudinally, the algorithm is capable of following the position of the truck across the image plane and through a wide variety of distances without updating the template in use. Altogether, the combined tracking and detection algorithm has been shown to be robust, accurate, and computationally efficient. As mentioned previously, improvements in these areas were the original goal for the development of the new algorithm.

2.5 LIMITATIONS OF CURRENT TRACKING AND DETECTION ALGORITHMS

Current tracking and detection algorithms can fall short of achieving their intended goal for a few reasons. In most vision-based tracking implementations, a region of the image is selected either manually or through some detection event. In each successive frame, the region of the image from the detection event is compared to the new image. Depending upon the algorithm, and based upon some weighted criteria, the region of the new image that most closely matches the tracked region in the old image is used to update the algorithm's weighting criteria for tracking the object in subsequent frames.

To draw a comparison to our algorithm, this would be equivalent to updating the tracked template with every frame. Unlike our implementation, there is only one detection event in which the user selects the object or region of the main image that it intends to track. The algorithm then makes its best judgment as to the location of the object in ensuing frames based upon the initial detection. In many tracking algorithms, there are no subsequent detection events. This lack of updating the tracked object can lead to an accumulation of error as time progresses.

Drift in Tracking Algorithms

Without training the algorithm or implementing other methods of artificial intelligence, a constantly updated template can lead to a great deal of error in tracking the object. Like our algorithm, other tracking algorithms are normally built to track objects at varying distances from the camera. The template being used is constantly updated to reflect these changes. As a result, it is common for the template to erroneously begin tracking areas behind or to the side of the initially selected object. This leads to compounding error over time. In vision-based tracking, this gradual increase in error is commonly referred to as "drift" in the tracking algorithm.

Currently Available Methods for Drift Mitigation

There are a few ways to combat the problem of drift in tracking algorithms. Convolutional neural networks can be used to compare the currently tracked object to detections in previous frames. By doing this, the algorithm learns while it runs, weighting current and previous frames as necessary. This weighting criteria can decrease or eliminate drift in the tracking algorithm.

Haar Cascades for Detection

Haar cascade classifiers can also be generated or designed based upon the intended application. If properly trained on the backside of the truck, the classifier algorithm can then be used to detect trucks in an image. This method can update the tracked object due to successive detections, requiring little information from previous templates. Drift is minimized by constantly detecting a new template for use.

The downside to using Haar cascades, however, is the accuracy and repeatability of the classifier algorithm. Incorrect or inaccurate detections are quite common. These inaccuracies were discovered in early attempts to use Haar cascades for tracking and detection algorithms. Often during testing, the cascade classifier would fail to detect the car in multiple successive frames, leaving gaps of more than a second between detections.

In the event that the car was detected consistently, the size of the region containing the detected car was highly variable. There was very little correlation between the size of the detected region and the size of the car in the image. As a result, it would be very hard to gather range information from this data. An example of size variability between successive frames can be seen in Figure 20.

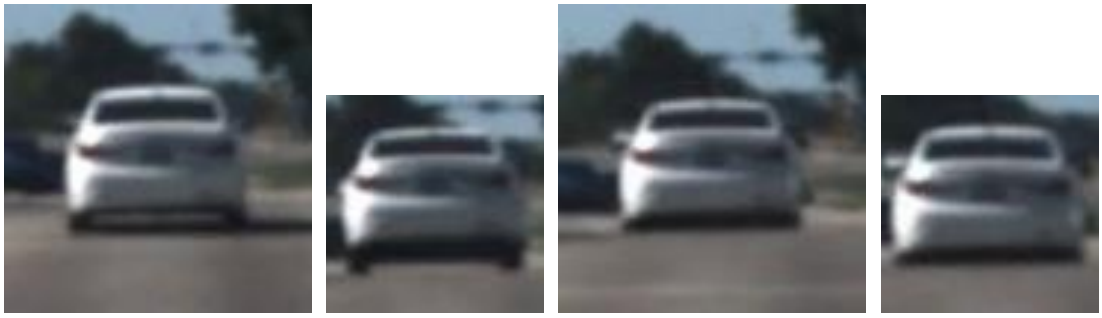


Figure 20: Size variability in Haar cascade detection between successive frames

All of the images in Figure 20 were taken from a time span of less than one second. It is clear that the detection commonly includes a great deal of the background and surroundings of the car, making it very hard to distinguish the car from its environment.

To remove some of the background information from the detections, attempts were made to crop the image to include only the rear of the car. In cropping the image, another inaccuracy in the detections was discovered. In many cases, the region containing the car was not always centered on the car. In other words, the detections provided more of an educated guess that a region may contain a car. This conclusion was drawn after recognizing that the cropped results commonly excluded sides of the car. Examples of this inaccuracy can be seen in the series of images shown in the following figure.

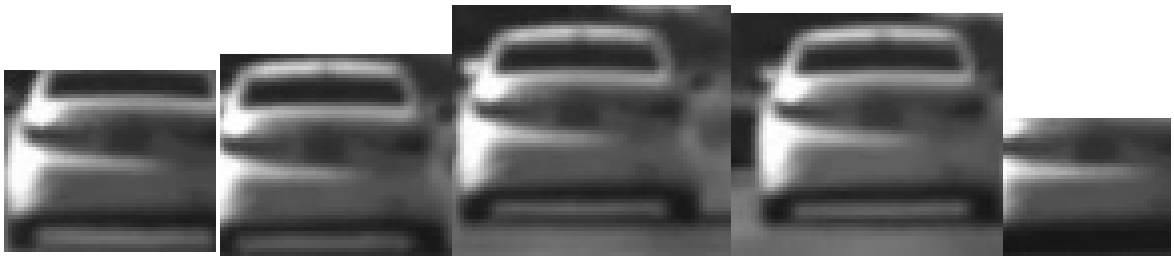


Figure 21: Center of template variability in Haar cascade detection between successive frames

The cropped regions shown in Figure 21 were all taken during the span of about a second. It can be seen that in some instances, the detection is centered toward the left side of the car. Only a few frames later, it is of a different size, and centered more toward the right of the car. The final image of the sequence shows the possible differentiation of the size of the detected region between frames. Attempting to gather concrete position information from the seemingly random detections would lead to instability in the tracking algorithm.

3. COMPARISON OF KCF TRACKING ALGORITHM AND OUR ALGORITHM

The initial intention of our research was to develop an advanced driver assistance system capable of supplementing the control scheme in the autonomous trucks. In testing the current implementation of the control algorithms, it was noted that any loss in DSRC or GPS communication, although extremely rare, caused a total system failure. In the seconds following a communication failure between the vehicles, the driver of the following vehicle had to retake control of the vehicle. In such instances, the steering wheel would reset to its default, straight orientation. This behavior was especially hazardous during turns, as it caused the vehicle to drive straight forward when it should have been turning to follow the lead vehicle.

Due to the overall failure of the control system in these instances, immediate driver intervention was required to bring the truck under control. In a real scenario, the driver may not have the awareness necessary to retake control of the vehicle in a safe and timely manner. For these situations, an altogether separate control system not based upon communication between the trucks can be implemented to increase the safety of the platooning system.

As described in the above chapters, we decided to supplement the current control scheme with a single-camera vision-based tracking algorithm hosted on a separate computer. An accurate and reliable tracking algorithm is critical to the success of this approach. Originally, it was assumed that currently available tracking algorithms could be implemented in the case of these system-wide failures. Depending upon the accuracy and reliability of the tracking algorithm, it could additionally be used for lateral error adjustments during regular platooning if needed.

3.1 INTRODUCTION TO THE KCF ALGORITHM

A great deal of accuracy in the tracking algorithm is required to safely control the vehicle. Initially, a Kernelized Correlation Filtering (KCF) algorithm was investigated for use in these scenarios. As mentioned previously, KCF algorithms are quite fast and robust, which made them a viable candidate for use in our scenario. One such implementation of a KCF algorithm was being used in our lab for other tracking purposes and was explored for use with the trucks. After some small modification, it was implementing on a desktop using a webcam. The algorithm was extremely fast, running at approximately 60 frames per second. Additionally, it was quite accurate in its ability to follow a tracked object across the screen and through a wide variety of distances.

3.2 POTENTIAL PROBLEMS WITH THE KCF ALGORITHM IF USED FOR CONTROL

In these tests, an object was manually selected for tracking by highlighting a rectangular portion of the image with a click and drag motion. Each time a new region is selected, the algorithm experiences a brief but noticeable pause that lasts for a second or more. During this time, it is unclear whether the incoming images are queued and processed once the pause has ended. Additionally, the algorithm also showed a potential for drifting as well as tracking the wrong object as time progressed.

Pause During KCF Initialization

After looking through the source code, it was noted that the algorithm is separated into three main functions. After starting the script, the algorithm simply displays the camera feed and waits for a user input. Once a region of the image has been defined by the user, the algorithm switches into a transitional “initialization” mode. The noticeable pause was observed during this initialization. Once initialized, the algorithm then begins tracking in real-time.

After working with the KCF algorithm and testing it for potential flaws, this pause during initialization was shown to have quite an effect on the effectiveness of the tracking algorithm. It appears that the KCF tracker initializes its tracked region based upon the coordinates of the image selected by the user. It does not appear to account for situations where the selected object might move before the tracker has finished its initialization.

To test this theory, we attempted to use the algorithm to track a cup. The rectangular region of the image containing the cup was selected using the mouse. Upon releasing the mouse click, the algorithm began its initialization. In the seconds during the initialization, the cup was moved slightly to be outside of the originally selected region. Once the initialization phase was

completed, the algorithm began tracking the initially selected region, as opposed to the cup. This behavior can be seen in Figure 22.

(a) (b)

Figure 22: KCF tracking algorithm is initialized on a region containing the cup (a) but begins tracking the wrong region if the cup is moved during initialization (b)

In Figure 22 (a), the mouse is used to select the region of the image containing the cup. Once the mouse click has been released, the algorithm begins initialization. Shortly thereafter, and before the algorithm has finished initialization, the cup is moved a few inches to the left. Once initialized, the algorithm appears to begin tracking the region specified by the initial mouse selection. Since the cup was moved in the moments between when the selection was made and the commencement of the tracking algorithm, it begins tracking the wrong region of the image. This error during initialization can be seen in Figure 22 (b).

Although the algorithm exhibits this flaw, it is not altogether realistic that the truck will move a great deal while the algorithm is initializing. Due to this assumption, this flaw with the algorithm was seen as something that could be worked around. Other tests were conducted to continue to analyze the effectiveness of the KCF tracking algorithm.

Drift in KCF Tracking

In addition to the pause during initialization, the algorithm showed a tendency to drift in some situations. If the target object moved toward or away from the camera, the tracking algorithm regularly began to track incorrectly. In some cases, the tracker would begin to incorporate some of the surroundings, eventually resulting in an unnecessarily large bounding box.

This behavior was again witnessed using the computer's webcam along with a cup. Figure 23 shows an example of this behavior. The tracker is initialized on the cup when it is close to the camera. As the cup moves away from the camera, the algorithm continues to track the cup, but begins to include the background in the result.

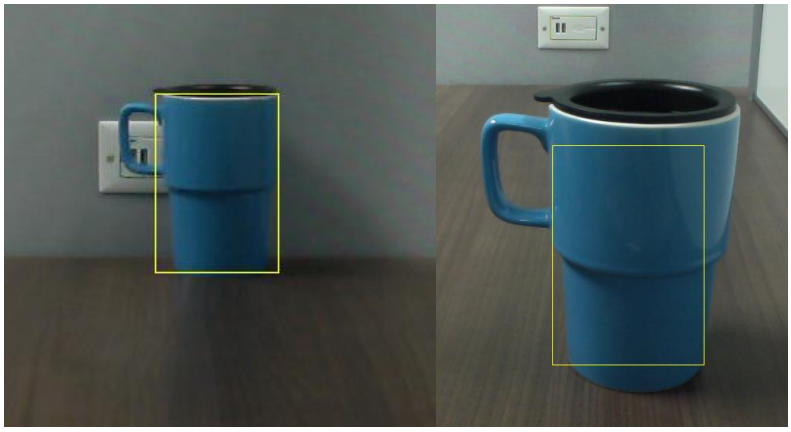
(a) (b)

Figure 23: KCF tracking algorithm is initialized on a region containing the cup (a) but undergoes a great deal of drift once the cup is moved away from the camera (b)

Although the tracker outlines a region that correctly encompasses the cup, the size associated with the tracked result is not consistent with size of the cup in the image. The ability to judge distance from the size of the tracked region is key to the success of our control algorithm. Having

a general idea of where the leading vehicle lies with respect to the camera is not sufficient for control.

