Lane Detection, Steering Control, and Object Detection for Self-Driving Cars

Ajay Anand, Shubham Gupta, Hari Santhanam, Anirudh Subramanyam, John (JT) Thompson

Introduction

Developing and testing algorithms for autonomous driving systems is an extremely expensive and time consuming process. Advances in learning and vision allows us to use high fidelity visual and physical simulation environments for experimentation and implementation. Leveraging these advances provides us with an open sandbox environment with a flexibility to create and innovate new methods to advance the goal of autonomous driving systems. Self driving technology allows a vehicle to autonomously perceive the environment and instrument a response to help it drive. The task of drive scene understanding in the perception space is dominated by two methods: Camera based 2D, or LIDAR based 3D perception. An autonomous car should be able to detect traffic participants and drivable areas, particularly in urban settings where a wide variety of object appearances and occlusions may appear. In this project we are attempting to utilize classical vision techniques and deep learning principles on 2D camera data to simulate a self driving car.

Background Research

Before beginning the process of implementing our project, we carried out a survey of existing means and methods to carry out the different tasks that were a part of the autonomous driving project. The following are the main elements of such an undertaking:

- Simulation Environment
- Lane Detection Subsystem
- Object Detection Subsystem

Simulation Environment

A quick search for a virtual sandbox for Autonomous Driving system development yielded the following results:

- CARLA
- Microsoft AirSim[6]
- GTA V Mod

We began by ruling out GTA V mod as it requires having the game installed and a variety of mods also installed to get the environment running successfully. CARLA is an open source simulator for autonomous driving research. Our plan was to find an existing implementation of a neural network integrated with the CARLA framework and test the performance on our local system. However, we encountered a host of issues with CARLA version control, Neural Net architecture, and system hardware issues which ruled it out. Finally, we found that we had the most success with Microsoft's AirSim, which provided us with a stable urban environment with easy integration through Python's API and a sizable documentation of methods and tools to get us started.

Microsoft AirSim

AirSim is an open-source, cross platform simulator that is built on Epic Games' Unreal Engine 4 as a platform for AI research. AirSim allows us to simulate the movement of drones and ground vehicles, such as cars and trucks. It also provides us a virtual sandbox to run experiments and carry out vision and learning tasks. For this project we used AirSim's City Environment, which is a large scale urban city map with generic features and assets. The Car Agent in AirSim gives us control over the car's state and is able to use basic inputs like throttle and steering. Additionally, inbuilt camera API's allows to capture real time images to be used as input for manipulations. The Camera API allows for a wide variety of camera types such as Depth, Segmentation, and Scene etc. The scope of this project is limited to the Scene camera which returns an RGB image. AirSim also allows user control on camera location relative to the Agent(Car). This project uses the default location as that provides us with a front-view of the road ahead.

Lane Detection Subsystem

The goal of the Lane Detection subsystem is to use the Camera input from the simulator, perform lane detection, and then project the predicted lanes onto the original image. This can be approached with traditional computer vision techniques or via the more state of the art deep neural networks. Deep neural networks for lane detection take RGB images as input and predict the location of lanes. Many of these architectures incorporate deep CNNs to extract lane marking features, which are then fit to find the presence of lanes [1] [8] [7]. However, not all architectures take into account weather differences and changing lighting conditions, such as shadows and visual bloom. Hence, incorporating a deep learning architecture that was trained on various roads, road markers, and differing weather/lighting conditions was attractive due to the city environment provided in the simulator.

We chose an architecture created by Zou, Jiang, Dai, Yue, Chen and Wang[9] called Robust Lane Detection to address these problems. This particular implementation uses several frames of continuous driving scene and a hybrid framework of a CNN and an RNN to outperform competing methods for lane detection, especially in difficult environments. Our implementation of this

model faced considerable challenges. The original model was trained on an extensive real world dataset while our experiments are carried out in a synthetic environment. Adding to this, our hardware capabilities were not sufficient to achieve good fps for such a heavy model. Hence any potential control scheme for lane keeping will have been severely compromised and set up for failure due to these two issues.

