Computer Vision T1, 2021 Group Report

A visual perception module for an autonomous vehicle.

Andrew Simpson
(z5109480)
andrew.simpson@unsw.edu.au

Daniel Pham
(z5209600)
z5209600@ad.unsw.edu.au

Dean Poulos
(z5122508)
dean.poulos@unsw.edu.au

Tom Wright
(z5207952)
thomas.wright@unsw.edu.au

I. Introduction

A. Specification

This report investigates an image-based approach utilising popular computer vision techniques to achieve four primary objectives. The first objective is to accurately detect and locate vehicles in a given image from the training set. Once achieved all vehicles are identified using a standard rectangular bounding box. The second primary objective is to accurately estimate the position of each vehicle in the frame, relative to the perspective of the camera. Once accomplished each vehicle is assigned a position data label for each frame. The third primary objective is to track vehicles between frames and perform velocity estimation based off the associated position data. The fourth primary objective is to detect and trace lane markings as poly-lines.

B. Data

The project makes use of the TUSimple datasets (CVPR 2017 Workshop). The datasets can be described by complexity, size and annotation type.

1) Velocity Estimation Dataset: The velocity estimation dataset is relatively broad consisting of variable traffic conditions and vehicles with relative distance ranging from 5 metres up to 90 metres. However, video is limited to daytime driving on a highway, and does not capture added complexity of local roads. The training data contains 1074 two-second clips recorded at twenty frames-per-second, with a total of 3222 annotated vehicles. The testing set contains 269 clips of videos with the same format. 5066 supplementary images are included with human labeled bounding boxes. Annotations are position and velocity generated by range sensors [1].

2) Lane Detection Dataset: The lane detection dataset supplies clips with mostly clear weather conditions at different times of day. Roads can have 2, 3, 4 or more lanes on highway roads. Traffic conditions also vary. The training set consists of 3626 video clips and a single annotated frame per clip. The test set consists of 2782 video clips. Each clip lasts 1s and contains 20 frames. The camera is oriented very close to the car orientation. Annotations are in the form of poly-lines for lane marking. More specifically, a frame of each clip is annotated with a list of `h_samples` which are uniformly-spaced samples along the y-axis of an image, spanning from the bottom of the image to above the vanishing point, and a list of lists `lanes` of x-values at which a y-sample intersects a lane, or -2 if it does not.

II. Literature Survey

A. Car Detection

A variety of studies have been performed in the space of real-time car detection and tracking including unmanned aerial vehicle (UAV) [2] [3] detection and ground-based pedestrian, traffic-sign and vehicle-detection [4] [5]. Various applications include traffic estimation [6], car-tracking [7], surveillance [8] and more. Various strategies for car-detection are utilised, including model-based feature prediction [2], neural networks [7], deep learning [3] and on-line boosting [9]. A meta-analysis of computer vision problems and datasets for autonomous vehicles which aggregates benchmarking data for various car-detection techniques [10] ranked deep convolutional networks the most successful.

However, we continue to explore only those methods which are allowed by the project specification, which detect objects using the so-called classical object-detection pipeline, as described by Janai et al. [10]. The classical pipeline begins with a pre-processing step, heuristic-based feature extraction, a model-based learning method then a sliding-window verification and classification of objects. Popular approaches to the latter step with regards to pedestrian detection are AdaBoost [4], which demonstrates fast false candidate removal, as well as Support Vector Machines (SVMs) with Histogram of Orientation (HOG) features, which are shown to work well for high-resolution sensor images [11]. However, it is shown that no specific classifier is better suited to the detection task, provided that sufficiently many features are supplied [12].

B. Position Estimation

Traditional methods for vehicle position detection in self driving car applications include LIDAR [13], microwave radar [14], and stereo camera imagery [15]. These methods are popular for vehicle position detection as they naturally produce 3d information as their output, with the exception of stereo imagery, which requires some computer vision feature matching techniques to calculate the depth information for each pixels. These methods do have many drawbacks however, including the need for expensive equipment and sensors, poor performance during bad weather such as rain and snow, and lack of detailed information that can be used for uniquely detecting the vehicles in the first instance. In this report we investigate ways in which vehicle positions can be calculated from two dimensional images alone. This greatly increases the complexity in accurately estimating the vehicle positions however reduces the cost and complexity of the sensor hardware requirement. It is common for existing two dimensional approaches to make use of known camera intrinsic and extrinsic properties. The intrinsic matrix which is paramatised by Hartly and Zisserman into the following form [16]:

$$K = \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}$$

The intrinsic camera matrix is used to map the 2 dimensional pixel space into the 3 dimensional world space. The parameters that define the matrix include focal lengths f_x and f_y, principle point offsets x_0 and y_0, as well as axis skew s. The focal lengths define the distance between the virtual image projection and the 'pinhole' of the camera. The pinhole camera is used as an approximation for the cameras inherent properties, since this accurately can be used to model the real world camera, assuming the intrinsic parameters are known.

C. Velocity Estimation

There are two main areas of research in the field of velocity estimation from a monocular camera system, they are Classical Distance-Velocity estimations, and Neural Network bases distance-velocity estimations. While the methods used in this report to calculate velocity are focused on the former approach, the latter does have significance for this problem. Kampelmuhler et al. describe their method for velocity estimation using a holistic system of vehicle detection, segmentation and trajectory/velocity estimations. [17]. Their method works by training a FlowNet2 Neural Network on various ground truth distance and velocities on different vehicles. This makes the estimation system more robust to differences in vehicle type.

The classical methods of velocity estimation build on the work done on Position Estimation where the 2-Dimensional distances are extrapolated over time to estimate the velocity of a vehicle [18]. The main challenge of this implementation is the inter-frame matching of like vehicles. Choi et Al. utilise Scale-Invariant Feature Transform (SIFT) features to match vehicles cross frames [19]. Their work notes that is method is most effective for vehicles that are close to camera, and have a high relative angle, ie. are perpendicular), to the camera. Comparatively, Badenas et Al. use vehicle segmentation matching to match vehicles between frames [20]. Both these methods perform well in their identification of similar vehicle between frames. Once vehicles are sucessfully matched, their positions can be mapped temporally to get an estimated velocity in 2-Dimensions [18].

D. Lane Detection

Research has been ongoing in lane detection in the twenty-first century and significant advancements have been made regarding algorithmic design and performance in specialised conditions. In it's simplest functional form, a lane perception module must detect the host lane in the local vicinity of the car. As summarised by A. Borkar et. al., This problem has been solved in roughly 80% of metro highway cases using Hough-based transformations [21] [22] [23] and improved upon by a layered-approach, consisting of Inverse Perspective Mapping (IPM), Random Sample Consensus (RANSAC) and Kalman Filtering [24], with roughly 90% accuracy.

