COMP9517 Group Project Report

Nathan Yin
School of Computer Science and Engineering @ UNSW Sydney
nathan.yin@student.unsw.edu.au

Usama Sadiq
School of Computer Science and Engineering @ UNSW Sydney
u.sadiq@student.unsw.edu.au

Uttkarsh Sharma
School of Computer Science and Engineering @ UNSW Sydney
uttkarsh.sharma@student.unsw.edu.au

Vishal Bondwal
School of Computer Science and Engineering @ UNSW Sydney
v.bondwal@student.unsw.edu.au

Ngoc Khanh Pham
School of Computer Science and Engineering @ UNSW Sydney
n.pham@student.unsw.edu.au

I. Introduction

We have seen major improvements in self-driving cars in the recent past, and even their adoption in some countries. Models by different manufacturers use different kinds of sensors to perceive the environment they operate in. Some use radar and LIDAR, while others (most notably Tesla), rely only on optical images, the same as a human would see. In this project, we used optical imagery from a single camera (aboard the vehicle) to build a visual perception capability for such a vehicle. We note that the camera is monocular, not stereoscopic, so depth information cannot be directly inferred from a pair of stereo images.

The task had multiple subcomponents, namely, to detect other vehicles, to estimate their distances and velocities, and to do lane detection.

For all these tasks, the TUSimple dataset was used [1]. It is originally from the CVPR 2017 competition as part of the Workshop Autonomous Driving Challenge. The dataset has two components. For velocity estimation, there are 1074 video clips of 2-second length (given as individual frames), while for lane detection, there are approximately 7000 video clips of 20 frames each (each one second long). For the first task, we are also given annotations for the last frame of each of the 1000+ videos. These annotations give the bounding box coordinates of detected cars, as well as their distances and velocities. Bounding boxes are not comprehensive. There are a total of 1442 bounding box annotations for the 1074 clips, though there are many more vehicles.

Real-time lane detection has surged in both popularity and importance since the boom in autonomous vehicles. The detection of lanes is critical to enable autonomous vehicles to position themselves on the road and finds uses in mapping trajectories and scene understanding. We train our model on the TuSimple lane detection dataset, which is populated by 3626 video clips with the corresponding annotations consisting of up to 5 polylines denoting lane instances. We implement the work of Neven et al. [2] by training our deep learning model for lane detection from code referenced in [3]. Lane detection is cast as an instance segmentation problem. Our model identifies distinct instances of lanes from a lane superclass in the image and is verified on the test set of TuSimple.

II. Literature Review

Vehicle detection in images is generally difficult. Vehicle models may vary in appearance, they can be present in different orientations, and changed contrast or illumination can cause the same vehicle to appear different. Background patterns and colours affect both human and digital perception. These problems are not limited to vehicles but are present in many object-detection tasks in computer vision, such as medical imaging or face recognition.

A. Vehicle Detection, Distance and Velocity Estimation

A literature review indicated that the most common application of vehicle detection is in the monitoring of traffic surveillance cameras, for purposes like toll collection or penalising traffic violations. Techniques used include frame difference, background subtraction, part-based feature-based methods, R-CNNs (regional convolutional neural networks), and many others. A good summary of techniques is given in [4].

For distance and velocity estimation, object tracking may be required, for this, feature-detection techniques like SIFT, histogram of oriented gradients (HOG), and Haar-like features are commonly used [4] [5]. For the general task of motion estimation, techniques including motion vectors, optical flow estimation, and background subtraction are used [6] [7] [8].

A highly successful object detector in recent years has been YOLO, and YOLOv3 was ultimately used for vehicle detection [9]. It is a deep learning-based model with 106 layers, of which 53 are CNN layers. It detects objects at 3 layers in the processing chain, to find instances of objects of different sizes. This was used with the Darknet architecture, which is pre-trained on the COCO dataset. COCO stands for 'Common Objects in Context', and has objects from 80 classes, including vehicles [10].