In other cases, the tracker was initialized on the cup. As time progressed and the cup was moved closer to the camera, it began tracking only the interior of the cup. This resulted in a much smaller bounding box that did not provide effective range information. Like with the larger bounding box discussed above, this result is not useful. An example of this behavior can be seen in Figure 24.



(a) (b)

Figure 24: KCF tracking algorithm is initialized on a region containing the cup (a) but undergoes a great deal of drift once the cup is moved toward the camera (b)

In both cases, it is clear that the algorithm can encounter some difficulty when tracking the cup through a wide range of distances from the camera.

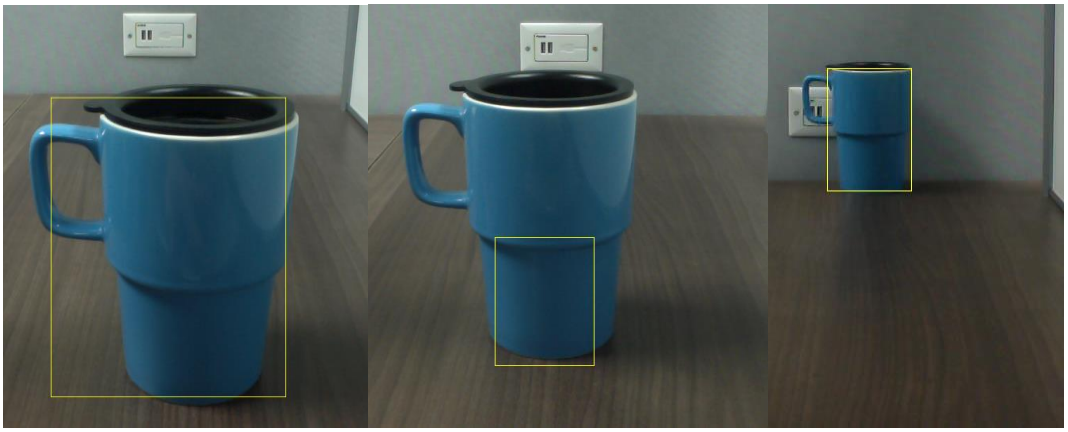
Random Size Variation in KCF Tracking

In some instances, the algorithm would jump and erroneously begin tracking the wrong portion of the cup, even if it was initialized on the correct portion of the cup.

On one such occasion, the algorithm was initialized to include the cup and its handle. After moving the cup around, the tracked result jumped to a much smaller size, including only the bottom half of the cup. It then continued tracking with the smaller region. As before, this random size variability when tracking is not useful for our purposes.

To accurately control the following vehicle in the case of a failure, accurate range and position information is required. These results can be seen in Figure 25. If using the size of the box for range information, the algorithm might sense that the cup moved extremely far away from the camera in just a few frames.

For example, the camera might sense that the cup is 12 inches from the camera in Figure 25 (a). Once the cup is moved backward by only a few inches and the algorithm jumps to include only the bottom portion of the cup, as shown in Figure 25 (b), the detected distance based upon the size of the yellow box is drastically different. If tracking correctly, this size difference would indicate that the cup has more than doubled its distance from the camera as shown in Figure 25 (c). In reality, the cup has moved only a few inches.



(a) (b) (c)

Figure 25: KCF tracking algorithm is initialized on a region containing the cup (a) but jumps once the cup is moved away from the camera (b). This suggests that the cup is actually much farther from the camera, as shown in (c)

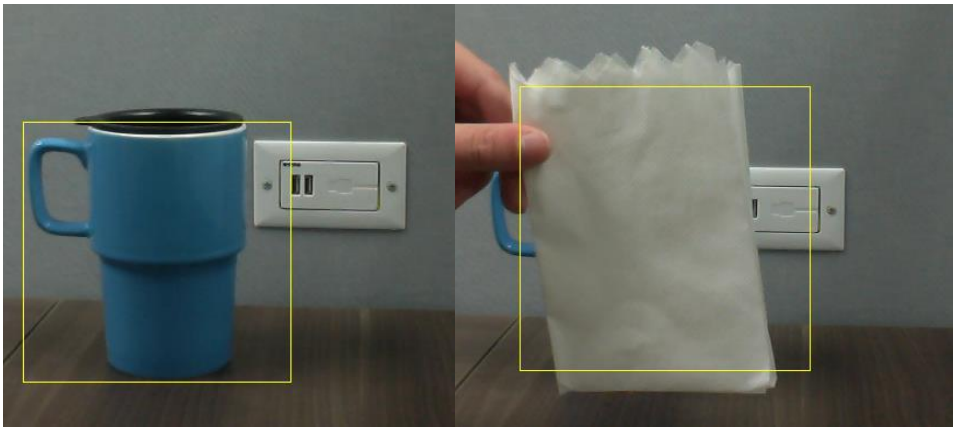
This type of size variation would result in a complete failure of the control algorithm. If the following vehicle was set to maintain a distance proportional to the size of the box shown in Figure 25 (a), the smaller box shown in Figure 25 (b) would indicate that the tracked object has moved far away from the camera as in Figure 25 (c). To maintain the distance represented by the bounding box in (a), the following vehicle would need to accelerate to catch up with the lead vehicle. In the event of this size variation error, this would cause the following vehicle to run into the back of the lead vehicle.

Tracked Object Replacement in KCF Tracking

One of the main differences between our algorithm and the KCF algorithm is the tracker's ability to update the object being followed. In the KCF algorithm, there is a single detection event in which the user defines the object to be tracked by using the mouse to select a portion of the image. In our algorithm, each frame is subject to detection, and the tracked object is updated frequently.

In testing the KCF algorithm, we noted that the singular detection event could, in some scenarios, cause the algorithm to begin tracking the wrong object. In the event that the originally tracked object is partially or fully occluded, the KCF algorithm begins tracking the object causing the occlusion. This suggests that the algorithm is not maintaining a large history of the object that it is tracking. Instead, it was assumed that the tracker relies mostly on recent information.

To test this theory, the cup was selected for tracking. Once the tracking algorithm was initialized, a piece of paper was placed between the cup and the camera, obstructing the camera's view of most of the cup. The piece of paper was then moved to the side, reintroducing the cup to the tracking algorithm. As expected, the tracking algorithm followed the paper. This progression can be seen in the following figures.

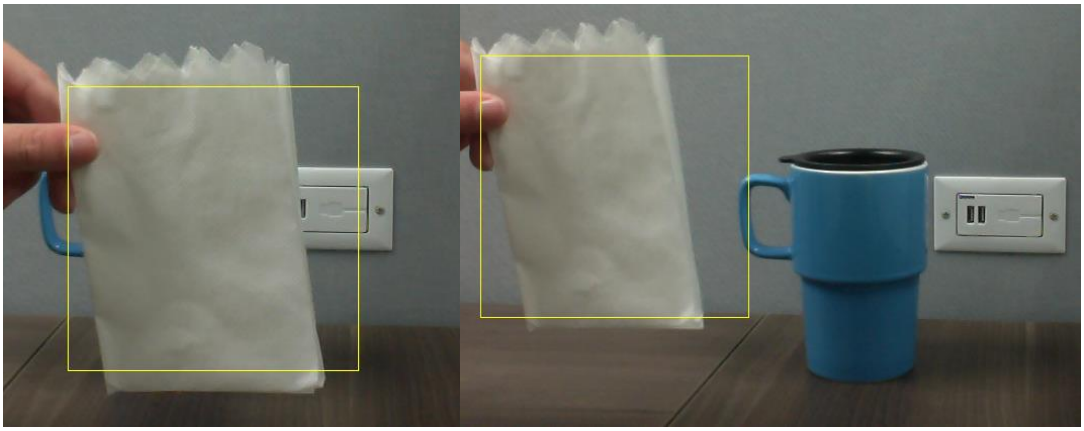


(a) (b)

Figure 26: KCF tracking algorithm is initialized on a region containing the cup (a) and is then occluded by a piece of paper (b)

In Figure 26, the cup is manually selected for tracking. After initialization, the paper is placed in front of the cup. Because the algorithm lacks any sort of detection event other than the initial

selection, it begins tracking the paper. As the paper is moved to the side, it follows the movement of the paper instead of continuing to track the originally selected cup. This result can be seen in Figure 27.



(a) (b)

Figure 27: The tracked up is occluded by a piece of paper (a). The KCF algorithm then begins tracking the paper instead of the cup (b)

In this scenario, replacement of the tracked object would result in a failure of the tracking algorithm.

3.3 KCF ALGORITHM MODIFICATION FOR USE WITH PRE-RECORDED VIDEO FROM TRUCK TESTING

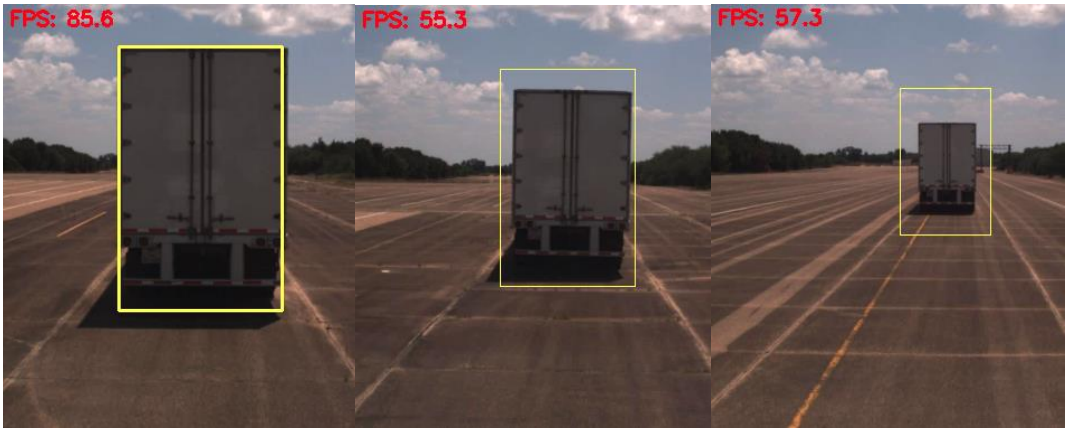
Although the tracking algorithm showed signs of weakness using the webcam, we additionally wanted to test the algorithm using video from real platooning scenarios. In the lead up to developing the detection and tracking algorithm, a large amount of video was recorded from the trucks during platooning. This video could then be played into the algorithm as if the algorithm was receiving the video in real-time, thus simulating the behavior of the algorithm if deployed in the trucks.

Addition of Our Detection Algorithm to KCF Tracking Algorithm

To simulate the tracking algorithm's behavior in the trucks, the KCF algorithm was modified to include our detection algorithm. The manual selection of a region of the image for tracking was replaced by the result from the detection algorithm. Once the appropriate detection criteria were met, the resulting bounding box was spliced into the KCF tracking algorithm in place of the normal, manual selection. The KCF tracker then initializes based upon this result and tracks the truck as the video plays.

Results of KCF Tracking Algorithm Used with Video from Platooning Scenario

As expected, the KCF tracker experienced a great deal of drift while tracking. The algorithm was prone to gathering background information as time progressed, resulting in a large bounding box. This inaccurate tracking result happened in all 11 videos that spanned more than 20 minutes in total. A short time after the first detection, the tracked region began to drift in each of the videos. These results can be seen in the following figures.