A generic lane detection system can be decomposed into four modules, namely image pre-processing, feature extraction, road/lane model fitting, temporal integration and image to world correspondence according to "Recent progress in road and lane detection: a survey" [25] (A. Hillel, R. Lerner, D. Levi & G. Raz). A description of each module has been detailed by the survey and summarised in the following four paragraphs.

The *image pre-processing* component can consist of illumination-based pre-processing such as exposure adjustment and colour space-transformations [26], as well as pruning-based pre-processing techniques such as obstacle removal [27].

Feature extraction: since lanes are distinct from the road in shape or colour, gradient-based approaches are applicable [28] to detect marking positions. So called "steerable filters" enable the extraction of edges along specified orientations [29]. Alternatively, the low-high-low intensity step of a lane marking motivates the use of a box filter or a step filter, which extracts the desired features [30]. Another common approach normalizes the lane marking width and direction using an inverse-perspective mapping [29] [24], which simplifies the problem of detecting lane markings which curve and thin toward the vanishing point.

Model fitting fits a geometric model to the lane in a top-down manner, eliminating the noise generated by bottom-up path detection. Models can be described by lines (parametric), splines and poly-lines (semi-parametric) or continuous boundaries (non-parametric). Random Sampling Consensus (RANSAC) assists a top-down approach in reducing the numerous outliers detected by the feature extraction step [24].

Temporal integration improves the accuracy of a true positive detection and rejects more false positives by requiring consistency across frames. Tracking lane markings across frames can be achieved by projecting the position of detected lane markings from a previous frame onto the current frame. Particle Filtering or Kalman [29] Filtering are popular methods for this task, however struggle because of smoothness assumptions made by a naive inverse-perspective mapping.

In more recent years, convolutional neural networks have surpassed the success of traditional computer vision approaches, enabled by modern computational power. Network architectures designed for instance segmentation have had success in supervised lane-detection. An example of this is ERFNet [31] which uses efficient residual factorisation in novel one-dimensional "non-bottleneck" layers to reduce parameter space and address the "degradation" problem observed in deep convolutional networks.

III. Method

A. Car Detection

In this task, multiple models were tested to find an accurate car classifier. An accurate car classify with tight bounding boxes is necessary for the improvement of the distance and velocity components.

The first model is a linear support vector machine that uses feature extraction methods such as hog descriptors and local binary pattern. It is an improvisation of the individual component that attempts to reduce false positives and have tighter bounding boxes.

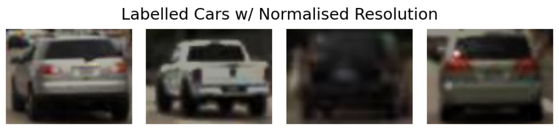

Fig. 1. Normalisation of labelled regions in the first eight clips of the training data.

1) Sampling Data: Images of positive examples (cars) were sampled using the annotated labels provided in the training and supplementary data-set. On the other hand, a data-set of negative examples are generated by randomly sampling a sub-image of the supplementary data that do not overlap with car regions.

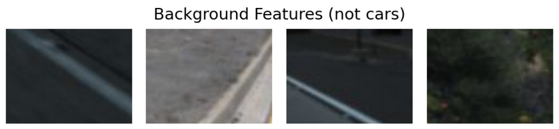

Fig. 2. First sixteen negative examples.

However, the result of random sampling can introduce bias and bring many false positives. To improve the classifier, the classifier is retrained with these false positives using a hard negative mining algorithm with a non-overlapping sliding window.

2) Pre-processing: Very dark or light exposures of light can dim the edges of cars affecting hog descriptors and spatial features. To reduce this effect, a histogram equalization method is utilized as shown in fig. 3. using the V channel on a HSV colour space. In combination to this, applying a different colour-space transformation, have been shown to improve detection performance [5]. For this project, images are converted to a YCrCb colour-space as this further improves the performance of car detection algorithms by reducing the effects of illuminations and shadows [32].

By eliminating the effects of lighting, the contrast of the image increases. An alternate method that was also tested includes normalising all channels on a given colour space in order to reduce the variation of colours spanning multiple images.

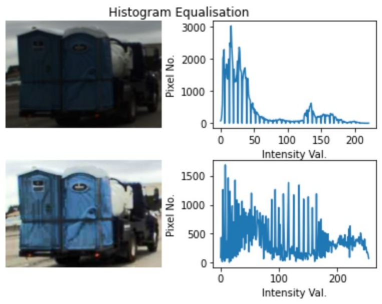

Fig. 3. Normalised brightness of clip twenty in the training data.

3) Feature Extraction: A supervised approach is used for feature extraction as sets of fully labeled data are accessible through the supplementary set. A HOG feature descriptor is chosen for its applicability to multi-scale object detection (i.e., image-pyramids), successful identification of shape-edge features and resistance to motion blur. In this step, the HOG feature descriptor reduces fixed-size image subsets into a fixed length, lower-dimensional feature vector by binning cell-majority orientation values. In order to achieve this, the ratios and resolutions of the labeled data must be normalised. Labeled car bounding boxes are normalised to a 1:1 aspect ratio and fixed resolution (64×64) as shown in Fig. 4.

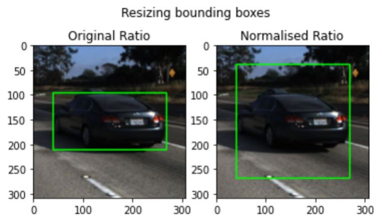

Fig. 4. Normalised bounding-box ratio using a length-increase method.

From the individual component, optimal hog hyper parameter include using 9 orientation bins with (8×8)px cells with (2×2) cells per block.

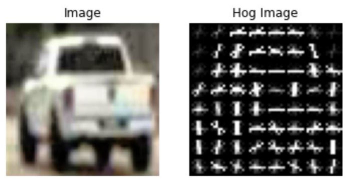

Fig. 5. Example HOG feature extraction on first two clips of training data.

Local binary pattern are used to extract spacial features from the images. This feature were shown to provide a performance boost when complimented with HOG feature descriptors in car classification and removes noisy false positives such as leaves and signs that may happen to create shape-edges similar to cars.

4) Model Training: In this step the positive and negative feature sets sampled from the first 1000 images of the training data and 5000 images of the supplementary data are supplied to a linear SVM for training. An SVM learns feature spaces based on some configurable kernel-method, gamma value and regularisation parameter C. Some estimates for best performing values can be made using those published in literature (e.g., N. Dalal and B. Triggs [33]). From the experimentation setup from the comparative studies of individual component, a C value of 1 was shown to perform optimally.

