Computer Vision Enhancement for a Teleoperated Vehicle

Clay Dodson

ME 8913

Dr. Bras

April 30, 2020

Introduction I.

The future of personal transportation is an autonomous one. McKinsey & Company predicts that 15% of all vehicles sold in 2030 will have full self-driving capability [1]. There are however, many challenges, both technical and societal, that must be solved before autonomous vehicles can become mainstream. Vehicle teleoperation is an emerging technology that eases this transition into full autonomy. Vehicle teleoperation is the concept of an operator controlling a vehicle remotely [2]. To a passenger, a teleoperation vehicle could be indistinguishable from a fully autonomous one. The vehicle in many situations could even be in full self-driving mode. The passenger, however, can have peace of mind knowing that if the driving environment becomes too challenging for self-driving operation, a human operator will take control remotely. One key research area in teleoperation vehicles is improving the human-machine interface [2]. It is vital that the human-machine interface effectively communicates the driving environment to the operator. Many computer vision concepts can be applied to the human-machine interface to aid in operator perception. This paper will focus on a proposed computer vision augmentation system shown in Figure 1. The subsystems covered are camera calibration, lane detection, and object tracking. The testbed used for this research is a golf cart with a front and a rear-facing camera. All calculations are based on input images of 720x480. All code execution times are calculated on a non-GPU accelerated compute system running an Intel i7-1065g7 and 16gb of ram.

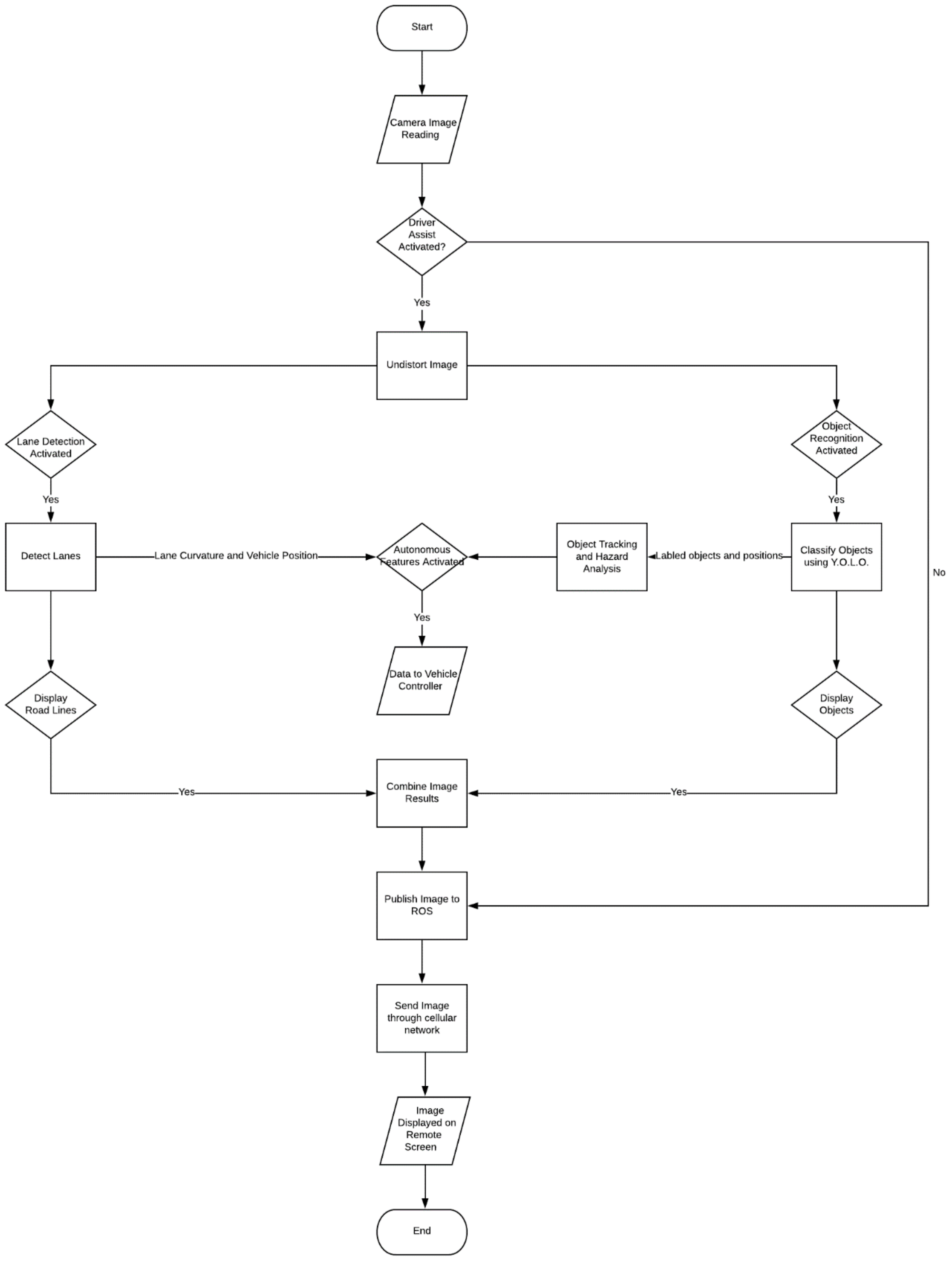
Start
Camera Image Reading
Driver Assist Activated?
Yes
No
Undistort Image
Lane Detection Activated
Object Recognition Activated
Yes
Yes
Detect Lanes
Lane Curvature and Vehicle Position
Autonomous Features Activated
Object Tracking and Hazard Analysis
Labled objects and positions
Classify Objects using Y.O.L.O.
Yes
Data to Vehicle Controller
Display Road Lines
Display Objects
Yes
Combine Image Results
Yes
Publish Image to ROS
Send Image through cellular network
Image Displayed on Remote Screen
End

Figure 1. System Flow for Computer Vision Augmentation System.

II. Related Work

Human-machine interface augmentation for teleoperation. Hosseini and Lienkamp [2] demonstrate a head mounted display for better operator emersion. Housseini and Lienkamp also use a predictive display to more accurately communicate vehicle position to the driver. Neumeier et al. [3] present a simulated human-machine interface that incorporates object detection in the video stream.

Lane Detection. Herman [4] validates a second-order polynomial can approximate lane marking geometry. He et al. [5] create a lane detection method that uses histogram statistics and determine that their approach is superior to previous Hough transform and connected component lane detection methods. Yan, Xuqin, and Li [6] utilize the canny edge detector to enhance their lane detection algorithm. Dhall et al. [7] demonstrate self-driving perception about a track made of cones.