B. Lane Detection

Lane detection models are expected to interpret the same perceptual cues as human beings on the road such as shape,

textural and geometric features. Thus, traditional computer vision methods in lane detection employ highly specialised, hand-crafted data pre-processing pipelines and feature extractors to identify lane instances. Hillel et al. [11] provides a detailed summarisation of traditional methods, which include applying the Kalman Filter [12], colour based features [13] and ridge features [14], Canny edge detection and the Hough Transform. Frequently, a post-processing step is applied to the binary lane segmentation results of a traditional feature extraction to categorize differing lanes.

The binary lane segmentations taken from a traditional computer vision pipeline are prone to robustness issues from scene variation. Likewise, approaches to the post-processing step often use heuristic methods that run into the same problem. Kim et al. [15] propose to alternatively forgo the disentanglement of binary maps with lane instances by casting the problem as a multi-class segmentation. This methodology predicts each lane instance as a separate class but cannot cope with a variable number of lanes outside the designated number of classes. Our chosen methodology is a middle ground between these two approaches. Our model predicts both binary segmentation and lane embeddings in a branched deep neural network. Our inspiring paper [2] then clusters based on a learnt perspective transform to get the lane instances.

III. Methods

A. Vehicle Detection, Distance and Velocity Estimation

1) Detection: We had varied experiences while making our object detectors during the individual component of this project. Basic models tended to have low accuracy, or give lots of false positives. The general method followed was to first build an image classifier that could recognize cars (usually via HOG), and then use sliding windows and image pyramids to use that classifier as an object detector. Manually implementing all these steps was quite inefficient and slow, so we looked for alternatives.

In the field of Computer Vision currently, deep learning is giving the best results for image classification and object detection tasks. Within deep learning models, YOLO is considered best-in-class for object detection and runs in a reasonable time. Thus it was utilised, along with a pretrained model with car weights (Darknet and COCO). Results can be viewed in the later sections.

2) Distance and Velocity: We tried multiple approaches, starting with the pinhole camera model and its intrinsic matrix. If we look at the system from the side and flatten it along one dimension, we can view it as the following schematic diagram. Figure 1.

From the laws of optics (as well as similar triangles), we know that $H/h = d/f$. Using this, we can calculate distance $d = H \times f/h$. This value would need to be adjusted according to the exact intrinsic matrix.

Note that instead of looking at the system from the side, we could also have seen it from overhead, and got a similar schematic, with vehicle width instead of height. However, we chose height instead of width because it does not change much

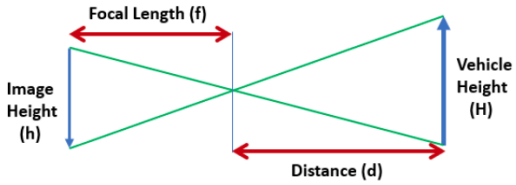

Fig. 1. Distance estimation using Pinhole camera model

with perspective. Vehicles can be in front of us, but also to the side. A tight box around a vehicle to our site would become a trapezoidal shape, tapering towards the vanishing point. The apparent width of its back would also reduce from its true width. Also, some part of its side would become visible, which would end up making a wider bounding box.

Refer to the diagram below, made with the same 3D car model placed in different ways, figure 2:

Fig. 2. Understanding bounding box geometry via 3D Views

The red bounding box represents the rear of the vehicle. In the central figure, it is almost a square. In the figures to the left and right, it is narrower, since the apparent width of the back of the vehicle has reduced. But if we look at green bounding boxes, which represent the whole vehicle, it has widened, because both its side and rear are being seen. We can also see the effect of perspective projection (the convergence of parallel lines at the top and bottom edges of a car). In projective geometry, length and angles are not preserved. Parallel lines converge towards a vanishing point.

However, the height of the vehicle would not change much with lateral movement, since the vehicle is not moving up and down, only left and right. Due to this, we are considering the height of the car for our algorithm, and not the width.