5) Verification and Refinement: Now, a sliding-window paired with an image-pyramid is used for detection on the supplementary set. If a sliding window determines that a car is there, the region is marked and incremented with a likelihood value. A threshold on these likelihood values allows us to obtain bounding boxes of cars. This filters multiple sub-optimal detections leaving behind a single, most-likely match, reducing the number of false positives or partial matches. A low stride of 8 was used during classification to improve the performance of the classifier whilst not being computationally expensive.

6) YOLOv5 Model: Another model that was also used to tackle the issues of an SVM. This model uses the You Only Look Once, version 5 (YOLOv5) [34] neural network architecture. YOLOv5 incorporates both classification and localisation methods within the same neural network being able to capture non-linear features and search for cars on a pixel by pixel basis. A problem with the linear SVM model was its speed and performance. The linear model uses modules from SKLearn [35] and a sliding window which are both CPU bound. YOLOv5 training and prediction algorithms are GPU bound which provides a massive speed-up in performance. HOG descriptors are rotation invariant. As a result, the detector has problems detecting the side of cars whose shape differs massively from the rear. Furthermore, edges are extremely difficult to extract when cars are only a few pixels in width. YOLOv5 uses a loss function that penalises based on the Intersection of Union (IOU), centre of the bounding box, width and height and classification object type. Because of this, this neural network is ideal at capturing vehicles with a tighter bounding box as required in the position and distance component of this project. The YOLOv5 github repository [36] provides a simple means to interface its functions and provide training which isn't supported in Pytorch or Tensorflow. The first 1000 supplementary images were reformatted and trained on using pretrained weights from YOLOv5l6 model which has weights trained on a range of images.

B. Position Estimation

Since there is not enough information to extract 3d depth information from a single 2 dimensional image we must make 3 primary assumptions to constrain the degrees of freedom of each detected vehicle.The first assumption is that all vehicles have the same fixed rear width that is equal to 2.0 meters. The second assumption is that all cars are located on the same horizontal plane, set at the point on the vertical axis where $z = 0$. The third assumption is that the camera can be modeled through the pinhole camera approximation, and that the intrinsic and extrinsic parameters are known.

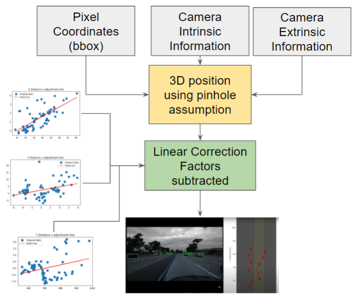
Pixel Coordinates (bbox)
Camera Intrinsic Information
Camera Extrinsic Information
3D position using pinhole assumption
Linear Correction Factors subtracted

Fig. 6. Video Clip to position Estimation Pipeline

Points in the image space are transformed into points in the world space through the use of the camera matrix. Using method of similar triangles, real world space is converted into pixel dimensions through the following equation [37]

$$Y = \frac{bb_{centre} - cx}{fx} \tag{1}$$

$$Y = \frac{width_{car} * fx}{width_{boundingbox}} \tag{2}$$

Using similar triangles both virtual camera and real world, we cam easily switch between real world and virtual distances by using both world and camera focal lengths. This is useful when for example displaying the location of a 3 dimensional point in a 2 dimensional image, or predicting a 3d world point based of a 2d image coordinate.

A linear correction factor is then applied to the y position output to reduce the error of this method. The correction factor is calculated through finding the linear regressions of various error metrics. 7 shows an example of one of these regressions. Since there is a repeatable trend in the error as a result of the perceived change in vehicle width as the vehicle approaches the sides of the image frame, this can be corrected for by subtracting the required adjustment according to the generated regression.

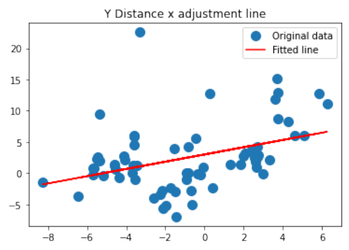

Fig. 7. y error versus x position

C. Velocity Estimation

Where Position Estimation is able to be calculated using a single bounding box with assumptions of the camera, velocity estimation requires temporal extrapolation with inter-frame vehicle matching. The pipeline of a video-clip to individual vehicle distance estimation is illustrated in Figure 8 is as follows.

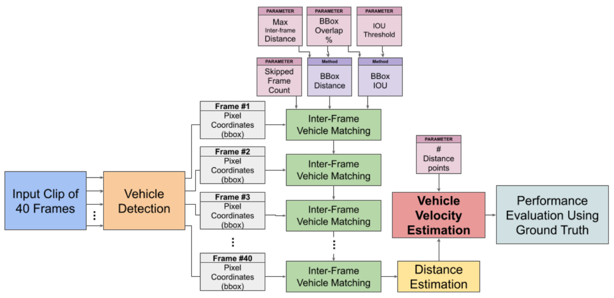

Fig. 8. Video Clip to Velocity Estimation Pipeline

1) Vehicle Detection: First step is the detection of vehicles, described in Car Detection.

2) Inter-Frame Vehicle Matching: Once bounding boxes are found for every vehicle in every frame, the inter-frame vehicle matching step occurs. This uses two main methods: minimisation of bounding box distance and maximisation of bounding box Intersection of Unions (IOU). These two methods are used in a cross-frame matching algorithm. These steps result in a list of bounding boxes, matched with the clip time in seconds.

a) Bounding Box Distance Minimisation: The bounding box distance minimisation method assumes that two bounding boxes with similar centroid positions are likely to be the same vehicle. The implementation of this algorithm accepts two adjustable parameters: the maximum inter-frame inter-centroid distance; and a minimum bounding box overlap percentage. The former parameter is a function of the maximum bounding box dimension, and the latter allows for a minimum percentage overlap between two similar bounding boxes in two frames.

b) Bounding Box IOU Maximisation: The bounding box IOU maximisation method assumes that two bounding boxes of the same vehicle will overlap to a certain percentage between clips. This assumption is generally consistent for vehicles in the same lane, and traveling at the same speed as the observing car. When this assumption breaks down (due to skipped frames, or high relative velocity), the Bounding Box Distance Minimisation Method is more robust. This method accepts two parameters: minimum bounding box overlap percentage (which is the same as in the Bounding Box Distance Minimisation Method) and IOU threshold to set a minimum IOU value to label two bounding boxes as belonging to the same vehicle.

c) Cross-Frame Matching Algorithm: The cross-frame matching algorithm uses a highly iterative approach to cycle through each bounding box in each frame, and compare it to each bounding box in the succeeding n frames. Since there is some error in the detection algorithm, n is a parameter which has been tuned to allow for vehicles to not be detected between frames, but still be matched as the same vehicle.

3) Distance Estimation: Using each bounding box for each matched vehicle the Distance Estiamtion Algorithm is used to generate a set of z and x points, associated with time values.