Object Detection. Redmon et al. [8][9] created YOLO, a program that can perform real-time object recognition. Li et al. [10] present an object tracking strategy using a Kalman filter. Chen et al. [11] demonstrate a collision avoidance system on a robot using object tracking.

III. Camera Calibration

The initial step of many computer vision systems is camera calibration. Camera calibration is essential for maintaining dimensional accuracy in the camera view. The system uses two cameras: a front facing Panasonic webcam and a rear facing Logitech webcam. Each camera has its own unique intrinsic and distortion parameters. The camera intrinsic matrix equation is

$$K = \begin{bmatrix} f_x & s & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \quad (1)$$

where f_x and f_y are the focal lengths, p_x and p_y are the principal points, and s is the skew coefficient. The distortion equation is

$$x = \frac{x_{undist} - p_x}{f_x}, y = \frac{y_{undist} - p_y}{f_y}, r = \sqrt{x^2 + y^2}$$

$$x_{n\_dist} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 xy + p_2(r^2 + 2x^2) \quad (2)$$

$$y_{n\_dist} = y(1 + k_1r^2 + k_2r^4 + k_3r^6) + p_1(r^2 + 2y^2) + 2p_2xy \quad (3)$$

$$x_{dist} = f_x x_{n\_dist} + p_x \quad (4)$$

$$y_{dist} = f_y y_{n\_dist} + p_y \quad (5)$$

where k_1, k_2, and k_3 are the radial distortion coefficients and p_1 and p_2 are the tangential distortion coefficients [12].

MATLAB's cameraCalibrator application is used to determine the intrinsic and distortion parameters of each camera. The cameraCalibrator application requires a set of checkerboard images taken by the camera at various orientations. Figure 2 shows an example calibration image.

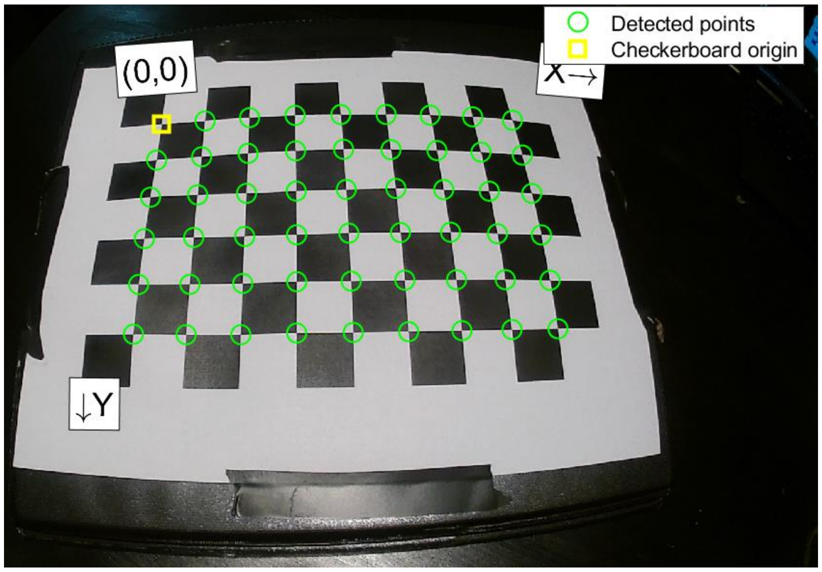

Figure 2. Checkerboard Calibration Image.

The parameters determined by the program are

Panasonic Webcam

$$K = \begin{bmatrix} 370.37 & 0 & 369.44 \\ 0 & 439.86 & 243.60 \\ 0 & 0 & 1.00 \end{bmatrix}$$

$$D = [k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3] = [-0.3721 \quad 0.1316 \quad 0.0006 \quad 0.0004 \quad -0.0206]$$

Logitech Webcam

$$K = \begin{bmatrix} 373.11 & 0 & 359.99 \\ 0 & 443.64 & 240.66 \\ 0 & 0 & 1.00 \end{bmatrix}$$

$$D = [k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3] = [-0.3803 \quad 0.1409 \quad 0.0003 \quad 0.0027 \quad -0.0231]$$

OpenCV's undistort function, using the K and D matrices as parameters, generates a corrected image. Figures 3 and 4 show the before and after correction.

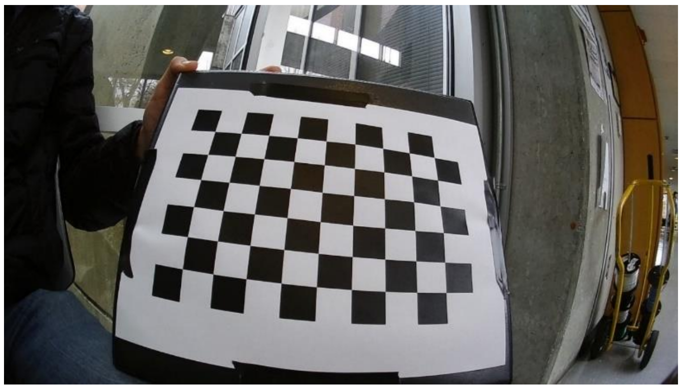

Figure 3. Distorted Image taken with Logitech Webcam.

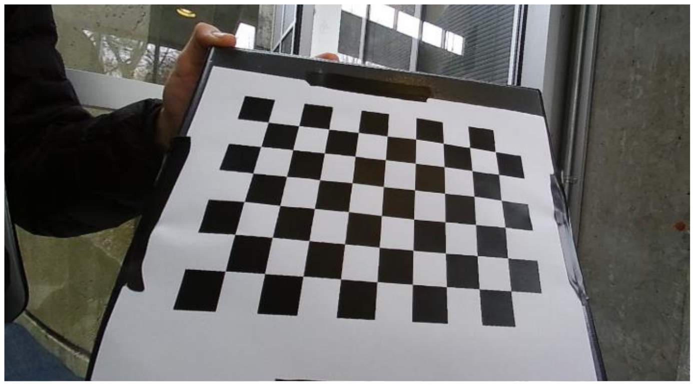

Figure 4. Corrected Logitech Webcam Image using OpenCV.

A loss of peripheral information is a concern with OpenCV's built-in undistort function. In some driving situations, it may be necessary to have as much peripheral vision as possible. A custom undistort function rectifies this issue. The function uses inverse warping, a method where the corrected image pixel location determines the corresponding pixel in the distorted image. After determining the correspondence, the distorted pixel value is copied into the corrected pixel location. Inverse warping ensures there is no image tearing in the transformation. Equations 4 and 5 describe the transformation. Figure 5 shows the new field of view resulting from the correction.

