Programming a Real Self-Driving Car

The Capstone Project

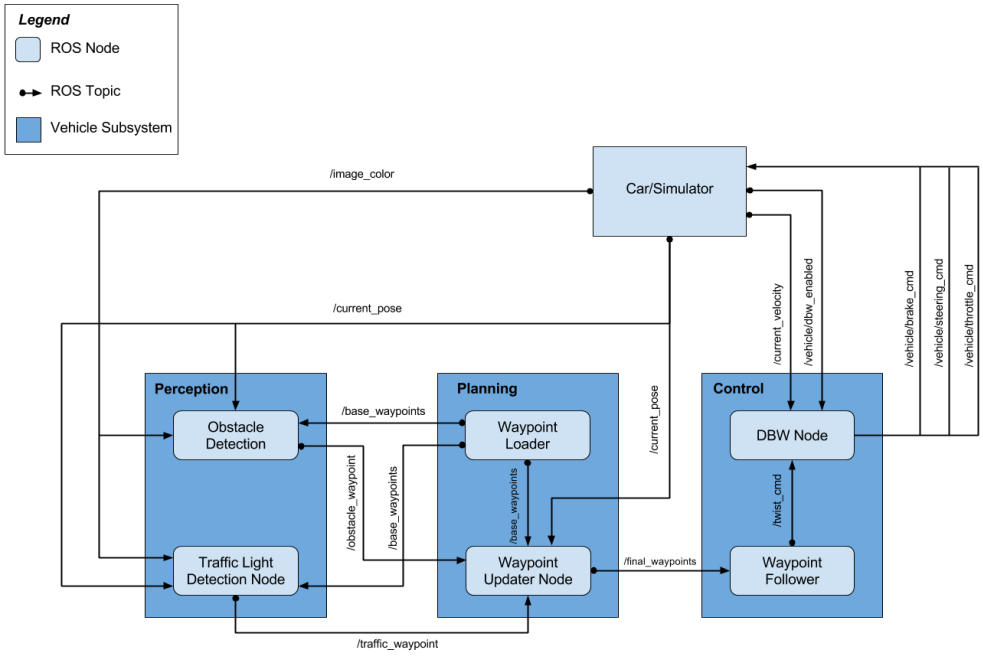
This is the project for the final project of the Udacity Self-Driving Car Nanodegree: Programming a Real Self-Driving Car. In this project, we will be writing ROS nodes to implement core functionality of the autonomous vehicle system, including traffic light detection, control, and waypoint following! We will test our code using a simulator, and when it's ready, our group can submit the project to be run on Carla.

Our Project Team - Experienced Drivers

Full Name	**E-mail**
Xiang Jiang (Team Lead)	jx.for.jiangxiang@gmail.com
Ahmed Khatib	ackhatib@gmail.com
William Wu	willywu2001@hotmail.com
Haribalan Raghupathy	haribalan.r@gmail.com
Ilkka Huopaniemi	ilkka.huopaniemi@gmail.com

Project Overview

The following is a system architecture diagram showing the ROS nodes and topics used in the project. You can refer to the diagram throughout the project as needed. The ROS nodes and topics shown in the diagram are described briefly in the **Code Structure** section below, and more detail is provided for each node in later of this paper.



Code Structure and Description

Below is a brief overview of the repo structure, along with descriptions of the ROS nodes. The code that you will need to modify for the project will be contained entirely within the (path_to_project_repo)/ros/src/directory. Within this directory, you will find the following ROS packages:

1. Perception Module
    A. Traffic Light Detection Node
2. Planning Module
    A. Waypoint Loader
    B. Waypoint Updater
3. Control Module
    A. DBW Node
    B. Waypoint Follower

1A. Traffic Light Detection Node

The Perception Module consists of a Traffic Light Detection Node. This node takes in data from the /image_color, /current_pose, and /base_waypoints topics and publishes the locations to stop for red traffic lights to the /traffic_waypoint topic.

