Siyuan Yu
ME499
Nick Marchuk

Building a Computer Vision Based Autonomous Vehicle

I. Abstract

This quarter, I worked with Aidan Jenkins to build a vehicle capable of computer vision based autonomous navigation. The main goals of this project were to learn how to use OpenCV, neural networks, the raspberry pi 3, and the pi camera in order to build an RC vehicle that could drive itself. Through the mistakes that were made and knowledge that was gained from the previous quarter, we managed to plan out our goals more realistically and meet the deadlines that we set out to achieve this quarter. Overall, we successfully downloaded and implemented OpenCV for image processing, began developing a neural network for computer vision learning, and built a custom RC car platform that can interface with the Raspberry Pi 3 and hold all of our electronics.

II. Goals and Planning

The main goal of this quarter's project was to develop an RC vehicle capable of navigating through the hallways of tech autonomously using computer vision algorithms and a camera. From last quarter's project, we learned that semi-autonomous navigation through GPS and IMU sensors was imprecise and not very robust. In addition, GPS navigation requires outdoor operation and does not work indoors. A computer vision approach is a much more elegant solution to this problem and has been successfully tested in projects done by students and by major companies such as Tesla and Google (see Research Appendix and Figure 1).

Figure 1: Examples of Computer Vision Cars

This computer vision approach involves combining image processing and machine learning techniques to train the vehicle to drive the way a human would by using primarily camera information. Although real autonomous vehicles rely on more than just cameras (sensor fusion from other sensors) and are based on sophisticated machine learning algorithms such as convolutional neural networks, we decided it would be

interesting to see how effective a cheap Raspberry Pi based RC car can be using a standard feedforward neural network with back propagation.

After the poor scheduling of last quarter's RC car project, one of the main focuses of this quarter was to develop a reasonable schedule and follow it as closely as possible (Figure 2). The intended goal for this quarter was to explore and develop all the hardware and software components needed to make an autonomous computer vision vehicle possible. This included downloading and learning OpenCV for image processing, learning how to use the Raspberry Pi 3 and the Pi camera, setting up our RC car to take Raspberry Pi controls, creating a neural network framework, and fabricating custom mounts for our electronics. Some of our reach goals included implementing all of our code on the Pi, training our vehicle, and testing the vehicle in tech. These goals were not met this quarter, but we have made plans to achieve these goals next quarter.

Task	Time
Get hardware	WEEK 1-2
Get OpenCV working on computer	
Test image processing w/ test images	
Turn image into binary arrays	
Setup local network on computer	WEEK 1-3
Setup pi (OS, applications, connect to wifi)	
SSH to pi, stream video from pi	
Fabricate mounts for Pi and sensing hardware for car	WEEK 4
Write neural net code	WEEK 4-end
Get interface between servos/esc for logging	
Automate neural net training with data logging	
Train and tune the neural net	
Reach Goals	
Get automation code working on computer w/ pi video streaming	
Get automation code working on pi	
obstacle avoidance	
Learn ROS, create a package	
Try to implement SLAM	
Attend Maker Fair	

Figure 2: Project Planning Schedule

III. Hardware

i) RC Car

For the RC car, we decided to stick with the 4-wheel drive 1:10 scale buggy that was used in last quarter's project. This car has great performance and adequate ground clearance for both outdoors and indoors use. In addition, the buggy has good mounting locations for our custom electronics platform. The buggy comes equipped with a 2.4 GHz TX/RX combo, an ESC, a brushless motor for throttle, and a servo for steering. Using the foundation we developed from last quarter's project, a Teensy 3.6 is used to interface with the car's ESC. The Teensy intercepts control inputs from either the Raspberry Pi or the RC controller and turns them into PWM inputs for the ESC (see Figure 3).