(a) (b) (c)

Figure 28: The KCF algorithm is initialized on the back of the truck using the detection algorithm (a) with the result of the KCF tracker after 10 seconds (b) and 20 seconds (c)

In Figure 28 (a), the back of the truck is detected using our detection algorithm. The KCF algorithm then initializes using this portion of the image. After 10 seconds, the leading vehicle appears to have accelerated relative to the following vehicle and appears smaller in Figure 28 (b). Figure 28 (c) shows the tracked result after 20 seconds. The lead vehicle has continued its acceleration, and the KCF algorithm is still tracking the truck in the image. However, the bounding box does not accurately reflect that the lead vehicle has accelerated in any significant manner. The bounding box in (a) does not reflect any background information about the truck, while more than half of the bounding box in (c) is made up of the surrounding environment.

The main difference between our algorithm and the KCF algorithm is the fact that the KCF algorithm does not update the tracked object if new detections are made. We attempted to add updated detection information to the KCF tracker with poor results. Upon each new detection, the lag during initialization caused the algorithm to become unresponsive with multiple detections. After these tests, it was apparent that a different tracking algorithm was necessary to provide the accuracy and reliability required for precise control of a vehicle.

3.4 RESULTS OF OUR TRACKING ALGORITHM IN SIMULATION AND COMPARISON TO KCF ALGORITHM

Although the KCF algorithm was capable in most cases of robustly tracking the truck across the image plane, it was not accurate enough to provide the precise range information necessary for control. Due to the accuracy of the detection algorithm, the detected region supplied to any standard tracking algorithm might be updated multiple times per second. As a result, we required an algorithm capable of processing a dynamically updated region of the image to track. This requirement drove the development of our tracking algorithm that is based upon template matching.

Advantages of Our Tracking Algorithm

Our tracking algorithm was developed to be reliable, accurate, and computationally efficient. The development was driven with the intention of providing tracking solutions in the areas where the KCF algorithm showed weakness. As a result, care was taken to avoid any type of separated initialization function. The tracking algorithm should be capable of providing reliable feedback regardless of how often the tracked object is updated. Additionally, the algorithm should not show signs of drift or error while running.

The resulting tracking algorithm was therefore based upon both template matching and optical flow principles. In each frame, an arbitrary template and search area are passed to the template matching algorithm. The size of the search area is defined according to the coordinates of previous detections and is representative of a feasible region that may contain the truck in the next frame.

In each frame of the tracking algorithm, any template or search area can be supplied to the template matching algorithm. This removes the necessity for an initialization phase and allows the tracked object and region to be updated continuously. This functionality is made possible due to the accuracy of the detection algorithm, which is capable of detecting the truck multiple times every second.

Results of Our Tracking Algorithm in Video Simulation

Once the tracking algorithm had been developed, it was simulated in a similar manner to the KCF algorithm. As before, the detection algorithm passes the appropriate information about the location of the truck to the tracking algorithm. One noticeable difference, however, is that our tracking algorithm was not tested using a single detection, but was updated in real time with new detections. This functionality is not comparable to the KCF algorithm due to the pause during initialization that rendered it unresponsive upon being supplied successive detections. The ability to rapidly update the tracked object and region were a driving force in the development of the tracking algorithm. The results of our tracking algorithm when processing the same video can be seen in Figure 29.

(a) (b) (c)

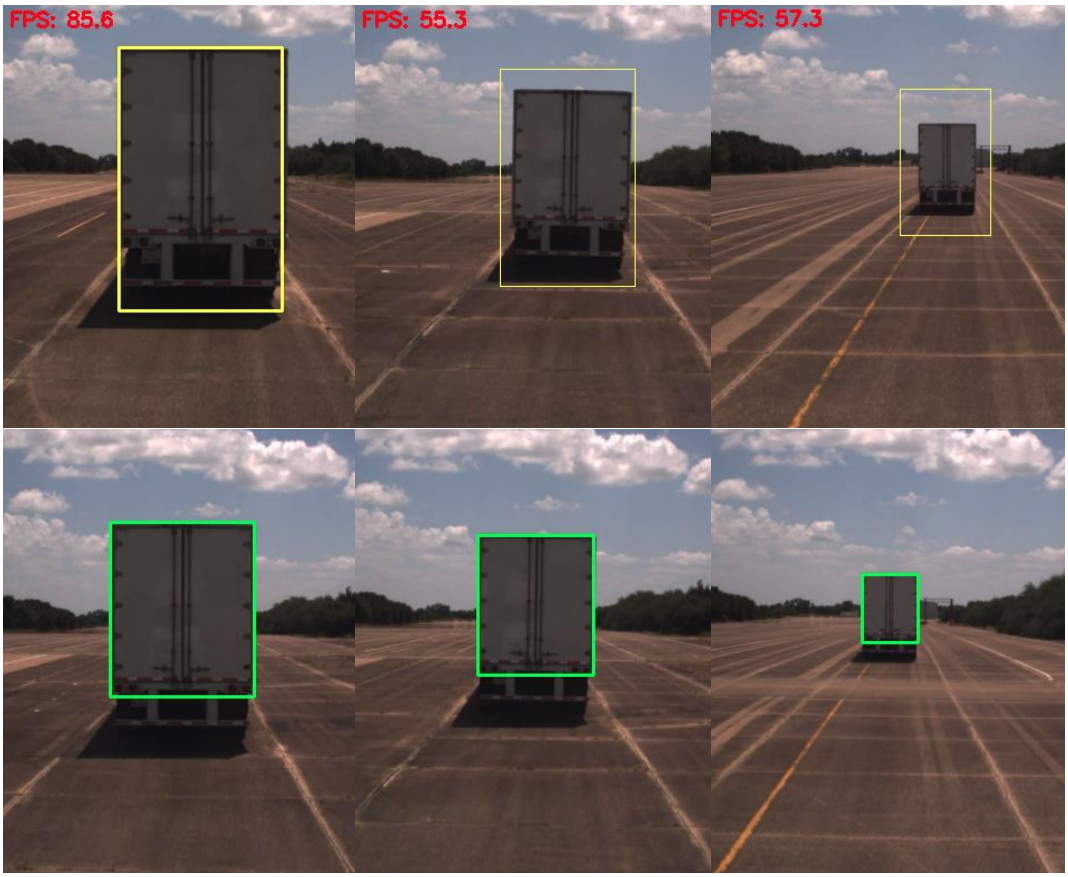
Figure 29: Our tracking algorithm is started on the back of the truck using the detection algorithm (a) with the results of the tracker after 10 seconds (b) and 20 seconds (c)

As with the KCF algorithm, our tracking and detection algorithm begins processing the incoming video stream. The first valid detection that begins the tracking algorithm is shown in Figure 29 (a). The lead vehicle again begins to accelerate away from the following vehicle, but the tracking

algorithm does not show signs of drift. It accurately tracks the change in size of the truck in Figure 29 (b) and (c). This accurate representation of the size of the truck provides the type of accurate feedback required for control of the following vehicle, as it provides both distance and position information for the truck.

Comparison Between Tracking Algorithms

A side-by-side comparison of the KCF algorithm and our algorithm can be seen in Figure 30.



(a) (b) (c)

Figure 30: Comparison of KCF tracking algorithm (top) and our tracking algorithm (bottom) at initialization (a) 10 seconds (b) and 20 seconds (c)

It is clear from the comparison in Figure 30 that our tracking and detection algorithm is much more accurate and robust than the KCF algorithm. When compared side by side, our algorithm shows no signs of the drift or tracking error common to the KCF algorithm.

False Detection Handling

The detection and tracking algorithm was tested on 11 pre-recorded videos totaling more than 20 minutes of footage. The videos were taken in multiple lighting scenarios combined with various maneuvers including lane changes, braking, acceleration, and wide turns. Overall, the algorithm behaved as desired for our application.

It was noted that in one scenario, the algorithm mis-detected the truck, causing the tracking algorithm to fail. After further investigation, we found that, in the middle of a turn, the parallax of the truck during the turn along with the lighting caused the algorithm to incorrectly identify the truck. This result can be seen in Figure 31.

Figure 31: False detection during simulated testing for control algorithm

To avoid passing these scenarios to the controller, a simple recursive check is performed after an apparent detection. If the newly detected region in the image is less than 80 percent of the size of the current template, the detection is ignored, and the algorithm continues to run using the old template. After implementing this function, the algorithm successfully detected and tracked the truck without error.

4. VISION-BASED CONTROL OF AN AUTONOMOUS VEHICLE USING OUR TRACKING AND DETECTION ALGORITHM

After testing the algorithm for weaknesses through desktop simulation, the program was modified for use in an autonomous vehicle. The algorithm was originally developed on a Mac OS using python 2.7 with OpenCV. Since most of our autonomous vehicles use the Robot Operating System (ROS) on a Linux machine running the Ubuntu 14.04 OS, multiple changes needed to be made to interpret the live camera feed for processing.

Although the tracking and detection algorithm was built and developed as a secondary safety feature for platooning research with the autonomous trucks, some unforeseen limitations prevented the use of our algorithm in the trucks. As a result of this setback, we shifted our focus toward implementing our algorithm in a drive-by-wire enabled Lincoln MKZ sedan. To successfully control the car, useful data was extracted from the result of the tracking algorithm and published via ROS topic to the car.

4.1 OPTIONS FOR DEVELOPMENT OF LATERAL AND LONGITUDINAL CONTROLLER

After examining the accuracy and reliability of our tracking and detection algorithm, we investigated ways to effectively use the results. Due to the technological capabilities already installed in the vehicles as well as additional equipment from the lab, we had multiple options for control. Among the options, we chose to use only the data from the camera to control the following vehicle.

Distance Estimation Based Upon Size of Detected Truck

As mentioned previously, precise estimation of the size of the truck would allow us to gauge its distance from the camera using simple ratios. In most cases, cameras have a fixed focal length corresponding to the size and concavity of the lens. In general, this focal length helps to relate the true size of an object to the size of the object's representation in the resulting image.

Since the focal length of the camera is fixed during normal operation, it can be assumed that there is a linear relationship between the actual size of an object and the size of the object in the resulting image. In the area of computer vision, this focal length, among other aspects of the camera's intrinsic parameters, can be calculated through a process known as camera calibration.

The calculation of the camera's focal length can be achieved easily using some known parameters and is commonly accomplished using a checkerboard pattern. Multiple pictures of the checkerboard are taken from different distances and orientations. If the width of the checkerboard squares is known, the intrinsic camera matrix can be calculated. OpenCV's *calibrateCamera* function can be used for this calculation.

The calibration results in a matrix representing the intrinsic properties of the camera. The central element of the intrinsic matrix represents the focal length the camera, normally in units of pixels.

Once the focal length is known, any line can be converted from a length in pixels to a length in meters using ratios and similar triangles. This concept can be seen in Figure 32.

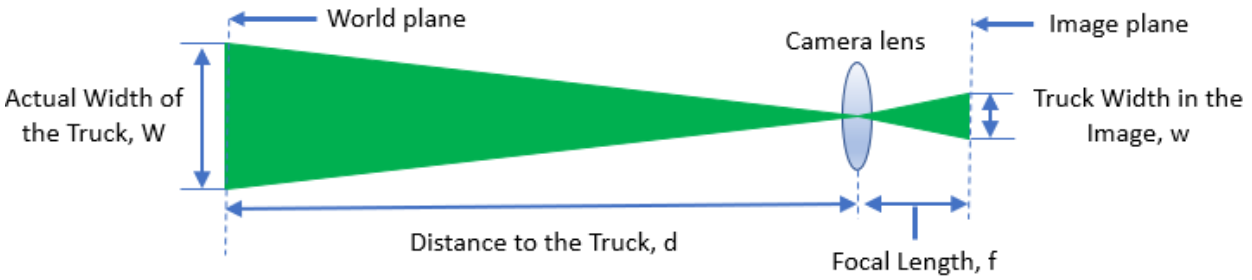


Figure 32: Method of similar triangles using detected width of the truck to determine distance for longitudinal control

In Figure 32, the ratio of the widths and heights of the triangles must be the same according to:

$$\frac{d}{W} = \frac{f}{w}$$

Now, since a standard 18-wheeler trailer is 2.6 meters wide and the focal length of the camera is known, we can solve for the distance to the truck by measuring the width of the truck in the image:

$$d = \frac{f}{w} * W$$

Since the relationship between actual and detected size is linear, the tracked region of the image can be used to provide an estimate of the camera's distance to the truck.

