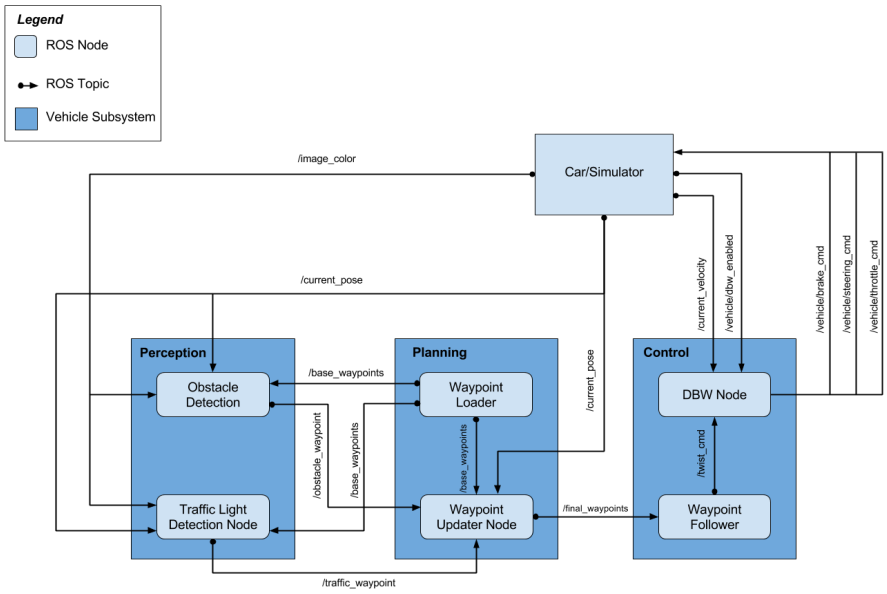
classroom.udacity.com

Self-Driving Car - Udacity

5-7 minutes

For this project, you'll be writing ROS nodes to implement core functionality of the autonomous vehicle system, including traffic light detection, control, and waypoint following! You will test your code using a simulator, and when you are ready, your group can submit the project to be run on Carla.

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 classroom concepts of this lesson.



Below is a brief overview of the repo structure, along with descriptions of the ROS nodes. The code that you will need to

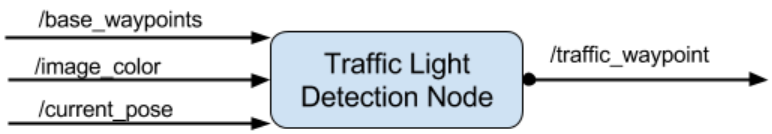
modify for the project will be contained entirely within the `(path_to_project_repo)/ros/src/` directory. Within this directory, you will find the following ROS packages:

(path_to_project_repo)/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.

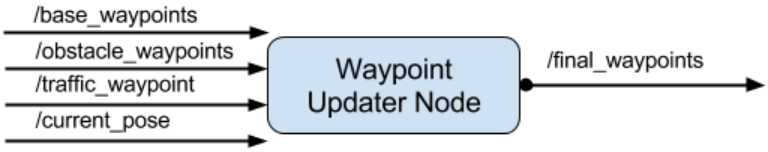
You will build both a traffic light detection node and a traffic light classification node. Traffic light detection should take place within `tl_detector.py`, whereas traffic light classification should take place within `../tl_detector/light_classification_model/tl_classfier.py`.



(path_to_project_repo)/ros/src/waypoint_updater/

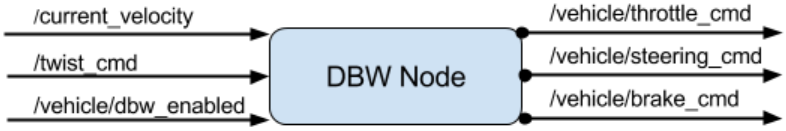
This package contains the waypoint updater node: `waypoint_updater.py`. The purpose of this node is to update the target velocity property of each waypoint based on traffic light and obstacle detection data. This node will subscribe to the `/base_waypoints`, `/current_pose`, `/obstacle_waypoint`, and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

topic.



(path_to_project_repo)/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, which are not necessary to change for the project. 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.

  Because you will be writing code across several packages with some nodes depending on messages published by other nodes, we suggest completing the project in the following order:

1. Waypoint Updater Node (Partial): Complete a partial waypoint updater which subscribes to `/base_waypoints` and `/current_pose` and publishes to `/final_waypoints`.
2. DBW Node: Once your waypoint updater is publishing `/final_waypoints`, the `waypoint_follower` node will start publishing messages to the`/twist_cmd` topic. At this point, you have everything needed to build the `dbw_node`. After completing this step, the car should drive in the simulator, ignoring the traffic lights.
3. Traffic Light Detection: This can be split into 2 parts:

- Detection: Detect the traffic light and its color from the `/image_color`. The topic `/vehicle/traffic_lights` contains the exact location and status of all traffic lights in simulator,

so you can test your output.

- Waypoint publishing: Once you have correctly identified the traffic light and determined its position, you can convert it to a waypoint index and publish it.

4. Waypoint Updater (Full): Use `/traffic_waypoint` to change the waypoint target velocities before publishing to `/final_waypoints`. Your car should now stop at red traffic lights and move when they are green.

   In the next few classroom concepts, we'll cover each component of the project. You can use these concepts as a guide for the suggested order of project completion, but feel free to ignore these steps and implement the project how you please!