We used the above similar triangles formula on some samples, and found confusing results. It could be due to some mistake at our end, or the actual value in the intrinsic matrix being different. We were also trying other approaches in parallel, and ultimately decided to use machine learning on the given training annotations to find the relationship between bounding box coordinates and distance.

For velocity, we tried background subtraction. However approaches as background subtraction or motion vectors work better when our camera platform is still. If our camera is moving, then static objects in the scene would appear to be moving, creating noise. Cars that are moving at the same speed as us would not be visible, only those that are moving. We can see this below, in subtraction of two frames in Clip 32. Two cars that are right ahead of us are barely visible, while

faster cars in the left and opposing lanes are visible because of relative motion. Background subtraction is useful for finding our velocity, and that can be helpful in the estimation of other velocities, but we went with another approach. Figure 3.

Fig. 3. Background subtraction (Clip 32)

Since we are given training annotations, we trained a machine learning model on location (bounding box) and distance data given in the annotations JSON, to determine their relationship. We got the best results with a custom deep learning architecture, composed of three densely connected layers, implemented in Keras.

After training, the model gives us x and y components of position (distance) as output, given a bounding box coordinates as input. The Earth-coordinate system used for distance is given below (this system is used by aircraft pilots). Figure 12 .

Fig. 4. Ground coordinate system used for distance estimation (image courtesy Ryan Anthony de Belen, Piazza @305)

For velocity, we can simply use the well-known formula $Velocity = Displacement/Time$. The displacement here is the distance travelled by the vehicle, which is the difference between distance estimations across two frames. We know that each clip is 2 seconds long. This gives a video frame rate of 20 frames per second, and time elapsed between two consecutive frames as 50 milliseconds. Note that the above would be the relative velocity of the vehicle to our camera, not its absolute velocity to the road. The annotations also give us relative velocity.

The above can be summarized in the flowchart below (figure 5) .

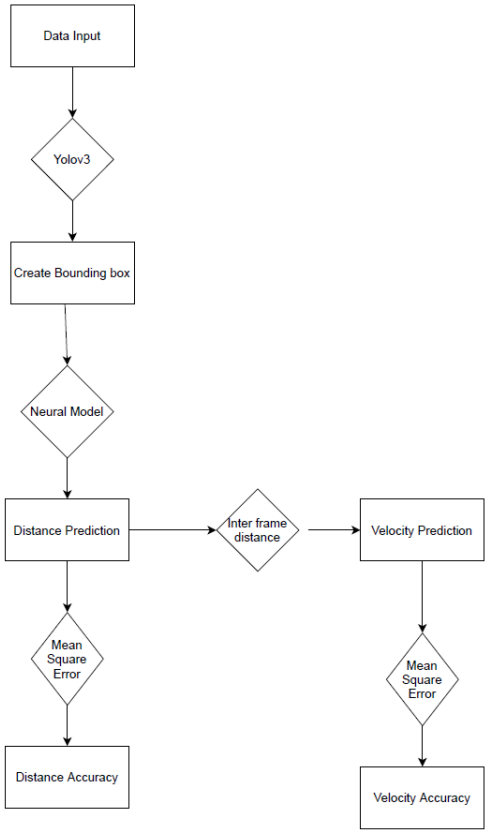

Fig. 5. Distance and velocity estimation flowchart

Since there can be multiple cars in a frame, we need to track individual cars across frames. We implemented object tracking based on Euclidean distances [16].

B. Lane Detection

We reshaped the input image to shape (256, 512) using nearest interpolation as a significant amount of the spatial information was redundant and could be removed whilst texture and shape are retained. Figure 6 visualises the network architecture.

The model architecture re-implements LaneNet from [2], which is an encoder-decoder architecture that produces both lane embeddings and a binary lane segmentation using code from [3]. The model architecture was chosen to have a branched output. The branching factor handles the invariance in the number of predicted classes compared to a multi-class segmentation model. Deep learning was chosen because of the improved robustness to occlusions, lane shape and other scene variations compared to traditional methods.

