Self-Driving Car

A Deep-Learning Approach

[1]Rutvik Shah
UG Student, Computer Dept,
BVM Engineering College,
V.V.Nagar, India.
rutvik2309@gmail.com

[2]Mit Patel
UG Student, Computer Dept,
BVM Engineering College,
V.V.Nagar, India.
mitbpatel0128@gmail.com

[3]Mayank Budhwani
UG Student, I.T Dept,
BVM Engineering College,
V.V.Nagar, India.
mayankbudhwani@gmail.com

[4]Ketan Upadhyay
Technical Director,
Access Computech Pvt Ltd,
Vadodara, India
ketan@acpl.ind.in

[5]Dr. Zankhana H. Shah
Professor, I.T Dept,
BVM Engg College,
V.V.Nagar, India
zankhana.shah@bvmengineering.ac.in

Professor, Computer Dept,
BVM Engineering College,
V.V.Nagar, India.
nmpatel@bvmengineering.ac.in

[6]Dr. Narendra M. Patel
[7]Dr. Darshak G. Thakore
Head, Computer Dept,
BVM Engineering College,
V.V.Nagar, India.
dgthakore@bvmengineering.ac.in

Abstract*: *Nowadays self-directed learning and automation are not restricted to human beings only. If you stare out at the automotive horizon, you can see a new exciting era coming into limelight: the age of self-driving cars. An age when humans will no longer need to keep their eyes on the road. No more concerns about distraction while driving or those stressful rush hour commutes, vehicles will whisk us where we want to go, blazingly fast and efficiently. This paper aims at demonstrating a system, which is able to drive a car on road without any human input. Both software and hardware parts are discussed here. The vehicle would contain certain sensors such as GPS, Ultrasonic Sensor, Camera and would contain an on-board computer for decision making. Waypoint data would be obtained from a nav provider like Google Maps. All of it would be simulated in CARLA, an open-source simulator.

***Keywords*: deep learning, neural networks, image processing, route planning, self-driving car.**

(Article history: Received: 6th February 2021 and accepted 17th May 2021)

I. Introduction

The idea of self-driving cars is a little dated, but its only in recent times that it has come anywhere near to being properly implemented. Companies like Google, Tesla, Waymo are involved in the research of self-driving cars and are even selling it commercially. Although the system is not perfect, it will soon be, with reducing hardware costs, increasing compute power and development in Deep Learning algorithms.

This system aims at demonstrating a full-fledged self-driving car, with a complete demonstration in software and a limited demonstration in hardware. The car would have all necessary features required for safely driving on a crowded street, such as object detection, traffic signal detection, maps integration and emergency stop.

II. Literature Survey

While implementing this system, we have used some already existing algorithms which are not developed by us. A summary of some of these algorithms is provided below.

A. Canny Edge Detection [1]

This algorithm is used for detecting various edges (sudden changes in gradient) in the image. The Canny filter is a multi-stage edge detector. It uses a filter based on the derivative of a Gaussian in order to compute the intensity of the gradients. Then, potential edges are thinned down to 1-pixel curves by removing non-maximum pixels of the gradient magnitude. Finally, edge pixels are kept or removed using hysteresis thresholding on the gradient magnitude.

B. Hough Transform [2]

The Hough transform is a method which can be used to extract features of a particular shape within an image, in this case, the lane lines. While edge detection helps in

reducing the data in an image considerably, the output of an edge detection algorithm is still an image.

Hough transform translates the points from the given image into Hough space and then determines the lane lines by finding which points lie on straight lines, and also gives the equations for said lines. The equations make it much easier to interpret the lane lines, compared to images.

C. Object Detection Algorithms

Several object detection algorithms were studied in order to find the most accurate algorithm for this application. Traditional CNN is not viable to use for object detection due to the huge computational load. R-CNN (Region-based CNN) [3] reduces the number of regions to just 2000 using selective search, but it is still not real-time, taking around 47 seconds per image. Fast R-CNN [4] is an improved version of R-CNN as it feeds input image to CNN to generate convolutional feature map. But selective search still consumes too much time in this algorithm. Faster R-CNN [5] improves further on Fast R-CNN, as instead of using selective search algorithm on the feature map to identify the region proposals, a separate network is used to predict the region proposals.

