Minimal Self-Driving Car agent

Aadesh Neupane
Brigham Young University
Provo, Utah, USA
aadeshnpn@byu.edu

Najma Mathema
Brigham Young University
Provo, Utah, USA
nmathema@byu.edu

Abstract

It is crucial not only to understand the specialized subsystem of an autonomous vehicle like lane detection, vSLAM, and traffic light detection to build a safe and reliable self-driving car, but understanding the subsystem interaction with each other is equally important. So for our CS704R project, first, we independently implement those specialized subsystems. Second, we combine those modules to build a minimal self-driving car agent in ROS. Finally, we test our agent in a simulated highway environment. Our minimal agent was successful in driving on the highway track.

1. Introduction

Automation in driving has the potential to allow excellent road safety by reducing risky driver behavior like Driving Under Influence (DUI), speeding, distraction, and many more. With so many miles to be traveled every year and so much time to spend in traffic, self-driving cars can help save humans' time and energy for other productive works, assuring more safety and efficiency in commuting. Additionally, it also serves as a practical approach for elderly and disabled people, allowing independence and comfort for traveling.

The autonomous driving field is diverse and consists of unified research among various disciplines, including robotics, computer vision, deep learning, control, and engineering. It takes accomplished research groups years to design and implement a self-driving car. This project's scope was to implement a few specialized subsystems such as Lane detection, vSLAM, and traffic light detection and understand their interaction in a minimal self-driving car agent.

An online learning platform called Udacity has introduced the course titled- "Become a Self-Driving Car Engineer," which is targeted towards learning and implementing the ideas from computer vision and deep learning to build a self-driving car. They have also provided a simulator platform to test the agent. We leverage the publicly available materials from this course to implement a self-driving car agent and tested the agent in their simulator. Note that we did not have access to any learning or additional materials from Udacity as we were not enrolled in the course. [1]

Figure 1 shows a general self-driving car architecture with four major components: Sensors, Perception, Planning, and Control. Paden et al. [27] in their survey paper emphasized that a) *Perception* and b) *Planning* are the crucial components. The perception system comprises modules responsible for the tasks such as localization, traffic light detection, static and dynamic obstacles detection, vehicle detection, road mapping, and tracking and recognition, and others. Whereas, the planning module comprises subsystems responsible for tasks such as route planning, path planning, behavior selection, motion planning, obstacle avoidance, and control. With our limited CS 704R course timeline in mind, we emphasized vision subsystems and cherry-picked to explore lane detection, traffic light classification, and localization. Then, we integrated those modules with a PID (Proportional–Integral–Derivative) controller and ROS (Robot Operating System) to create a minimal self-driving car that can drive in a simulated highway road.

The remainder of the paper is structured as follows. Section 2 presents the literature review in this area. Section 3 describes the modules we explored for designing the self-driving car, detailing the algorithms implemented to achieve the desired functionalities in each of the modules. Section 4 describes the experiments. Section 5 describes the minimal self-driving car agent. Section 6 presents the conclusion and future work.

2. Related work

'Shakey the Robot' was the first general-purpose robot with rudimentary manipulative abilities and capable of visual, tactile, and acoustic signal processing and pattern

[1] We used the knowledge and intuition built from classes CS 650 and CS 704R. The self-driving course costs $2034, assuming that the student completes the course in six months. We only had access to public repositories that outlines the project description.