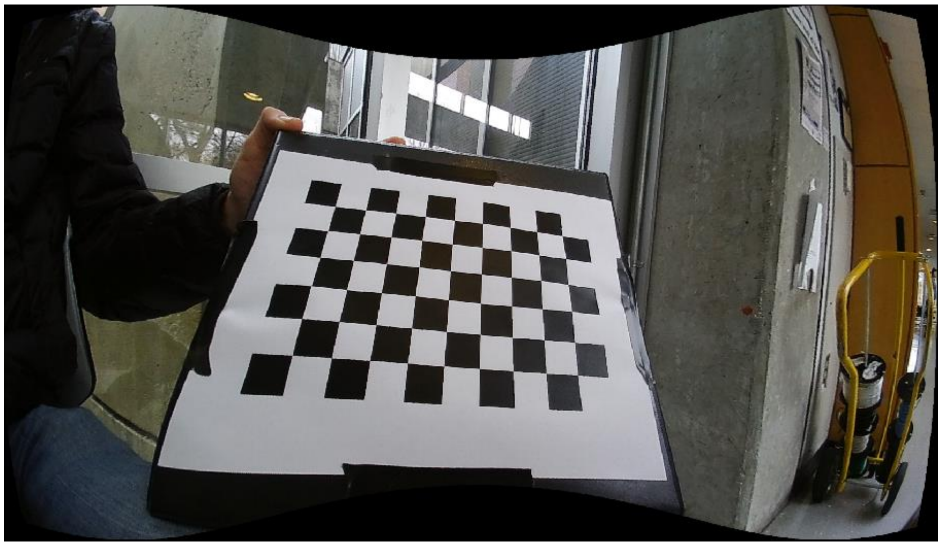

Figure 5. Corrected Image with all Visual Data Preserved.

There is a drawback to the custom undistort function. The execution time of OpenCV's undistort is 0.004 seconds while the custom undistort function is 0.726 seconds. Numba, a GPU acceleration library for Python, could be used to solve this problem by giving a 500-times performance boost. Numba, however, requires a CUDA enabled computer which was not available during testing [13]. Since speed is a necessity, OpenCV's undistort function is selected for the system.

IV. Lane Detection

The lane detection process closely follows the structure of He et al.'s proposed lane detection method [5]; however, there are differences in the line isolation methods. The initial step in the lane detection algorithm is applying the image correction function to the video feed. Figure 6 shows a screenshot of the raw video feed from the front-facing camera. Figure 7 shows this image after the correction is applied.

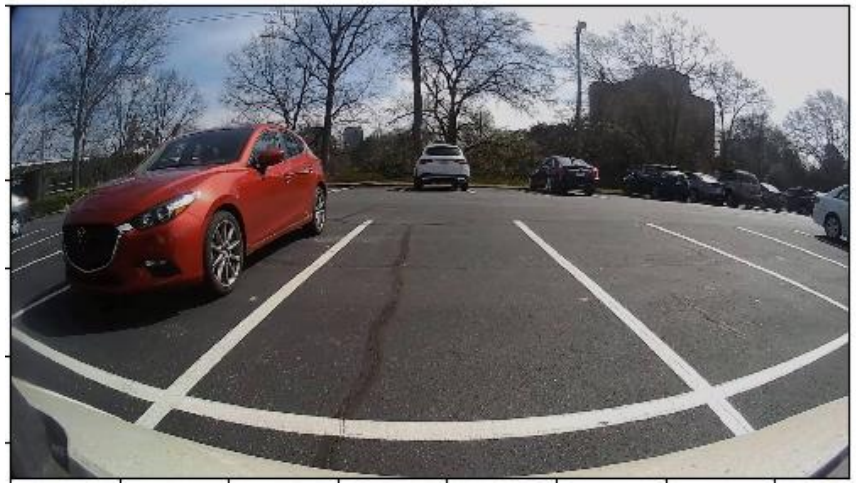

Figure 6. Screenshot of Raw Video Feed.

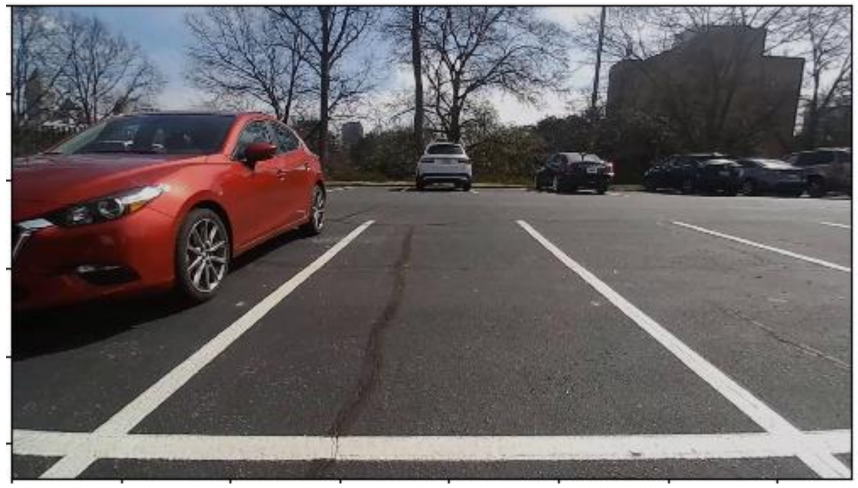

Figure 7. Screenshot of Corrected Video Feed.

With the image corrected, the next step is to create a binary image containing only the lane lines. First, a Gaussian smoothing filter removes noise in the image. The Gaussian filter has a 3x3 shape with a $\sigma = 1$. Second, the color space is converted from RGB to HLS. The HLS color space allows for more accurate yellow and white thresholding in changing lighting conditions [14]. Once converted to HLS, the image is binarized using an L and S channel threshold. The L channel threshold is 150. The S channel threshold is 100. Figure 8 shows the resulting binary mask.

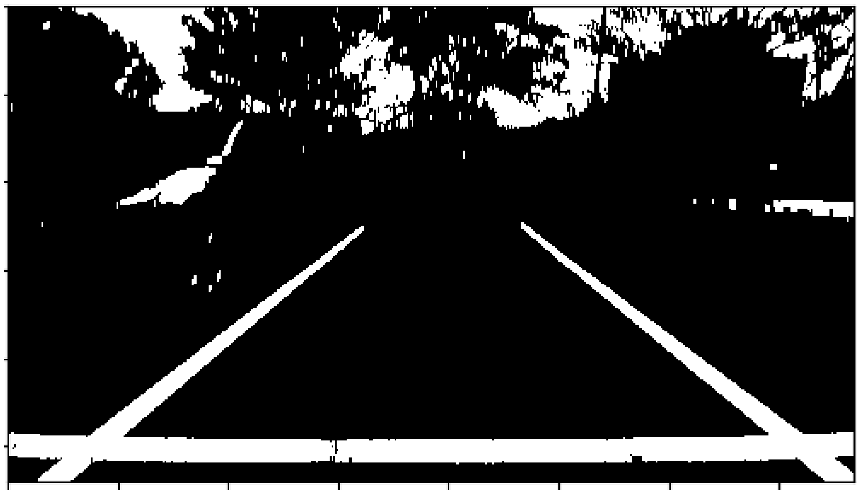

Figure 8. HLS Threshold Mask.