Traditional methods for lane detection and prediction are based on the use of edge detectors to detect the lane markings and Hough transforms to fit lines to the detected lanes. This method usually uses a front view of the vehicle as an input and thus the range of available information is limited. Lee and Dorj[3] describe a novel method of transforming the regular front view to an overhead view to access a wider range of information. This led us to choose a hybrid model where we captured an image of the road using a FPV camera on the vehicle, transformed it into top down (birdseye) view using homography, and then detected lanes using canny edge detection and fit them using Hough Transforms.

Object Detection Subsystem

Object detection refers to the task of recognizing and locating objects in a frame or images. Specifically object detection draws a bounding box around the object and predicts a class label for it. In terms of autonomous driving, object detection extends to the detection and localization of vehicle, pedestrians, animals, road signs, traffic lights etc and can be extended to tasks like object tracking. Most state of the art object detectors use a deep neural network as a backbone and detection networks to extract features from input images(or video)[4]. Pre-existing image object detectors usually can be divided into two categories: two-stage detectors like Faster RCNN[5] and one-stage detectors like YOLO[2]. Two-stage detectors have high localization and object recognition accuracy, while one-stage detectors have high inference speed. Due to the constraints we faced with hardware capability and interest in using a lightweight model, we decided to use YOLO for object detection.

Methodology

In this section we describe the lane detection and object detection pipeline we implemented. We first elaborate on the two different lane detection methods we use: one using traditional homography and canny edge detection, and another using a deep learning architecture called Robust Lane Detection. Then we describe the algorithm implemented to control the autonomous vehicle. Finally, we illustrate object detection and identification using a YOLO deep learning network.

Traditional Homography and Canny Edge

We first take an image from AirSim of the front view of the car and apply homography, like how we did in class. We select 4 points in the source image and find 4 points on the destination where we would like to map each of the points in the source. An example is shown in Fig 1, where we connect the 4 points in the source we are using for the mapping to the destination. Then we perform the homography using opencv2's getPerspectiveTransform function, and forward warp the source's points to the destination. The resulting transformation is shown in Fig 2.

Figure 1: Front camera view of the car with the 4 points for homography connected.

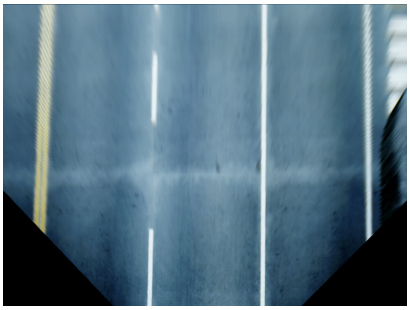

Figure 2: Homography View

To detect the lanes, we use canny edge detection. We set a low threshold of 100 and a high threshold of 200. Then, we empirically define a mask separately for the right and left lanes, resulting in two images displaying the canny edge output of just the left lane and one with just the right lane. The mask is used to ensure that edges other than the road lines are not outputted. Then we use Hough transforms to fit the lines from canny edge and we overlay these lines on top of the original image. Specifically, we use HoughTransformsP, which is a probabilistic interpretation to Hough Transform, and returns the coordinates of the starting point and end point of a line. An example is shown in Fig 3.

Figure 3: The detected lane using canny edge and Hough transform.

Notice that if you look carefully, each lane actually has multiple lines fitted to it. In order to design a controller later on, we need to extract the average fitted line for both the left and right lanes, which will give us a more accurate picture of the car's true position between the lanes. We also set a gradient threshold in order to not have the car follow lines with large gradients that may happen at crosswalks or regions affected by shadows. We do this by determining the slope of the lines, and if the slope ever surpasses a threshold value, we have the car follow the line from the previous time step that was under this threshold. Instead of following the lines with large gradients, to keep the vehicle in lane during these phases, we used the previous successful lane iteration to guide the vehicle until it successfully perceives a new lane. This method worked extremely well along straight and slightly curving roads.

