Self Driving Car Nanodegree - Capstone Project: System Integration

Project objectives

The objective of this project is to implement a ROS-based core for an autonomous vehicle. The vehicle shall be able to complete a closed-circuit test-track, detecting the traffic lights and stopping whenever required. The code will be evaluated in a Unity simulator and a real-world Lincoln MKZ

Specifications

The car should:

- Smoothly follow waypoints in the simulator.
- Respect the target top speed set for the waypoints.
- Stop and restart depending on the state of the traffic lights.
- Publish throttle, steering, and brake commands.

We implemented using ROS 3 different modules for the autonomous vehicle:

- Perception: Traffic Light Detection Node
- Planning: Waypoint Updater Node
- Control: DBW Node

Project video

https://drive.google.com/file/d/1CplAmb5ED4I56STiP_9a3j5S6NzMbmpz/view?usp=sharing

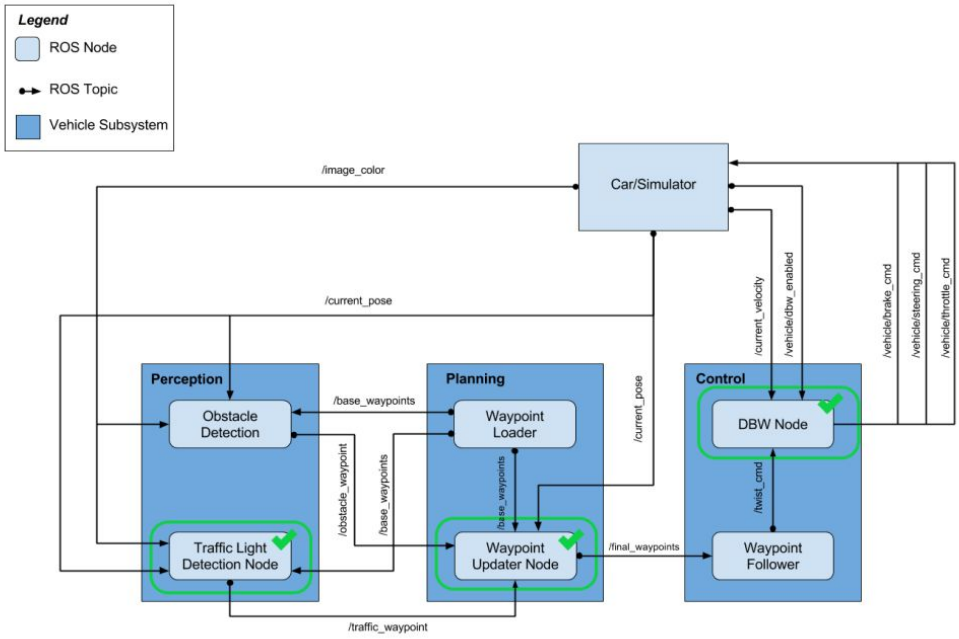
Team members

- Hugo Trinidad - TL : hugot305@aol.com
- Leandro Borgnino : leo.borgnino@gmail.com
- Mariam Mostafa : mariam.hamdy.mosta@gmail.com
- Stefano Castagnetta : stefanocastagnetta@gmail.com
- Tobias Zwiehoff : T.Zwiehoff@gmail.com

Overview

This repository contains the code for the final project of the Udacity Self-Driving Car Nanodegree: Programming a real self-driving car which can detect traffic lights and respond to it.

The code consists of a set of implemented ROS (Robot Operating System) Nodes which interact to maneuver CARLA, Udacitys self driving car platform, autonomously around a test track. In particular the project handles three different subsystems of the vehicle: *Perception*, *Planning* and *Control*:



System Architecture

Among the components showed in the picture above, the team focused on the implementation and integration of the following 3 nodes:

- Traffic Light Detection: Part of the *perception* sub-system, the node takes care of detecting traffic lights and classifying their state
- Waypoint Updater: Part of the *planning* sub-system, takes care of generating a trajectory (as a set of waypoints with their respective target velocities) taking into account the detected traffic lights in the environment
- Drive by Wire Controller: Part of the *control* sub-system, takes care of translating the Twist Messages generated by the waypoint follower into throttle, brake and steering values

