Aidan Jenkins
March 17, 2017
ME499

Autonomous Vehicle 499: Winter 2017
Advisor: Nick Marchuk

Introduction

This second quarter of work on the autonomous vehicle project was completed with Ryan Yu and we sought to expand on lessons learned from Fall 2016 and investigate more popular and sophisticated technologies. In this stage of the project computer vision and machine learning technologies were explored. The goal of the project was to have the car navigate the main loop on the second floor of the Technological Institute at Northwestern University (Tech). This was selected because of its proximity to the Mechatronics Lab in the Ford building, its large hallways with distinct tile patterns, and absence of fire doors. A second chassis was acquired so work could be performed without dismantling the car from the Fall. An additional goal of this quarter's project was to keep hardware costs down. Machine learning and computer vision algorithms are computationally expensive, however we wanted to maintain accessibility for this project, so popular hobby components were used. This report will document the steps taken this quarter, the reasons why, and offer suggestions for future efforts. Ryan and I wished to have a demonstration at the 2017 Chicago Maker Faire in April, however that event was cancelled in the second week of March.

Initial Steps

The first steps taken in this quarter of the project were to apply lessons learned from Fall 2016. Ryan and I spent the first week planning and getting a better understanding of project and what we thought it would take to complete it while being mindful of how it would fit in our schedules. The focus was on getting as much set up as possible the first half of the quarter so we could spend the rest tackling the unknowns, namely the neural network. We decided to make sure we found a location indoors to test in. Fall in Chicago was not particularly conducive to outdoor testing, particularly at the end of the quarter in December, Winter would not be any better. That is why we selected the second floor of Tech. It was safe from the elements and had regular and easily identifiable features on the floor, something we thought would be beneficial for line following and edge detection. The hallways are about 8 feet wide and form a square with over 100 feet per side. A photo of one of the hallways is shown in Figure 1. Moreover, after 7 pm the hallways were largely unoccupied so we would have ample time to test.

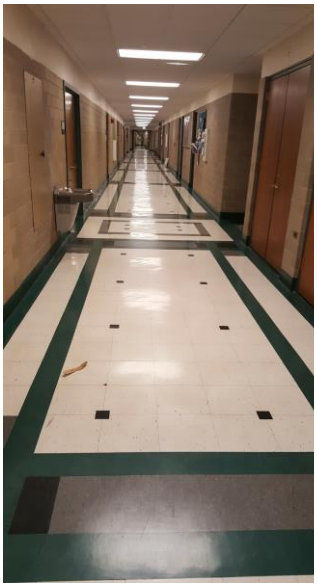

Figure 1 Proposed hallway for testing the car. Note linear arrangement of tiles on the floor that can be used for edge detection

Hardware selection was completed in parallel to planning, because scope and location was closely related to the quality of hardware needed. Because of the cost goals and familiarity with R/C

hobby components from Fall 2016, so similar parts were used. This time a 1:10 scale buggy with 4-wheel drive was selected. Its increased ground clearance over a 1:10 “touring car” made it better suited to diving outside, or in inclement weather if we got to a point where we were testing outside. On the electronics side, hobby components, a 2.4 GHz Tx/Rx combo, brushless motor, electronic speed control (ESC), and servo were used. Similar to the Fall 2016 phase, these components were interfaced with a Teensy 3.6 microcontroller. A Raspberry Pi 3 B was selected to handle the computer vision and neural net calculations on board the car. This was the most up to date model of the Raspberry Pi featuring a 1.2GHz quad core processor, 1GB ram, and built-in wireless communication, and a dedicated graphics chip to accelerate graphics performance. This computer was selected, as opposed to a more purpose-built platform like the NVIDIA Jetson board, because of its widespread adoption, cost vs. performance, and open source support. Additionally, if the Pi had performance problems, it could stream images to a laptop or desktop.