Distance Estimation Based Upon Radar

Another option that was proposed for longitudinal distance estimation was using radar data for longitudinal control. Both the Lincoln and the autonomous truck are equipped with Delphi radar systems. If using the radar data for longitudinal following distance estimation, we could build a longitudinal controller to maintain the car at some set following distance. We could then use the position of the vehicle in the image for lateral control.

GPS Waypoint Following for Lateral Control

In the past, we have utilized the Lincoln car for research using GPS waypoints for lateral control. In these scenarios, longitudinal control was normally handled by limiting the speed of the car. If using some method of longitudinal control, the GPS coordinates of the leading vehicle could be used to create a path for the following vehicle. However, this would require some form of communication between the trucks.

To adequately control the following vehicle using GPS waypoints from the lead vehicle, DSRC communication is necessary. This again is not a problem because both the car and the truck are equipped with these capabilities. If using GPS along with DSRC communication for waypoint lateral control, we could incorporate the range estimate from the detection algorithm for longitudinal control.

Although an attractive solution, this control strategy was passed over for one main reason. The reason for developing the tracking and detection algorithm was to provide some type of backup safety system to aid the driver in the event that the control system experiences some type of catastrophic failure. As discussed previously, a failure in the GPS and DSRC communication is

precisely the reason for this type of system malfunction. Relying on a solution that requires both the GPS and the DSRC to be functional might not be the best course of action in the future.

Lateral Control via Error Estimation from Detection Algorithm

Another proposed alternative for lateral control of the following vehicle involved using the detected position of the leading truck for lateral error correction. This is possible because the camera is mounted to the center of the car, pointed straight forward. If the detected truck lies to either side of the center of the image we can assume that we need to turn in that direction to minimize the lateral error.

Lateral and Longitudinal Control Method

After reviewing the possible configurations for lateral and longitudinal control, we decided to attempt to implement the simplest and yet untried method from those listed above. In an effort to rely solely upon camera data, we chose to develop both the lateral and longitudinal control based solely upon the results of the tracking algorithm. Since the data from the tracking algorithm seemed to be extremely accurate and reliable, we assumed that it was possible to provide the type of lateral and longitudinal control that might be developed from data sources such as GPS or radar.

4.2 DERIVATION OF CONTROL STRUCTURE BASED UPON THE RESULTS OF OUR TRACKING ALGORITHM

To develop the lateral and longitudinal control algorithms, we first needed an understanding of the existing control structure in the car. At present, the car is controlled through a combination of steering and acceleration commands by passing inputs of steering wheel position and desired speed. To use our algorithm, we converted the truck's position and size in the image to something meaningful for the controller.

Velocity-Based Longitudinal PD Control from Distance Estimates

The first challenge was developing a longitudinal controller that output speed commands. Due to the accuracy of the detection algorithm, we were able to gather an approximate distance to the lead vehicle. The car, however, requires a desired speed input similar to a car's cruise control. Since the current speed of the car is available, we were able to vary the car's following distance by sending a different desired speed to the car's cruise control.

For example, if the car is travelling at a current speed of 10 m/s and a following distance 40 meters, we can decrease the following distance to 30 meters by commanding a higher velocity for a short period of time. If the car is travelling at the correct following distance, there is no correction necessary. In this case, the desired speed command supplied to the car is simply the current speed of the car.

To send a command reflecting the desired speed of the car based upon distance estimation from the tracking algorithm, we implemented a summing element in the controller. At all times, the current speed of the car is supplied to the algorithm. The current speed is then changed based upon the error feedback from the camera. If the distance error is close to zero, very little speed

adjustment is required, and the current speed of the car dominates the control input to the plant. If the error term is large, however, the speed adjustment is large enough to warrant some type of acceleration or deceleration. The eventual structure of the PD longitudinal controller can be seen in Figure 33.

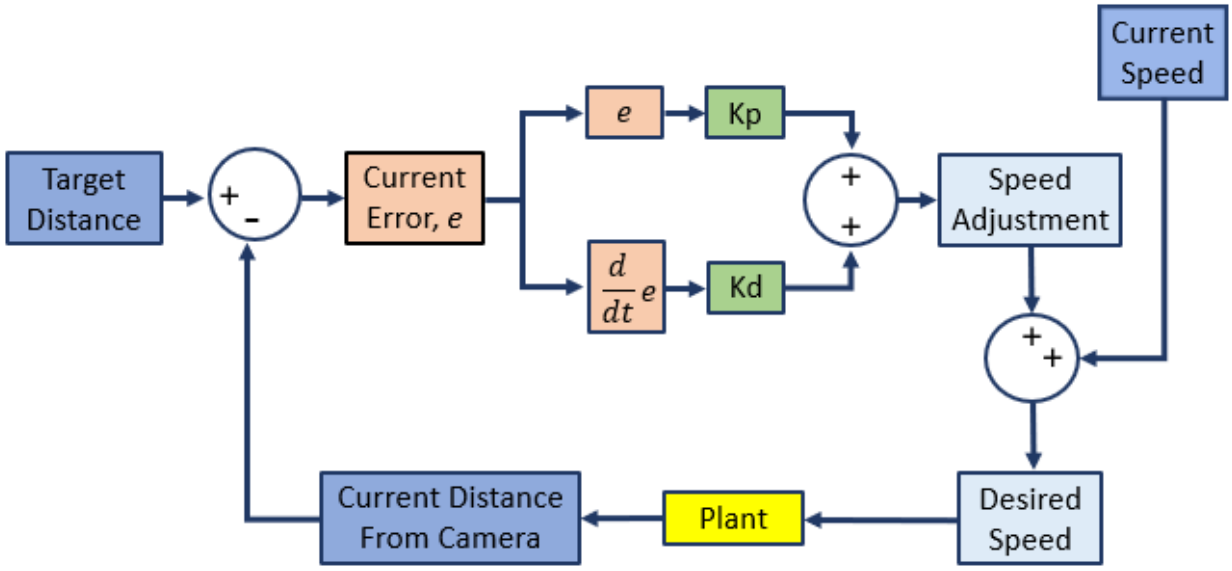
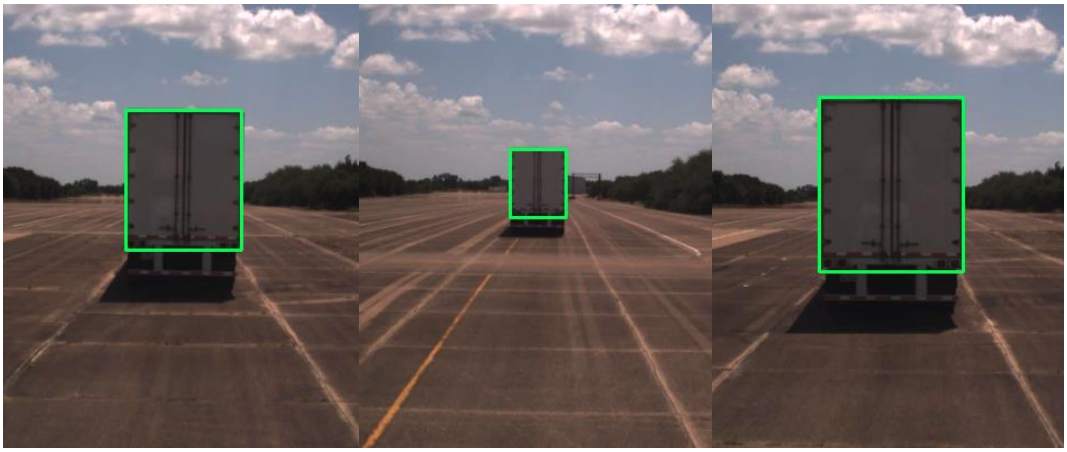


Figure 33: Diagram of the longitudinal PD controller

To better visualize the longitudinal control procedure, we can assume that the control distance was set corresponding to the detected size of the trailer in Figure 34 (a).



(a) (b) (c)

Figure 34: Assuming the target following distance is shown by (a), the controller would send acceleration commands in the case of scenario (b), and deceleration commands in the case of scenario (c)

In Figure 34, if the target size is shown by (a) and the detected size is also (a), only the current speed of the car is supplied to the controller, resulting in no acceleration or deceleration. If the size of the trailer is found to be smaller, as is the case with (b), the target speed of the car is increased gradually until the appropriate distance is met. Likewise, if the trailer is larger than the target image as in (c), the car is issued a decreased target speed. The decreased speed can be achieved through braking or simply releasing the accelerator, depending upon the required amount of deceleration.

Error-Based Lateral PD Control from Position Estimates

After developing the longitudinal controller, we applied the same proportional-derivative control principles to the lateral controller. Since the camera was mounted in the center of the car and was pointed straight forward, we could assume that the detected position of the truck could be used to correct the lateral error of the following vehicle. If the truck was detected more toward the left or right of the image, there was some amount of lateral error between the center of the detected truck and the center of the image. We could then command the steering wheel position to minimize the lateral error. The resulting lateral PD controller can be seen in Figure 35.

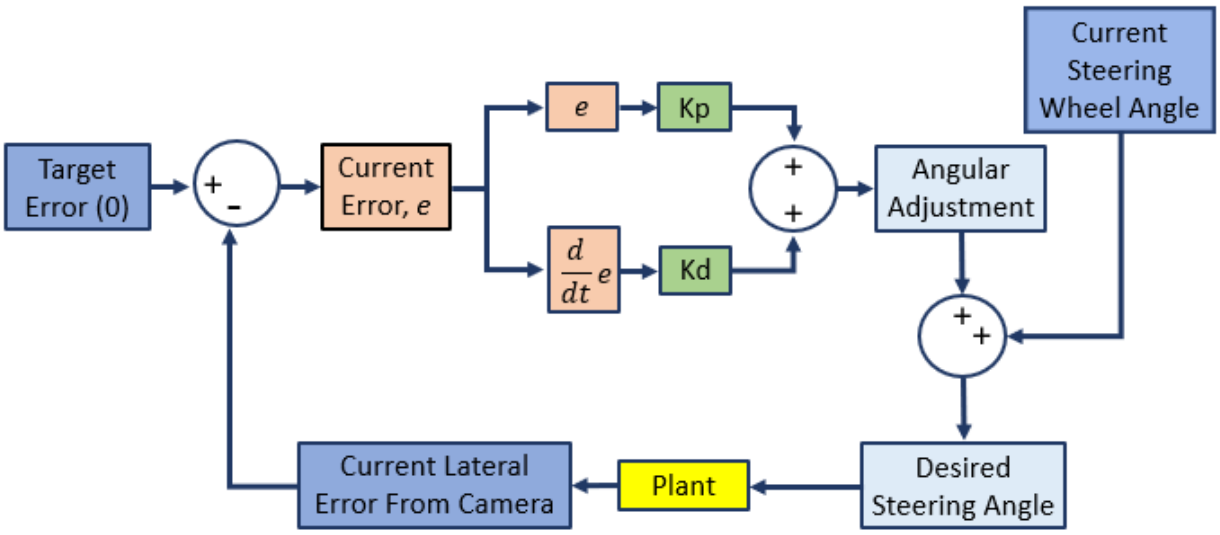
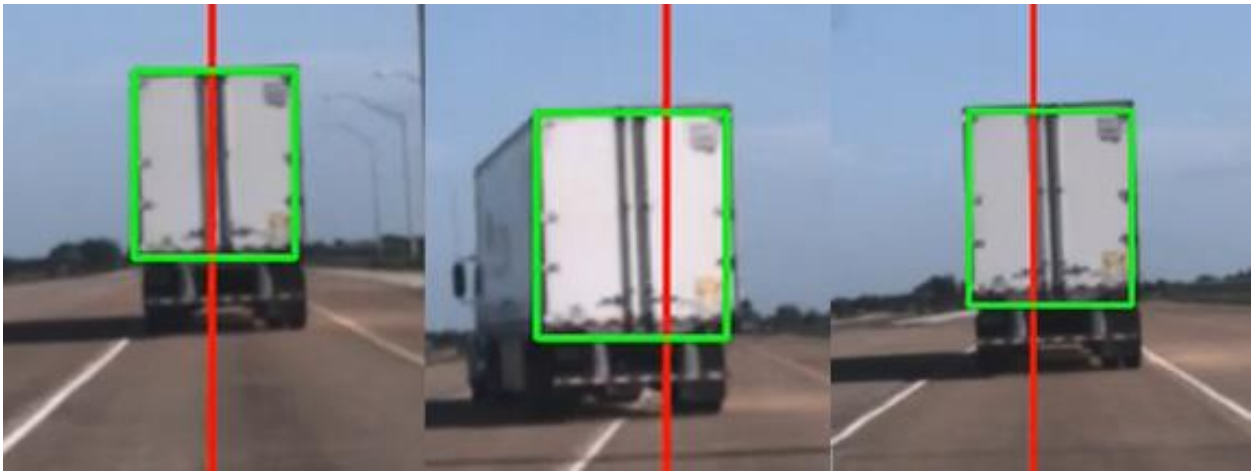


Figure 35: Diagram of the lateral PD controller

An example of the error-based lateral control scenario can be seen in Figure 36. For visual reference, the lateral controller attempts to keep the red line centered on the backside of the truck.