A separate binary mask is created with a Canny edge detector. The lower and upper thresholds of the Canny edge detector are 30 and 100. Figure 9 depicts the result of the Canny edge detector.

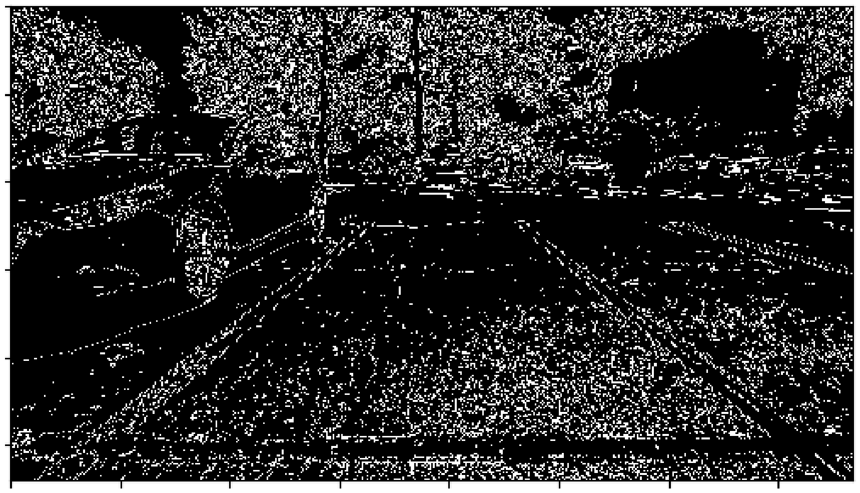

Figure 9. Canny Edge Detection.

A closing operation fills in the lane lines of the Canny edge detector mask. The closing operation consists of a dilate and then an erode operation each with 10 iterations. The structuring element in the closing operation is

$$kernel = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

The shape of the kernel helps fill in vertical lines. Figure 10 shows the result of the closing operation.

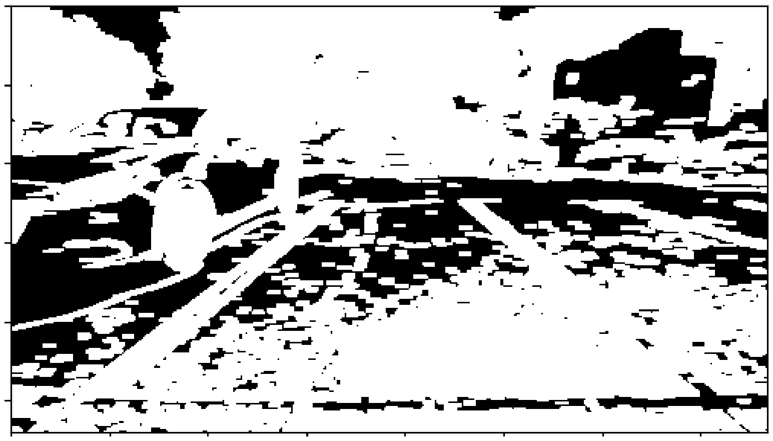

Figure 10. Canny Edge Closing Operation.

In many road situations, such as at intersections, there is a strong horizontal line present in the view. This horizontal line can skew the results of the lane detection. An opening operation isolates the horizontal line. The same kernel used in the closing operation is used in the opening operation. The opening operation consists of 15 erode and dilate iterations. Figure 11 shows the mask of the isolated horizontal line.

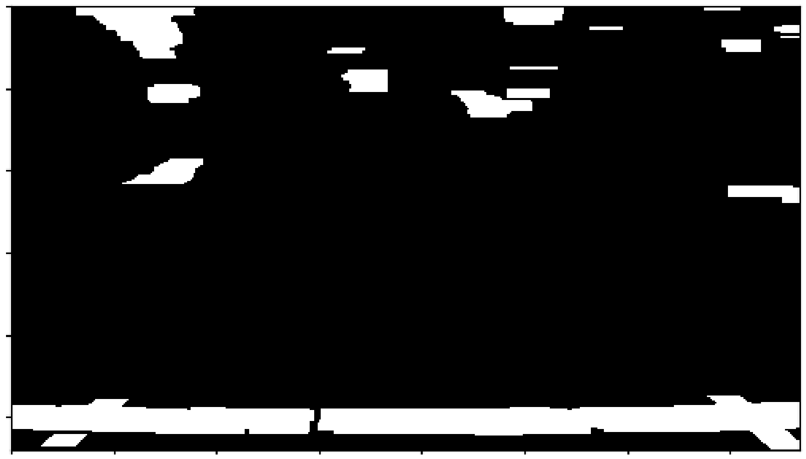

Figure 11. Horizontal Road Line Isolation.

An inverse operation is applied to the horizontal mask, and then the three masks are bitwise ANDed. Figure 12 shows the final binarized line isolation.

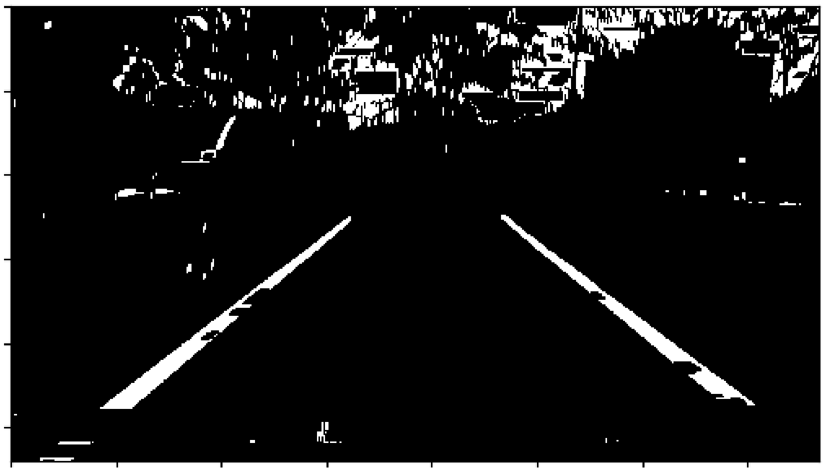

Figure 12. Union of all Masks.