1. File tl_detector.py

The traffic light detection node is implemented in program tl_detector.py. After initialization, this Node subscribes to the following topics: /current_pose, /base_waypoints, /vehicle/traffic_lights and /image_color (code lines 30 to 41).

Next, the function image_cb() (code lines 75 to101) identifies red lights in the incoming camera image and publishes the index of the waypoint closest to the red light's stop line to /traffic_waypoint. It publishes upcoming red lights at camera frequency. Each predicted state has to occur STATE_COUNT_THRESHOLD number of times (defined in line 14) until we start using it. Otherwise the previous stable state is used.

To achieve this, the helper function process_traffic_lights() is used (cade lines 141 to 177), which finds closest visible traffic light, if one exists, and determines its location and color. It returns the integer index of the waypoint closest to the upcoming stop line at a traffic light (-1 if none exists) and the color index of the detected traffic light if any (0: RED, 1: YELLOW, 2: GREEN, 4: UNKNOWN).

2. File tl_classifier.py

In code line 139 of file tl_detector.py, we also make use of the imported TLClassifier function ‘get_classification’ inside of the file tl_classifier.py. It classified the incoming images and returns the color index of the detected color or returns unknown if nothing is detected.

Since we know the locations of the traffic lights and the vehicle, we can get reduce the classification problem to transformation and detection problem. Color is easier to detect in HSV space. In our use case, red light is very important, and in HSV space red has two different ranges, since we want to be very sensitive to red light, we include both range in the mask. Further improvements can be made when dealing with unknown locations and complex data by applying NN solutions.

2A. Waypoint Loader (in Planning Module)

The Waypoint Loader Node is implemented in ./ros/src/waypoint_loader/waypoint_loader.py. There the node loads the static waypoint data and publishes to /base_waypoints.

2B. Waypoint Updater (in Planning Module)

The Waypoint Updater Node is implemented in ./ros/src/waypoint_updater/waypoint_updater.py. The purpose of this node is to update the target velocity property of each waypoint based on traffic light and obstacle detection data. This node will subscribe to the /base_waypoints, /current_pose, /obstacle_waypoint, and /traffic_waypointtopics, and publish a list of waypoints ahead of the car with target velocities to the /final_waypoints topic.

The node is initialized (line 65) and subscribe to /current_pose, /current_velocity, /base_waypoints,

/traffic_waypoints and /obstacle_waypoint topics (lines 70 to 74). Then in line 78, we define the topics we

want to publish: /final_waypointsfor the Control Module. Also, in line 28 to 36 we already defined and set important

parameters. LOOKAHEAD_WPS determines the number of waypoints ahead of the vehicle to be published for the Control Module.

From line 82 to 88, we initialize further relevant parameters. In lines 93 to 94 function pose_cb() returns the current position of the car in the environment when querying the /current_pose topic. Then in line 96 & 97 we call get_waypoint_velocity() and query the current velocity when accessing the subscribed topic /current_velocity. Furthermore, waypoints_cb() (code lines 99 to 110) loads the basic path to follow coming from the waypoint_loader Node only if it has not been done already, and traffic_cb() (code lines 112 to 113) returns the index of the waypoint to stop at if a red or yellow light is detected.

Next, from lines 115 to 204 we define some helper functions which we will not describe in detail.

3A. DBW Node (in Control Module)

The DBW Node is implemented in ./ros/src/twist_controller/dbw_node.py. The goal for this part of the project is to implement the drive-by-wire node which will subscribe to /twist_cmd and use various controllers to provide appropriate throttle, brake, and steering commands. These commands can then be published to the following topics:

/vehicle/throttle_cmd, /vehicle/brake_cmd, /vehicle/steering_cmd

1. **File dbw_node.py**

To mimic human driving, we pass a variable specifying driving mode of the car (acceleration/deceleration) and relax the precision of following suggested speed. If we are accelerating and the car speed increased a bit more than expected the controller will just release throttle and let the car cruising so it can slow down on its own (lines 77 to 96).