(a) (b) (c)

Figure 36: The lateral controller attempts to keep the middle of the truck centered on the red line. As the truck changes lanes to the left (b) the controller steers left (c), completing the lane change

As the leading vehicle attempts a leftward lane change, the detected region is centered to the left of the red line, shown in Figure 36 (b). Due to the lateral error between the middle of the truck and the red line, the steering wheel is turned to the left. A few frames later, the center of the truck is detected slightly to the right of the red line, indicating that the lane change has completed. At this point, the steering wheel is straightened out, and the car continues to follow in a straight line.

4.3 DETECTION AND TRACKING RESULTS FROM TESTING LONGITUDINAL AND LATERAL CONTROLLERS

After tuning the Kp and Kd gains for the longitudinal and lateral controllers, we were able to successfully control the car. Since the intent of the algorithm is to increase the safety of the platooning system, the gains were chosen to provide smooth inputs to the controller. In this way, the response of the car imitates the control inputs that a human might make. Additionally, a smoothing filter was placed on the inputs coming from the distance estimates from the tracking algorithm.

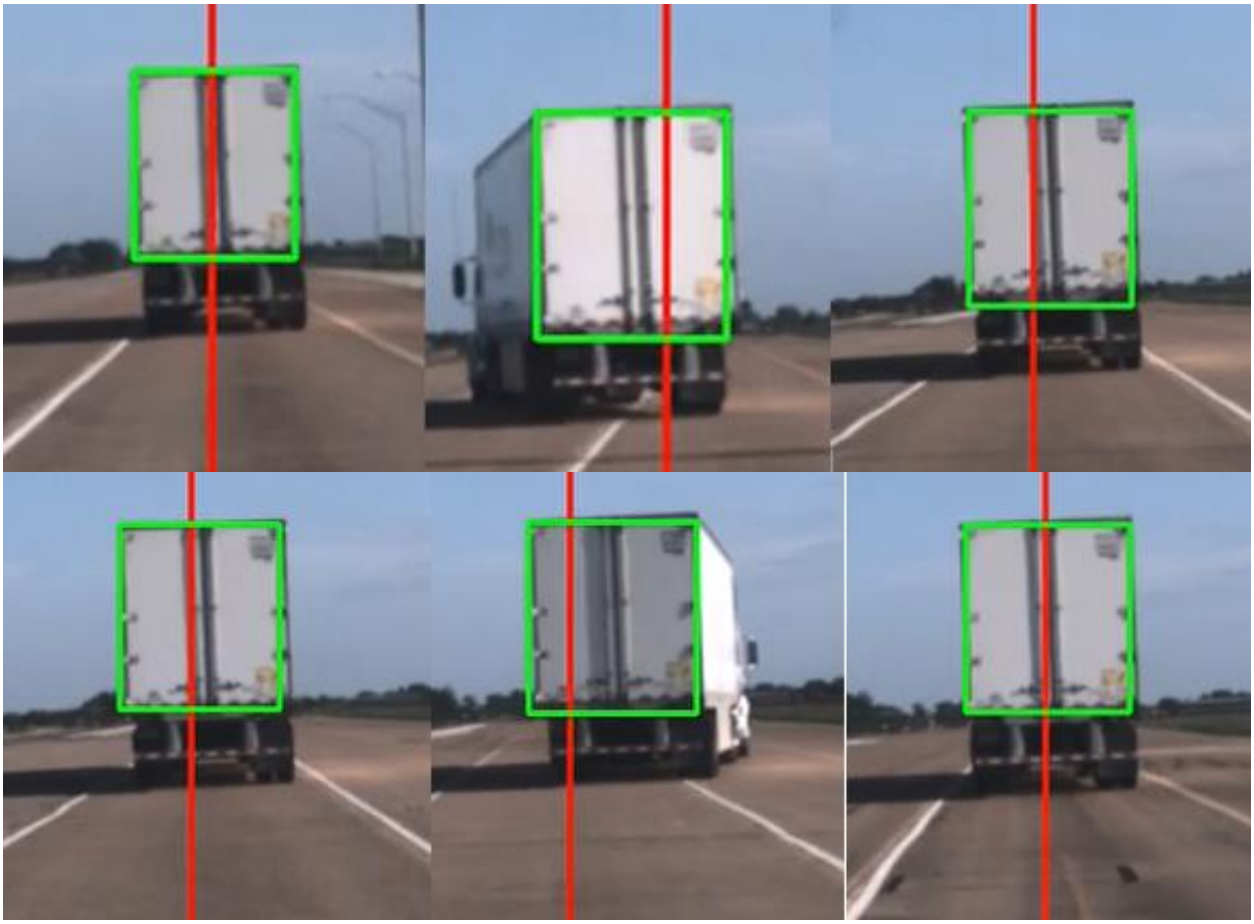
For example, if the car is not at the correct following distance, the controller gradually increases or decreases the desired speed of the car over a period of seconds. This is in contrast to some instantaneous acceleration or deceleration that might attempt to keep a precise following distance. During testing, it was noted that small adjustments to the speed or steering angle were uncomfortable for the passengers in the car. In most cases, these minor adjustments additionally had no noticeable impact on the car's overall response.

Results of Lateral and Longitudinal Control Implementation

After successfully implementing the longitudinal and lateral controllers in the car, we were able to test their responses in a variety of scenarios. In general, the test scenarios took the same common form. First, the detection and tracking algorithm was enabled. Once the car was at a safe following distance and speed, the control algorithm was enabled as well. Once the car was successfully following, the driver of the truck was instructed to make some controlled maneuver to test the car's response.

Lane Change

In one scenario presented earlier, a very basic lane change was performed at approximately 25 miles per hour. Initially, both the car and truck were travelling in a straight line at similar speeds. As the lead vehicle began to change lanes, a fair amount of lateral error developed. The car then steered to follow. Once the lateral error is minimized as shown in Figure 37 (c), the car corrects itself, steering back to the to complete the lane change. Once the gains were tuned, lane changes to both sides were tested multiple times without incident, as shown below.



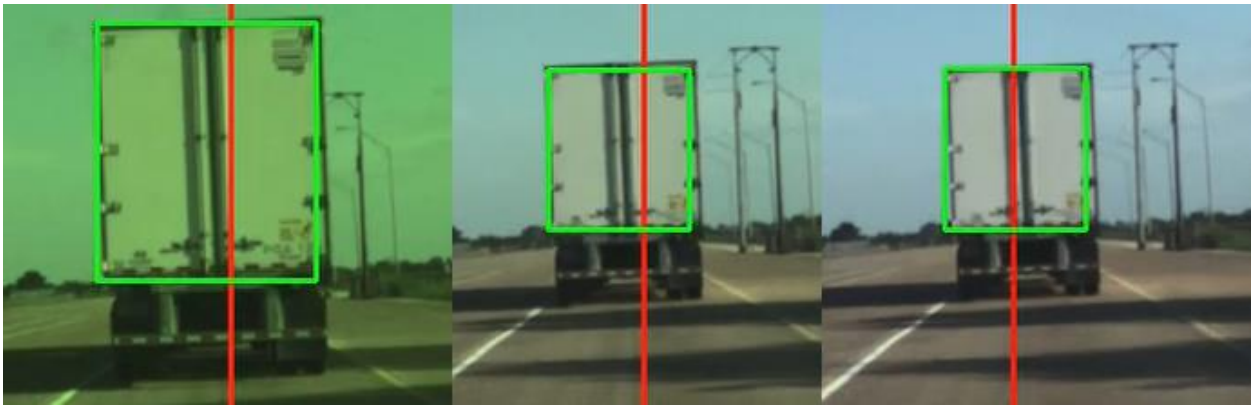
(a) (b) (c)

Figure 37: The lateral controller attempts to keep the middle of the truck centered on the red line. As the truck changes lanes (b) the controller steers to complete the lane change (c)

Stationary Start with Lateral Offset

Another type of maneuver that was tested involved a “start from stop” scenario. With the truck stationary, the car was manually driven both inside of the following distance and offset from the center of the truck. The controller was started, and the car remained stationary while the truck began to pull away.

Once the detected distance reached the target distance of 40 meters, as shown in Figure 38 (b), the car began to both accelerate and steer to reduce the apparent offset. A short time later, the lateral error was corrected, and the car reached the appropriate following distance as shown in Figure 38 (c).

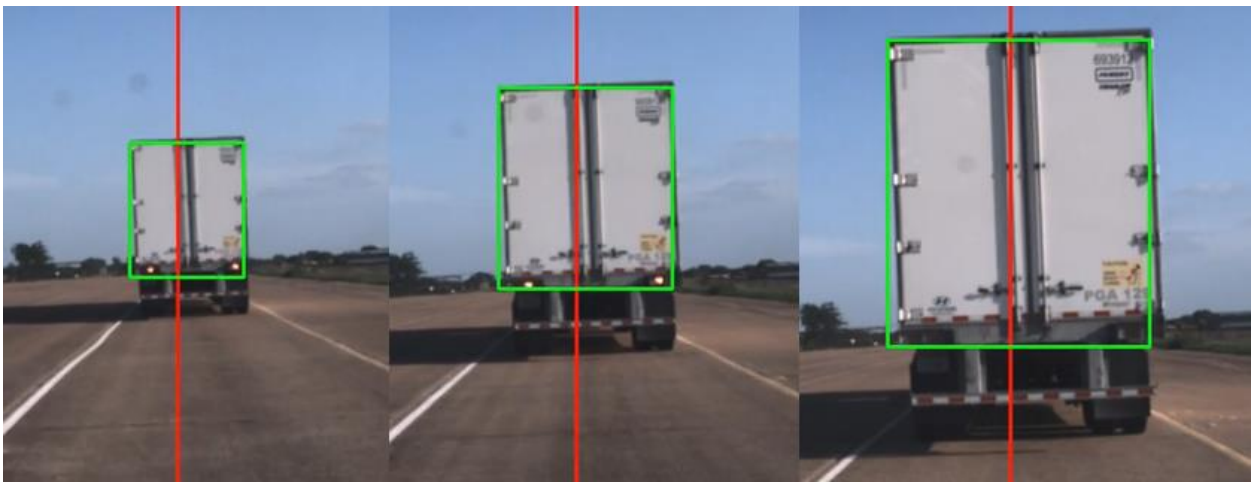


(a) (b) (c)

Figure 38: The truck begins too close to the car with some lateral error (a). Once it reaches the appropriate following distance, the car begins to accelerate (b) and quickly corrects the lateral error (c)

Emergency Stop

After testing the control algorithms in a variety of lane change, acceleration, and deceleration scenarios, we decided to test the algorithms in an emergency stop scenario. In this test, the following distance was set to 40 meters and both vehicles were travelling at approximately 25 miles per hour. When the car had reached the steady state following distance, the driver of the truck was instructed to brake aggressively to a halt. This was meant to imitate a situation in which the truck might need to slow down to avoid a collision. In this scenario, the car reacted by braking appropriately and coming to a stop behind the truck. The result of this test can be seen in Figure 39.