4) Velocity Estimation from Temporally-Based Distances: The method for estimating velocity from a set of time-related distances is to get the time derivative. Since the aim of the experiment is to determine the velocity at the end of a clip, an adjustable parameter of i data-points is used. Using these i data-points, a linear regression line is fitted, with the gradient of this line equating to the velocity of the vehicle. This can be seen in Figure 9.

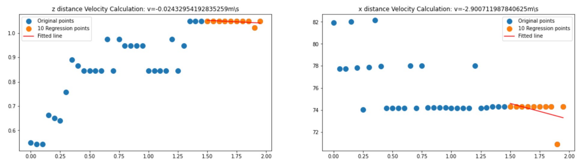

Fig. 9. Example of Positional Regression with a sample of n=10 Points

D. Lane Detection

Since TuSimple contains annotated frames, a supervised approach was considered favourable to most traditional, unsupervised gradient and filter-based approaches. Due to the success of deep convolutional networks in recent years, a deep-learning instance segmentation pipeline was elected for this task. The pipeline will consist of a pre-processing step, an instance segmentation step and finally a model fitting step.

1) Pre-Processing & Batching: Images are down-sampled by a factor of two to reduce training size. At this stage, poly-lines annotations must be converted into a segmented image as a target for the model.

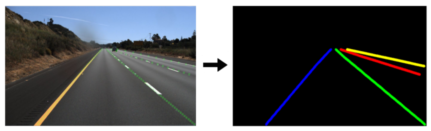

Fig. 10. Mask of training example annotations applied applied as poly-lines to a black background, used for generating a segmented target image.

Next, input images were converted into tensors alongside targets, and these were batched according to a `batch_size` hyper-parameter.

2) Instance Segmentation: An instance segmentation CNN architecture was chosen to learn lane markings, as they have had success in recent years. In particular, a semantic segmentation network ERFNet [31] which implements a residual connection layer and factorized convolutions to achieve a strong balance between efficiency and accuracy. Recent research has shown that bottleneck layers still suffer from degradation [38]. ERFNets redesign of the non-bottleneck layer is an attempt to alleviate this. The network structure is shown below in Figure 11 from the original paper [31].

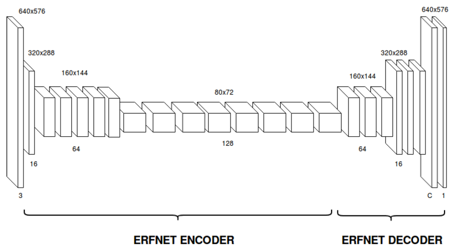

Fig. 11. ERFNet architecture from the original paper [31].

The residual (skip) connections in ERFNET connections supply context to lower layers to alleviate vanishing or exploding gradients characteristic of degradation. ERFNet's novel non-bottleneck residual layer structure uses a one-dimensional factorisation to reduce the parameter space of the original design, as shown in Figure 12 from the original paper [31].

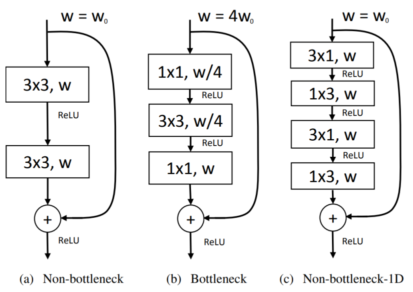

Fig. 12. Where feature maps w are "bottlenecked" or compressed by a factor of 4 in traditional bottleneck layers.

A PyTorch implementation of ERFNet [39] and an instance-loss function similar to binary cross entropy was integrated into a Python pipeline with the aforementioned and following steps. Here, an optimiser is chosen according to some `optimiser` hyper-parameter and some `learning_rate` for gradient descent, a parametrised model-fitting scheme and finally, an evaluation procedure.

3) Model Fitting: The output of ERFNet is an image with six activation channels. The value of each pixel represents the probability that it belongs to the class represented by its channel. The `argmax` operation is performed on the output resulting in a single-channel segmented image. From this image, poly-lines must be fit. Two different methods of fitting poly-lines are implemented as a hyper-parameter, which include mean and linear regression, as shown in Figure 13.

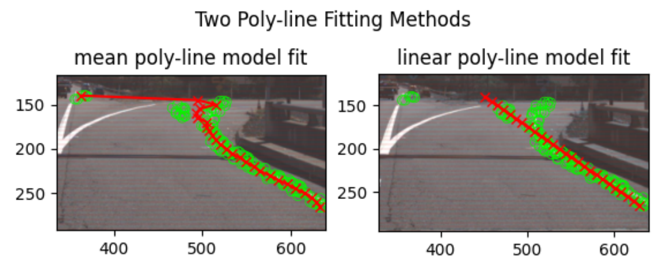

Fig. 13. Two implemented methods of fitting poly-lines to a segmented image.

IV. Experimental Setup

A. Car Detection

1) Evaluation Metrics: Two popular performance metrics are used for evaluating detection, namely, precision (Eq. 3) and recall (Eq. 4). Using these metrics, a precision-recall curve can be constructed to further evaluate each method. In this regime, we categorize detections into true-positive, false-negative and false-positive.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{3}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{4}$$

To make these categorisations, the intersection-over-union ratio should be considered. This is simply the fraction of overlapping region and total region areas. If half or more of the detected bounding box area overlaps that of the ground truth, a true positive detection has been made. This is commonly known as the PASCAL VOC metric and will be used in the implementation to constitute a match. Otherwise, a false-positive has occurred when a detection has been made on something other than a car, and a false-negative when a labeled car has not been detected by this standard. The time-performance of the methods is also recorded, specifically training duration and detection duration-per-image.

2) Experimental Parameters: Many parameters have been already exhausted during the individual component of the assignment. Instead, a different set of parametrisation will be evaluated. Furthermore, the new proposed model will need to be evaluated as well.

a) SVM parametrisation: Different colour spaces were not explored in the individual component. RGB, HSV and YCrCb all have effects on lighting levels which need to be tested. Evaluation of different lighting levels also must be considered, whether adjusting the V channel in histogram equalisation will have an effect of the normalisation of all colour channels. Furthermore, testing is required to determine whether hard negative mining approach improves the performance of the classifier.

b) YOLOV5: The evaluation of YOLOv5 module will be tested using the default parameters with an epoch of 10 and a training sample size of 1000 and a validation size of 200 from the supplementary data set. Images were resized 1280×1280 for training.

B. Position Estimation

1) Experimental Environment: The position detection algorithm was evaluated using 1074 static frames from the training data set. Each image included a ground truth position and this was used to calculate the error for each car.