We then also mimic brake booster functionality in lines 119 to 123. When the desired speed is very low we will activate extra braking force. Moreover, if the car almost stopped we will apply full braking to keep the car from moving.

Finally, since a safety driver may take control of the car during testing we consider the DBW status in our implementation which can be found by subscribing to /vehicle/dbw_enabled.

2. **File twist_controller.py**

This file contains the Controller class. You can use this class to implement vehicle control. For example, the control method (lines 31 to 55) can take twist data as input and return throttle, brake, and steering values. Within this class, you can import and use the provided pid.py (line 18) and lowpass.py (lines 28 to 29) if needed for acceleration, and yaw_controller.py (line 22) for steering.

3B. Waypoint Follower (in Control Module)

A package containing code from Autoware which subscribes to /final_waypoints and publishes target vehicle linear and angular velocities in the form of twist commands to the /twist_cmd topic.

Compiling and Running the Code

The usage is as following

1. Clone our project repository in the Github

```
git clone https://github.com/William-Wu56/CarND-Capstone.git
```

2. Install python dependencies

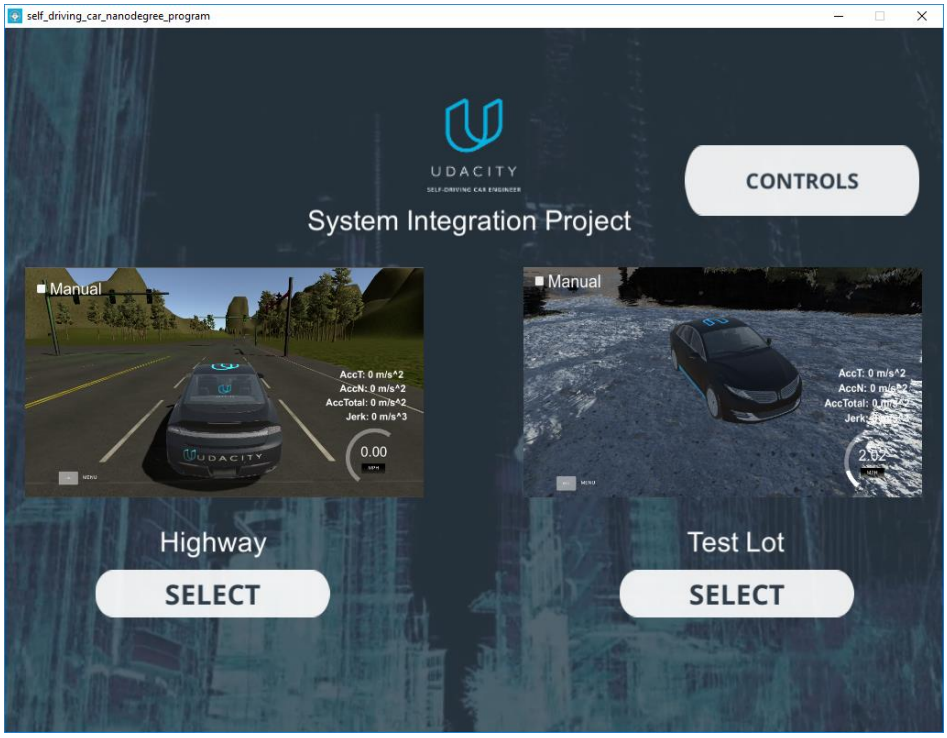
```
cd CarND-Capstone

pip install -r requirements.txt
```

3. Make and run styx

```
cd ros

catkin_make

source devel/setup.sh

roslaunch launch/styx.launch
```

Downloading the simulator for our host operating system and using this outside of the VM. We will be able to run project code within the VM while running the simulator natively in the host using port forwarding on port 4567.

Testing in simulator and Results



The picture above shows the interface of the simulator. We could test our project either on Highway, or in Test Lot.