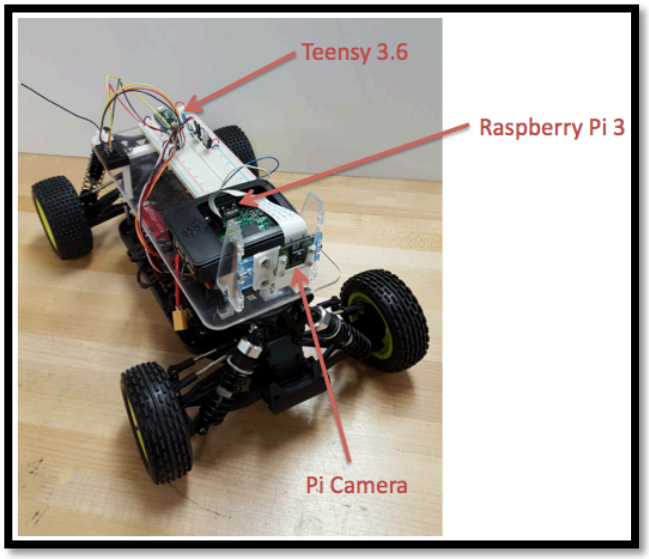

Figure 3: RC Car Setup

ii) Onboard Computer

We chose to use the new Raspberry Pi 3 B as our primary on board computer. We chose the Pi for its cheap price, compact size, dedicated graphics chip, and compatibility with a good selection of small but powerful camera hardware. In addition, there is lots of online support for Raspberry Pi projects (and specifically computer vision projects), which turned out to be extremely useful for us since we were first time users. Overall, the Pi directly interfaces with the camera and handles all of the image-processing and neural network computations. The Pi communicates with the Teensy using serial communication protocols and the Teensy is in charge of all the RC car control algorithms. All of this can technically be done on just the Raspberry Pi 3 alone, but due to time constraints and our previous work, we decided it would be easier for the Pi to output control commands and let the Teensy deal with the control signals.

iii) Electronics Platform

In order to hold all of these electronics, we lasercut a platform out of acrylic and 3D printed plastic supports that bolted onto the frame of the car. We also 3D printed an adjustable camera mount for the Pi camera at the front of the car. Since the camera is the only sensor that we have on the vehicle, it is extremely important that it is well mounted. It is also important that its height and angle are adjustable for distance calculations and field of view purposes.

iv) Router

In case the Raspberry Pi was not powerful enough to do the neural network computations onboard, we decided to flash a Linksys router and create a server-client network to facilitate streaming from the Pi to our laptops. This backup allows us to stream images from the Pi and perform the necessary computations on our laptops on the

go. This type of streaming setup introduces delay into the system, but is a good backup to have in case all the computations cannot be performed onboard. This networks is also helpful for debugging and training purposes.

IV. Software

i) OpenCV

One of the most important goals of this project was to learn how to download, setup, and use the OpenCV library. OpenCV is an open source image-processing library that provides building blocks for computer vision projects such as image filters, machine learning dataset structures, and other useful image manipulation functions. In order to develop and debug more easily, we decided to download OpenCV on the Pi and our laptops. We quickly found that downloading the library on Windows 10 was extremely difficult and that it was much easier to setup on Linux and Ubuntu. We found that PyImageSearch.com has a great series of tutorials for how to download OpenCV on the Pi as well as computers that are running Linux and Ubuntu.

After downloading OpenCV, we decided to perform a few image processing tests. The first test was to check out some of the image filters. For computer vision learning algorithms, the Canny edge detector is an extremely popular filter and helps to simplify images and recover lines of interest. We processed a few images of hallways using only a few lines of code and the OpenCV library (see Github for code). The code ran extremely quickly thanks to the efficiency of the OpenCV functions and the results showed that it could be a useful filter for our purposes (Figure 4). The utility and ease of implementation of OpenCV filters makes it possible to simplify and distill images to their most important features before feeding them into a neural network. This makes it easier and more likely for the neural network to identify and learn patterns in the images. Because we are using a simple and conventional neural network, reducing the complexity of the input images is extremely important in increasing the likelihood of success.