2) Evaluation Metrics: From the ground truth data some evaluation metrics were created. The two main metrics used were percentage error for x position where values are usually far from the origin and this metric makes sense, and absolute error for y position as cars often move across $y = 0$ which causes large percentage errors despite the absolute error being quite Small. 5 shows how the absolute error is calculated and 6 shows how the absolute error is calculated.

$$Error_{abs} = |Position_{GroundTruth} - Position_{Predicted}| \quad (5)$$

$$Error_{\%} = |\frac{Error_{abs}}{Position_{GroundTruth}}| \quad (6)$$

3) Experimental Parameters: The two main parameters which were adjusted to optimise the algorithm output included the defined width of the vehicle, as well as the correction factor for y versus x position. These values were crafted to reduce the overal error and improve the algoritms performace.

C. Velocity Estimation

1) Experimental Environment: The input for the velocity estimation performance evaluation section is the Training data-set, with parameters for the Car Detection and Position Estimation set as above. All 40 frames are passed into the Velocity Estimation Algorithm, with a velocity estimation of at the 40th frame returned.

2) Evaluation Metrics: Using the ground truth from the annotated data, the primary metric for velocity analysis is the absolute error calculated in Equation 7.

$$Error_{abs} = |Velocity_{GroundTruth} - Velocity_{Predicted}| \quad (7)$$

Since there are significantly more vehicles detected by the Vehicle Detection algorithm than the labeled ground truth, the Inter-Frame Matching Methods were used to match the ground truth bounding box to the nearest detected vehicle, evaluating the error between these.

3) Experimental Parameters: There are five adjustable hyperparameters available to improve the performance of the velocity detection implementation. As seen in Figure 8, these are: maximum inter-frame centroid distance, minimum bounding box overlap percentage (for the distance and IOU methods), the IOU threshold, skipped frame count and distance-regression data-point count. The methods for their determination are discussed below.

a) Inter-Frame Matching Parameters: Without any numerical analysis methods available for analysis of Inter-Frame Matching performance, determination of the accuracy of this algorithm was done qualitatively. Figure 14 shows two adjacent frames before they are matched, notice the inconsistent colour of bounding boxes between frames. Conversely, Figure 16 shows the same two frames after the matching algorithm has been run with weightings set to the necessary hyperparameters.

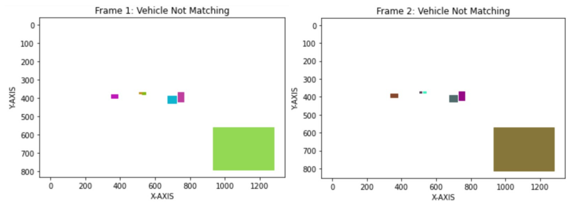

Fig. 14. Bounding Box Representation, not Inter-Frame Matched

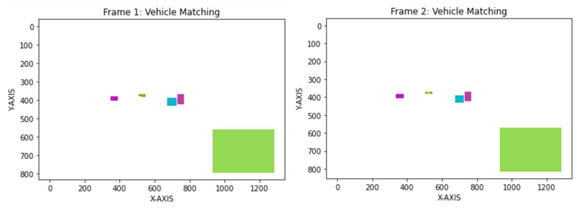

Fig. 15. Bounding Box Representation, Inter-Frame Matched

To set the skipped frame count, observational analysis was done on a random selection of vehicle-detected frames, with the maximum gap between the same vehicle being detected being three (3) frames.

b) Distance-Regression Data-Point Count: By iterating through all values of n, where n is the number of frames before the 40th frame, and determining the distance regression from those points (as visualised in Figure 9), Figure **??** is created. This shows that that by including all points in the velocity estimation, error is minimised.

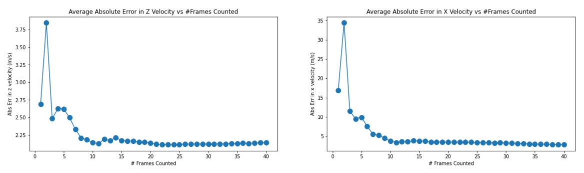

Fig. 16. Number of Frames Counted vs Absolute Velocity Error

D. Lane Detection

The following experiments are performed Python via PyTorch and a NVIDIA GeForce RTX 2070 graphics card used for training.

The accuracy formula used by the TuSimple challenge [40] is

$$\text{accuracy} = \frac{\sum_{clip} C_{clip}}{\sum_{clip} S_{clip}} \quad (8)$$

where C_{clip} is the number of correct points in the last frame of the clip, and S_{clip} is the number of requested points in the last frame of the clip. A prediction is classified as correct if the difference in distance between ground-truth and prediction is below some threshold. In TuSimple's evaluation module, this threshold is calculated using the formula:

$$\text{threshold} = \frac{20\text{px}}{\cos(\theta)} \quad (9)$$

where k is the coefficient (or gradient) of the line of best fit through the ground-truth lane markings. This scales the threshold according to the angle of the lane, enforcing tighter thresholds for lanes with smaller angles. Since this threshold is constant for all y-samples, it is more lenient for poor predictions further into the distance, as shown in Figure 17

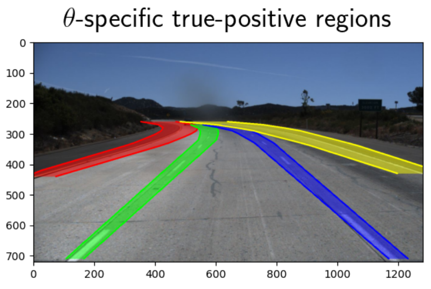

Fig. 17. Region for correct-prediction highlighted green, where lanes 1, 2 with sharper θ-angles have larger thresholds, therefore wider TP regions.

Finally, the algorithm run-time is recorded for comparison between different approaches and hyper-parameter combinations.

Various hyperparameters become available in the deep-learning segmentation pipeline. Hyper-parameters are selected from instance segmentation and model fitting.

Instance Segmentation: Hyper-parameters at the training stage are varied, all else kept constant. These include:

1) *Optimiser*: Adaptive Momentum (Adam) and Stochastic Gradient Descent (SGD) back-propagation algorithms are used to train separate models, which are then evaluated.
2) *Batch Sizes*: Batch sizes are varied over values `1, 5 & 10` for a reasonable number of epochs so training times and model performance can be evaluated.
3) *Learning Rate/Momentum*: Optimiser-specific hyper-parameters are varied over the course of experimentation to avoid convergence to local minima.

Model Fitting: After instance segmentation, the output image is labelled with five classes, e.g., background and four lane markings. Poly-lines must be fit to the segmented image, and can be done in a variety of ways. Poly-lines were fit in two different ways and evaluated to determine best performance:

1) *simple mean*: poly-line is constructed using the mean x-value of a class (segmented lane-marking) at each `y_sample`.
2) *linear regression*: a line is fit through all class labels and sampled at each `y_sample`.