You only look once (YOLO) [6] is an object detection system targeted for real-time processing. YOLO has better performance compared to other algorithms and good mean average precision, and it predicts all objects in a single pass with locations. Image is divided into grids and each grid is responsible to find objects. A single CNN predicts multiple bounding boxes and also their class probabilities. After non-max suppression, it outputs the boxes with their most probable class. YOLO uses a network involving 106 layers. The basic network structure of YOLOv3 is shown in Fig. 1.

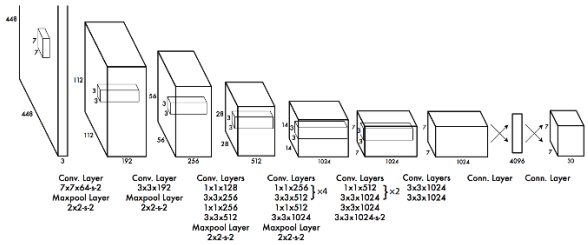

Fig. 1: YOLOv3 Model [6]

D. Transfer Learning

Transfer learning is a technique to use existing knowledge for solving another similar problem. Generally, this strategy is being considered while dataset is limited, and accuracy is also matter in final outcome. One model is trained using big dataset that is called pre-trained model and people use that model to fulfill their requirements. Pre-trained models have defined weights value which helps to get good accuracy with lesser loss. Here, model training is performed using new dataset.

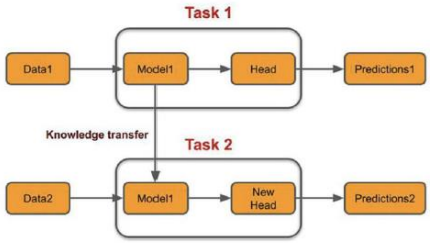

Fig. 2: Block Diagram of Transfer Learning [7]

E. Model Predictive Control

The main idea of MPC is in the robustification of physical control processes under uncertainty carried-out in an online manner. Model Predictive Control (MPC) can be used, in this system, to calculate the actuator values for the car. The control action is obtained by minimizing an objective function at each time step over a finite horizon subject to the equations and constraints of the model. The major advantage of MPC is its straight-forward design procedure. Given a model of the system, including constraints, one only needs to set up an objective function that incorporates the control objectives.

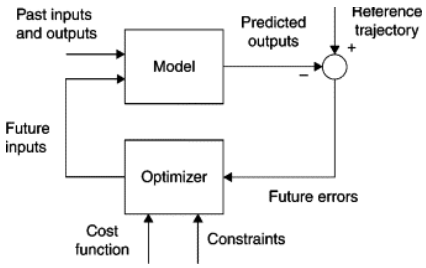

Fig. 3: MPC Controller [8]

III. Proposed Methodology

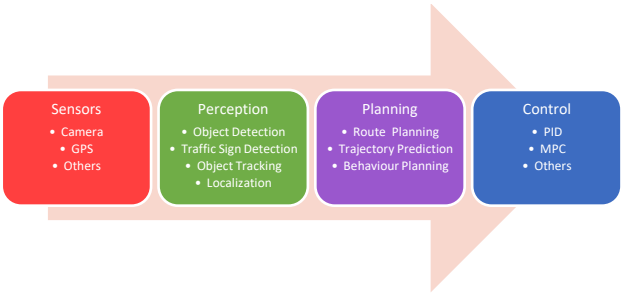

Fig. 4: System Design

The system involves various sensors to collect real-time information about the driving environment. Further, the collected data is processed to extract useful information. Based on the extracted information, the vehicle's route is planned in order to drive the vehicle safely and to reach destination from source. Further, based on this determined route, the controllers will actually determine the vehicle parameters (eg. steer, throttle, brake) so as to actually drive the vehicle on the said route.

IV. Implementation and Results

The pipeline of our system along with brief module breakdown can be summarized by Fig. 5.