The pixel embedding branch outputs the logits; the logarithmic probabilities over 5 channels. Each channel represents a potential lane. Thus, combined with a binary segmentation of the lanes in the image, we were able to detect the instances of a lane after clustering. The pixel embeddings for each lane

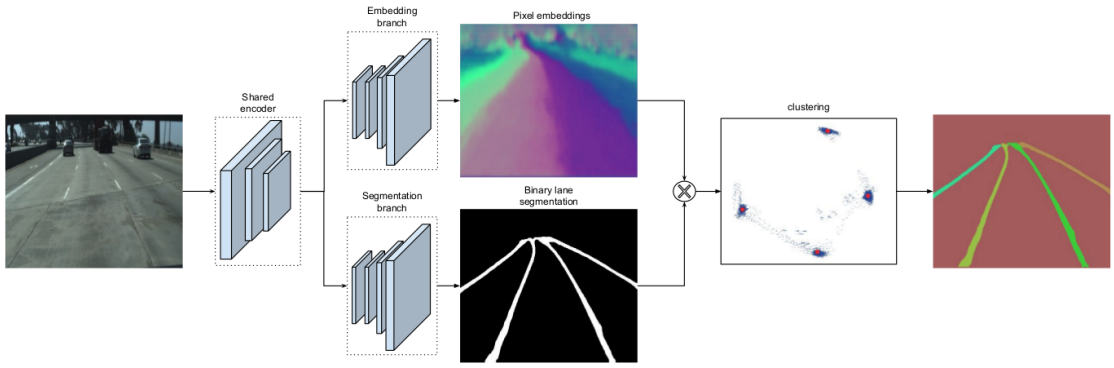

Fig. 6. Lane Instance Segmentation Architecture from [2]

are clustered to disentangle the binary segmentation. The paper implements a one-shot discriminative loss function proposed by Branbandere et al. [17] for clustering.

VGG was used as an encoder architecture, and a fully convolutional network decoder was used as a decoder for both the binary and instance segmentation branches. Neven et al. [2] propose applying a learnt perspective transform and then fitting a 2nd-order polynomial to the lanes. We did not implement the proposed perspective transform or the curve fitting in our project. Instead, our method performs a post-processing step of thresholding the binary segmentation with sizes outside of a specified range.

The post-processing step applies morphological closing to the binary pixel embeddings. This closing of the image fills small holes whilst preserving the shape of the binary segmentation. Then, the Hough transform is applied to the closed image to detect the lines or parametric curve of the lane. Finally, connect component analysis is applied to the results of the Hough transform to create distinct lane instances. Our rationale behind this decision was to improve the performance using traditional computer vision techniques, as our model heavily overfitted to the training set.

IV. Experimental Setup

A. Vehicle Detection, Distance and Velocity Estimation

For detection, we used a 70% confidence interval threshold in the YOLOv3 model discussed previously, and make a bounding box once this threshold is crossed. Examples are shown in the results section.

For distance and velocity, we got the best results with a custom deep learning architecture, consisting of 3 densely connected layers implemented in Keras. The model had 100 input nodes, 32 nodes in the middle layer, and 2 output nodes (one each for x and y coordinates). Different architectures were tried, and their results are tabulated in the next section.

As discussed previously, we trained the model on location (bounding box) and distance data as given in the annotations JSON, to determine their relationship. Once trained, the model outputs x and y components of distance as output, given bounding box coordinates as input. Note that the distance coordinates are real-world coordinates on the ground plane (as shown in Figure 4), while bounding box coordinates are the x and y coordinates in the image plane. The velocity is also determined using these distance coordinates, by using with the inter-frame distance and frame rate.

B. Lane Detection

Our pre-processing step utilised an image reshaping through nearest-neighbour interpolation to reshape the data to a three-channel colour image of shape (256, 512, 3). We also generated binary and instance maps from the annotations in the test and train set. The Mean intersection over the union between the output from the 20th image frame and the ground truth was used as a metric.