The next step is warping the road into a bird's-eye view. A homography transformation accomplishes the warping. The first step in the homography transformation is selecting a region of interest shown in Figure 13. The region is a trapezoid where the top and bottom sides are horizontal, and the remaining two sides are in line with the perspective. This area in the binary mask is then warped into a 300x300 pixel square, as shown in Figure 14.

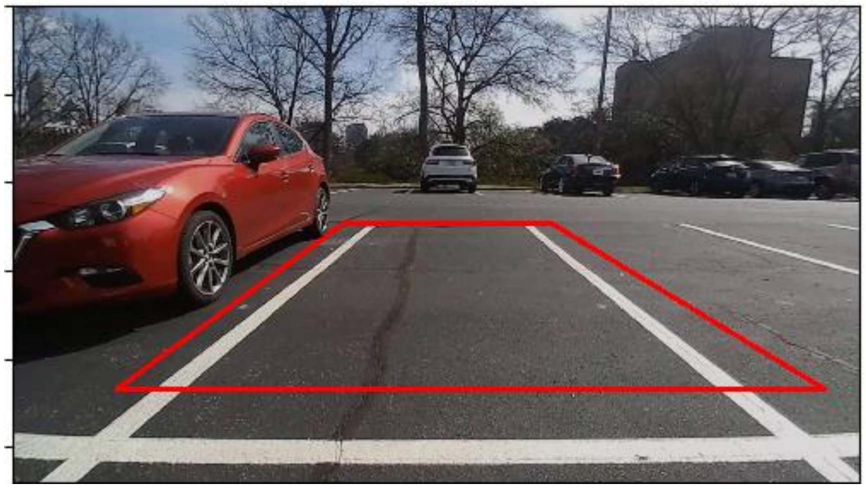

Figure 13. Selected Region of Interest.

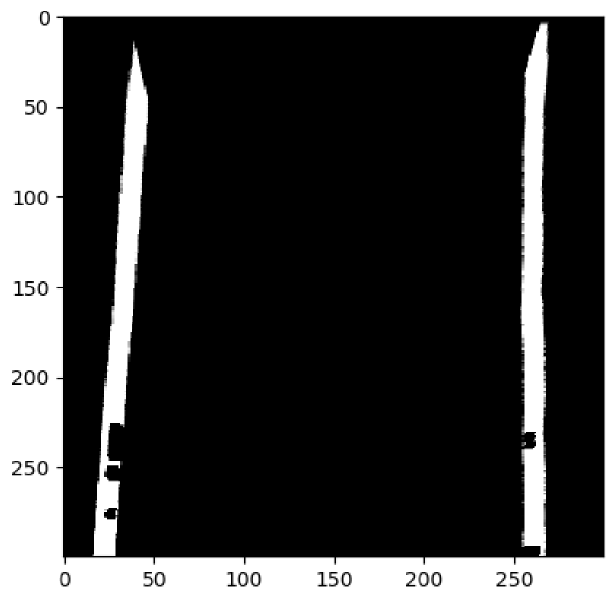

Figure 14. Bird’s-eye View of Lane Lines.

With the road lines isolated in a bird’s-eye view, the next step is to calculate the lane geometry. To calculate the lane geometry, the pixel locations must be grouped into their respective lines. A pixel summation is taken about the columns to get an x-axis density distribution. This operation results in two peaks corresponding to the initial x-position of the two lines. A sliding window algorithm is initialized with the peak locations [5]. The algorithm uses a box window to scan up each road line while selectively grouping pixels. A second order polynomial fit determines the function describing each line grouping. The averaged result of the polynomial fit determines the road centerline. The vehicle’s lateral position can be determined with respect to the centerline. The camera position is centered to ensure that the middle of the image (150 px) corresponds to the vehicle centerline. The lateral position error is recorded in pixels and must be converted into real world measurements. The scale factor determined for the camera position is 12.95 mm per pixel in the x-direction. The lane’s radius of curvature can also be determined by the centerline polynomial. The radius of curvature equation is

$$R = \frac{((1+\dot{x})^2)^{3/2}}{|\ddot{x}|} \text{ [15] (6)}$$

After the calculations, the polynomial fit lines are drawn in the bird’s eye view. The bird’s eye view is then inverse warped and overlaid on the original image. The output image displays values for lane curvature and lateral position error in the upper left corner, as shown in Figure

15. Figure 16 demonstrates the system working on highway conditions. The code execution speed for this subsystem is 0.036 seconds per frame.

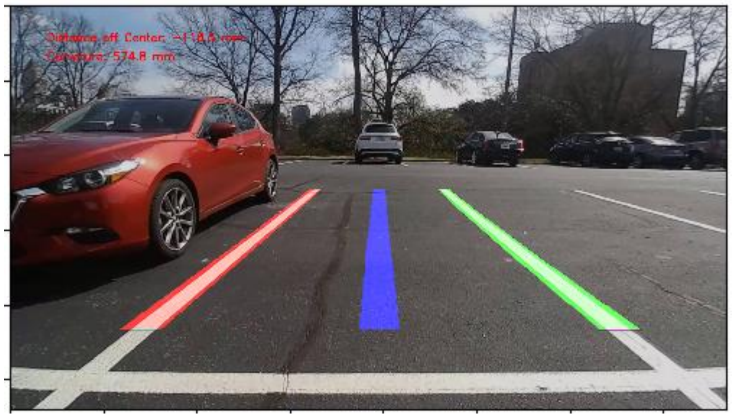

Figure 15. Output of Lane Detection Subsystem.

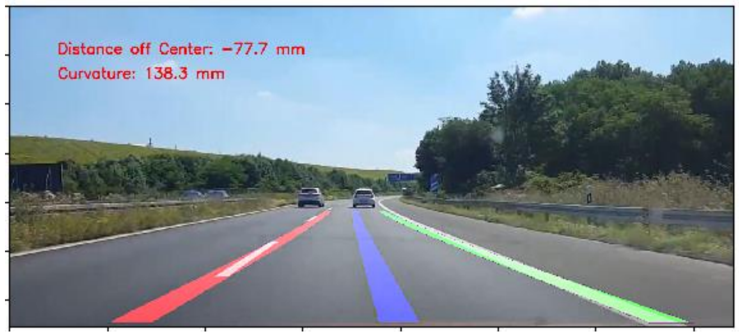

Figure 16. Lane Detection in Highway Scenario. Note: Values in upper left corner have not been calibrated for this image. The test image source is [23].

