Name: Christoph Hinniger

Email: christoph.hinniger@web.de

Writeup Capstone Project (individual)

A lot of the code is implemented by following the walkthrough videos by udacity. To implement the traffic light classification, I used the Object Detection API by tensorflow. For training I used images, that were extracted from the simulator with the save_image function inside the tl_detector file. I used labelImg to label all traffic lights in those images and used the faster_rcnn_inception_v2 model for transfer learning. Since I was not able to get the object detection API running with Tensorflow 1.3, I had to update the tensorflow version inside the udacity workspace. To get the program to run, I navigated to the workspace directory and deleted the "CarND-Capstone folder. Afterwards, I opened the terminal and calld the following:

```
git clone https://github.com/Christoph9402/CarND-Capstone-Project.git && cd CarND-Capstone-Project && pip3 install -r requirements.txt && pip install tensorflow==1.15.0 && pip install matplotlib && cd ros && chmod -R +x src && cd src && cd tl_detector && cd light_classification && PYTHONPATH=$PYTHONPATH:/home/workspace/CarND-Capstone-Project/ros/src/tl_detector/light_classification && export PYTHONPATH && PYTHONPATH=$PYTHONPATH:/home/workspace/CarND-Capstone-Project/ros/src/tl_detector/light_classification/object_detection && export PYTHONPATH && PYTHONPATH=$PYTHONPATH:/home/workspace/CarND-Capstone-Project/ros/src/tl_detector/light_classification/slim && export PYTHONPATH && cd ../ && cd ../ && cd ../  && catkin_make clean && catkin_make && source devel/setup.sh && roslaunch launch/styx.launch –screen
```

This automatically installs every necessary dependency. As you can see in the video "Capstone_final.mp4", the Udacity workspace is very slow. I had to limit the throttle to not go over 0.1 (twist_controller file line 48 and 49). This was necessary, because the car could barely follow the waypoints. In the submitted version I commented those lines, so the throttle is not limited. As you can see in the video, the car follows the waypoints and new waypoints are being published. The vehicle is also capable of detecting traffic lights, when they are closer than 300 waypoints. Once the vehicle is inside this distance, the detected light state is being published, as it is visible in the terminal output on the left side of the video. The detection works well and only occasionally fails to predict the right state. Using more training images could be useful here, but means much more work for labeling. As you can see, once a red traffic light was detected, the car slows down and stops at the line in front of the traffic light. The slow performance is especially visible, when the light changes to green and the car can move it takes a few seconds until the vehicle moves – nevertheless, the green light is detected correctly. Another example where the slow workspace limits the quality of the output can be seen in the video, where the traffic light switches from green to red, but the vehicle still processes the green traffic lights. The vehicle does in the end classify the traffic light as a red one, but that was too late to slow down. With more performance, the car should have stopped just as it did at the other red traffic light.

In conclusion, the vehicle moves smoothly along the published waypoints. It is also capable of detecting and classifying states of traffic lights. One a red light is detected, the vehicle stops at the line and waits until the state switches to green. Then it accelerates and follows the track. Workspace speed was a very limiting factor, but I am very satisfied with the result, since the car navigates around the track successfully and the specifications of the rubric was met.