Then, we convolved the image with the encoding and decoding layers of the neural network to produce both binary and lane instance pixel embeddings. The encoder consisted of 5 blocks. Each block consisted of a sequential convolution layer, batch norm and ReLU activation function. Max pooling with a kernel size 2x2 and a stride of 2 was used to downscale the input to the next layer. We selected the VGG encoder because we could not get good results using the ENet [18] architecture suggested by Neven et al. The decoder consisted of 1x1 convolution layers followed by two final deconvolution layers. The rationale behind the decoder was that a simple decoder would suffice, as the encoder is designed to output good, representative features. The loss of the encoder with computed and backpropagated from the combined loss of both branches.

The morphological close is applied with a kernel size of 5. Then, a Hough lines transform was applied to the image. Finally, we performed connected components labelling a thresholded output where the size of a lane was within [50, 1000] pixels. We did not implement the learnt perspective transform or curve fitting from [2]. Our model was trained over 400 epochs with varied learning rates between 0.01 to 0.001.

V. Results and Discussion

A. Vehicle Detection, Distance and Velocity Estimation

1) Vehicle Detection: For the YOLO detector, we used the previously-mentioned confidence threshold of 70%. On all the clips tested, we got good results, with no false positives, and false negatives only on very distant or obscured cars.

Two examples are below. Distant cars in both cloudy and bright conditions are being detected, as well as different classes of vehicles: car, truck, and motorcycle, indicated by different colours (Clips 32 and 10). Figure 7.

Fig. 7. Vehicle-detection bounding boxes (clips 32 and 10)

The results are much better than the training annotations provided, which have an average of 1.3 detections per image (1442 bounding boxes across 1074 clips). In all clips tested, we could find at least 3 bounding boxes (usually many more, as seen above).

2) Distance and Velocity Estimation: The coordinates of our bounding boxes were used as an input to the neural network model for distance and velocity estimations. The following are the tests conducted on the estimation of the distance velocity from the output of the yolov3 model.

Since we relied on the creation of the Neural Network model, we had to normalize the data for providing effective accuracy apart from the structure of the model.

The following table (Table I) demonstrates the different structures we tested, along with the loss in these structures. Note that there is a high mean square error because it is

TABLE I
Deep Learning Architectures Tested

Models Architecture (Input — Middle — Output)	Activation Function	MSE
64, 32, 2	sigmoid,sigmoid	60.467
100, 32, 2	sigmoid,sigmoid	55.429
100, 64, 2	tanh, tanh	78.3216
100, 64, 2	relu, relu	52.2348
80, 64, 2	relu, relu	50.5689
80, 64, 2	leaky relu, leaky relu	51.756
90, 64, 2	relu, relu	45.8765
90, 32, 2	relu, relu	40.5689
100, 32, 2	relu, relu	13.256

comparing absolute values between the predicted value and the original data. Since values are floating point, a predicted value of 3.21, for example, would be considered a mismatch from a real value of 3.20. To account for this, and inherent floating point precision errors, we allowed upto +/- 1 pixel difference between the predicted and actual values. After factoring this in, we observed accuracy in the neighbourhood of of 80% after 10 iterations.

The following graph gives the accuracy plotted across iterations for our model (figure 8).

Fig. 8. Accuracy of 79-80% and above in distance calculation (MSE). Iterations are on the x-axis.

Since velocity is a derivative of distance, we could use the same model in the velocity estimation component of the project and observed that accuracy results were similar to distance estimation, as shown in the graph above.