Robust Lane Detection

Since Hough transforms fit linear lines, we also tried a more robust network to deal with curves in roads. As a result, we use a deep learning network called Robust Lane Detection. The input to the network is 5 consecutive frames of a video, which are first fed into an Encoder CNN. The features from these images are extracted by the encoder and fed into an LSTM which helps analyze time-series data. The LSTM then learns features and makes lane predictions. The outputs of the LSTM are fed into a decoder CNN, resulting in binary masks that indicate the presence/absence of the lanes [9]. We use published code of this algorithm and extrapolate to our specific application. In AirSim, we change the data preprocessing mechanism to feed the raw data pixels straight into the network, rather than using a dataloader in typical deep learning frameworks. This helps reduce run-time and aids in real-time processing of the input. An example output of the lane detector is shown in Fig. 4. However due to GPU restrictions, we were unable to use this method in our implementation of lane keeping.

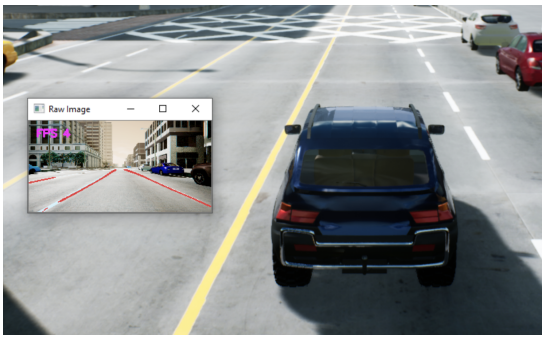

Figure 4: The robust lane detection output in AirSim is shown in the small window.

Steering Control and Lane Keeping

This section describes a simple implementation of proportional steering control for Lane Keeping. The objective of the control algorithm was to assist the Agent(Car) in maintaining its bearing and keep to the lane we define using homography and canny edge detection. The goal of the controller is to keep the heading of the vehicle in line with center of the lane. The lane center is determined from the output of the lane detection as the midpoint of the two lanes, and the center of the image is the default heading of the Agent. The heading error is calculated as the difference between the lane center and the vehicle heading. A proportional gain, K_p, then scales that error and is used directly as corrective steering input.

```
car_controls.steering = Err_heading * K_p
```

Object Detection Using YOLO

Figure 5: The detection of cars using YOLO.

Detecting objects in a driving scenario is very crucial for perceiving the car environment. In a real-time scenario, the object detector should have low latency and high accuracy. Two stage networks based on R-CNN are more accurate but take up a lot of computation. We decided to use a one-stage detector YOLO which is much faster. It frames the object detection problem as a regression problem and directly predicts the bounding boxes and class probabilities. The input image is divided into grids and each grid cell predicts B bounding boxes(x,y,w,h) and their confidence scores. Each grid cell also predicts C class probabilities, so the output is a S x S x (5*B + C) shaped tensor. Finally, post processing needs to be done to decode the bounding boxes. We use an open source implementation for the project and remove the dataloader to feed the image directly to the network. An example image is shown in Fig. 5.

Results & Analysis

The following section describes the performance of the methods and our findings related to the general tasks at hand.

Our first implementation of lane detection, using the deep learning method called Robust Lane Detection, had both strengths and drawbacks. The network was very accurate in detecting the lane markers in both sunny and dark regions, while also performing well on curvy roads. Also, it handles the degradation of lane markers well and the noise that occurs in an image. However, the method showed low inference speed, which coupled with the complicated architecture, led to poor performance on our hardware. Since we are sampling 5 frames at a time to feed into an encoder and then an LSTM, there was a large delay in between updates of the detected lane markers. As a result, we could not use this to write a controller for autonomous driving.