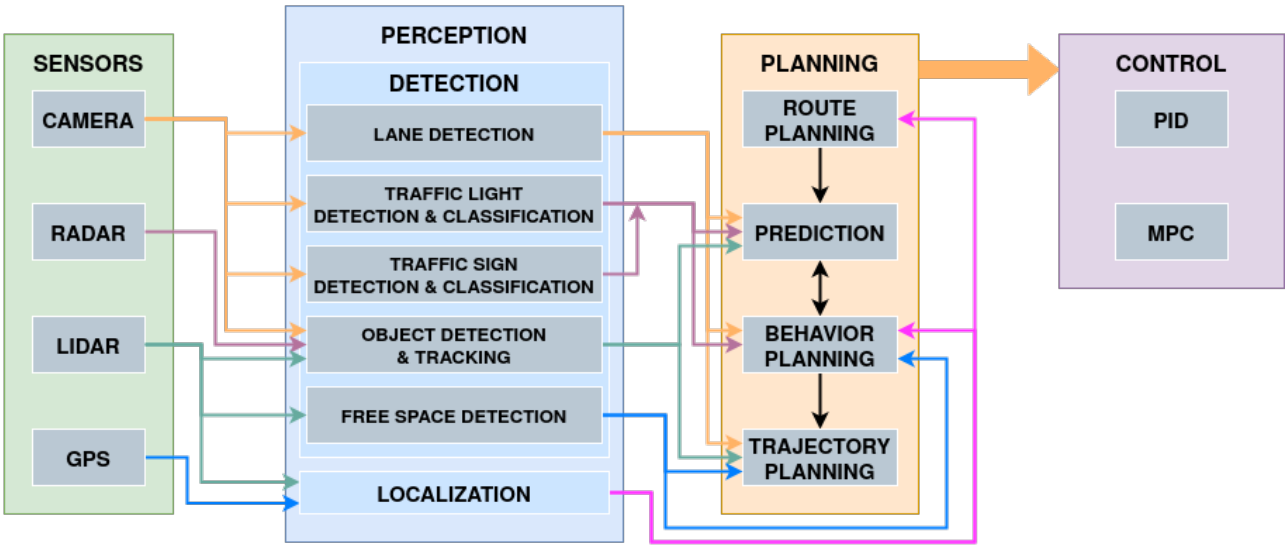

Figure 1. A typical self-driving car architecture.

recognition [25]. Shakey is significant for three distinct reasons: a) its control software was structure b) today's robots still follow its integrated computer vision, planning, and navigation methods, and c) Shakey served as a proof that encouraged later development for more advanced robots. 'Allen' was the first robot based on the subsumption architecture, which had sonar distance and odometry on board. It could avoid both static and dynamic obstacles and move to the distant goal location. Though Shakey and Allen inspired other advanced robots, most of them were built for indoor controlled and lab environments.

DARPA initiated 'The Grand Challenge' in 2003 to spur innovation in unmanned ground vehicle navigation capable of traversing unrehearsed off-road terrain. In the first competition in 2004, out of 15 teams, none of them completed the task. In 2005, five teams finished the task, but 'Stanley' finished first completed the course in 6h 53 minutes [32]. Stanley team treated the autonomous driving task as a software problem by dividing the task into four distinct categories: sensor interface, perception, planning, and control. 'The Grand Challenge' was about autonomous driving in a desert trial, whereas 'The Urban Challenge' was about developing unmanned vehicles that could navigate traffic in a mock urban environment. CMU's *Boss* [34] was the winner followed by Stanford's *Junior* [23] in the urban challenge. The global architecture for Stanley, Junior, and Boss autonomous vehicle were almost similar.

The architecture has barely changed for today's era from Shakey, Allen, Stanly, and Junior, but significant advancement has been made on each module: sensors, perception, planning, and control. One of the vital task for a self-driving car is to identify the lanes, segment the static and dynamic objects, detect and identify the traffic signs. Road lane detection had received much attention in the computer vision community.Kluge et al. [19] proposed an algorithm for lane detection in noisy road images where edge-based lane detection schemes struggle. They used a deformable template model of lane structure to locate lane boundaries without thresholding the intensity gradient information. Lai et al. [20] described a novel algorithm to extracts complete multiple lane information by utilizing prominent orientation and length features of lane markings and curb structures to discriminate against other minor features. Wang et al. [35] propose a Catmull-Rom spline-based lane model that describes the perspective effect of parallel lines for lane detection. Chen et al. [6] proposed a real-time lane detection algorithm that described the road boundary as two parallel hyperbolas on the ground plane. By fitting points with hyperbolas, their model was able to make full use of road boundaries with partial occlusion. Kim et al. [18] described a robust lane detection and tracking system to deal with challenging scenarios based on random-sample consensus and particle filtering algorithms. Parashar et al. [28] introduced the Sparse CNN network that improved performance and energy efficiency by exploiting the zero-valued activations in ReLU operator. SCNN has been successfully applied for efficient lane detection. Zou et al. [39] used multiple frames instead of a single image with a hybrid CNN and RNN network for robust lane detection from continuous driving scenes.

The classical approaches for road traffic sign detection and classification include taking advantage of the traffic sign's unique colors and shapes using simple color thresholding and shape analysis [9, 36]. Detection of traffic signs in real-world images is a difficult problem. To address this issue, Houben et al. [17] create a real-world benchmark

dataset for traffic sign detection and baseline results. Zhu et al. [38] create a huge benchmark dataset bigger than [17]'s dataset and proposed a robust end-to-end CNN model similar to Faster RCNN to detect and classify traffic signs.

Simultaneous Localization and Mapping (SLAM) is the concurrent construction of a map and the estimation of the robot's pose. SLAM is an important component for autonomous vehicles. There has been extensive research in SLAM over the last 30 years. Taketomi et al. [31] did an excellent survey of current visual SLAM systems. They also identified the relationship between visual odometry (VO) [26], vSLAM [8] and structure from motion (SfM) [1]. Estimating the pose of the robot based on the sequential changes in the images is visual odometry. Structure from motion (SfM) is a technique to estimate the camera motion and 3D structure of the environment in a batch manner where vSLAM does it online with individual images. There are also direct methods to perform vSLAM such as DTAM [24], LSD-SLAM [12] and DSO [11].