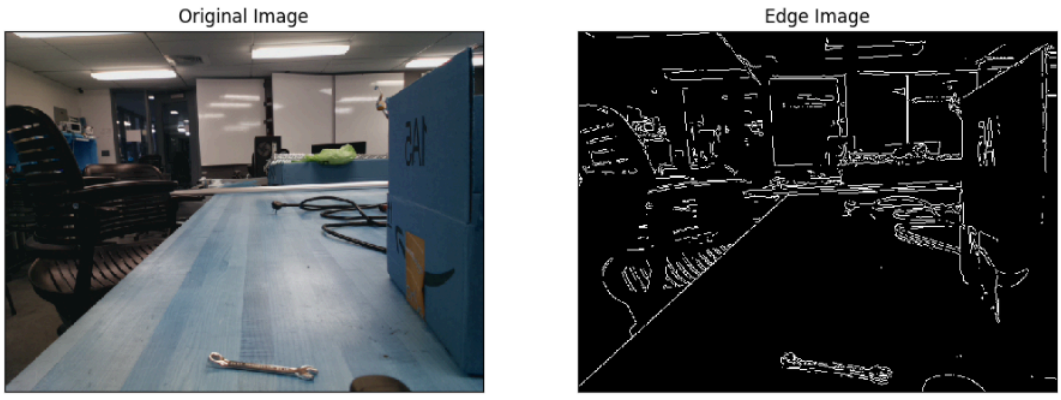

Figure 4: Canny Edge Detection Test

In addition, we decided to implement a simple computer vision algorithm using OpenCV as a proof of concept for our project. This was done by implementing a face detection algorithm taken from a PyImageSearch OpenCV tutorial (see Research Appendix). The algorithm uses Haar feature-based cascade classifiers to identify facial features and classify them as faces. Because of the built in features of OpenCV and the availability of training libraries online for facial detection, this code was very easy to implement and was successful in identifying faces (see Github and Figure 5). This proved that implementing a similar algorithm for identifying road features is possible using OpenCV features.

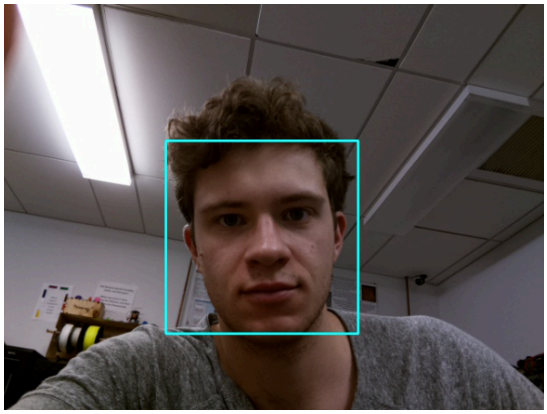

Figure 5: Haar Cascade Face Detection Test

ii) Python and Virtual Environments

For this project, we decided to code in Python since it is an easy to learn and widely used language. In addition, there is lots of online support for OpenCV and neural network projects in Python. In order to maintain project organization and stability, we decided to setup virtual environments for our Python 2.7 projects. These virtual environments ensure that everything that is needed to run the project files is contained in the environment and will not be disrupted by new versions of Python or other packages.

iii) Raspbian OS

On the Pi, we decided to run the very popular Raspbian OS. This system is a Linux based OS, which has a much better compatibility with the OpenCV library as mentioned earlier. The OS was setup using the New Out Of the Box (NOOBs) software provided by the Raspberry Pi foundation and was downloaded onto a 16GB memory card.

iv) Neural Network Algorithm

The neural network that we designed is a standard feedforward neural network with back-propagation. The neural network is made up of multiple layers of nodes called neurons. Each neuron is connected to all the neurons in the layer directly before and after the layer it is in. These connections all have weights and a function called an activation function. The first layer is called the input layer and has as many neurons as there are

desired inputs. The last layer is called the output layer and has as many neurons as there are desired outputs. All the layers in between are called hidden layers and that's where all the mathematical connections are made. As you train the neural network with specified input and output pairs, the weights of the neuron connections recursively change and adapt until an error function is minimized. This allows the neural network to intelligently identify complex patterns and relationships without the need of human assistance.

In our case, the input to our neural network is a flattened array of all the pixel values from an image of the road taken by the Pi camera. Our outputs for now are simply left, forward, or right steering angles (Figure 6). This methodology was taken from a very similar project done by David Singleton that was found online (see Research Appendix).