V. Results

A. Car Detection

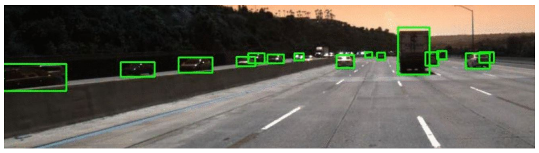

Fig. 18. Result of best classifier

Classifier	Precision	Recall
Linear SVM w/ YCrCB	0.32	0.19
Linear SVM Retrained w/ HSV	0.38	0.16
Linear SVM Retrained w/ RGB	0.40	0.15
Linear SVM Retrained w/ YCrCB	0.48	0.15
YoloV5	0.92	0.75

TABLE I
Performance of Vehicle Classifier Models

B. Position Estimation

The position detection algorithm was tested on 1074 sample images, and the errors for each frame were saved. The errors were then analysed and a summary table was produced. II shows the summary of results from the position detector algorithm.

	Initial Algorithm	After Correction
Mean y (ground truth)	35.1	
Mean y (ground truith)	0.184	
Abs mean error x (m)	3.63	2.12
SD error x	3.36	3.36
Abs mean error y (m)	1.19	0.65
SD error y	0.998	0.998

TABLE II
Performance of position detector

C. Velocity Estimation

Analysing the performance of the Velocity Estimation numerically results in Table III. This shows that there is some consistent error in the velocity estimation.

Velocity Results	
Avg Absolute Error Z	2.14085m/s
Avg Absolute Error X	2.76011m/s

TABLE III
Performance of Vehicle Velocity

To gain a better understanding of the performance of the velocity estimation model, the generated regressions of the distance values for each frame were compared to the ground truth velocities. These results can be seen below in Figures 19 and 20. Analysing these graphs visually indicates that the linear regression method is capable of consistently estimating velocities of vehicles, despite a discrepancy between the estimated value and the ground truth value. This may indicate that upstream processes (vehicle detection or distance estimation) introduce some systematic error.

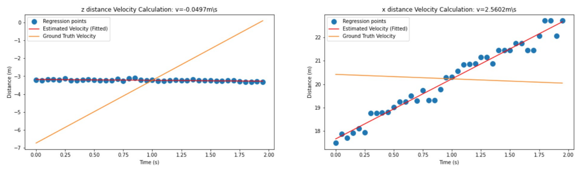

Fig. 19. Estimated vs Ground Truth Velocity with Distance Points

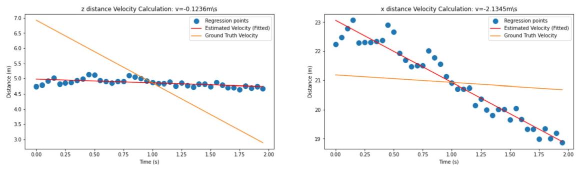

Fig. 20. Estimated vs Ground Truth Velocity with Distance Points

In Figure 21, it can be seen that the z distance is non linear. The linear regression estimation is not able to account for this, and such outputs an incorrect result.

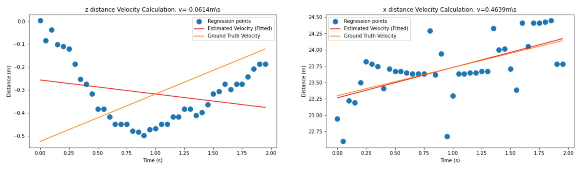

Fig. 21. Non-Linear Velocity. Estimated vs Ground Truth

D. Lane Detection

1) Example Detections: Some outputs of the pipeline are shown in Figures 22 & 23.

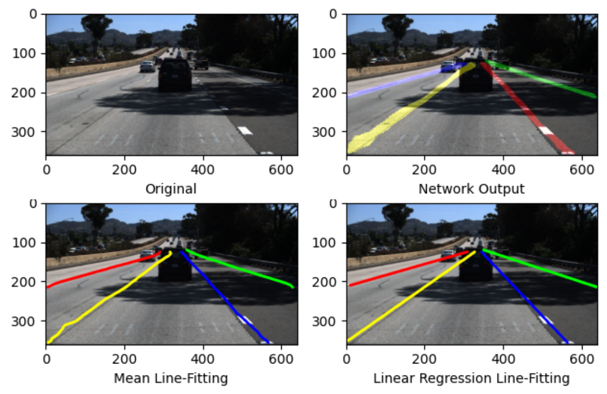

Fig. 22. Example of good, 4-lane detection resistant to occlusion and shadows.

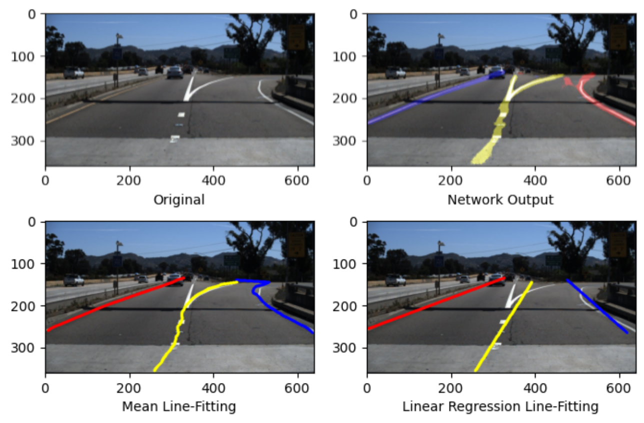

Fig. 23. Example of ambiguous lane detection.

2) Gradient-Descent Algorithm Experiment: The model was trained and evaluated using Adam and SGD optimisation, with `batch_size` $\in [1,2,5]$. Learning rates used were $\eta = 5 \times 10^{-4}$. Loss evolution for `batch_size = 1` is shown in Figure 24 below.

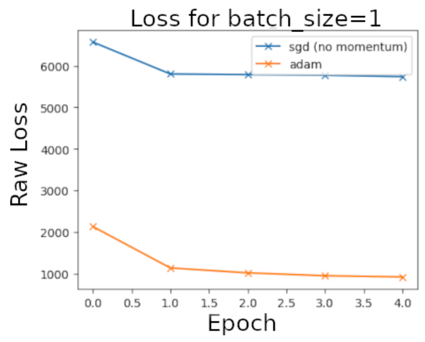

Fig. 24. Results from gradient descent with `batch_size=1`, SGD with default parameters (no momentum).

Since SGD was very converging at some local minima, `momentum=0.05` was added and the learning rate was raised to $\eta = 0.15$. The network was then trained using `batch_size=5` over 10 epochs, then using `batch_size=10` over 15 epochs, as shown in Figure 25.