A self-driving car needs to be aware of its surroundings. Semantic segmentation allows the vehicle to classify the surroundings in semantically meaningful ways. Mask-RCNN [16] is the most popular and state-of-the-art method to perform semantic segmentation. Ha et al. [15] proposed an encoder-decoder network for semantic segmentation of images of street scenes for autonomous vehicles based on RGB-Thermal dataset. Feng et al. [13] reviewed the methodologies for deep multi-modal object detection and semantic segmentation in autonomous driving.

DNN based models are viable to replace the standard pipeline of autonomous vehicles such as depth estimation, optical flow, semantic segmentation, and others [22]. Also, there is are works where the standard pipeline has been replaced by end-to-end deeplearning [3, 21]. Burnett et al. [4] described self-driving car architecture and specialized algorithms that enabled the team to win SAE AutoDrive Challenge to develop a level 4 autonomous vehicle in just six months. Toromanoff et al. [33] developed a novel technique called implicit affordances to effectively leverage RL for urban driving, including lane-keeping, pedestrians and vehicles avoidance, and traffic light detection, and their system won the Camera Only track of CARLA challenge.

3. Methodology

This section describes the sub-systems used to build a self-driving car. A typical self-driving car architecture is shown in Figure 1. *Sensors* includes everything needed to understand its surroundings and location, including cameras, lidar, GPC, radar, and IMU. The *Perception* system includes software modules and pipelines that analyze the massive amount of data coming from sensors into valuable semantic information like lanes, traffic lights, other vehicles, pedestrians, trees, and other static and dynamic objects. Perception also includes localization that maps the object in the environment and estimates the vehicle's current pose concerning the maps. The *Planning* component is responsible for the high-level decision about the vehicle's path that includes path prediction for other dynamic objects, behavior selection, and trajectory estimation. The *Control* module is responsible for converting the planner's high-level decision into actuators signals in terms of the throttle, steering angle, and breaking power. Usually, the control module employs PID or MPC (Model Predictive Control) to smooth the control signals.

Although each subsystem was built and tested individually, each of these was not executed in the final product to test the Udacity self-driving car simulator. The simulator already had a few of the subsystems built-in for ease and efficiency in testing. The details will be presented as the section proceeds. The following are the subsystems developed for the project: Lane finding, Traffic Light Classifier, and Visual Odometery.

3.1. Lane Finding

With road safety as one of the significant purposes of self-driving cars, lane detection is one of its major critical components of the perception system. Lane detection is vital for understanding the driving scene and positioning the car to track the driving route. Lane detection is the process of detecting the allowed, safe drivable region in front of the car and vehicle tracking, where we determine the vehicle's position relative to the road for navigation purposes. There are several ways available in the literature on this problem. In fact, in real-world implementation, it requires the usage of several other technologies including radar[29], lidars[10], ultrasonic sensors[5] along with several cameras to have a complete understanding of the world. However, components like radar and lidars are costly even for industrial manufacturing, whereas cameras serve as an efficient yet economical way to understand the environment.

In this project, we have tried three different ways to tackle this task by implementing it via the traditional computer vision approach, a deep learning-based model (SCNN)[28], and Advanced Lane Finding methodology. We try to differentiate these methods based on their performance based on computational time and generalization ability.

3.1.1 Classical Computer Vision Approach

The classical approach uses traditional computer vision approaches where the properties of images like the color spaces, gradients, edges, and specific regions of interest within the image are processed. The process is comparatively a lot easier and more straightforward than other deep learning approaches and advanced processes. The entire

pipeline processing is done on a pre-recorded video using the OpenCV library.

The video is loaded, and processing is done on each frame. The first step is to convert the frame to grayscale and clean up the frame by removing noise by applying a Gaussian Blur. This is one of the crucial steps for edge detection, which is the objective of lane detection. This is followed by a combination of Binary Thresholding and Otsu's Thresholding so that the algorithm itself finds the appropriate threshold value. The process of applying a Gaussian blur followed by Otsu's Binarization yielded the best results.

Canny Edge Detection is applied for Edge Detection. The threshold value is the same as identified by Otsu's thresholding algorithm. This image is dilated to enhance the image's structure by convolving the image with a 5*5 filter. This allows the brighter areas or the foreground to grow and darker areas or background to reduce. Thus, it allows the objects of interest to be more prominent in the image. The final step is to apply Hough Transform to recognize the edges or lines and their position in the image. Thus the algorithm extracts all the lines passing through each edge point and group them by similarity.

