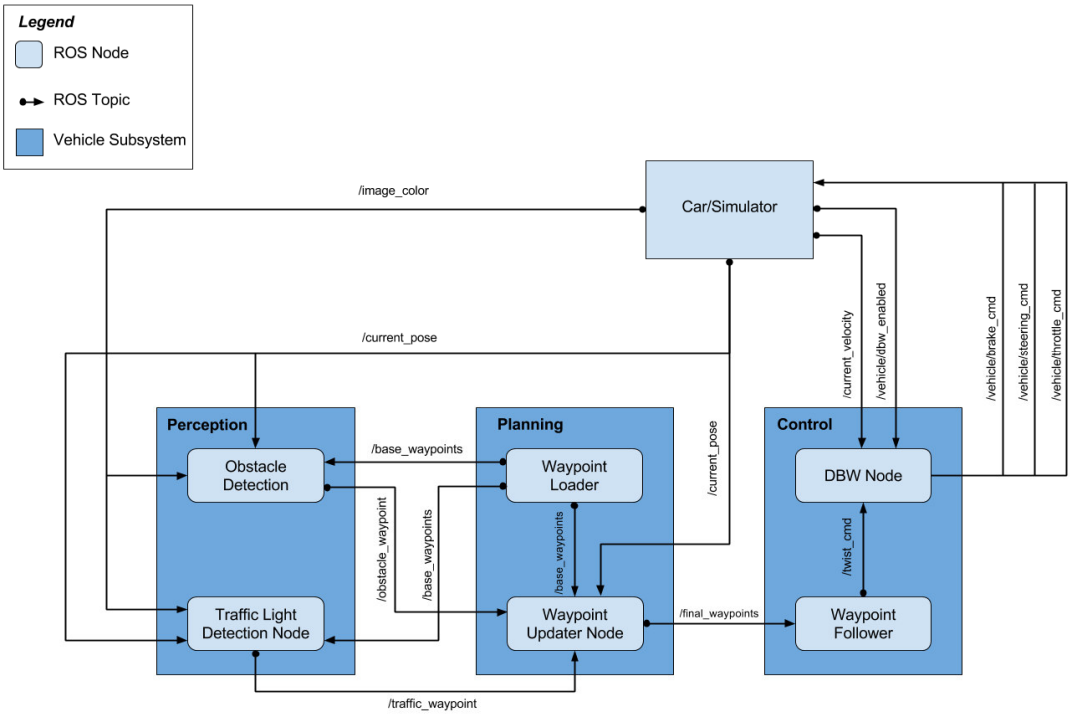
Self-Driving-Car_System-Integration

Using ROS to program a self-driving car.

Note: This project does not include a neural network traffic light detection. The traffic light detection is using simulator ground truth detected.

System Architecture Diagram

The following is a system architecture diagram showing the ROS nodes and topics used in the project. They are described briefly in the Code Structure section below.



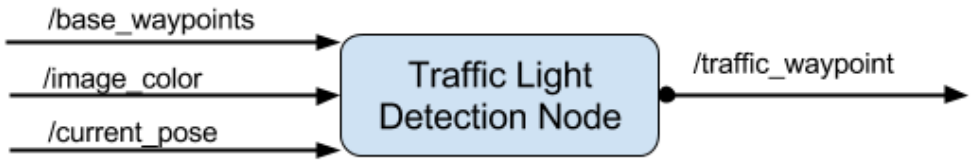
Twist commands that the DBW Node subscribes to are basically linear and angular accelerations.

Code Structure

The code is contained entirely within the (path_to_project_repo)/ros/src/ directory. Within this directory, you will find the following ROS packages:

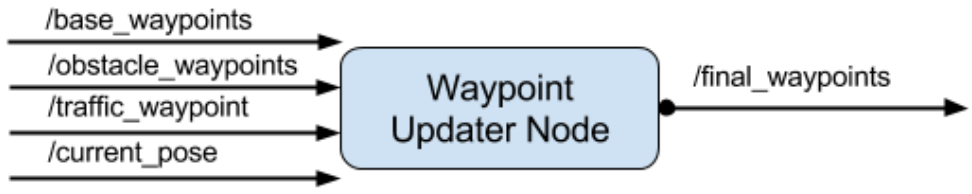
/ros/src/tl_detector/

This package contains the traffic light detection node: tl_detector.py. 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. The `/current_pose topic` provides the vehicle's current position, and `/base_waypoints` provides a complete list of waypoints the car will be following. I built both a traffic light detection node and a traffic light classification node. Traffic light detection takes place within tl_detector.py, whereas traffic light classification takes place within `../tl_detector/light_classification_model/tl_classfier.py`.



/ros/src/waypoint_updater/

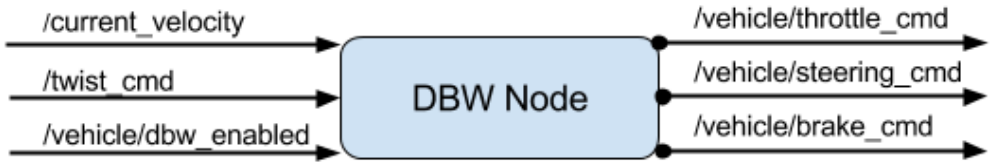
This package contains the waypoint updater node: `waypoint_updater.py`. The purpose of this node is to update the target velocity property of each waypoint based on traffic light and obstacle detection data. This node will subscribe to the `/base_waypoints`, `/current_pose`, `/obstacle_waypoint`, and `/traffic_waypoint topics`, and publish a list of waypoints ahead of the car with target velocities to the `/final_waypoints` topic.



/ros/src/twist_controller/

Carla is equipped with a drive-by-wire (dbw) system, meaning the throttle, brake, and steering have electronic control. This package contains the files that are responsible for control of the vehicle: the node dbw_node.py and the file twist_controller.py, along with a pid and lowpass filter that you can use in your implementation. The dbw_node subscribes to the `/current_velocity` topic along with the `/twist_cmd topic` to receive target linear and angular velocities. Additionally, this node will

subscribe to `/vehicle/dbw_enabled`, which indicates if the car is under dbw or driver control. This node will publish throttle, brake, and steering commands to the `/vehicle/throttle_cmd`, `/vehicle/brake_cmd`, and `/vehicle/steering_cmd topics`.



In addition to these packages you will find the following. The styx and styx_msgs packages are used to provide a link between the simulator and ROS, and to provide custom ROS message types:

- (path_to_project_repo)/ros/src/styx/ A package that contains a server for communicating with the simulator, and a bridge to translate and publish simulator messages to ROS topics.
- (path_to_project_repo)/ros/src/styx_msgs/ A package which includes definitions of the custom ROS message types used in the project.
- (path_to_project_repo)/ros/src/waypoint_loader/ A package which loads the static waypoint data and publishes to /base_waypoints.
- (path_to_project_repo)/ros/src/waypoint_follower/ 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`.

Usage

1. Clone the project repository

```
git clone https://github.com/udacity/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
```

4. Run the simulator