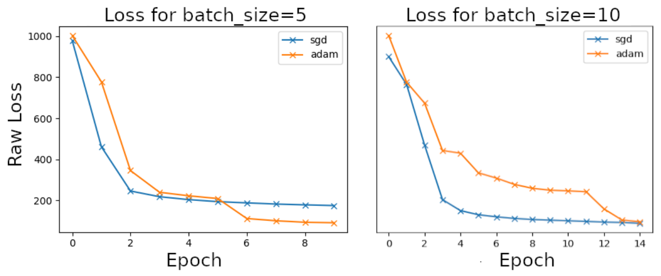

Fig. 25. Loss evolution of gradient descent with `batch_size=5,10`.

Training time for each model was recorded and their performance was evaluated. Their performances are summarised in Figure 26 below.

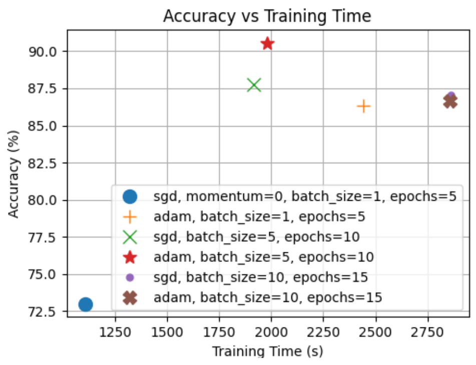

Fig. 26. Average accuracy (calculated with Eq. 8) versus training time in seconds.

3) Model-fitting Experiment: In the model-fitting experiment, the network trained with the best optimisation hyperparameters (SGD and Adam with `batch_size=5` had poly-lines fit using both the mean and linear-regression methods, and were evaluated as shown in Figure 27.

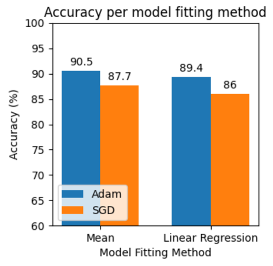

Fig. 27. Average accuracy (calculated with Eq. 8) versus model fitting method.

VI. Discussion

A. Classification

Using the YOLO module, classification performed very well with tight bounding boxes. However, the classifier failed to capture extremely distant cars. A proposal to address this issue could be to used more accurate detection neural network architecture that trade off speed for accuracy.

B. Position Estimation

The position calculation function performed well, considering the limited information provided to it. In the forward (x) direction we see less 3.6 meter mean standard error at 36 meters which is equal to approximately 10% percentage error. The standard deviation is small, around 9%.

In the y axis the absolute error is smaller, around 0.63m with standard dev of 0.99m.

The detector is accurate enough to correctly identify which lane a vehicle for x distances in excess of 20m.

Future work could focus on addressing the 3 assumptions, for example using a classifier to detect car width based of known car types instead of making a fixed assumption. Overall the car position detection worked well considering the limited 2 dimensional information available to the detector.

C. Velocity Estimation

As velocity estimation is a downstream calculation, with the results relying heavily on the consistency and accuracy of Vehicle Detection and Position estimation, the most significant improvements in velocity estimation will come from upstream improvements.

The main area for improvement for velocity estimation however, is the change from a linear regression estimation for velocity to a non-linear regression. This would allow for the fact that velocities (both in the z and x directions) do not need to be linear. The inter-frame vehicle matching algorithm could also be improved by implementing a SIFT key-point algorithm between bounding boxes to more accurately match vehicles.

D. Lane Detection

As shown in Figure 22, the proposed pipeline was resistant to occlusion and shadows, a property which is not necessarily shared by classical lane detection approaches. The model typically struggled with ambiguous or diverging lanes like in Figure 23. In these examples, a linear fit produces smoother detections and addresses some of the jitter caused by the mean polyline fit.

The first experiment demonstrated that Adam was the better performing optimiser, and that the loss landscape is complex, with potentially many local minima. For this reason, SGD without momentum performed very poorly, as shown in Figure 24. Adam made impressive improvements on the network where SGD would converge, as shown in 25, which shows the loss of the best performing network, that which was trained using adaptive momentum and a batch size of five. All networks were evaluated as seen in Figure 26. In general, Adam and SGD took rougly the same amount of time to train when using momentum.

It was found that the mean poly-line fitting method outperformed linear regression by about a few percent. Polynomial regression, RANSAC or further segmentation of the image to address an example like 23 are possible extensions to the model-fitting step.

Some other areas for improvement include temporal integration filters such as Kalman filtering. These would reduce the number of sporadic false positives. Also, adding SCNN-style message passing layers has been shown to enhance CNN performance in lane detection, and could be implemented as an extension to ERFNet.

VII. Conclusion

Although further experimentation could be used to improve the results in the problem domain, the performance of our algorithms have performed at a satisfactory level. Classification and Lane Detection have an overall performance metric of over 90% whilst lane and velocity detection have a low deviation rate despite the limited 2 dimensional information provided as input. The Velocity Estimation method was able to consistently estimate velocity, while removing random error caused by upstream artifacts. With an average error of 2m/s in both axes, the analytical performance of the algorithm was considered satisfactory. Lane detection was performed with a very high accuracy, efficiently generating poly-line lane markings resistant to occlusion and shadows.

Ultimately, the set of algorithms implemented to detect vehicles, estimate their position and velocity, and detect lane lines shows the first steps in creating a vehicle mounted monocular camera system for self-driving.

VIII. Contribution of Group Members

1) *Daniel P., z5209600:* Improved Car Classification.
2) *Andrew S., z5109480:* Implemented Distance Detection.
3) *Tom W., z5207952:* Implemented Velocity Detection.
4) *Dean P., z5122508:* Implemented Lane Detection.

References

[1] TuSimple, "Velocity estimation challenge." github.com/TuSimple/tusimple-benchmark. Accessed: 2021-03-31.

[2] T. Zhao and R. Nevatia, "Car detection in low resolution aerial images," *Image and Vision Computing*, vol. 21, no. 8, pp. 693–703, 2003.

[3] N. Ammour, H. Alhichri, Y. Bazi, B. Benjdira, N. Alajlan, and M. Zuair, "Deep learning approach for car detection in uav imagery," *Remote Sensing*, vol. 9, no. 4, 2017.

[4] P. Viola, M. J. Jones, and D. Snow, "Detecting pedestrians using patterns of motion and appearance," *International Journal of Computer Vision*, vol. 63, pp. 153–161, Jul 2005.

[5] I. M. Creusen, R. G. J. Wijnhoven, E. Herbschleb, and P. H. N. de With, "Color exploitation in hog-based traffic sign detection," in *2010 IEEE International Conference on Image Processing*, pp. 2669–2672, 2010.

[6] N. Buch, S. A. Velastin, and J. Orwell, "A review of computer vision techniques for the analysis of urban traffic," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 3, pp. 920–939, 2011.