3.1.2 Advanced Lane Finding Methodology

Building on top of the classical approach, advanced lane finding was done to apply the logic to raw video. Advanced approaches include distortion correction, image rectification, color transforms, and gradient thresholding. The lane's radius of curvature and vehicle position is identified using perspective transform and polynomial fit. The pipeline was created to process a video to detect the lanes on a highway road.

It is necessary for the processing to have a reasonable estimation of camera parameters to measure the planar objects. The camera images could be affected by radial or tangential distortion, which needs to be removed. Thus, the first step is to compute the camera calibration matrix to calibrate the camera and use it for distortion correction to the raw image. For further processing, the frames should be converted to a binary image with accentuating the desired feature, i.e., the lanes. This is obtained by using image manipulation techniques like color space transformations and thresholding. Since most of the lane lines are either yellow or white, these colors are focused on identifying lane lines in the frames. Existing works show that the color space transformation from the RGB(Red, Green, Blue) space to the HLS(Hue, Lightness, and Saturation) space, and using the Saturation channel has been the most effective to detect lane lines based on colors. Thus, we used the RGB to HLS color space transformation to get the saturation channel's right threshold to highlight the white and yellow lane lines. The Sobel operator was used to detect edges in both horizontal and vertical directions based on the image gradients in all directions. Then the obtained threshold was applied to the undistorted image, resulting in the binary image.

The next step is to obtain the region of interest which is the trapezoidal region of the lane to generate the birds eye view image using perspective transform. The vertex coordinates for the trapezoidal region for the source and destination image are determined to calculate the perspective transformation matrix. Then with the matrix, the image is warped accordingly. Then the pixel coordinates of the lane lines from the transformed image need to be identified. We computed a histogram of the bottom half of the transformed binary image in the y-direction to find the x positions with the highest pixel intensities. A sliding windows search is performed in the image regions to find the pixel coordinates for both the right and left side of the lane. After obtaining the pixel coordinates, the equation of the line curve is calculated by computing a second-degree polynomial using python's Numpy's polyfit to obtain the radius of curvature and vehicle position from the centerline. A basic assumption of the lanes' width and the length of lanes are made for the calculations. For vehicle positioning, the car is assumed to be in the center of the image. The deviation from the center of the lane and the center of the picture is calculated. If the deviation is negative, the car is on the lane's left side, and otherwise on the right side.

3.1.3 SCNN-a deep learning based model

SCNN(Spatial Convolutional Neural Network)[28] is a CNN based traffic lane detection algorithm specially developed for capturing the spatial relationship of the row and column pixels of an image. The algorithm is specifically suitable for long continuous shape structure, with a strong spatial relationship but fewer appearance clues. It has shown impressive results on detecting traffic lanes on occluded images where the regular CNN architectures do not do very well.

The authors show that their model significantly improved the performance on the very challenging Cityscape dataset[7], and also outperformed the Recurrent Neural Network (RNN) based ResNet and MRF+CNN (MRFNet) in the lane detection dataset by 8.7% and 4.6% respectively. This model's fundamental idea is to view both the rows or columns of the feature maps as layers and apply convolution and non-linear activation and sum operations sequentially. This gives rise to a deep neural network, and the information is propagated between the neurons in the same layer. As the spatial information is reinforced through the interlayer propagation, this makes it so useful for structured objects like lanes or poles with occlusions.

Instead of working with a CNN's output, the top hidden layer is split into slices, and each slice is sent to a convo-

lutional layer. Unlike a traditional CNN, where the output of a convolution layer is sent to the next layer, each slice's output is added to the next slice to form a new slice. The new slice is then sent to the next convolution layer, and this process continues until the last slice is updated. Thus, the convolution kernel weights are shared across all slices, making it a kind of RNN. Additionally, this method is repeated for slices in directions downward, upward, rightward, and leftward. The main three advantages of using this model are its computational efficiency, message propagation, and flexibility to incorporate in any part of the CNN, not just the top hidden layer.

3.2. Traffic Light Classifier

One of the popular deep learning models, Faster R-CNN[30], was trained on the Bosch Small Traffic Lights Dataset[2] for identifying the traffic light signals. Faster R-CNN is the third and the most widely used iteration of the original R-CNN paper[14] for object detection.

Though the architecture of Faster-RCNN is a bit complex, it is the most computationally efficient version and yields a significant performance improvement. Initially, the images are passed through a pre-trained ResNet50 to obtain a convolutional feature map. These extracted features are passed through a Region Proposal Network(RPN) to find a pre-defined number of regions called the "bounding boxes," which are the locations of possible objects in the image. This was one of the authors' most prominent problems in the paper by generating a variable-length list of bounding boxes by introducing the concept of "Anchors". Anchors are fixed-sized reference bounding boxes placed uniformly throughout the original image. Instead of detecting where the objects are, the anchors are used to identify if they contain relevant objects and how the anchors could be better adjusted to fit the relevant object. The RPN is a fully convolutional network that takes all the anchors and gives two predictions per anchor, the "objectness score"(the probability that an anchor is an object) and bounding box regression for better adjusting the anchor.