B. Lane Detection

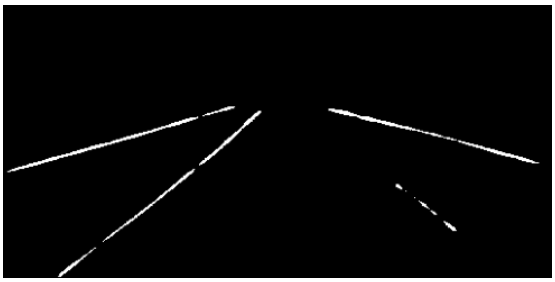

Fig. 9. Binary image after post-processing

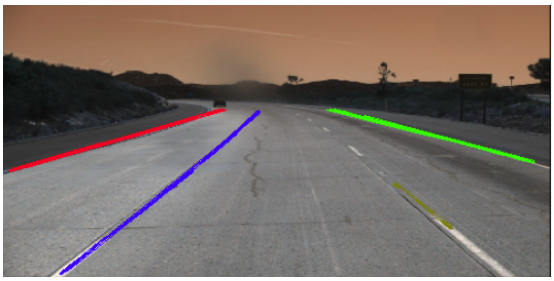

Fig. 10. Lane Instance Image after post-processing

Our LaneNet model was heavily overfitting on the training dataset. Our performance on the lane detection training set averages over a 0.85 mean IoU, and consistently creates salient lane instances. The output on the test set results were frequently disjointed or resulted in no output. The Hough transform and connected components improved the performance on instance segmentation. However, a considerable amount of time was wasted on generating and training the

Fig. 11. Example model outputs. (Left Image) Instance segmentation on the train set. (Right Image) Instance segmentation on the test set.

weights for the pixel embeddings of lane instances, as it was not used in lane detection pipeline. Alternatively, applying transfer learning with a pre-trained model on our dataset may have improved the performance. Figure 11 shows an example of our model's performance on the train and test dataset, before the post-processing was added. Our model performed well on the training set, however it was prone to errors on the test set. Empirically, we observed the misclassification of single lanes into multiple instances, or blank binary segmentation results after post-processing for a few images.

The post-processing step removed the fragmentation of singular lanes into multiple instances. As shown in Figure 10, the disjoined binary segmentation of a lane from Figure 9 is correctly classified. Our post-processing extends the simple connected-components implementation and produces good results from the final binary classification. Table II shows the results of our model on the test set.

TABLE II

Checkpoint	Accuracy	False positive	False negative	IOU
199th	0.52	**0.53**	**0.74**	0.2905
299th	**0.53**	0.54	**0.74**	**0.2928**
399th	0.51	0.54	0.76	0.2817

Multiple methodologies were attempted and applied to the binary segmentation and pixel embeddings. We attempted to draw straight lines using the spatial coordinates of the lane instance outputs from the connected components analysis. Lines with a similar gradient and offset from the coordinate (0, 0) would be merged. However, this method was not robust to scene variation and did not account for the lane curvature.

VI. Conclusion and Further Work

We could see from this work that even with a single optical camera, a fairly good visual perception capability can be built in a vehicle. However, given the very high stakes in self-driving cars, including human lives, much more work is required. For example, a single optical camera may prove ineffective in dark conditions, such as night driving. It could also become obscured by weather conditions such as mist or rain. Augmentation of the optical camera with other sensors would help make the system more robust.

Our work utilised mainly a deep-learning-based approach. This approach can be enhanced by combining it with traditional computer vision methods. Statistical models like Kalman filtering can be used to give a second opinion and verify the primary algorithm's output. For example, a momentary obscuration of the camera, such as by a flying leaf or water splashed on the camera, can lead to accidents if only instantaneous camera output is being used. By using an approach that calculates trends from past data (such as Kalman filtering), we can determine in such cases that there is a high likelihood of current camera output being wrong.

Fig. 12. Ground coordinate system used for distance estimation (image courtesy Ryan Anthony de Belen, Piazza @305)

The Kalman filter uses a linear model assumption for tracking (assumes that changes are linear). This would hold if a car is moving at a constant speed to our camera, but would not hold if a car is accelerating (for example to overtake us or another car). Extended Kalman filtering can be used to account for acceleration and deceleration.