V. Cone Detection

To test the lane detection subsystem, a considerable amount of time and tape was used to create test tracks. Instead of using lines to draw out the test track, cones could instead be used. Cone test tracks would be significantly faster to build, alter, and teardown. Two adjustments are made to allow the previous lane detection system to work with cone lane markers. The first adjustment replaces the bitwise line mask with cone locations.

YOLOv3 is a fast object recognition algorithm that has found popularity due to its real-time recognition [9]. The standard YOLOv3 model can detect 75 objects trained from the COCO data set and can be trained to detect new objects such as traffic cones [16]. Mark Dana [17] trained and created a program to detect traffic cones based on YOLOv3. Dana's program is used as the base for the cone detection and is altered to be compatible with ROS. The output of the new code is a ROS message containing an array of boundary boxes corresponding to detected cones. The cone lane detection program takes these boundary boxes as inputs. It is assumed that the bottom edge midpoint of each boundary box is the ground location of the cone. Ground location points that fall in the region of interest are warped into the bird-eye view coordinate plane. The next step is to group the location points into their respective lines.

The sliding window approach performed poorly with grouping the sparse cone points. A new approach is taken. The new method uses the slope to optimize the search area orientation. Figure 17 depicts a visual representation of the algorithm.

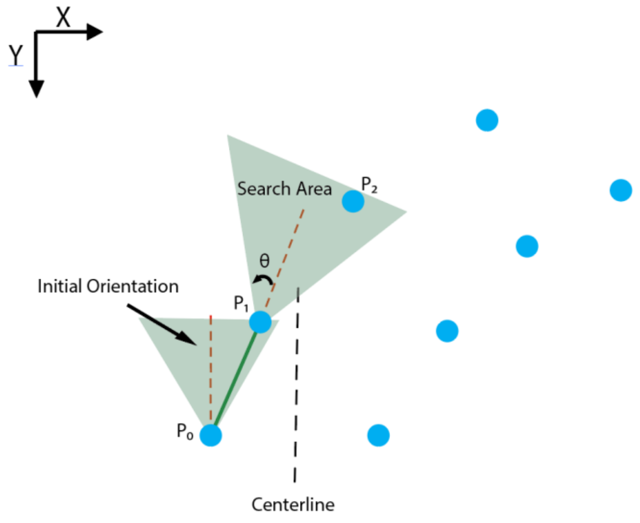

Figure 17. Visual Representation of the Point Grouping Method.

The algorithm starts with a set of points sorted from largest to smallest y-value. The point with the largest y-value left of the image centerline is selected as the initial starting point P_0 . Then, a viewing angle θ generates a search area. The algorithm searches for the point that has the next largest y-value and is within the search area. This point is P_1. If that point is found, the slope angle of segment $\overline{P_0P_1}$ is used to reorient the next search region. This process continues until there are no more points detected. The assumptions of this algorithm are that the vehicle is in between the initial points, there are only 2 line groupings, each grouping contains at least 2 points, and the lines trend in the negative y-direction. The new grouping of points is then fitted with a polynomial curve, and the remainder of the program follows the previously described lane detection subsystem. An example output for the cone lane detection system is shown in Figure 18. The execution speed for this subsystem, including the YOLOv3 detection, is 0.077 seconds per frame. If YOLOv3 detection is not factored in, the execution speed is 0.011 seconds per frame.

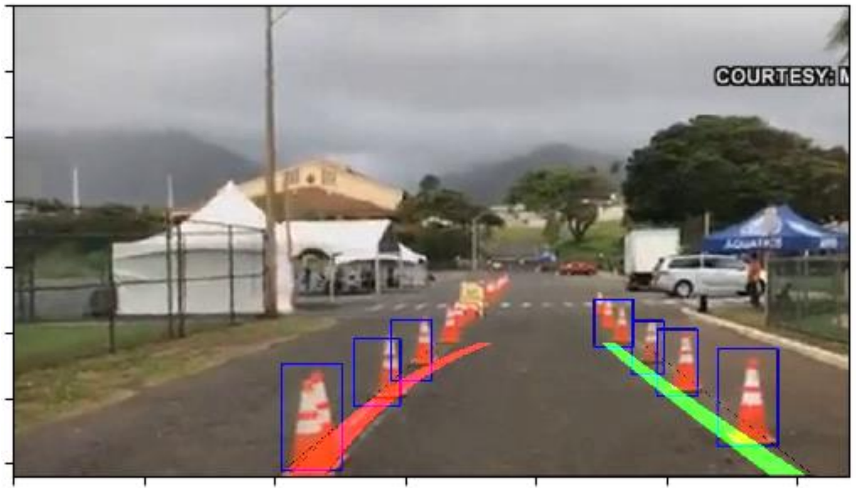

Figure 18. Cone Lane Detection Example. The test image source is [25].

VI. Object Tracking

An essential aspect of vehicle operation is the ability to react to the environment. There are many situations where obstacles cross the vehicle's path, and evasive maneuvers must be taken. It is especially crucial in a teleoperated environment that the operator can distinguish potential hazards.

The defining feature of the YOLOv3 algorithm is it only uses one frame of data to determine objects [8]. This defining feature, however, makes it impossible to track the future object trajectories using YOLOv3 alone. The tracking subsystem builds off YOLOv3 by incorporating additional functions to track and estimate future states of detected objects. The program is initialized with the results of YOLOv3. The results of YOLOv3 are a list of object labels (i.e. Person, Car, Ball) and their respective bounding boxes. Each YOLO object is converted into a tracking object. The tracking object properties are name, bounding box, tracker, Kalman filter, and threat level. When a new tracking object is created, its tracker property is initiated using OpenCV's built in CSRT tracker. The CSRT tracker is selected due to its high bounding box accuracy [18]. The CSRT tracker estimates bounding boxes for each new frame; however, it cannot predict future object states. The Kalman filter can predict the future path of the object. "A Multiple Object Tracking Method Using Kalman Filter" [10] describes the Kalman filter structure used in this subsystem. The object is highlighted green if the object has no potential to cross the vehicle's path, as shown in Figure 19. The object is highlighted orange if it can potentially enter the vehicle's path, as shown in Figure 20. Figure 21 shows the behind the scenes processing to determine potential hazards. The object is highlighted red if it is currently in the vehicle's path, as shown in Figure 22. The tracking subsystem reads the results of YOLOv3 every 10 frames to recalibrate the trackers and remove any objects no longer present. The average execution time is 0.037 seconds per frame.