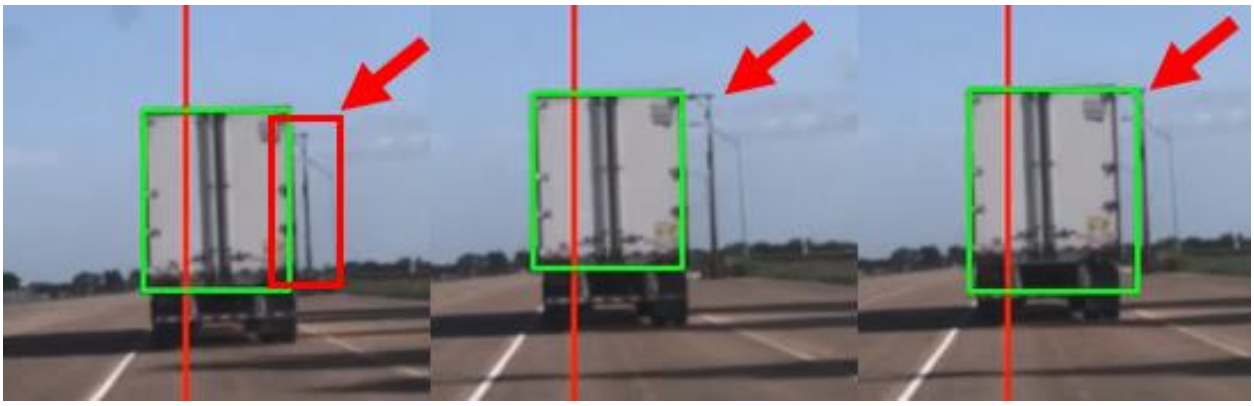
(a) (b) (c)

Figure 39: The truck begins to brake (a). As the car slows down, the tracking algorithm still correctly characterizes the truck (b) until it comes to a halt behind the truck (c)

In Figure 39 (a), the truck begins its aggressive deceleration while the car is still following at 40 meters. The car appropriately responds by continuously tracking the truck while slowing down, and eventually comes to a stop behind the truck, as shown in (c).

Inaccurate Detection and Tracking Drift During Testing

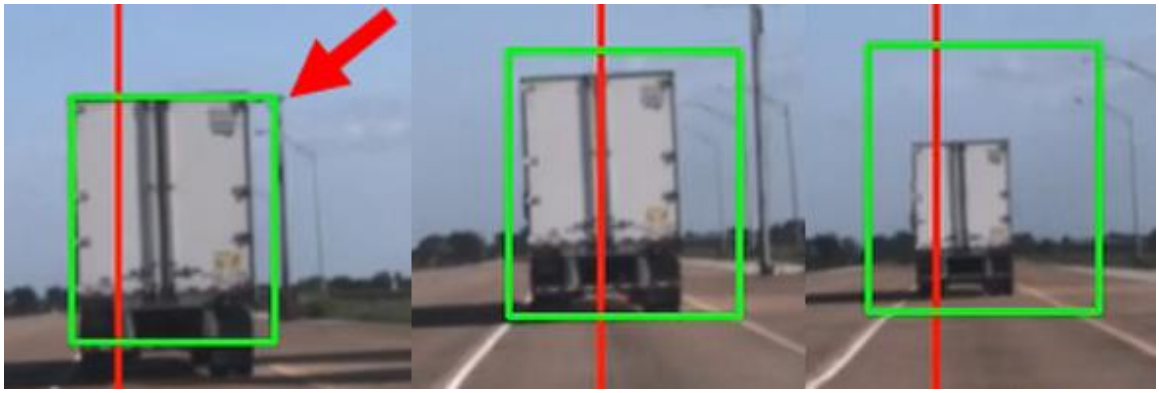
In most of the above cases, the car handled the test scenarios effectively. Before the lateral controller gains were tuned appropriately, the detection algorithm inaccurately identified the truck in one of the runs, causing the algorithm to fail. After reviewing the video, the inaccurate detection was found to be the result of an oddly shaped light pole. Due to the inaccurate detection, the algorithm began to show signs of drift. This unique scenario is shown in Figure 40.



(a) (b) (c)

Figure 40: A square shaped light pole (a) directly lines up with the trailer (b) causing the detection algorithm to mis-identify the truck (c)

After some time, the algorithm began to show signs of drift due to this detection. This drift eventually caused the tracking algorithm to fail, as shown in Figure 41.



(a) (b) (c)

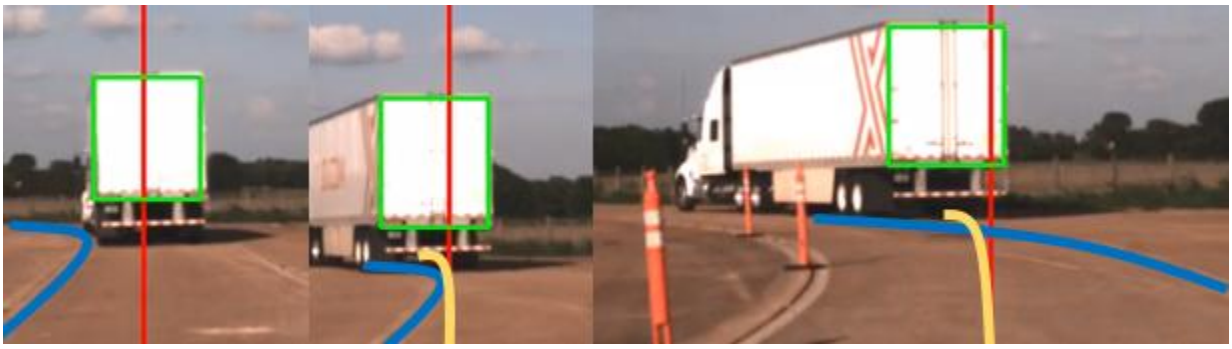
Figure 41: The mis-detection from the light pole (a) resulted in drift in the tracking algorithm, causing it to fail (b), (c)

It should be noted that the detection incident shown in Figure 40 and Figure 41 was an isolated incident. During testing, this mis-detection and resulting failure of the tracking algorithm was not found to be repeatable. As is seen in Figure 38, the runway containing the pole that caused the error was consistently in use, and no similar incidents were recorded.

Error in Lateral Control During Wide Turns

The lateral controller that we developed proved to be quite successful in handling most driving scenarios. In some situations, however, the lateral controller encountered difficulty with wide turns. This is due mainly to the basic structure of the lateral controller and its lack of any type of path planning functionality. Although these wide turn situations are outside of the scope our algorithm's use case, this functionality was tested for robustness.

At an extended following distance and during wide turns, it was noted that the lateral controller began to direct the car straight toward the back of the truck, instead of following an arc-like turn. In these situations, driver intervention was required, and the algorithm required re-initialization. An example of this situation can be seen in Figure 42.



(a) (b) (c)

Figure 42: Stages of a wide left turn in which the apparent path of the car (yellow) does not coincide with the path of the truck (blue)

In Figure 42, the truck makes a wide left turn, following the path represented by the blue line. As it begins its turn in (a), the car appears to adjust accordingly. As the turn becomes sharper as shown in (b), the apparent path of the car, shown in yellow, points directly toward the back of the truck. The car continues to steer straight toward the truck in (c), eventually needing driver

intervention to avoid hitting the orange cones. Due to the lack of a path planning algorithm, this behavior was witnessed every time a large turn was attempted.

Braking and Coasting Functionality for Changes in Longitudinal Distance

To demonstrate the comfort of the longitudinal controller in varying the following distance, we tested the vehicle in a scenario that highlighted both a drastic and a gradual change in following distance. In this test, the controller was activated while the car was accelerating toward the lead vehicle. In addition, the car was much closer to the truck than the intended 40 meter following distance.

When the controller was enabled, the car braked gradually until it reached the correct following distance of 40 meters. This initial braking to achieve the desired following distance caused a drastic difference in speed between the car and the truck. Due to this difference in speed, the car was not able to instantaneously hold the following distance at 40 meters. As a result, there is a fair amount of overshoot in the detected following distance, and the gap increased to further than 40 meters.

To make up this gap, the car began to accelerate gradually, again overshooting the target distance and reaching a following distance of close to 35 meters. As before, this distance is too close. In this case, however, the car did not use the brakes to increase the gap. Instead, it released the accelerator and decelerated naturally with drag and other forces. A few seconds later, the car had settled and was following at a steady state distance of 40 meters. The results of this scenario can be seen in the following figure.

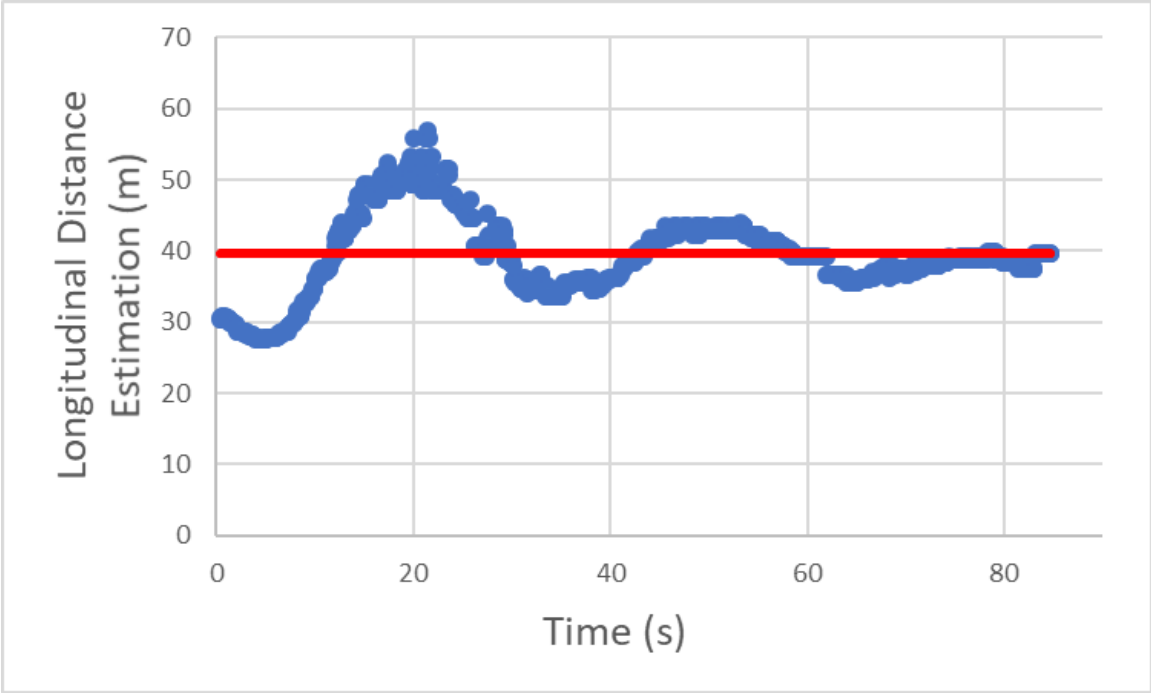


Figure 43: Example results of camera distance estimation while following at a target distance of 40 meters

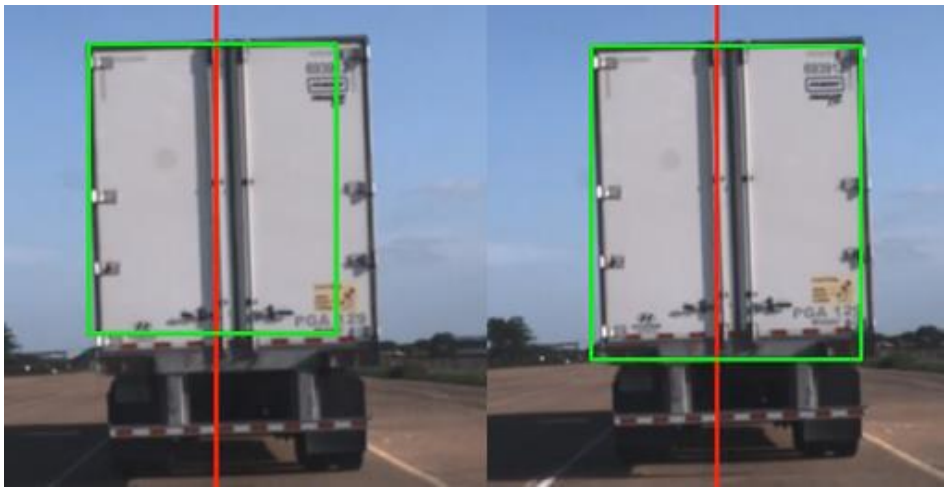
It should be noted that the results shown in Figure 43 are not representative of the car's actual following distance. The chart shows the camera's detected distance with time, which is just an estimation of the true following distance. To validate whether the tracking, detection, and control algorithms were working correctly, it is necessary to compare our results to some form of a ground-truth distance measurement.