The Region of Interest Pooling is used to classify the possible object detected by RPN by extracting fixed-sized feature maps for each possible object. The R-CNN(Region-based CNN) consists of two fully connected layers; one is the classification layer that uses the extracted features to predict the class that the object belongs to, and the other is a regression layer that generated adjustments to be made to the bounding box for precision. The classification also consists of a background class which is assigned to the bad proposals.

3.3. Visual Odometry

Generally, Odometry refers to estimating the sequential changes of position over time using sensors such as wheel encoder to acquire relative sensor movement. When camera images are used instead of special sensors such as wheel encoder or IMU to estimate the robot's pose, it is called Visual Odometry (VO) [26]. Based on the type of camera setup, VO can be either monocular or stereo. Usually, there are three commonly used VO motion estimation techniques called: 3D to 3D, 3D to 2D, and 2D to 2D methods [37]. The first two methods are only applicable when 3D data are available. When depth information is unavailable, 2D to 2D method is used by exploiting the epipolar geometry to estimate the relative transformation between the calibrated image frames. Usually, these procedures are followed to compute VO:

- Features extracted in the first frame F_1 and second frame F_2 using ORB features descriptors
- Match features between the two consecutive frames. Estimate a transformation between the first two frames using the 5-point algorithm and triangulate the corresponding points using this transformation. OpenCV makes it relatively easy to perform this step with *cv2.findEssentialMat* and *cv2.recoverPose* function.
- Extract features in the following frame and matches them with previously extracted features
- Use RANSAC is refined matches in the current frame and the points reconstructed from earlier frames.
- Use the estimate transformation to triangulate the currently matched features between frames
- Report from 4 for every iteration

3.4. ROS nodes for Self-driving Car

Figure 3 shows the architecture of the self-driving car in ROS that was used for the final Udacity's capstone project. Figure 3 is simplification of the general architecture shown in Figure 1. The main task to be completed in this section was to integrate traffic light detection and classification and setup PDI controller to follow waypoints.

3.4.1 Perception

Perception sub-system sees the environment with its sensors and publishes valuable information to other sub-systems. Notably, we only have implemented traffic light detection for our project. It is possible to implement segmentation and obstacle detection algorithms for safe and reliable autonomous vehicles in this sub-section.

Traffic Light Detection Node A crucial part of the vehicle's self-driving capabilities comes from detecting and classifying upcoming traffic lights. This node processes

Figure 2. Three different methods for Lane detection. a) Uses Hough lines, b) Uses perspective transform and histogram, and c) Uses SCNN model

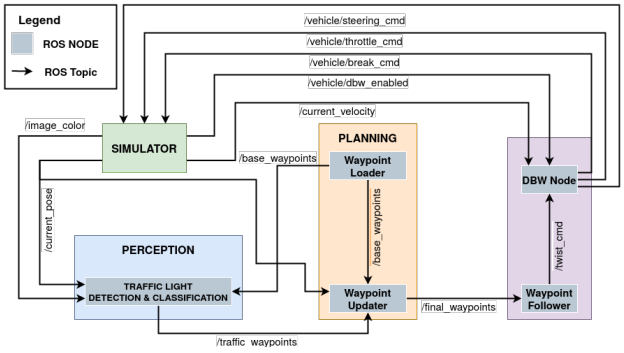

Figure 3. ROS nodes and topics

images provided by the vehicle's onboard camera and publishes upcoming traffic light information to the /traffic_waypoint topic. The Waypoint Updater node uses this information to safely determine if/when the car should slow down to stop at upcoming red lights. We use the model trained in section 4.2 to detect and classify traffic lights. The traffic light detection system was found to be always lagging several seconds behind the simulator. The vehicle would often pass the traffic light when the node finally published the results for that particular light. One of the reasons that the detection node being slow was due to the massive Neural Network parameters.

3.4.2 Planning

The planning module plans the vehicle's path based on the vehicle's current position and velocity, along with the state of upcoming traffic lights. A list of waypoints to follow is passed to the control module.

Waypoint Loader It is a simple node that reads the waypoints along the track and pushes them to the topic /base_waypoints. This node can be used to publish waypoints based on the advanced lane detection module or other modules that track the environment's features.