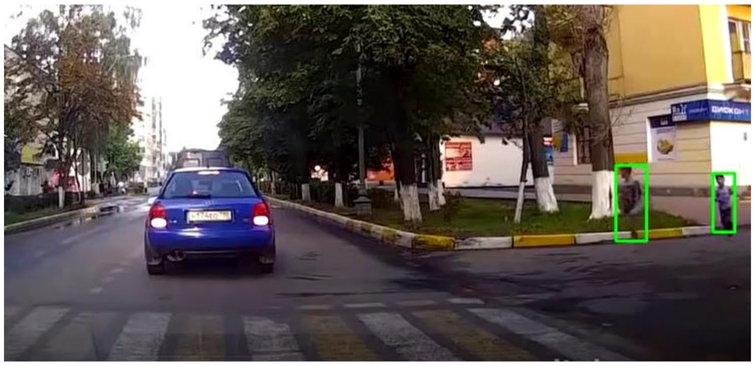

Figure 19. No Threat Tracking. Test video source is [26].

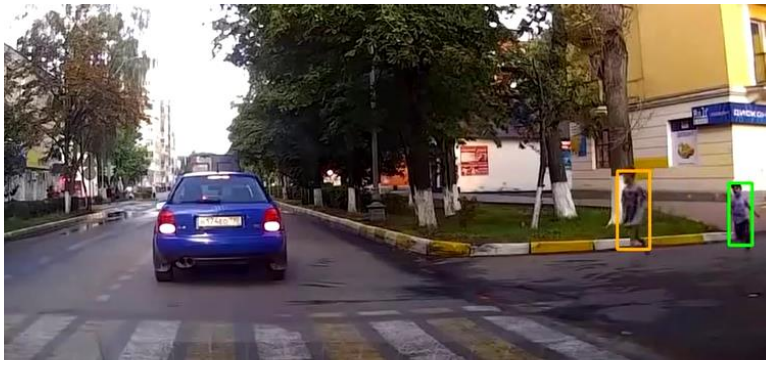

Figure 20. Potential Hazard Tracking.

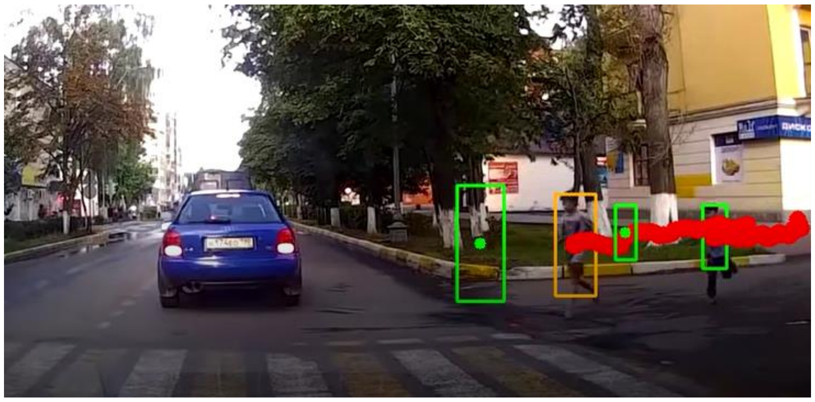

Figure 21. Path Prediction.

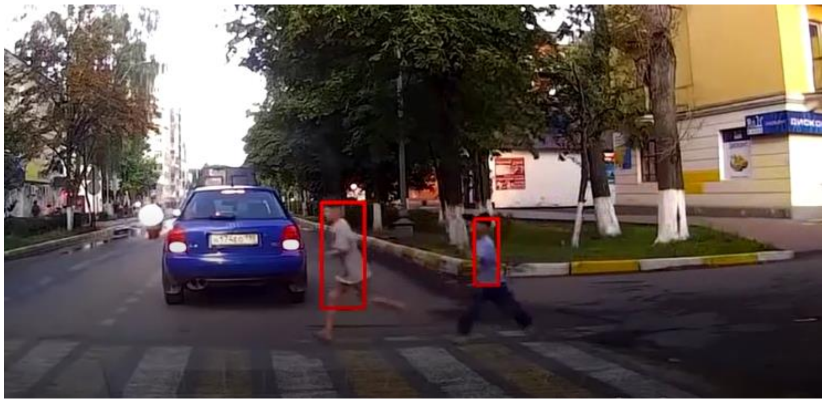

Figure 22. Hazard Tracking. Test video source is [26].

VII. Conclusion and Future Work

This paper describes a computer vision augmentation system for teleoperation vehicles. The system utilizes camera calibration to get an accurate view of the scene. This scene is further processed by the lane detection subsystem. The subsystem determines the road geometry and the vehicle’s lateral position within the lane and works with roads marked by bright lines or cones. The average video playback for lane detection is 28 frames per second. Finally, the object detection subsystem aids the operator in highlighting possible driving hazards. The average framerate for object detection is 27 frames per second. Figure 23 shows the combined object and lane detection view. The average framerate of the combined lane and object detection is 24 frames per second. The display of the system can be customized such as turning on or off the augmented reality view, changing item colors, and changing the collision threat sensitivity.

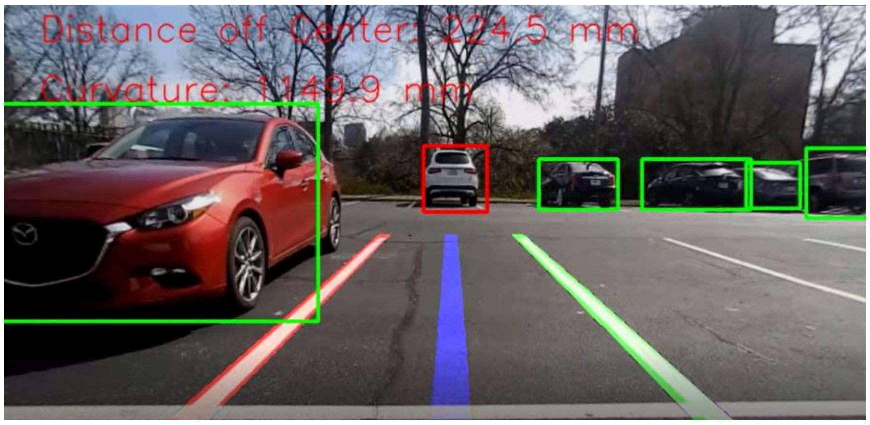

Figure 23. Combined Lane and Object Detection.

Several system improvements can be made in the future. The first improvement is execution time. The code is primarily written in Python, an interpreted language. The code could be rewritten natively in C++, a compiled language, for faster execution. The system also needs to be further evaluated on a graphics accelerated system. The primary contributor to the code's execution time is the matrix operations. Matrix operations run significantly faster on a graphics processor (GPU) than a CPU. Bowley [18] demonstrates that OpenCV can run 3,700% faster on a CUDA enabled GPU. The YOLOv3 program can also run upwards of 49,990% faster on a CUDA enabled GPU [15].