Testing on the Highway

In the simulator, when clicking the SELECT button on the left (below the word ‘Highway’), we started the testing on the Highway.

Above are six snapshots for the car driving on the highway. The car was able to successfully complete track lap while meeting all the objectives.

- The 1st snapshot shows the car was going to start but detected the RED traffic light. So, the car almost stoped there (1.02 MPH).
- The 2nd snapshot shows the car detected (the RED traffic light turned into) the GREEN light. Then, the car increased the speed to pass traffic light at 15.10 MPH.
- The 3rd snapshot shows the car detected the RED traffic light. As the result, the car slowed down and closer to the traffic light at 3.37 MPH.
- The 4th snapshot shows the car detected the GREEN traffic light. Not changing the speed, the car passed traffic light at 23.99 MPH.
- The 5th and 6th snapshots show the car didn't find a traffic light. So, the car is with a speed around 25 MPH.

Testing in the Test Lot

In the simulator, when clicking the SELECT button on the right (below the word 'Test Lot'), we started the testing in the Test Lot.

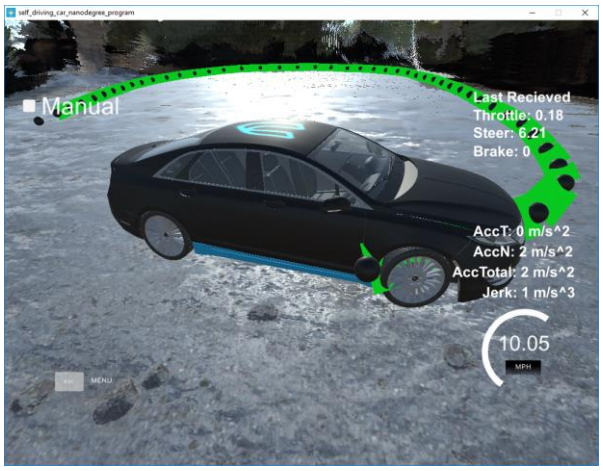
NOTE: The `./ros/src/waypoint_loader/launch/waypoint_loader.launch` file is set up to load the waypoints for the first track (Highway). To test using the second track (Test Lot), you will need to change

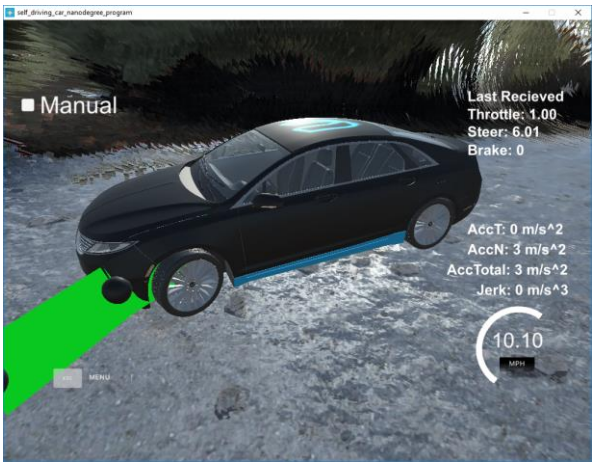
```
<param name="path" value="$(find styx)../../../data/wp_yaw_const.csv" />
```

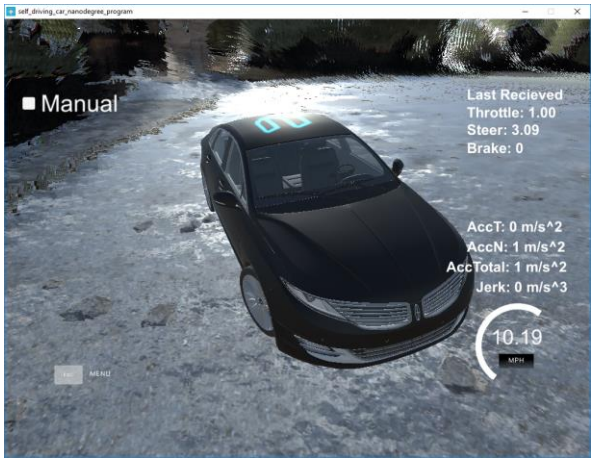
to use the `churchlot_with_cars.csv` as follows:

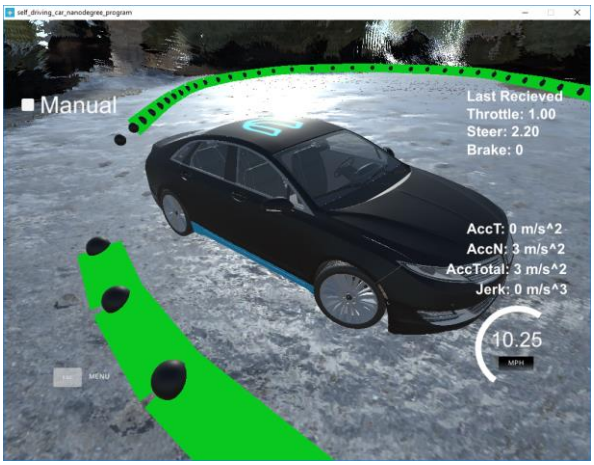
```
<param name="path" value="$(find styx)../../../data/churchlot_with_cars.csv"/>
```

Below are six snapshots for the car driving in the Test Lot. Without sticking, the car keeps as U turn in a counter-clockwise direction. The car's speed is about 10 MPH.







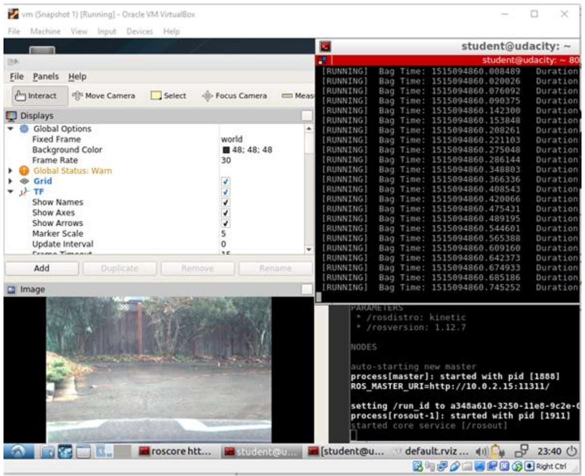


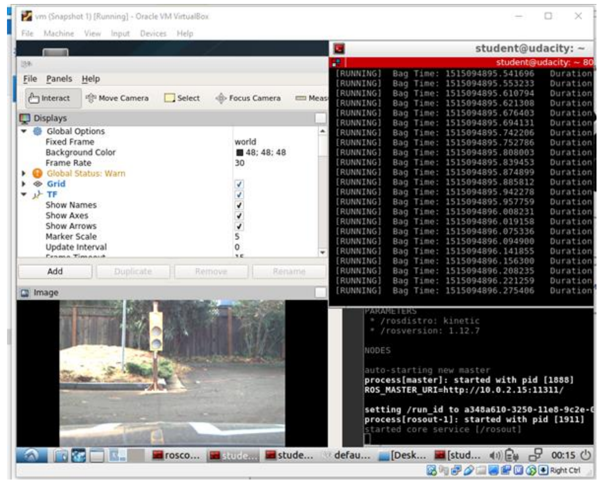
Real world testing

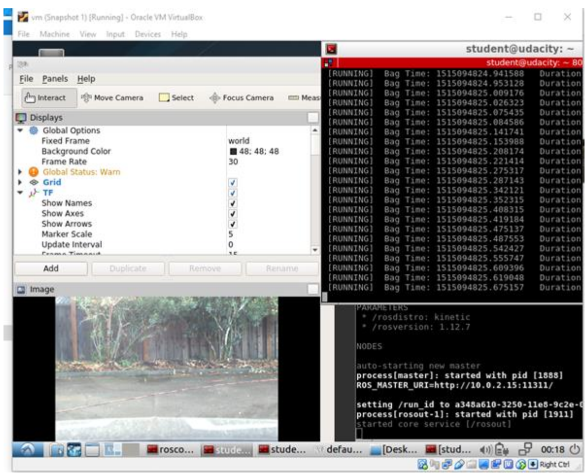
To make a 'real world testing', we have to download the training bag that was recorded on the Udacity self-driving car, and unzip the file. Then we need:

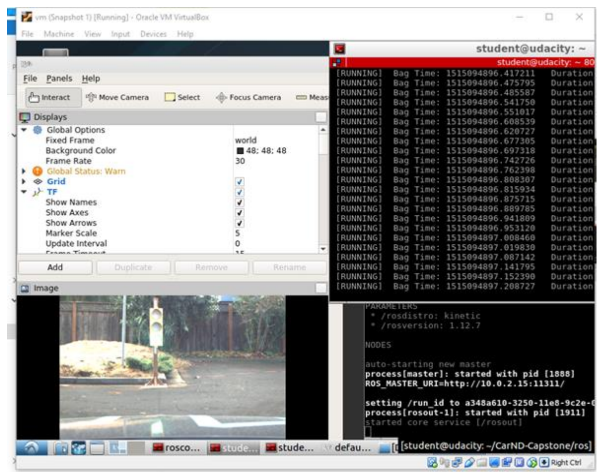
1. Download the rviz config file.
2. Open a terminal and start `roscore`.
3. Open another terminal, run `rosbag play -l traffic_light_bag_file/traffic_light_training.bag`
4. Open one more terminal and run `rviz`.

The testing results shows as the following four snapshots.


vm (Snapshot 1) [Running] - Oracle VM VirtualBox
File Panels Help
Interact Move Camera Select Focus Camera
Displays
Global Options
Fixed Frame world
Background Color 48; 48; 48
Frame Rate 30
Global Status: Warn
Grid
TF
Show Names
Show Axes
Show Arrows
Marker Scale 5
Update Interval 0
Add Duplicate Remove Rename
Image
student@udacity: ~
process[master]: started with pid [1888]