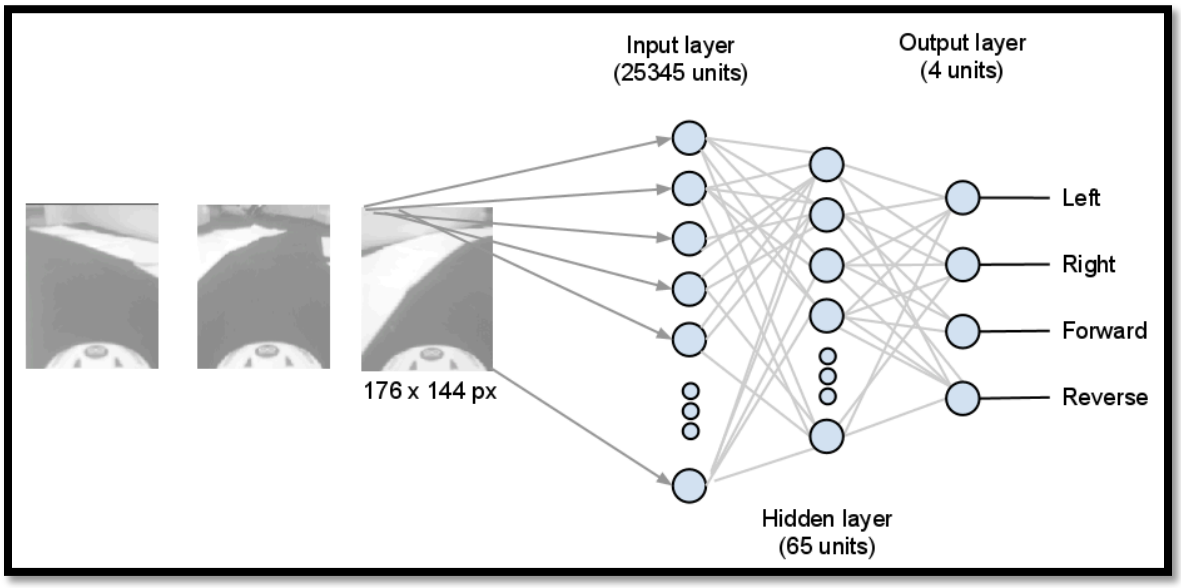

Figure 6: Neural Network Configuration

Once the neural network is setup, it needs to be trained by feeding the net a wide and diverse set of RC car driving images from the Tech hallways. Each image must also be labeled with a left, forward, or right steering angle (this can be expanded to however many discrete levels of steering angles we deem is necessary). A large number of training images are required to train the net to be accurate and this is determined through trial and error. For testing purposes, neural networks are typically trained on a fraction of the training data, and the remaining data is used for evaluation and verification.

During the training step, tuning must be performed as well. The tunable parameters of a standard neural network include the number of hidden layers, the number of nodes in each hidden layer, the type of activation function used, and the tolerance for the error function. In addition, the amount and quality of the training data will also affect the accuracy of the prediction results.

After the neural network is properly trained, the final weight variables are saved in a CSV file. The neural network can then be implemented onboard the RC car and perform real-time predictions based on these saved weight variables.

Currently, our training data collection method has not been designed and implemented yet. Therefore, the current iteration of the neural network is configured to take pictures of dogs and cats as inputs and output a prediction of which animal is in the picture. This was done because training datasets of cats and dogs are widely available online. Once this version of the code is debugged and working as intended, it will be easy to reconfigure the net to work for road images and output steering angles.

IV. Future Steps

As of now, the hardware for the autonomous car is almost completely ready to go. The Raspberry Pi still needs to be hooked up to power onboard the RC car.

In terms of software, there are a few things left to do. The most important thing now is to design a code that will automatically capture, save, and label road images while driving the RC car through Tech hallways. Manually labeling the massive amounts of training data that we need would be a huge undertaking, so automating this process is necessary.

For the neural network, the dog/cat classification test still needs to be debugged and tested. In addition, a function for saving the weight variables to a CSV after training still needs to be coded. After all of this is complete, the code needs to be reconfigured for our application and implemented on the Raspberry Pi.

Once all of these things are completed, the neural networks needs to be trained and evaluated using our RC car driving training data. Lots of time needs to be set aside for tuning the parameters and performance of the neural network.

V. Conclusion

Overall, this project was a great learning experience and crash course into computer vision and machine learning. In the process, I was introduced to the Raspberry Pi and became more comfortable coding in Python. In addition, I learned so much about the capabilities of OpenCV for image processing and found many useful resources on how to download the library successfully on the Pi, Mac, and Linux computers (see Research Appendix). If this neural network is successful in automating the navigation of the RC car around Tech, I think it would be interesting to design and implement a convolutional neural network for the future. This learning algorithm is more complex, more intelligent, and better suited for computer vision projects and would drastically increase our understanding of the current state of the technology.