We attempted to implement an encoder-decoder deep learning model for lane detection. For future reference, we would attempt hand-crafted feature detectors for greater interpretability. A credible direction of investigation is a traditional computer vision pipeline [19].

VII. Contribution of Group Members

All members participated in the initial planning and task distribution. Work was divided into two subteams: one for detection, distance and velocity estimation, and the other team for lane detection. Further work was as follows (in alphabetical order of the first name):

- **Nathan Yin**: Lane detection team. Model inference, visualisations, training and testing. Contributed to presentation and report.
- **Ngoc Pham**: Lane detection team. Model inference results, post-processing and training. Contributed to presentation and report.
- **Usama Sadiq**: Distance and velocity estimation team. Built the machine learning model for distance learning, which was also adapted for velocity later. Also worked on the YOLO object detector. Contributed to presentation and report.
- **Uttkarsh Sharma**: Distance and velocity team. Built the YOLO object detector, and the velocity estimation model. Contributed to presentation and report.
- **Vishal Bondwal**: Distance and velocity team. Built object detector and object tracking classes, adapting previous project code and adding more. Contributed to presentation and report.

REFERENCES

[1] T. H. Inc. Tusimple competitions for cvpr2017. [Online]. Available: https://github.com/TuSimple/tusimple-benchmark

[2] D. Neven, B. Brabandere, S. Georgoulis, M. Proesmans, and L. Van Gool, "Towards end-to-end lane detection: an instance segmentation approach," 06 2018, pp. 286–291.

[3] A. Klintberg. Pytorch lanenet. [Online]. Available: https://github.com/klintan/pytorch-lanenet

[4] V. Kiran, P. Parida, and S. Dash, *Vehicle Detection and Classification: A Review*, 01 2021, pp. 45–56.

[5] D. Lowe, "Object recognition from local scale-invariant features," vol. 2, 02 1999, pp. 1150 – 1157 vol.2.

[6] H.-C. Chu and H. Yang, "A simple image-based object velocity estimation approach," 04 2014, pp. 102–107.

[7] A. Ali, A. Hassan, A. Ali, H. Khan, W. Kazmi, and A. Zaheer, "Real-time vehicle distance estimation using single view geometry," 03 2020, pp. 1100–1109.

[8] S. Zhenbo, J. Lu, T. Zhang, and H. Li, "End-to-end learning for inter-vehicle distance and relative velocity estimation in adas with a monocular camera," 05 2020, pp. 11 081–11 087.

[9] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 04 2018.

[10] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. Zitnick, "Microsoft coco: Common objects in context," vol. 8693, 04 2014.

[11] A. bar hillel, R. Lerner, D. Levi, and G. Raz, "Recent progress in road and lane detection: A survey," *Machine Vision and Applications*, vol. 25, 04 2014.

[12] Z. Kim, "Robust lane detection and tracking in challenging scenarios," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 9, pp. 16 – 26, 04 2008.

[13] K.-Y. Chiu and S.-F. Lin, "Lane detection using color-based segmentation," vol. 2005, 07 2005, pp. 706 – 711.

[14] A. López, J. Serrat, C. Cañero, F. Lumbreras, and T. Graf, "Robust lane markings detection and road geometry computation," *International Journal of Automotive Technology*, vol. 11, pp. 395–407, 06 2010.

[15] J. Kim and C. Park, "End-to-end ego lane estimation based on sequential transfer learning for self-driving cars," 07 2017, pp. 1194–1202.

[16] S. Canu. Object tracking with opencv and python. [Online]. Available: https://pysource.com/2021/01/28/object-tracking-with-opencv-and-python/

[17] B. Brabandere, D. Neven, and L. Van Gool, "Semantic instance segmentation with a discriminative loss function," 08 2017.

[18] D. Spoorthi and B. Ashwini, "Efficient neural network for real semantic segmentation," 06 2019, pp. 340–343.

[19] C. Low, H. Zamzuri, and S. Mazlan, "Simple robust road lane detection algorithm," 06 2014, pp. 1–4.