Setup

To avoid late stage technology setbacks, we sought to set up our hardware to be tolerant to changes—we built in contingencies. One of the first orders of business was flashing a Linksys router with the open source dd-wrt firmware. This unlocks a lot of router features and would give us more options if we needed to use a server-client relationship to control the car.

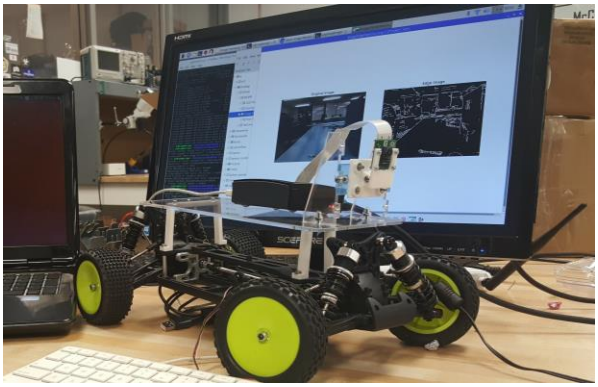

Figure 2 Buggy with image processing on display

The next objective was to install OpenCV, an open source image processing package. I attempted to install this in Windows 10, but this proved incredibly challenging. After two weeks of no progress, I decided I needed to move on. I created a 40GB partition on my hard drive, installed Ubuntu 16.04, and quickly installed the necessary software from the command line. Although Ubuntu can run with a lot less disk space, datasets for neural networks have gigabytes of data—it made more sense to have one larger partition than redo it later. To leverage my laptop’s graphics card for training the neural net later, I needed to install an NVIDIA driver. Initially I used the sudo apt-get command, this resulted in my computer booting, but not allowing me to log in. The latest NVIDIA driver for Linux operating systems should be downloaded from their website as a .tar file and installed.

Installing the Linux Raspbian OS on the Raspberry Pi was quickly accomplished using the New Out Of the Box (NOOBs) software from the Raspberry Pi Foundation. With Raspbian installed, downloading and installing OpenCV was trivial. For long term support and project stability Ryan installed Python virtual environments in Raspbian. This allowed us to run code using multiple versions of Python. This was important because although the newest version is Python 3.6, version 2.7 is still extremely popular and a lot of code that works on one will not work on the other. The baseline setup for hardware and software took approximately a week and a half longer than expected.

Progress

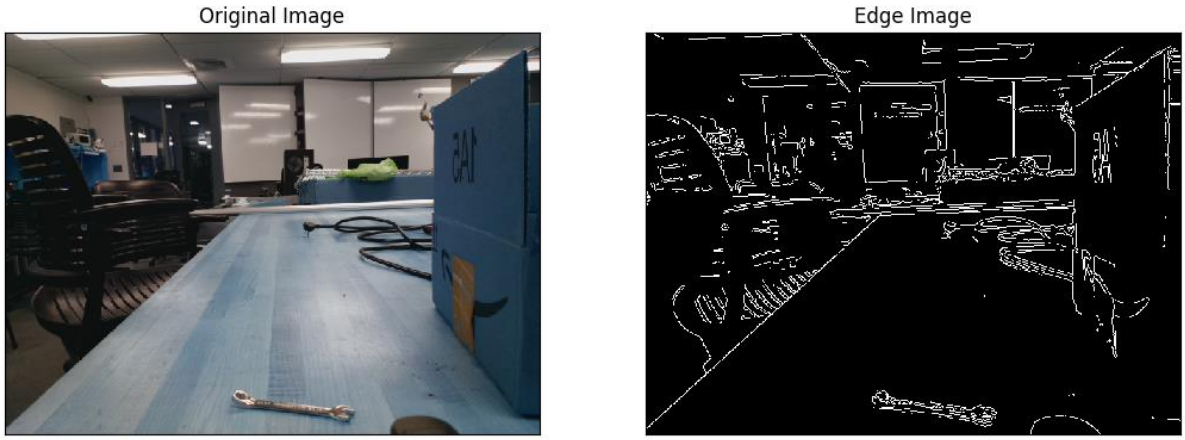

Figure 3 Original image captured from the Pi Camera mounted on the buggy and the Canny edge processed image