4.4 VALIDATION OF GROUND-TRUTH DISTANCE MEASUREMENTS USING PORTABLE GPS UNITS

The previous sections demonstrated that the detection and tracking algorithms were capable of supplying relatively precise information to the longitudinal and lateral controllers. Additionally, the controllers were able to use this data to provide a smooth ride for the car's passengers during normal driving maneuvers. In the current implementation of the algorithm, however, there was no way to validate that the information being passed to the controllers was at all accurate.

Detected Size Variation in Successive Images

As a passenger, the car appeared to follow at a safe, steady distance. The detection algorithm updated the tracked template multiple times per second, each time providing a new range estimation to the car. This range estimation appeared to vary from frame to frame but did not cause any type of sudden acceleration or deceleration due to the filters and the design of the controller. An exaggerated example of the varying range estimation can be seen in Figure 44.



(a) (b)

Figure 44: Exaggerated variation in detected size between frames separated by less than 0.1 seconds

The size variation seen in Figure 44 shows the possible fluctuation in following distance that might be supplied to the controllers. The example shown above is representative of the largest amount of variation seen in any of the tests, which was during the emergency stop scenario. An example of a more standard size variation between frames can be seen in Figure 45.

Figure 45: Normal variation in detected size between frames

The size variation shown in Figure 45 seems relatively small and is on the order of a few pixels. Although seemingly small, this type of change can have a much greater impact on the detected distance to the truck due to the scaling of the focal length. The focal length of the camera in the car is approximately 2850 pixels. Using the distance estimation equation presented previously, this means that a tracking error of 1 pixel can result in a variation of more than 20 centimeters in the longitudinal distance estimation. At a following distance of 40 meters, a five-pixel variation can result in a longitudinal estimation variation of more than one meter. To quantify the amount of error and its impact on the control of the vehicle, we were able to use GPS for distance estimation.

Current GPS Capabilities

Although the precision of the tracking and detection algorithms was suitable enough for successful control of the vehicle, there was no way to verify the accuracy of the detections and resulting movement of the car. Both the car and the truck had previously been equipped with GPS capabilities, which were explored in the development of the control algorithms. After looking into using these sensors for validation of our algorithms, we determined that the installed location of the sensors would not be helpful in determining accurate position data for the car and the truck. The installed locations can be seen in Figure 46.

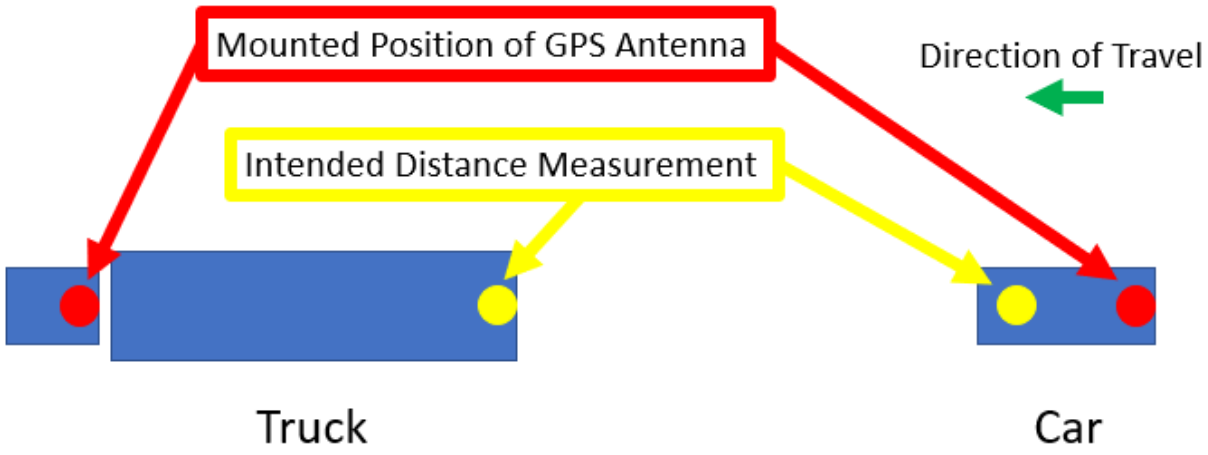


Figure 46: Mounted positions of GPS antennas in car and truck relative to intended distance measurement

Due to their relative positions at the front and rear of each vehicle, gathering concrete position information from these units was not feasible. To adequately capture the intended distance as shown in Figure 46, a great deal of initial estimation would be required to account for the offset between the intended distance and the mounted distance in each vehicle. Additionally, estimating the truck's offset might not be accurate in the case of a lane change or other maneuver. This is

because the front of the truck containing the GPS unit would move drastically before any movement of the trailer is detected.

Portable GPS Units for Accurate Ground-Truth Distance Measurement During Testing

Since the previously installed GPS units were not able to accurately provide the position information required to validate the performance of our algorithms, we installed portable GPS units into both the car and the truck. The portable GPS units, made by Swift Navigation, are small, easily transportable devices that can be used nearly anywhere. As long as there is a source of 12-volt DC power, the GPS units are able to communicate with each other. To record the corresponding GPS data, a computer is additionally required. For best results, the GPS implementation requires a stationary base station positioned toward the middle of the testing area. The satellite units are then mounted to the moving vehicles. Using this setup, we were able to record accurate ground-truth distance information for comparison with the tracking algorithm's data. An example of the testing setup is shown in Figure 47.

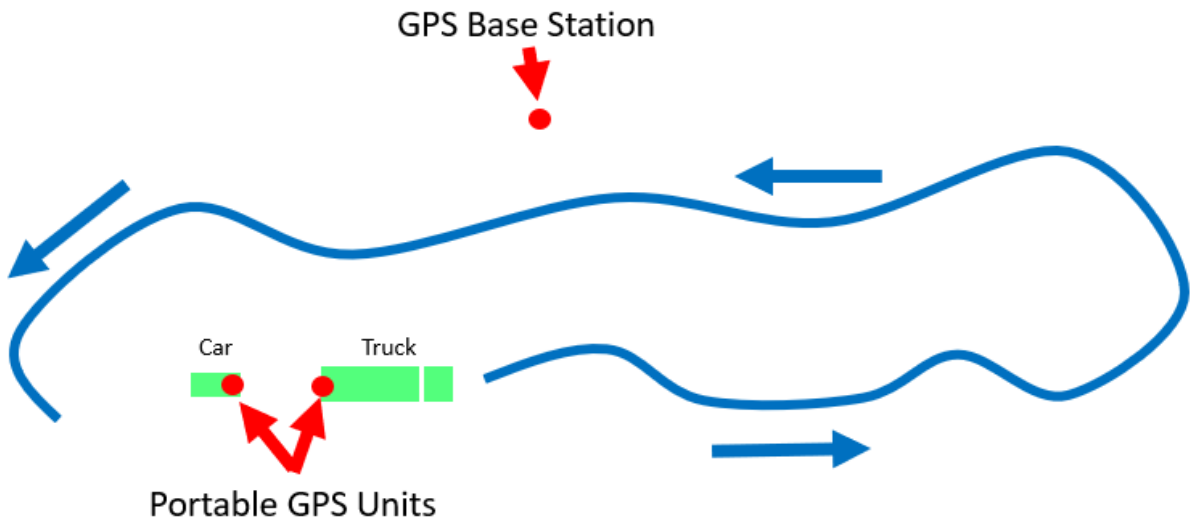


Figure 47: Example implementation of portable GPS units for validation of tracking and control algorithm accuracy

Results of GPS Testing

After testing the detection, tracking, and control algorithms while recording GPS data, we were able to validate the accuracy of our driver assistance system. In one scenario, the car and truck were brought to a steady-state speed and following distance. The truck was then instructed to perform an emergency stop procedure. In this scenario, the car was able to follow the truck as expected. These results can be seen in Figure 48.

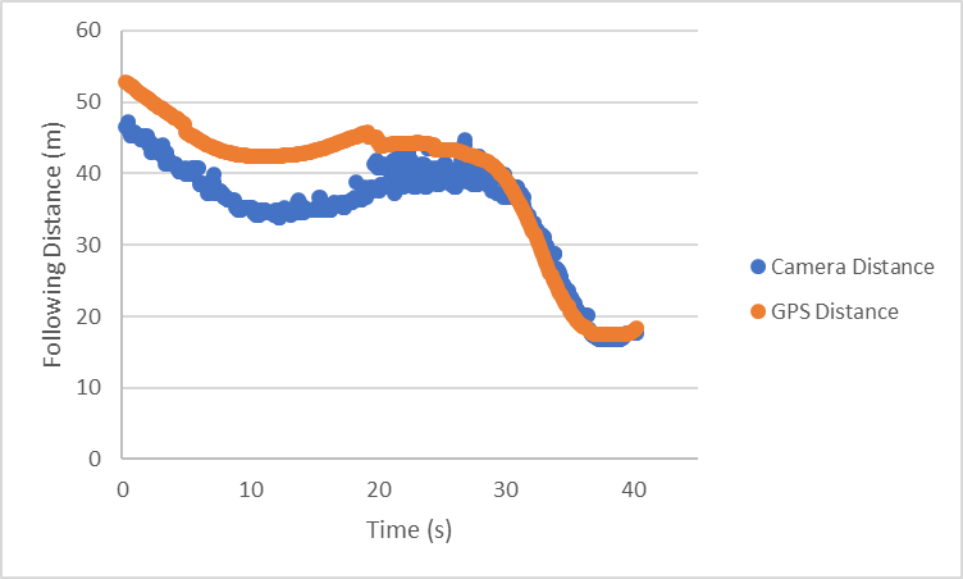


Figure 48: Camera distance estimate and GPS distance during emergency stop scenario

The detection and tracking error at longer following distances can be attributed to a combination of camera calibration and scaling. If the camera was calibrated incorrectly, the inaccurate focal length would cause the estimation error to grow with the following distance. Additionally, it is more difficult to accurately detect the precise width of the truck at long distances. An error of a few pixels in this estimation at long distances could cause quite a large difference in the following distance calculation. At 45 meters, a five-pixel error would cause a longitudinal

distance estimation error of more than 1.5 meters. At a following distance of 15 meters, however, it would only cause the following distance estimation error of close to 15 centimeters.