ROS_MASTER_URI=http://10.0.2.15:11311/
process[rosout-1]: started with pid [1911]


vm (Snapshot 1) [Running] - Oracle VM VirtualBox
File Panels Help
Interact Move Camera Select Focus Camera
Displays
Global Options
Fixed Frame world
Background Color 48; 48; 48
Frame Rate 30
Global Status: Warn
Grid
TF
Add Duplicate Remove Rename
Image
student@udacity: ~
process[master]: started with pid [1888]
ROS_MASTER_URI=http://10.0.2.15:11311/
process[rosout-1]: started with pid [1911]


vm (Snapshot 1) [Running] - Oracle VM VirtualBox
File Panels Help
Interact Move Camera Select Focus Camera
Displays
Global Options
Fixed Frame world
Background Color 48; 48; 48
Frame Rate 30
Global Status: Warn
Grid
TF
Add Duplicate Remove Rename
Image
student@udacity: ~
process[master]: started with pid [1888]
ROS_MASTER_URI=http://10.0.2.15:11311/
process[rosout-1]: started with pid [1911]


vm (Snapshot 1) [Running] - Oracle VM VirtualBox
File Panels Help
Interact Move Camera Select Focus Camera
Displays
Global Options
Fixed Frame world
Background Color 48; 48; 48
Frame Rate 30
Global Status: Warn
Grid
TF
Add Duplicate Remove Rename
Image
student@udacity: ~
process[master]: started with pid [1888]
ROS_MASTER_URI=http://10.0.2.15:11311/
process[rosout-1]: started with pid [1911]
[student@udacity: ~/CarND-Capstone/ros]