An additional subsystem could be created for a semi-autonomous driver-assist feature. A current strategy for a loss in communication between the operator and vehicle is to brake [23]. Instead of braking when a loss of communication occurs, the vehicle could enter a mode similar to the Cadillac Super Cruise, where the vehicle continues to travel while autonomously lane-keeping [20]. This mode could also determine a safe area to pull off and stop. The lane detection subsystem gives both lateral vehicle position and road geometry, which are parameters necessary for a lane keep controller [21]. The controller could also take inputs from the object detection subsystem to slow down or stop the vehicle if a hazard is present. The object detection subsystem could also be used for an object following system. The object following system would use visual servoing to follow behind an object such as a lead car [22].

References

[1] Collins, Luke, et al. “Self-Driving Cars and the Future of the Auto Sector.” *McKinsey & Company*, McKinsey & Company, Aug. 2016, www.mckinsey.com/industries/automotive-and-assembly/our-insights/self-driving-cars-and-the-future-of-the-auto-sector.

[2] A. Hosseini and M. Lienkamp, "Enhancing telepresence during the teleoperation of road vehicles using HMD-based mixed reality," *2016 IEEE Intelligent Vehicles Symposium (IV)*, Gothenburg, 2016, pp. 1366-1373.

[3] Stefan Neumeier, Michael Höpp, Christian Facchi, "Yet Another Driving Simulator OpenROUTS3D: The Driving Simulator for Teleoperated Driving", *Connected Vehicles and Expo (ICCVE) 2019 IEEE International Conference on*, pp. 1-6, 2019.

[4] Herman, Martin, et al. “Visual Road Following without 3D Reconstruction.” *SPIEDigitalLibrary*, International Society for Optics and Photonics, 25 Feb. 1994, www.spiedigitallibrary.org/conference-proceedings-of-spie/2103/1/Visual-road-following-without-3D-reconstruction/10.1117/12.169472.full.

[5] J. He, S. Sun, D. Zhang, G. Wang and C. Zhang, "Lane Detection for Track-following Based on Histogram Statistics," *2019 IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC)*, Xi'an, China, 2019, pp. 1-2.

[6] Yan, Xuqin, and Yanqiang Li. “A Method of Lane Edge Detection Based on Canny Algorithm.” *2017 Chinese Automation Congress (CAC)*, 2017, doi:10.1109/cac.2017.8243122.

[7] Dhall, Ankit, et al. “Real-Time 3D Traffic Cone Detection for Autonomous Driving.” *2019 IEEE Intelligent Vehicles Symposium (IV)*, 6 Feb. 2019, doi:10.1109/ivs.2019.8814089.

[8] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 779–788, 2016.

[9] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. CoRR, abs/1804.02767, 2018.

[10] Li, Xin, et al. “A Multiple Object Tracking Method Using Kalman Filter.” *The 2010 IEEE International Conference on Information and Automation*, 20 June 2010, doi:10.1109/icinfa.2010.5512258.

[11] Chen, Chung-Hao, et al. “A Moving Object Tracked by A Mobile Robot with Real-Time Obstacles Avoidance Capacity.” *18th International Conference on Pattern Recognition (ICPR06)*, Aug. 2006, doi:10.1109/icpr.2006.106. [11] Zhang, Z. “A Flexible New Technique for Camera Calibration.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 22, No. 11, 2000, pp. 1330–1334.

[12] Zhang, Z. “A Flexible New Technique for Camera Calibration.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*. Vol. 22, No. 11, 2000, pp. 1330–1334.

[13] M. Rocklin, “Python, Performance, and GPUs,” *Towards Data Science*, 28-Jun-2019. [Online]. Available: https://towardsdatascience.com/python-performance-and-gpus-1be860ffd58d. [Accessed: 22-Apr-2020].

[14] S. Raviteja and R. Shanmughasundaram, "Advanced Driver Assistance System (ADAS)," 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India, 2018, pp. 737-740.

[15] Svirin, Alex. “Curvature and Radius of Curvature.” *Math24*, math24.Net, 2020, www.math24.net/curvature-radius/.

[16] Redmon, Joseph. *YOLO: Real-Time Object Detection*, pjreddie.com/darknet/yolo/.

[17] Dana, Mark. “Real Time Cone Detection.” *GitHub*, GitHub, Inc., 14 Mar. 2019, https://github.com/MarkDana/RealtimeConeDetection.

[18] Rosebrock , Adrian. “OpenCV Object Tracking.” *PyImageSearch*, PyImageSearch, 30 July 2018, www.pyimagesearch.com/2018/07/30/opencv-object-tracking/.

[19] J. Bowley, “OpenCV 3.4 GPU CUDA Performance Comparison (nvidia vs intel),” *James Bowley*, 28-Feb-2018. [Online]. Available: https://jamesbowley.co.uk/opencv-3-4-gpu-cuda-performance-comparison-nvidia-vs-intel/. [Accessed: 27-Apr-2020].

[20]“SUPER CRUISE,” *Cadillac*, 2020. [Online]. Available: https://www.cadillac.com/ownership/vehicle-technology/super-cruise. [Accessed: 28-Apr-2020].

[21] A. Benine-Neto, S. Scalzi, S. Mammar and M. Netto, "Dynamic controller for lane keeping and obstacle avoidance assistance system," *13th International IEEE Conference on Intelligent Transportation Systems*, Funchal, 2010, pp. 1363-1368.

[22] K. Usher, P. Ridley, and P. Corke, “Visual servoing of a car-like vehicle - an application of omnidirectional vision,” *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, Sep. 2003.

[23] P. M. Dorey, A. Hosseini, J. Azevedo, F. Diermeyer, M. Ferreira, and M. Lienkamp, “Hail-a-Drone: Enabling teleoperated taxi fleets,” *2016 IEEE Intelligent Vehicles Symposium (IV)*, 2016.

[24] Toyota GT86 DRIVER. “"Hood View Drive," *YouTube*, July, 2016. [Video file]. Available: https://www.youtube.com/watch?v=UI6RpagY86U [Accessed: 22-Apr-2020].

[25] *Cone Lane Maui*. Nexstar Broadcasting, Inc., 2020.

[26] Amazing Vidz Planet. “CRAZY Cross Walk Accidents and CLOSE CALLS – BEST Compilation of 2018,” *YouTube*, July, 2018. [Video file]. Available: https://www.youtube.com/watch?v=dwBb7D91cDA [Accessed: 22-Apr-2020].