[7] C. Goerick, D. Noll, and M. Werner, "Artificial neural networks in real-time car detection and tracking applications," *Pattern Recognition Letters*, vol. 17, no. 4, pp. 335–343, 1996. Neural Networks for Computer Vision Applications.

[8] B. Coifman, D. Beymer, P. McLauchlan, and J. Malik, "A real-time computer vision system for vehicle tracking and traffic surveillance," *Transportation Research Part C: Emerging Technologies*, vol. 6, no. 4, pp. 271–288, 1998.

[9] H. Grabner, T. T. Nguyen, B. Gruber, and H. Bischof, "On-line boosting-based car detection from aerial images," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 63, no. 3, pp. 382–396, 2008.

[10] J. Janai, F. Güney, A. Behl, and A. Geiger, "Computer vision for autonomous vehicles: Problems, datasets and state of the art," 2021.

[11] M. Enzweiler and D. M. Gavrila, "Monocular pedestrian detection: Survey and experiments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 12, pp. 2179–2195, 2009.

[12] C. Wojek and B. Schiele, "A performance evaluation of single and multi-feature people detection," in *Pattern Recognition* (G. Rigoll, ed.), (Berlin, Heidelberg), pp. 82–91, Springer Berlin Heidelberg, 2008.

[13] G. Li, X. Fang, K. Khoshelham, and S. O. Elberink, "Detection of cars in mobile lidar point clouds," in *2018 3rd IEEE International Conference on Intelligent Transportation Engineering (ICITE)*, pp. 259–263, 2018.

[14] L. Giubbolini, "A multistatic microwave radar sensor for short range anticollision warning," *IEEE Transactions on Vehicular Technology*, vol. 49, no. 6, pp. 2270–2275, 2000.

[15] Y. Shima, "Inter-vehicle distance detection based on keypoint matching for stereo images," in *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, pp. 1–6, 2017.

[16] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second ed., 2004.

[17] M. Kampelmühler, M. G. Müller, and C. Feichtenhofer, "Camera-based vehicle velocity estimation from monocular video," *arXiv preprint arXiv:1802.07094*, 2018.

[18] J. Farrelly and P. Wellstead, "Estimation of vehicle lateral velocity," in *Proceeding of the 1996 IEEE International Conference on Control Applications IEEE International Conference on Control Applications held together with IEEE International Symposium on Intelligent Contro*, pp. 552–557, IEEE, 1996.

[19] J.-Y. Choi, K.-S. Sung, and Y.-K. Yang, "Multiple vehicles detection and tracking based on scale-invariant feature transform," in *2007 IEEE Intelligent Transportation Systems Conference*, pp. 528–533, IEEE, 2007.

[20] J. Badenas, J. M. Sanchiz, and F. Pla, "Motion-based segmentation and region tracking in image sequences," *Pattern recognition*, vol. 34, no. 3, pp. 661–670, 2001.

[21] D. Schreiber, B. Alefs, and M. Clabian, "Single camera lane detection and tracking," in *Proceedings. 2005 IEEE Intelligent Transportation Systems, 2005.*, pp. 302–307, 2005.

[22] A. A. Assidiq, O. O. Khalifa, M. R. Islam, and S. Khan, "Real time lane detection for autonomous vehicles," in *2008 International Conference on Computer and Communication Engineering*, pp. 82–88, 2008.

[23] M. Aly, "Real time detection of lane markers in urban streets," in *2008 IEEE Intelligent Vehicles Symposium*, pp. 7–12, 2008.

[24] A. Borkar, M. Hayes, and M. T. Smith, "Lane detection and tracking using a layered approach," in *Advanced Concepts for Intelligent Vision Systems* (J. Blanc-Talon, W. Philips, D. Popescu, and P. Scheunders, eds.), (Berlin, Heidelberg), pp. 474–484, Springer Berlin Heidelberg, 2009.

[25] 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, pp. 727–745, Apr 2014.

[26] H.-Y. Cheng, B.-S. Jeng, P.-T. Tseng, and K.-C. Fan, "Lane detection with moving vehicles in the traffic scenes," *IEEE transactions on intelligent transportati on systems.*, vol. 7, no. 4, pp. 571–582, 2006.

[27] K. Yamaguchi, A. Watanabe, T. Naito, and Y. Ninomiya, "Road region estimation using a sequence of monocular images," in *2008 19th International Conference on Pattern Recognition*, pp. 1–4, 2008.

[28] M. Nieto, L. Salgado, F. Jaureguizar, and J. Arrospide, "Robust multiple lane road modeling based on perspective analysis," in *2008 15th IEEE International Conference on Image Processing*, pp. 2396–2399, 2008.

[29] J. McCall and M. Trivedi, "Video-based lane estimation and tracking for driver assistance: Survey, system, and evaluation," *IEEE transactions on intelligent transportati on systems.*, vol. 7, no. 1, pp. 20–37, 2006.

[30] H. Sawano and M. Okada, "A road extraction method by an active contour model with inertia and differential features," *IEICE transactions on information and systems*, vol. 89, no. 7, pp. 2257–2267, 2006.

[31] E. Romera, J. M. Alvarez, L. M. Bergasa, and R. Arroyo, "Erfnet: Efficient residual factorized convnet for real-time semantic segmentation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 1, pp. 263–272, 2017.

[32] C. J. Jin, G. R. Chang, W. Cheng, and H. Y. Jiang, "Background extraction and vehicle detection method based on histogram in ycbcr color space," *Applied Mechanics and Materials*, vol. 182-183, pp. 530–534, 2012.

[33] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, pp. 886–893 vol. 1, 2005.

[34] G. Jocher, A. Stoken, J. Borovec, NanoCode012, A. Chaurasia, TaoXie, L. Changyu, A. V, Laughing, tkianai, yxNONG, A. Hogan, lorenzomammana, AlexWang1900, J. Hajek, L. Diaconu, Marc, Y. Kwon, oleg, wanghaoyang0106, Y. Defretin, A. Lohia, ml5ah, B. Milanko, B. Fineran, D. Khromov, D. Yiwei, Doug, Durgesh, and F. Ingham, "ultralytics/yolov5: v5.0 - YOLOv5-P6 1280 models, AWS, Supervise.ly and YouTube integrations," Apr. 2021.

[35] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[36] Ultralytics, "yolov5."

[37] k. simek, "Dissecting the camera matrix, part 3: The intrinsic matrix."

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[39] F. Pizzati, M. Allodi, A. Barrera, and F. García, "Lane detection and classification using cascaded cnns," 2019.

[40] TuSimple, "Lane detection challenge." github.com/TuSimple/tusimple-benchmark. Accessed: 2021-03-31.