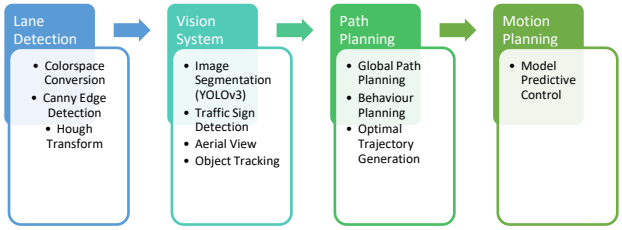

Fig. 5: Implementation Pipeline

In this section, various modules of the system are discussed in detail, along with the results obtained from the implementation of these modules.

A. Lane Detection

Identifying lane lines on the road is a pretty common task performed by all human drivers to ensure their vehicles are inside lane constraints when driving, so as to make sure smoother traffic and to avoid chances of collisions with other objects due to lane misalignment. Similarly, it is a most crucial task for an autonomous vehicle to perform. Eventually recognizing lane markings on roads is possible using state of the art computer vision techniques.

1) The pipeline [9]

Lane-detection is not a single-shot process. When performing lane detection in images, the image needs to passed through multiple transformations in order to obtain an accurate position of the lane lines in the given image. This system uses a standard proven lane detection pipeline, involving color space conversion, Canny Edge Detection and Hough Transform.

Fig. 6: RGB Image

Fig. 7: Grayscale Image with Extracted Lane Lines

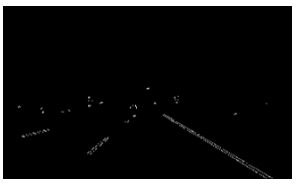

Fig. 8: Canny Edge Detection

Fig. 9: Detected Lane Lines

B. Object Detection

Object Detection is essential for a self-driving system. Images are used to detect objects and their classes. Object detection is part of the Vision system and it is important to recognize obstacles on the road. It helps us to plan the path of our vehicle as we can track motion of surrounding objects. YOLO algorithm is used for detecting surrounding objects such as cars, pedestrians etc.

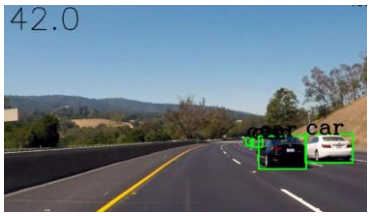

Fig. 10: Object Detection

C. Traffic Sign Detection

Every country has its own set of rules and regulations. There are many traffic rules in the world but the sign and its meaning is deferring according to the province. Traffic signs play an important role in driving so vehicles must be moved accordingly. Indian traffic rule book has ninety traffic signs but only 27 signs are implemented for this project.

1) Transfer Learning using YOLOv3

HyperLabel was used to create a dataset involving various traffic lights and signs and label them. After creating the dataset the model was trained on cloud (Tesla K80).

Model was trained for 220 epochs, because the error had sufficiently declined by that point. This model predicts many signs in one image so for better performance we made an algorithm. The algorithm is made in such a way that it finds specific traffic signs in the picture. Loss of the model is 4.173. Model accuracy was 0.91.

2) Non-Maximal Supression

This model generates images which have many bounding boxes on detected objects. This step filters out all other bounding boxes and only keep that box whose class probability is maximum.

Fig. 11: Traffic Signs Detected

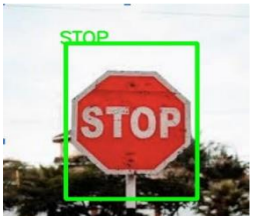

Fig. 12: After Non-max Suppression

D. Aerial View

The vehicle has multiple cameras pointing in all direction. A six-camera system was used in this case for data collection. After that, a neural network was implemented to reconstruct the aerial (top down view). An aerial view has several advantages as it combines useful information from multiple camera angles into one and can be used for local path planning.

1) Dataset Collection and Preprocessing

The model was trained on semantically segmented images gathered in the default town available in CARLA (Town02), 40 episodes, each episode 1000 frames long. For validation and testing, separate episodes were used. The image shape used was 144 x 96.