Waypoint Updater This node performs the core path planning. As seen in Figure 3, it subscribes to three topics to get the entire list of waypoints, the vehicle's current pose and the state of upcoming traffic lights. The node publishes a list of waypoints to follow, where each waypoint contains a position on the map and a target velocity. Every time the vehicle changes its position, a new path is planned. First, the closest waypoint to the vehicle's current position is found, and then a list is created containing the next 200 waypoints. Second, the upcoming traffic lights are analyzed, and speed is adjusted based on the light's state. Following rules control the vehicle speed:

- If the vehicle is within 30 waypoints of green light, speed it slowed to 5 mph.
- If the vehicle is within 50 waypoints of red/yellow light, slow down with a linearly interpolated speed that would stop at the light.
- If the car is within 25 waypoints of a red/yellow light, slow down to 5 mph.
- If the car is less than 4 waypoints from the stoplight, command full stop.
- In all other cases, command the max speed specified.

The final list of waypoints is then published on the /final_waypoints topic.

3.4.3 Control

The control module publishes control commands for the vehicle's steering, throttle, and brakes based on the list of waypoints to follow.

Waypoint Follower This node parses the list of waypoints to follow and publishes proposed linear and angular velocities to /twist_cmd topic.

DBW (Drive By Wire) DBW module is the main module that sends commands to the vehicle to drive. Recall, we have the target linear and angular velocities for each waypoint in the /twist_cmd topic. So the DBW needs to adjust the vehicle's state accordingly with the help of three controllers. *Throttle Controller* is a simple PID controller that compares the current velocity with the target velocity

and adjust the throttle. The throttle gains were tuned that allowed reasonable acceleration without oscillation around the set-point. *Steering Controller* translates the proposed linear and angular velocities into a steering angle based on the vehicle's steering ration and wheelbase length. In order for a smooth, jerk-free ride, the maximum linear and angular acceleration rate is constrained. The controller's steering angle is also passed through a low pass filter to reduce jitter from noisy data. *Braking Controller* is a simple proportionally controller that takes the difference between the current velocity and the proposed velocity. Similar to the throttle controller, the proportional gain is tuned to ensure reasonable stopping distances.

4. Experiment

Four separate experiments were conducted for related sub-systems required for the self-driving car.

4.1. Lane Detection

As described in section 3.1, we experimented with three different approaches for lane detection. Figure 2 shows the output from these approaches on a test image. On average, basic takes around 1.442s to process six images, advance takes around 1.803s, and SCNN takes around 9.806s. Note that SCNN is being run on the CPU. Similarly, on average, basic takes around 27.117s to process a challenging video of length 10s, advance takes around 36.759s, and SCNN takes around 5m 54.175s. Moreover, for a simple video of length 27s, basic takes around 16.616s, advance takes around 52.633s, and SCNN takes around 15m 19.947s [2]. The output from SCNN was not satisfactory as it was detecting extra lines outside the lanes. The advanced method was reasonable faster for the lane detection data and performed well on typical images and challenging scenarios.

4.2. Traffic Light Classification

Bosch Small Traffic Lights Dataset [2] is an accurate dataset for vision-based traffic light detection. This dataset contains 13427 images at a resolution of 1280x720 pixels. There are 15 different labels based on the state of the traffic lights. In our experiments, we merged the 15 different labels into just four labels: Green, Red, Yellow, and Off. A pre-trained FasterRCNN model with a ResNet50 backbone was used to the speedup training process. We trained the network for 100 epochs taking around 9hrs on an AWS server with one NVIDIA Tesla V100 GPUs [3].

4.3. Visual Odometry

The camera matrix was provided with the KITTI dataset. For features detection, cv2.FastFeatureDector was used with minimum features to track set to 1500. Features were tracked in subsequent frames using cv2.calcOpticalFlowPyrLK. cv2.findEssentialMat was used to compute the essential matrix by triangulating the corresponding matching points in subsequent frames, and cv2.recoverPose was used to compute camera pose from the essential matrix solving Perspective-n-Point(PnP) problem. The required camera pose is equivalent to the extrinsic camera parameter with the camera's translation and rotation in the global coordinate system.

[2]The output videos and images from lane detection can be found at _ box folder.

[3]We skipped evaluation on testset to save time

Figure 4. FasterRCNN model successfully detecting traffic lights in urban environment.

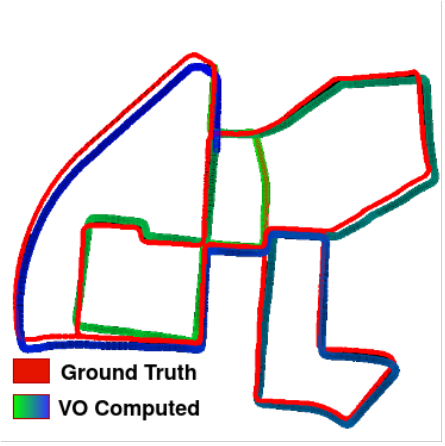

Figure 5. Visual odometry trajectory and ground truth trajectory from the vehicle in KITTI dataset 00. Red path is the ground truth and the gradient path is the estimated pose.