The other lane detection method we implemented was using Homography with Canny Edge/Hough transforms. Homography allowed us to gain a better vantage point of the scene and the lanes we were detecting. This coupled with Canny Edge and Hough Transforms yielded very good results in shadow regions with linear and slightly curved roads. Most likely this is due to the curved roads still being able to be approximated with linear lines as long as the speed of the car was not too high. However, a limitation of this approach was that it suffered in regions of greater lighting conditions and markings on the road. We attempted to handle this by using a gradient threshold, but this was not satisfactory in regions where there were long periods of time between updates of the lane location. For example, this could occur in long instances between consecutive shadow regions. Therefore, this method worked extremely well in certain instances, but suffered in others.

The control scheme presented in this is a simplified implementation of proportional control. For the use cases in this project, direct steering input performed exceptionally well at keeping the vehicle in the lane. The limitations of the method lie in taking sharp turns and low information zones where the lanes are not detected for a period of time.

YOLO based object detection was able to correctly identify and classify the most frequent classes of cars and trucks accurately and with good speed. The inference speed of the network was high and rendered good fps. This made it viable for use with manual control of the vehicle as displayed in the project demo. However, a drawback of YOLO is the high localization error. We found YOLO has trouble localizing smaller objects, like road signs and traffic lights, which leads to very irregular bounding boxes for these classes.

Conclusion

In this project, we successfully developed an autonomous driving system in the open-source AirSim environment. We achieved this by first developing a lane detection algorithm using homography and canny edge detection. We utilize the front camera on the car to select four points on the road in front of us and conduct homography to generate a birds-eye view of the road in front of the car. We then implement canny edge on the birds-eye view image of the road to detect the lane in front of the car. With the detected lane, we then implement a simple proportional controller to keep the vehicle in the center of the lane. Finally, we utilize the YOLO network to detect cars, trucks, and traffic lights in the AirSim environment captured through the car's frontal camera. With more time and processing capability, this project can be expanded upon to have more robust lane detection using neural networks. The object detection software can be greatly expanded upon to classify nearly any object that one would expect to see while driving a car in numerous environments. With more comprehensive object detection software, the control architecture can be greatly expanded upon. Some examples of future control implementations would be controlling speed based on speed limit sign detection, stopping at traffic lights and stop signs, turning at intersections, and even reacting to other cars in the environment. This project builds a substantial

foundation to achieve all of these tasks and proves that good performance for these tasks can be achieved using traditional computer vision techniques in a novel way.

Bibliography

[1] Ping-Rong Chen, Shao-Yuan Lo, Hsueh-Ming Hang, Sheng-Wei Chan, and Jing-Jhih Lin. Efficient road lane marking detection with deep learning. In *2018 IEEE 23rd International Conference on Digital Signal Processing (DSP)*, pages 1–5. IEEE, 2018.

[2] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun. Monocular 3d object detection for autonomous driving. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2147–2156, 2016.

[3] Byambaa Dorj and Deok Jin Lee. A precise lane detection algorithm based on top view image transformation and least-square approaches. *Journal of Sensors*, 2016(1), 2016.

[4] Licheng Jiao, Fan Zhang, Fang Liu, Shuyuan Yang, Lingling Li, Zhixi Feng, and Rong Qu. A survey of deep learning-based object detection. *CoRR*, abs/1907.09408, 2019.

[5] S. Ren, K. He, R. Girshick, and J. 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, 2017.

[6] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017.

[7] Hoo-Chang Shin, Holger R Roth, Mingchen Gao, Le Lu, Ziyue Xu, Isabella Nogues, Jianhua Yao, Daniel Mollura, and Ronald M Summers. Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning. *IEEE transactions on medical imaging*, 35(5):1285–1298, 2016.

[8] Yang Yang Ye, Xiao Li Hao, and Hou Jin Chen. Lane detection method based on lane structural analysis and cnns. *IET Intelligent Transport Systems*, 12(6):513–520, 2018.

[9] Q. Zou, H. Jiang, Q. Dai, Y. Yue, L. Chen, and Q. Wang. Robust lane detection from continuous driving scenes using deep neural networks. *IEEE Transactions on Vehicular Technology*, 69(1):41–54, 2020.