The GPS coordinates of the lead truck were recorded continuously during one round of testing that included lane changes and emergency stop scenarios. These results can be seen in Figure 49.

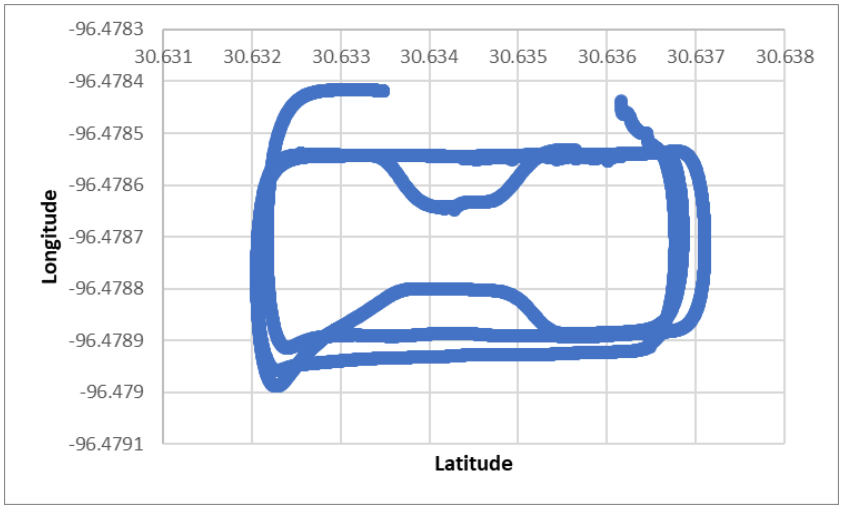


Figure 49: GPS coordinates of lead truck during testing

When overlaying the GPS coordinates taken from the car during multiple tests, it is clear that the lateral controller is quite capable of following the truck in the tested scenarios. These results can be seen in Figure 50.

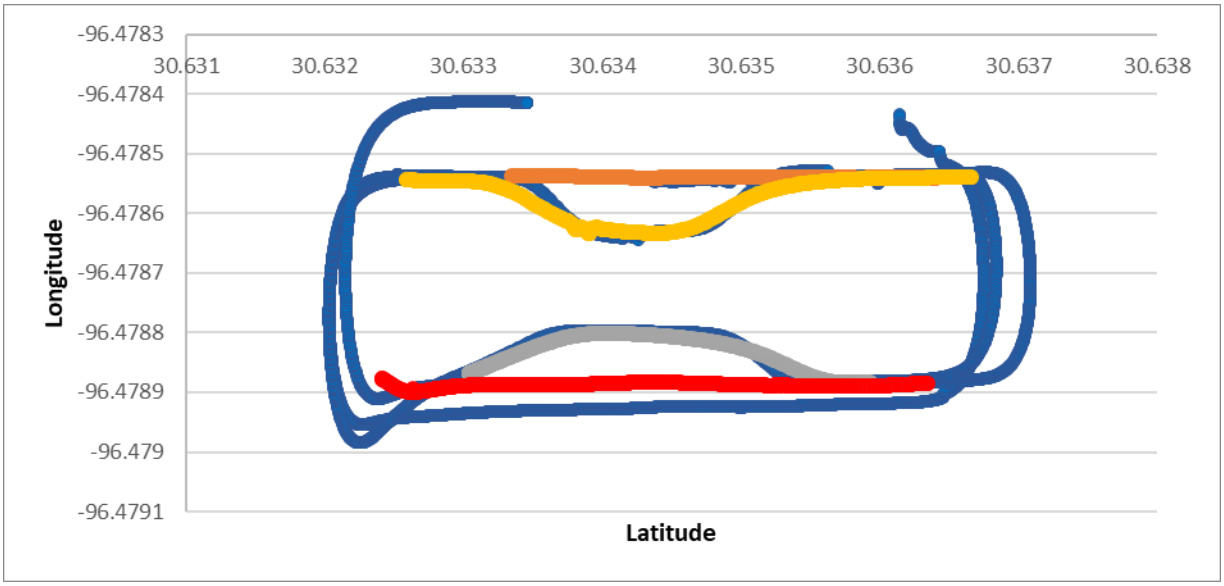


Figure 50: GPS coordinates of both the car and truck during multiple tests

5. CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

The previous chapters have shown an in-depth look at the development of an advanced driver assistance system capable of controlling a drive-by-wire enabled autonomous vehicle using only a camera. The initial intent of the research was to develop a safety system for an autonomously controlled 18-wheeler truck that is engaged in a platooning activity. Due to the relatively close following distances associated with platooning, a control system malfunction due to communication loss with the lead vehicle could result in an extremely unsafe scenario. Although the driver of the following vehicle should be capable of regaining control of the vehicle instantaneously, this may not be the possible in some cases.

To increase the safety of the platooning system, we developed a vision-based tracking, detection, and control algorithm that is capable of safely controlling the platoon's following vehicle in the event of some system failure. The stand-alone system continuously tracks the lead vehicle and provides the precision and speed necessary for control. Once developed, our algorithm was tested against an in-house implementation of a Kernelized Correlation Filtering (KCF) algorithm with noticeable success.

Due to unforeseeable difficulties, we were not able to implement the tracking and detection algorithm in the autonomous truck. As an alternative, we successfully used the algorithm for control of an autonomous car. In testing, the car handled most driving scenarios effectively, including lane changes, emergency stops, and other maneuvers. Due to the lack of a path planning algorithm in the lateral controller, the car regularly encountered difficulty with wide turns at a large following distance.

5.2 FUTURE WORK

Overall, the detection, tracking, and control algorithms presented in this paper demonstrated the successful implementation of an advanced driver assistance system. Once fully implemented, our algorithms were capable of successfully directing a drive-by-wire enabled autonomous car to follow a manually driven commercial vehicle through a variety of scenarios. As discussed above, there is one main area for improvement of the existing algorithm.

Although our algorithms were able to control the car safely and smoothly through a wide range of test scenarios, the structure of the lateral controller did not account for situations involving wide turns and path planning. It should be noted that these turning situations are outside of the scope of our algorithm's use case, but future improvements can be made to increase the robustness of our algorithm. The inability to follow the lead vehicle through wide turns as described in Section 4.3 has room for some improvement, which might be accomplished in several ways.

Lateral Control in Turns Using GPS Waypoints from the Lead Vehicle

In examining the different forms of lateral control available for use with our algorithm, GPS waypoint following was overlooked due to its inherent implementation conflict with our intended use case. The advanced driver assistance system was developed for cases in which the GPS or DSRC communication fails. If the lateral control was based solely upon GPS data transmitted via DSRC from the leading vehicle, the lateral control algorithm can only be used while there is available data from the lead vehicle.

If the GPS or DSRC fails while platooning, the lateral control algorithm would only be able to function based upon the data transmitted before the failure. In other words, the controller would

be able to direct the following vehicle up until the point of GPS or DSRC failure but would thenceforth be useless. To account for this situation, a combination of our current lateral controller and a GPS waypoint controller could be implemented.

In this scenario, the GPS waypoint controller could be used for any time there is a solid GPS and DSRC connection. If the connection fails, the waypoint controller would be used for planning the path of the following vehicle up to the extent of the GPS information. After this point, the currently implemented lateral controller presented in Section 4.1 could be used.

Lateral Control Using Detected Waypoints from Camera

Like the GPS waypoint controller mentioned above, a lateral controller could be developed that is dependent upon detections from the tracking algorithm, as opposed to GPS coordinates from the lead vehicle. In this implementation, the location of the lead vehicle can be localized based upon the size and position of the detected truck. Since the GPS coordinates and heading of the following vehicle is known, the detections can be used to map out a path.

If the current position and heading of the following vehicle is known, a coordinate frame can be established. If the lead vehicle is detected a distance of 40 meters in front of the following vehicle, this position in x-y space can be converted to a GPS coordinate based upon the current heading. In this manner, a continuous sampling of the heading and tracked distance can provide GPS waypoints for use with the lateral controller. An example of this implementation can be seen in Figure 51.

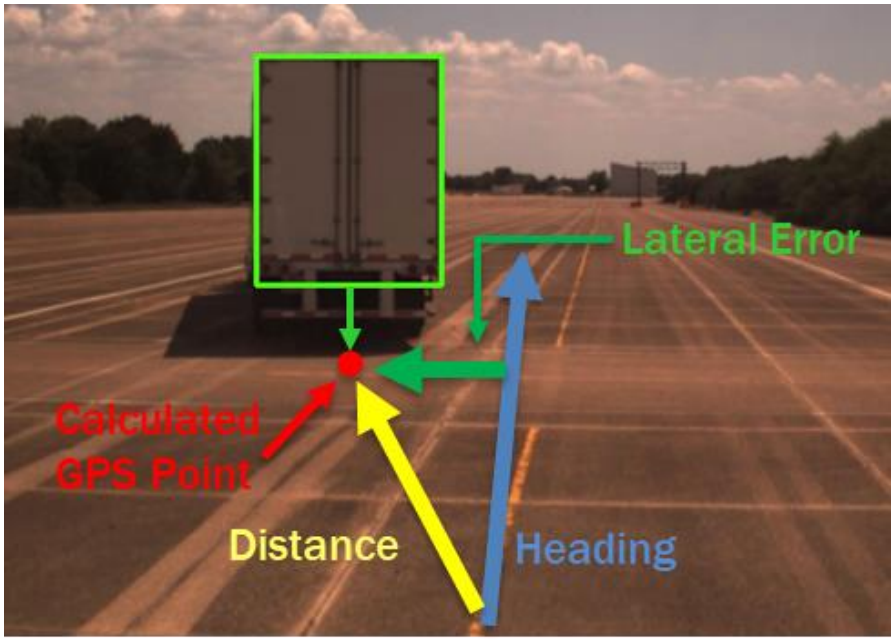


Figure 51: Example implementation of GPS waypoint calculation for lateral control

Lateral Control in Turns Using Object Pose Estimation

A final method for lateral control in scenarios involving large turns is a very popular computer vision algorithm called pose estimation. Pose estimation algorithms are based upon established characteristics of a tracked object and the camera's intrinsic matrix. If an object is known to have a certain size and shape, its 3D position and orientation can be estimated from an image.

Aspects of pose estimation are currently being used in estimating the longitudinal distance to the truck. In addition to simply estimating the truck's distance, a correlation can be drawn between the detected angle of the trailer and the yaw angle of the truck. An example of this scenario can be seen in Figure 52.

(a) (b) (c)

Figure 52: The detected truck (a) with its corresponding Canny edge map (b) and angle estimation (c)

With further research, a correlation can be drawn between the angle θ in Figure 52 (c) and the actual yaw angle of the truck. Building upon the detected waypoints method presented above, some simple calculations can provide the appropriate waypoints and desired headings for the following vehicle.

REFERENCES

[1] M. Schwarzinger, T. Zielke, D. Noll, M. Brauckmann and W. von Seelen, "Vision-based car-following: detection, tracking, and identification," *Proceedings of the Intelligent Vehicles `92 Symposium*, Detroit, MI, 1992, pp. 24-29.

[2] Shih-Shinh Huang, Chung-Jen Chen, Pei-Yung Hsiao and Li-Chen Fu, "On-board vision system for lane recognition and front-vehicle detection to enhance driver's awareness," Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on, 2004, pp. 2456-2461 Vol.3.

[3] Kuo, Ying-Che; Pai, Neng-Sheng; Li, Yen-Feng. In Advances in Nonlinear Dynamics, Computers and Mathematics with Applications. 2011

[4] Ponsa, Daniel; Serrat, Joan; López, Antonio M. Transactions of the Institute of Measurement & Control. Oct 2011, Vol. 33 Issue 7, p783-805. 23p.

[5] Hartley, Richard, and Andrew Zisserman. *Multiple View Geometry in Computer Vision, Cambridge University Press, 2017.*

[6] OpenCV Documentation (2018, August 15). *Color Conversions.* Retrieved from https://docs.opencv.org/3.1.0/de/d25/imgproc_color_conversions.html