5. Self-driving car

We cloned the Carnd-Capstone project from the official Udacity GitHub. The ROS project skeleton was already provided in the project. The project codebase was based on ROS Indigo. It only supported python2.7. The first task was to convert the codebase to Python3 compatible. We painstaking figured out which packages were needed for the latest ROS Noetic so this project would run. The package configuration and making the codebase compatible with ROS Noetic and Python3 was the most challenging part. The final version of the self-driving car agent

can be found at https://github.com/aadeshnpn/CarND-Capstone

The next step in building a minimal self-driving car was to integrate all the ROS nodes, as shown in Figure 3. The agent architecture has three main components: perception, planning, and controller. We use the traffic light detection module described earlier for perception. The simulator already provides the agents with waypoints or landmarks to follow, so lane detection was not used here. Based on the status of traffic lights, the waypoints are updated with preferred velocity and steering angle. The DBW (Drive by Wire) node uses that to generate control signals using a PID controller. The simulator subscribes to the control commands published by DBW (Drive by Wire) to drive.

We ran the agent several times and noticed that the car was not following the traffic lights through the traffic light node was publishing correct information about the lights' state. After careful analysis, we found out that running the simulator and the agent in the same machine lags the system as the simulator consumes the system resources. Also, the traffic light detection module takes a few seconds to make an inference on an image. When the traffic light modules finally publish the state of traffic lights, the car has already passed the light.

We tried training the Faster-RCNN with VGG as a backbone network to mitigate the issue, assuming that it would reduce the model size, thus speeding inference time. To our surprise, the estimated time for training the network was around 78 hrs on AWS p2.xlarge instance (K80 GPU), so we decided to try a different approach. We found out some discussion threads that listed the potential issue with the lagging was to do with how ROS initializes and stores the DNN model. We tried increasing the ROS buffer, but that did not help.

Neglecting the car's issue of not following traffic lights, the car was successfully in droving on the highway road. Our objective to build a minimal self-driving car agent was completed but with few caveats:a) the simulator provides the waypoints/landmarks, b) the vehicle pose/state is also accessible from the simulator, and c) PID controller was used though Model Predictive Control (MPC) would be more efficient. We tried to address these limitations of the current implementation by integrating the advanced lane detection to create waypoints/landmarks on the fly and use visual odometry to estimate the vehicle's pose to feed to the MPC. This effort to enhance the current implementation with dynamic waypoints, vSLAM, and MPC became demanding than anticipated. With the interest of time, it would be interesting to integrate those new modules as future work.

6. Conclusion and Future Work

We explored and implemented core components like lane detection, visual odometry, and traffic lights detection in this project. We integrated those components for a minimal self-driving car agent. There is definitely room for improvement. Priority would be to create a unified DNN that can do semantic segmentation and Traffic light detection in real-time without lagging. It would be interesting to run the agent and simulator in different machines to understand the computation resource constraints better. Also, visual SLAM needs to be used instead of the manually created waypoints that would generalize better for other environments. Currently, the PID controller is used for the final autonomous vehicle, but MPC use needs to be evaluated in place of the PID controller.

References

[1] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M Seitz, and Richard Szeliski. Building rome in a day. *Communications of the ACM*, 54(10):105–112, 2011. 3

[2] Karsten Behrendt, Libor Novak, and Rami Botros. A deep learning approach to traffic lights: Detection, tracking, and classification. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1370–1377. IEEE, 2017. 5, 7

[3] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016. 3

[4] Keenan Burnett, Andreas Schimpe, Sepehr Samavi, Mona Gridseth, Chengzhi Winston Liu, Qiyang Li, Zachary Kroeze, and Angela P Schoellig. Building a winning self-driving car in six months. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 9583–9589. IEEE, 2019. 3

[5] Alessio Carullo and Marco Parvis. An ultrasonic sensor for distance measurement in automotive applications. *IEEE Sensors journal*, 1(2):143, 2001. 3

[6] Qiang Chen and Hong Wang. A real-time lane detection algorithm based on a hyperbola-pair model. In *2006 IEEE Intelligent Vehicles Symposium*, pages 510–515. IEEE, 2006. 2

[7] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Scharwächter, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset. In *CVPR Workshop on the Future of Datasets in Vision*, volume 2, 2015. 4

[8] Andrew J Davison. Real-time simultaneous localisation and mapping with a single camera. In *null*, page 1403. IEEE, 2003. 3

[9] Arturo De La Escalera, Luis E Moreno, Miguel Angel Salichs, and José María Armingol. Road traffic sign de-

tection and classification. *IEEE transactions on industrial electronics*, 44(6):848–859, 1997. 2

