**Self-driving Car Engineer Nanodegree Capstone Project: Program an Autonomous Vehicle**

Marek Kolodziej, Ph.D. Tim Limyl, M.S., Alejandro Terrazas, Ph.D.

The purpose of this capstone project is to combine the skills garnered by the students in Udacity Self-driving Car Engineering Nanodegree in order to program Udacity’s Carla self-driving car platform. The students combined skills in computer vision, vehicle control systems, and integration through the Robot Operating System (ROS; ).

**Project Aims**

The project’s aims were scaled down from a full self-driving car and, instead focused on a) staying in a single lane and b) stopping at red traffic lights. Goals such as changing lanes, planning

**Project Stages**

The project proceeded in two stages. First, basic elements of the system were developed to run Udacity’s simulator (Figure x). The simulator is a virtual reality 3D environment with three lanes of traffic and eight lights. There are no other cars in the environment. In order to pass the first test, the simulated vehicle must stop “safely”, that is smoothly, at each red light, decide to stop or continue through each yellow light, and continue through each green light at an appropriate speed.

Figure x: Simulator showing car at the starting gate..



Once the software system passes the first test on the simulator, it is transferred to Udacity’s Carla to run on a real-world—albeit greatly simplified—track. The traffic lights encountered on the test track are substantially different than the traffic lights encountered in the simulator. Moreover, conditions on the test track include glare and poor contrast among other challenges.

**Object Detection and Classification System**

In order to react correctly to the traffic lights, the software system must a) detection, identify the traffic light housing with a bounding box and b) classification, look within the bounding box to determine the state of the light (green, yellow, red). A single shot detector (SSD; reference) was used for this purpose.



The SSD used transfer learning in which a pretrained network is adapted for use in a specific project. The pretrained network selected was from the TensorFlow Zoo (www…). The virtual camera on the simulator produces 800 x 600 pixel (height by width) tri-color (blue, green red) images at a rate of x Hz. These images were captured to disk for use in training and testing the computer vision module. In order to address the differences between the simulator and the real-world test track images, a mixture of real and simulator images were used for training, validation, and testing of the computer vision algorithms. Early development of the computer vision system relied on simulation images only. Following success with the simulator testing, additional images from the Bosch Small Traffic Light Dataset were mixed in.

***Figure x: Traffic light images from Carla (left) and the Udacity simulator (right).*** *Significant differences can be seen between the two images. In the Carla real-world image, there is significant reflection from the dash upon the windshield that is not present in the simulator image. Moreover, the simulator traffic lights are always of the overhead variety and come in groups of three housings, while the Carla image contains a single housing mounted on a post. The sizes of the images are also different. The differences between the simulator and Carla images make it necessary to train the light detection and classification module using a mixture of real world and simulator images.*



Figure x, below, shows an example image with bounding boxes around detected objects.

***Figure x, Example image showing bounding boxes surrounding detected objects.*** *The image demonstrates correct detection of two traffic light housing and one incorrect detection of trees. The second stage of the detection and classification is to determine whether the lights are green, yellow, or red.. Moreover, it is necessary to eliminate the false positive (lower right corner) surrounding the trees. The false positive can be eliminated based on the shape of the bounding box because traffic lights are distinctively shaped as vertically oriented rectangles, while, the false positive is more square in shape. As shown in Figure x, above, the simulator does not contain overhead lights; therefore, looking in the upper part of the image would not work well for the Carla images.*



**Performance of the Traffic Light Detection and Classification System**

In order to characterize the performance of the detection and classification system, the following metrics were used: 1) precision, 2) recall, 3) false detection.

The object detection and classification software was written in Python 3.7 and TensorFlow version

**Obstacle Detection**

Data Sources

Bosch small traffic lights detection database

**Robot Operating System (ROS)**

ROS (Robot Operating System) provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more. ROS is licensed under an open source, BSD license.

The robot operating system is an open source software framework developed initially by Willow Garage (www) and is ideally suited for complex robot operations that often involve multiple systems running independently. For example, the multiple modules in Carla include: 1) a camera that produces raw images, 2) the object detection and classification system which consumes the raw images and produces the state of the light (i.e., green, yellow, red), 3) a decision-making system that decides whether to stop at a light or continue through it, and 4) the vehicle control system that is responsible for steering, accelerating and decelerating the car, just to name a few. Indeed, most robots and autonomous vehicles are may contain dozens of interacting modules. Writing a single program that integrates all of the modules in a self-driving car would be very challenging, to say the least.

ROS allows these systems to run independently while, at the same time, exchanging messages. The main mechanism for communicating between modules is through a “publisher-subscription” (commonly referred to as a “pub-sub” model. In certain cases, asynchronous communications—in which … -- are required. ROS supports both message passing protocols.