Fig. 13: Semantic Segmentation in Carla

There are total 12 categories of objects in Carla and for ease, it was decided to split it out into 3 categories (roads, road lines), (vehicles), (everything else).

2) Training

The model architecture designed for obtaining aerial view is shown in Fig. 14.

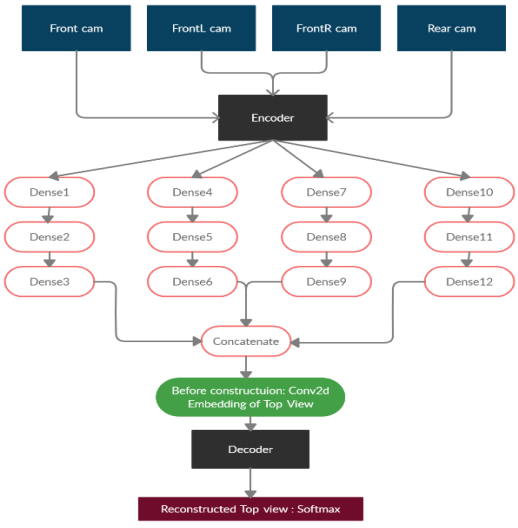

Fig. 14: Network Architecture

In this setup training took ~10hrs on Tesla P100. Inference on same machine takes 3.8ms (checked in real-time via synchronous mode). However, in reality (RGB image to segmented image) some time would also be required to obtain segmented image. Loss of said model was 0.061 on validation set.

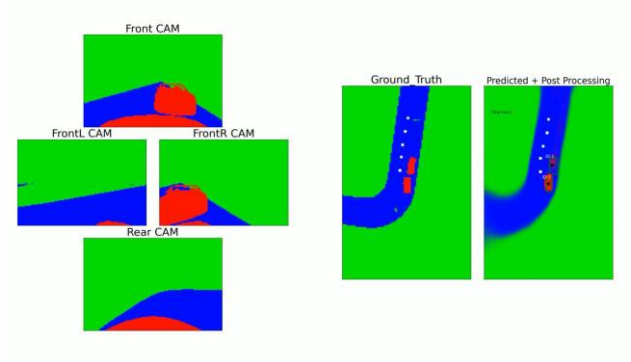

Fig. 15: Top View Comparison between Ground Truth and Prediction

The rightmost top down view was obtained by taking an argmax of the predicted class probabilities. This approach has a detrimental drawback: at edge cases the cars seem to be re-emerging out of the blue, even though actually the predicted probability for their class is not void.

3) Object Tracking in Aerial View

After predicting top view, we have extracted pixels with higher car class probability. In order to find disjoint clusters of car, flood fill algorithm was used to extract out group of connected pixels. In Fig. 16 black dots are representing centroid of each cluster.

Finally, to get an extra edge, we can track motion of these objects by assigning them unique id. However, we have to solve multiple issues, starting from duplicate detection to random reappearance of an object. We have used SORT (Simple Online and Realtime Tracking) algorithm [10] which is most commonly used in object detection algorithms for tracking surrounding objects and it requires only bounding box information of detected objects. In Fig. 16, bounding boxes corresponding to id are results of SORT algorithm.

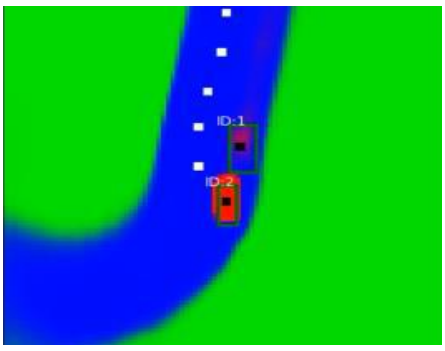

Fig. 16: Tracking of Cars using Unique ID

E. Local Path Planning

Local path planning is the process of planning out the path of the vehicle in order to avoid other objects, switch lanes, overtaking, making turns and so on. It is a very complex task as it involves sensor data, localization and prediction.