[10] Raúl Dominguez, Enrique Onieva, Javier Alonso, Jorge Villagra, and Carlos Gonzalez. Lidar based perception solution for autonomous vehicles. In *2011 11th International Conference on Intelligent Systems Design and Applications*, pages 790–795. IEEE, 2011. 3

[11] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *IEEE transactions on pattern analysis and machine intelligence*, 40(3):611–625, 2017. 3

[12] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European conference on computer vision*, pages 834–849. Springer, 2014. 3

[13] Di Feng, Christian Haase-Schütz, Lars Rosenbaum, Heinz Hertlein, Claudius Glaeser, Fabian Timm, Werner Wiesbeck, and Klaus Dietmayer. Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges. *IEEE Transactions on Intelligent Transportation Systems*, 2020. 3

[14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014. 5

[15] Qishen Ha, Kohei Watanabe, Takumi Karasawa, Yoshitaka Ushiku, and Tatsuya Harada. Mfnet: Towards real-time semantic segmentation for autonomous vehicles with multi-spectral scenes. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5108–5115. IEEE, 2017. 3

[16] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017. 3

[17] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The german traffic sign detection benchmark. In *The 2013 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2013. 2, 3

[18] ZuWhan Kim. Robust lane detection and tracking in challenging scenarios. *IEEE Transactions on Intelligent Transportation Systems*, 9(1):16–26, 2008. 2

[19] Karl Kluge and Sridhar Lakshmanan. A deformable-template approach to lane detection. In *Proceedings of the Intelligent Vehicles' 95. Symposium*, pages 54–59. IEEE, 1995. 2

[20] Andrew HS Lai and Nelson HC Yung. Lane detection by orientation and length discrimination. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 30(4):539–548, 2000. 2

[21] Mathias Lechner, Ramin Hasani, Alexander Amini, Thomas A Henzinger, Daniela Rus, and Radu Grosu. Neural circuit policies enabling auditable autonomy. *Nature Machine Intelligence*, 2(10):642–652, 2020. 3

[22] Stefan Milz, Georg Arbeiter, Christian Witt, Bassam Abdallah, and Senthil Yogamani. Visual slam for automated driving: Exploring the applications of deep learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 247–257, 2018. 3

[23] Michael Montemerlo, Jan Becker, Suhrid Bhat, Hendrik Dahlkamp, Dmitri Dolgov, Scott Ettinger, Dirk Haehnel, Tim Hilden, Gabe Hoffmann, Burkhard Huhnke, et al. Junior: The stanford entry in the urban challenge. *Journal of field Robotics*, 25(9):569–597, 2008. 2

[24] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. Dtam: Dense tracking and mapping in real-time. In *2011 international conference on computer vision*, pages 2320–2327. IEEE, 2011. 3

[25] Nils J Nilsson. Shakey the robot. Technical report, SRI INTERNATIONAL MENLO PARK CA, 1984. 2

[26] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 1, pages I–I. Ieee, 2004. 3, 5

[27] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55, 2016. 1

[28] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. Scnn: An accelerator for compressed-sparse convolutional neural networks. *ACM SIGARCH Computer Architecture News*, 45(2):27–40, 2017. 2, 3, 4

[29] Ralph H Rasshofer and Klaus Gresser. Automotive radar and lidar systems for next generation driver assistance functions. *Advances in Radio Science*, 3, 2005. 3

[30] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1137–1149, 2016. 5

[31] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. Visual slam algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 9(1):16, 2017. 3

[32] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, et al. Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, 23(9):661–692, 2006. 2

[33] Marin Toromanoff, Emilie Wirbel, and Fabien Moutarde. End-to-end model-free reinforcement learning for urban driving using implicit affordances. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7153–7162, 2020. 3

[34] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008. 2

[35] Yue Wang, Dinggang Shen, and Eam Khwang Teoh. Lane detection using spline model. *Pattern Recognition Letters*, 21(8):677–689, 2000. 2

[36] Yi Yang, Hengliang Luo, Huarong Xu, and Fuchao Wu. Towards real-time traffic sign detection and classification.

IEEE Transactions on Intelligent transportation systems, 17(7):2022–2031, 2015. 2

[37] Khalid Yousif, Alireza Bab-Hadiashar, and Reza Hoseinnezhad. An overview to visual odometry and visual slam: Applications to mobile robotics. *Intelligent Industrial Systems*, 1(4):289–311, 2015. 5

[38] Zhe Zhu, Dun Liang, Songhai Zhang, Xiaolei Huang, Baoli Li, and Shimin Hu. Traffic-sign detection and classification in the wild. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2110–2118, 2016. 3

[39] Qin Zou, Hanwen Jiang, Qiyu Dai, Yuanhao Yue, Long Chen, and Qian Wang. Robust lane detection from continuous driving scenes using deep neural networks. *IEEE transactions on vehicular technology*, 69(1):41–54, 2019. 2