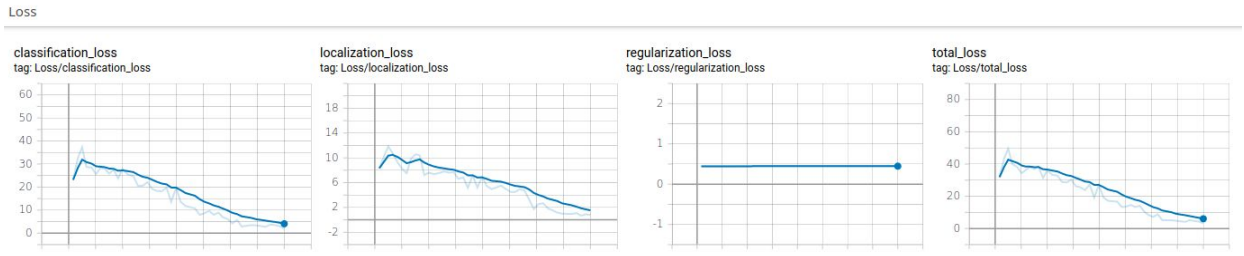
Perception

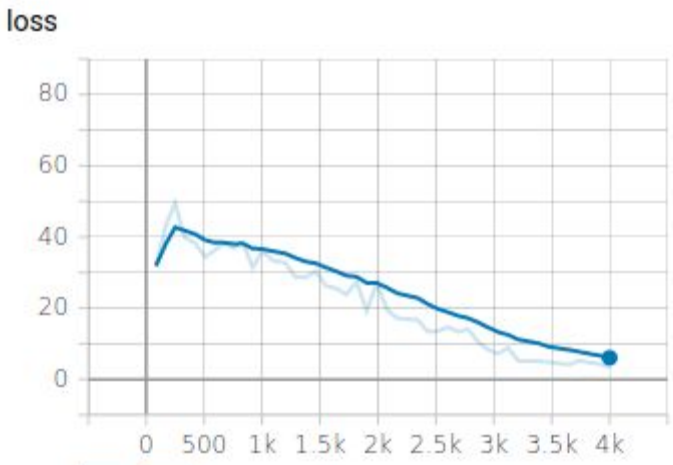
The sensing and perception subsystem's main objective is to collect data to aid in localization, navigation, spot detection, and obstacle avoidance. In this project the objective is to detect traffic lights, classify its state and take actions according to the result.

The detection and classification was done with a processing model based on SSD architecture (an end to end training and classification for object detection) using transfer learning to customize an existing model for our purpose. Using transfer learning allowed us to save time during the training process.

We used the SSD mobilenet v2 as the base model for transfer learning to our TF Object Detection API. A datasets was extracted from udacity ROSbags and labeled with LabelImg. The model generation scripts can be found at https://github.com/leoborgnino/traffic-light-model-generation.

The training process was slow and no more than 4k steps were done during a 12hs training period; in the images below we can see that the loss was still decreasing when the training finished so the model can still be improved.





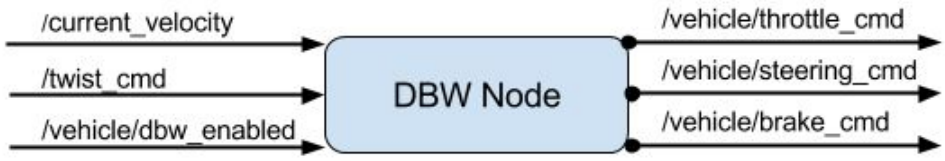
After training, several tests were made with random images from the rosbag and then in the simulator with site mode. The images from this tests can be seen below:

Planning

In this subsystem there are 2 nodes: the waypoin_loader and the waypoint_updater. The first one was already given and it loads the static waypoint data and publishes to /base_waypoints. The waypoint_updater was developed by us and it subscribes to the /base_waypoint topic, the /obstacle_waypoint topic and the /current_pose topic. With this information, it computes velocities for the waypoints that are in front of the vehicle. The vehicle always tries to keep a speed close to the speed limit unless there is a red light in front of it. In that case it decelerates and stops before the stop line. Once the light turns green the car resumes motion.

Controls

In this subsystem, there is a DBW(drive-by-wire) node that is responsible for controlling the vehicle. The dbw node subscribes to /current_velocity, /twist_cmd, and /vehicle/dbw_enabled topics. This node will publish throttle, steering, and brake topics. The steering angle is calculated using a YawController while the throttle is calculated using PID. A low pass filter is used to filter out the velocity's high frequency noise.



DBW was tested using dbw_test.py, we tested the code against the bag recorded at https://s3-us-west-1.amazonaws.com/udacity-selfdrivingcar/files/reference.bag.zip
Three csv files are produced which we used to process how DBW node is performing on various commands. The files are found in https://github.com/leoborgnino/CarND-Capstone/tree/master/ros/src/twist_controller