Planning a path that is both safe and efficient is one of the hardest problems in the development of an autonomous vehicle. In fact, this step, known as Path Planning, is still an active area of research. The reason why Path Planning is such a complex task is because it involves all components of a self-driving vehicle, ranging from the low level actuators, the sensors which are fused to create a "snapshot" of the world, along with the localization and prediction modules to understand precisely where we are and what actions different entities in our world (other vehicles, humans, animals, etc.) are more likely to take in the next few seconds.

1) Core Logic

We humans transition our vehicle into different states based on our driving style, outside information and destination. We can codify states for machines and instruct them which states they could move to. In this case, the finite state machine is quite straightforward and illustrated below:

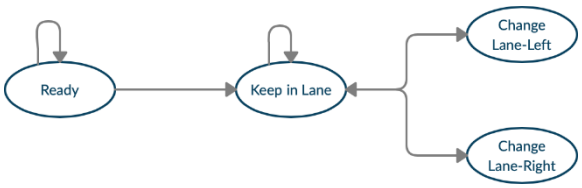

Fig. 17: FSM of Local Path Planning

2) Waypoints Generation using Aerial View

Now, the truth is we only three classes are required to be able to drive around and not collide with others: road, vehicles, everything else.

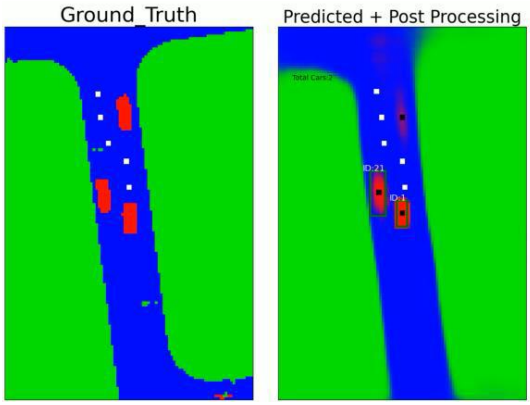

Fig. 18: Path Planning based on predicted Top View

The white dots denote the waypoints of the path planning procedure. The path is found using a greedy algorithm that scans a range of possible waypoints, takes a sphere centered on a given waypoint, and calculates the "average road" for that sphere. Then, the waypoint with the highest "average road" is chosen, and the procedure is repeated. This component generates waypoints which can be followed by standard controller such as MPC, PID etc.

3) Trajectory Models

This module is responsible for generating accurate trajectory based on ego car's current position, which is then supplied to the controller. We have considered first 10 waypoints from the position of our vehicle and to get rid of random gaps between these points, they are fit onto a curve using spline interpolation [11] as it uses piecewise construction.

F. Motion Models

We created two types of controllers – a PID and a MPC controller. A PID controller [12] continuously calculates error value and applies a correction based on proportional, integral and derivative terms. However, a PID controller is inconsistent and may result in abnormal driving behavior in certain conditions. Hence, we decided to use MPC for this project, which is a better controller which involves much more parameters than a PID controller.

MPC predicts the next action by taking advantage of viewing the results of a longer future plan (1 sec). In our example, we use waypoints generated by path planner to fit a 3-rd order polynomial function, which is used to compute y-coordinate and heading.

$$y = f(x) = a_3x^3 + a_2x^2 + a_1x + a_0$$

$$\tan(\varphi_{dest}) = \frac{dy}{dx} = 3a_3x^2 + 2a_2x + a_1$$

Next, we will define a dynamic model for predicting the car's state at time *t+1* from the last state at time *t*. Using the kinematic bicycle model, we can deduce the location, the heading direction and the speed from the last state. In addition to that, we have added 2 more states to measure the cross-track error and the heading error for ψ.

$$x_{t+1} = x_t + v_t \cos(\varphi_t)dt$$

$$y_{t+1} = y_t + \sin(\varphi_t)dt$$

$$\varphi_{t+1} = \varphi_t + \frac{v_t}{l_f}\delta_t dt$$

$$v_{t+1} = v_t + a_t dt$$

$$cte_{t+1} = f(x_t) - y_t + v_t \sin(e\varphi_t)dt$$

$$e\varphi_{t+1} = \varphi_t + \varphi_{dest}\frac{v_t}{l_f}\delta_t dt$$

Finally, we define a cost function to optimize our path with the trajectory. In our model, our cost includes

CTE, heading error, speed error, steer error, acceleration cost, steering rate change and acceleration rate change.