Once the software was installed, we immediately started testing OpenCV, the Pi Camera, and simply scripts in Python. We practiced operations like resizing and rotating images. To verify the install of OpenCV and better understand image processing techniques we did a "hello world" like activity with Haar cascades for face recognition. Additionally, because line following was of interest, we implemented a Canny edge detection algorithm to extract edge data from captured images. An example of an original image and an image with edge detection overlaid is shown in Figure 3. Preprocessing images is important for this project because it will greatly simplify the inputs to the neural network and lower the computation load. Eliminating "noise," unneeded features, from the pictures before they are converted to numpy arrays allows the neural network to focus on classifying fewer elements from the images.

In order to get practice with neural networks, Ryan worked on a basic neural net that classified images as either containing a cat or dog. There are many ways to approach neural network design and it is difficult to understand what the exact outcome will be for the network design. This is one of the reasons why machine learning and neural networks are such popular fields, there is much more to be understood. We wanted our network to be supervised—take inputs classified by our training data code as an image paired with a steering angle and known as "good". For a simple neural net implementation, we wanted to have an output layer with only a few nodes. In other words, for any given input, we wanted the neural net to return right, left, straight, or stop. Research into this type of implementation, however, suggested that it may not be sufficient. Information in lectures from MIT's class 6.S094: Deep Learning for Self-Driving Cars and Udacity's self-driving car project suggested we would need a Convolutional Neural Network (CNN), or a Long Term-Short Term neural network to incorporate some history into the training of the model. As the amount of code and number of devices grew; four operating systems spread across three laptops and the two Raspberry Pi's, we cloned the Pi SD cards for backup and created a GitHub repository. This way data losses would not hold us back and keep code consistent across devices. Furthermore, it allows this project to document itself, should anyone else like to work on something similar.

Conclusion

In the Fall portion of the project some parts of the code were hacked together in the last week of the quarter to make the car work. When problems arose, the code proved nearly impossible to debug. As the quarter drew to a close, we rapidly realized creating and implementing the neural network was more nuanced than we initially expected. Other courses had final projects that were taking more time away from the car project, so we focused on understanding neural networks better and how we are going to implement a more robust design in the spring instead of cobbling together something of poor quality.

The major remaining parts of the project are the data recording and classification code, building the neural network, and training the network. If the Pi cannot handle real time calculations it needs to be configured as a client that streams images to the laptop for classification and the laptop will return a steering value. The data recording code is straight forward in concept. When the car is driven in manual mode it will save images and the corresponding steering angle. When the neural network is trained, it will learn to identify the correct steering angle for what the camera is seeing. The Pi should be able to handle this using serial communication with the Teensy and storing data on its SD card, or an external USB drive. Udacity, for example, is using ROS, an open source platform for robotics, to manage the training data. This needs to be investigated as well as using a machine learning package. TensorFlow is very popular and will be much easier to use to create a CNN or a neural net with memory. Because of the difficulty of programming the more complicated neural nets in Python and the industry acceptance of TensorFlow, we strongly recommend considering it for the neural net creation. The large amount of data generated also poses a problem because of the amount of time needed to train the network. We believe the last component needed to make this project robust is creating a USB bootable version of Linux that can be run on a computer with a powerful graphics card that can accelerate training time from days to hours.

The goals originally defined for this phase of the project were ambitious and although they were not all completed, the project is better understood and well framed for success in the future. At this point, Ryan and I are very comfortable with finishing the remaining code needed to get our prototype working. Additionally, we will have more time to devote to the project. Ryan has finished taking classes and is doing some travelling. I am only enrolled in one class and working in the mechatronics lab. It would have been very satisfying to finish the project this quarter, however we are well set up for next quarter and are looking forward to completing the project the right way.