Another important benefit of using ROS is the large number of device drivers that are included with the distribution. These device drivers support many of the most popular sensors (cameras, LIDAR, RADAR, IMU) and actuators (motors and controls). The easy integration of the supported devices means that developers are able to focus more effort on integration.

ROS has several tools that aid in the development of an integrated system. The system developer can “peer into” the messages being exchanged and store these in a “bag” file, which can can be replayed to the system. Replaying past data ensures that conditions can be replicated during failure analysis. Moreover,

**Carla**

Udacity’s Carla contains subsystems for sensing, perception, planning and control. These subsystems are described below.

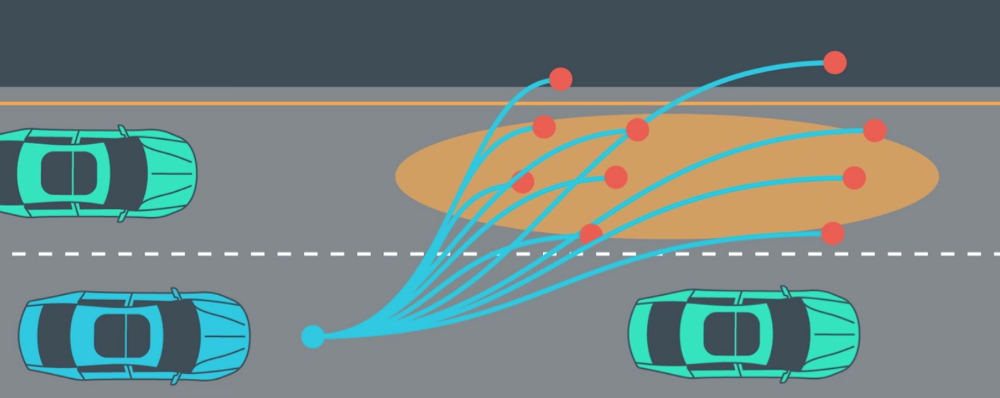
***Sensing*** The sensing is largely comprised of several cameras, an inertial measurement system (IMU), RADAR and LIDAR sensors. The camera images provided for this project are mounted behind the windshield. Not included in this project (but part of Carla) are a RADAR sensor and a LIDAR sensor both of which provide the distance to nearby (xx meters in distance) objects. The RADAR is forward facing and only senses the distance to objects in front of the car. The LIDAR system has a 360-degree view that provides distance to objects in all directions. The GPS provides an estimate of position in global coordinates with 1-2 meters resolution and an Inertial Measurement System (IMU) that estimates displacement of the car in the x, y coordinates along with the direction of movement.

None of the individual sensors are sufficient for localizing the car to the degree needed for safe driving. Therefore, sensor fusion (combining sensor data) is necessary. GPS coordinates for the lights and car location are provided in the simulator and on the test track.

***Perception.*** As in neural and cognitive science, perception differs from sensation in that the perception takes the raw input from the sensor and elaborates it into meaningful information. Meaningful information for a car include traffic light detection and classification (as included in this project), obstacle detection, traffic sign detection, detection of other cars, and, pedestrians. A great deal of the effort in the Capstone project involved perceiving the state of the traffic lights.

***Planning.*** Self-driving cars (and the drivers they will replace) always need to think ahead. The planning subsystem is built on the concept of waypoints—a series of coordinates that are updated as the car moves through the environment. For example, a lane change to the left might be comprised of 20-40 equidistant waypoints in the form of a smooth s shape over a 30 meter span. Note: there is no requirement that the waypoints are equidistant, extend 30 meter, nor that there are 20-40 waypoints that describe the trajectory. In dense traffic, the parameters may vary substantially from the parameters in sparse traffic. A real-world planning subsystem would be adaptive to the situation. Again, this is similar (in concept, if not in detail) to how neural planning systems work. Figure x shows examples of waypoints from a previous project in the Self-driving Car Engineering Nanodegree program.

***Figure x: Example Path Planning Trajectories for lane changing in heavy traffic.*** *In this example, a multitude of possible paths are considered; however, most of the path do not have the desired outcomes. The goal of an effective path planning module is to smoothly transition between cars (and other objects) while maximizing safety and comfort for the passenger. The path planner must consider the speed of the car ahead in the lane as well as the car to the side and generate the correct combination of acceleration and steering to avoid other cars and land in the gap. Example taken from path planning course materials from Udacity’s Self-driving Car Engineer Nanodegree*



***Control.*** Finally, after planning comes execution of the commands that control the car. As in human driving, there are a limited number of controls that, in combination, create a complex series of behaviors. Carla is equipped with a drive-by-wire controller to control a) acceleration, b) braking and c) steering. Carla uses a proportional-integral-derivative controller.

**The Testing Track**

After passing the simulator test, the testing moved to the real world using Carla.. Some minor modification to the code (setting flags) was necessary in order to run the code on Carla.

Figure x. Architecture of Carla, the Udacity Self-driving Car..