Fig. 25 shows output of MPC for 1 step which includes actuator values of 10 time-steps separated by 0.1 seconds. We have also compared reference trajectory (generated by path planner) and predicted trajectory (MPC) in this figure.

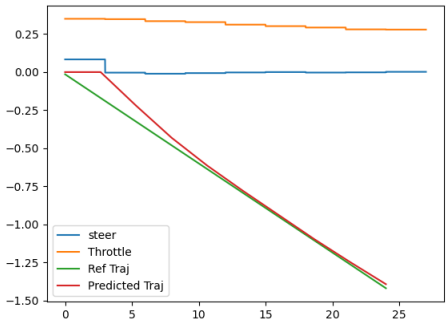

Fig.19: Snapshot of MPC's Action Plan

Apart from following trajectory very accurately we have noticed MPC gives clear idea about what vehicle should do.

G. Global Path Planning

A global route planner guides the vehicle regarding which route to follow in order to reach the destination from the source. Detailed implementation of a global route planner is out of the scope of this paper. A basic global route planner works by interpreting the map as a graph, in which vertices represent various positions in the map, and then calculating shortest valid path between two points.

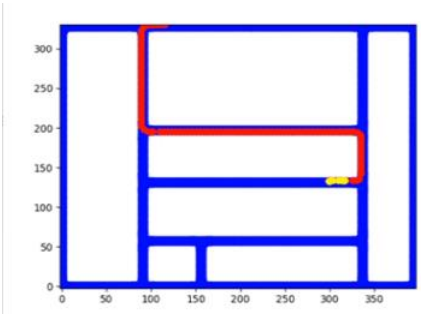

Fig.20: Route Calculated by Global Planner

H. Hardware Based Implementation

In order to demonstrate the abilities of the created models and controllers, a limited version was implemented on a small hardware model car build using a Raspberry Pi. The system for hardware-based implementation is similar to that of simulator approach.

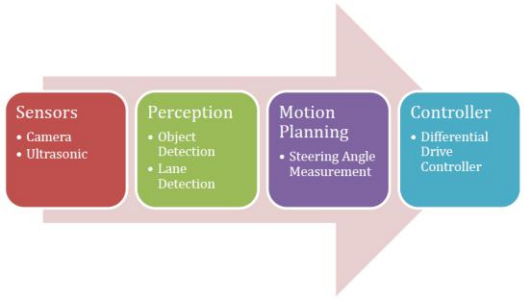

Fig. 21: Hardware Approach

1) Differential Drive Control

The differential drive is a two-wheeled drive system with independent actuators for each wheel. The difference between voltages supplied to motors at both sides makes car turn to a certain radius either right or left.

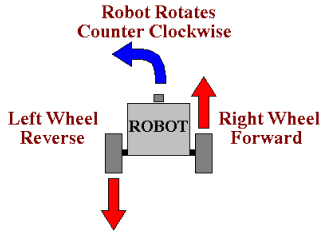

Fig. 22: Differential Drive Control [13]

2) PWM – for controlling speed [14]

The speed of a DC motor can be controlled by varying its input voltage. A common technique for doing this is to use PWM (Pulse Width Modulation), where average voltage value is adjusted by sending a series of pulses. The average voltage is proportional to the width of the pulses known as Duty Cycle.

3) From Software to Hardware

Since the hardware model contains a single camera, it is capable of performing lane detection. The camera is mounted on the front of the car and will detect lanes and according to it determine the center line and try to drive the car on it.

Some examples of detected lanes:

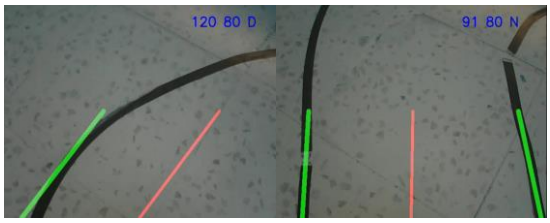

Fig. 23: Lanes Detected by Hardware

Using this technique, it is possible to detect lanes. After detecting lanes, the steering angle can easily be calculated. If only one lane is visible, the car tries to steer

in the opposite direction till the other lane becomes visible.

However, there are some implementation-related issues. Due to a lack of high-speed camera, clear images in motion cant be captured. Which may lead to the algorithm not working in some cases. To continuously drive the car accurately in lane, multiple high-speed cameras need to be mounted on the vehicle.

The next step would have to be object detection. However, object detection would not be possible with the limited compute capabilities of a Raspberry Pi. Object detection algorithms are very compute-extensive, and hence not possible to implement on a model. Cloud processing also has extreme latency issues. An ultrasonic sensor is used on the model to implement emergency-stop and to detect other objects in front. Most such applications use an on-board high-performance computer for processing tasks and decision-making tasks.

V. Conclusion

The idea of self-driving cars does not seem so distant anymore. With ever-increasing compute power and more powerful deep learning algorithms emerging, self-driving cars will soon become much more accessible to everyone.

While implementing this project, some issues arose such as lack of compute power, requirement of high-speed cameras and sensors such as LIDAR for better recognition of surrounding, limited availability of open-source tools for simulating a driving environment, inaccurate detection of some traffic signs such as stop sign, red light due to limited dataset size, etc. For example, in this demonstration(hardware) the vision system runs a little slow due to limited compute power of a single-board computer such as Raspberry Pi. However, most issues are easily solvable with the proper hardware. Over time, it's likely that the improvements in self-driving cars make manually driven cars obsolete.

References

[1] Y. Open CV Tutorials, "Canny Edge Detection Tutorial", https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html

[2] University of Edinburgh "Hough Transform", http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm

[3] Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik "Rich feature hierarchies for accurate object detection and semantic segmentation Tech report (v5)", 2014 IEEE Conference on Computer Vision and Pattern Recognition

[4] Ross Girshick. "Fast R-CNN" 2015 IEEE International Conference on Computer Vision (ICCV)

[5] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks" June 2015 IEEE Transactions on Pattern Analysis and Machine Intelligence

[6] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi "You Only Look Once: Unified, Real-Time Object Detection", 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)

[7] Sebastian Ruder, Matthew E. Peters, Swabha Swayamdipta, Thomas Wolf "Transfer Learning in Natural Language Processing", Jun 2019

[8] A. Giovanni Beccuti, Manfred Morari, "Analysis and Design of Hybrid Systems", 2006

[9] Nachiket Tansksale "Finding Lane Lines – Simple Pipeline for Lane Detection", https://towardsdatascience.com/finding-lane-lines-simple-pipeline-for-lane-detection-d02b62e7572b

[10] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, Ben Upcroft "Simple Online and Realtime Tracking", Feb 2016

[11] Steven Chapra, Raymond Canale "Numerical Methods for Engineers", McGraw Hill Education, May 2010

[12] Paul Avery "Introduction to PID Control" https://www.machinedesign.com/automation-iiot/sensors/article/21831887/introduction-to-pid-control

[13] Society of Robots "Programming – Differential Drive" https://www.societyofrobots.com/programming_differentialdrive.shtml

[14] Dejan "Arduino DC Motor Control Tutorial – L298N | PWM | H-Bridge" https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/

Author Profiles

Rutvik Shah
UG Student, Computer Department,
BVM Engineering College,
V.V.Nagar, India

Mit Patel
UG Student, Computer Department,
BVM Engineering College,
V.V.Nagar, India

Mayank Budhwani
UG Student, I.T. Department,
BVM Engineering College,
V.V.Nagar, India

Ketan Upadhyay
Technical Director,
Access Computech Pvt Ltd,
Vadodara, India

Dr. Zankhana H. Shah
Associate Professor, I.T. Department,
BVM Engineering College,
V.V.Nagar, India

Dr. Narendra M. Patel
Associate Professor, Computer Department,
BVM Engineering College,
V.V.Nagar, India

Dr. Darshak G. Thakore
Head, Computer Department,
BVM Engineering College,
V.V.Nagar, India