December 2, 2 0 1 7 B Y VVS TEAM SPARTAN

System Integration - Traffic light detection and classification

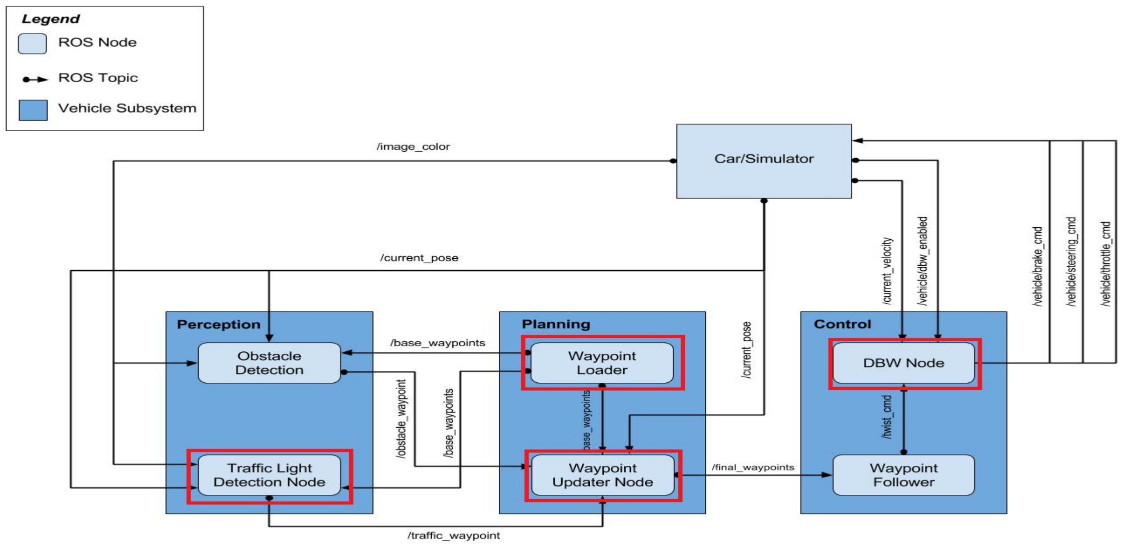
In this project, we are suppose to writing ROS nodes to implement core functionality of the autonomous vehicle system, including traffic light detection, control, and way point following. This project requirement is to work as team of 5 udacity students to share set of tasks before final integration to project . Our team successfully able to complete given each task and integrate final software to get expected results on simulator .

Architecture:

The following is a system architecture diagram showing the ROS nodes and topics used in the project. Our team was mainly working on four important modules as highlighted by red blocks in diagram.

Following are list of tasks assigned to team :

1) Way point Loader Node (publish next set of way points from current position of car)
2) Traffic light detection node which includes
   a) Traffic light classification (classify as red , yellow , green).
   b) Nearest Traffic light way point index detection and publish.
3) Final way point updater node (updates velocity for each base way point based on stop distance to traffic light)
4) Drive by wire node (DBW) use various controllers to provide appropriate throttle, brake, and steering commands



Waypoint Loader :

This module is designed and implemented by our teammate **Sergio Valero**.

Before the car can move in the simulator, it will be necessary to write a first version of the waypoint_updater node. We can find class and method stubs for the node in

(path_to_project_repo)/ros/src/waypoint_updater/waypoint_updater.py

The eventual purpose of this node is to publish a fixed number of waypoints ahead of the vehicle with the correct target velocities, depending on traffic lights and obstacles. The goal for the first version of the node should be simply to subscribe to the topics

- /base_waypoints
- /current_pose

and publish a list of waypoints to

- /final_waypoints

The /base_waypoints topic publishes a list of all waypoints for the track, so this list includes waypoints both before and after the vehicle (note that the publisher for /base_waypoints publishes only once). For this step in the project, the list published to /final_waypoints should include just a fixed number of waypoints currently ahead of the vehicle:

- The first waypoint in the list published to /final_waypoints should be the first waypoint that is currently ahead of the car.
- The total number of waypoints ahead of the vehicle that should be included in the /final_waypoints list is provided by the LOOKAHEAD_WPS variable in waypoint_updater.py.

The next section includes details about the message type used to publish to /final_waypoints.

Traffic light detection node :

Once the vehicle is able to process way points. Way pointer updater module will select set of base way points and publish to other modules .

Traffic light Detector:

This module was implemented by our teammate **Swaroop**.

It is important to know current way point of vehicle and closest obstacles way point to detect exact stop distance to obstacle way point . Traffic lights are the first obstacle that we'll focus on.This module will detect way point close to traffic lights will help us to plan steering and throttle commands to stop vehicle without jerks and incidents. This module will also call traffic light classifier to update statues of traffic light to nearest traffic light way point .

Finally , This module will publish nearest way point to traffic light and traffic light status to final way point updater module.This module has four following simple steps:

1) Detect way point close to vehicle position.

2) Keeping vehicle waypoint as reference detect traffic light way point index close to vehicle waypoint .

3) Detect stop line way point point index close to nearest traffic light way point. This stop line index will be further used to detect distance to vehicle and change velocity to stop at stop distance.

4) Distance to stop line is also used to enable and disable classifier when stop line is at far distance .

Traffic light classifier :

The main aim of this module is to classify traffic light as red , green and yellow. We as team explored couple of methods to and classify traffic light. Please find bellow couple of methods decided to implements progressively,

1) Simpler thresholding method to just threshold on red channel to determine presence of red color in image . This method was proposed as simple work around which will helped us in basic integration testing of other modules developed in this project . This work is done by our teammate **FELIX DELGADO**.

2) Hough Circle detection: This method was proposed and implemented by our teammate **Swaroop**. This method involves following steps in final solution ,

A) Convert BGR to HSV image .

B) Dividing image in to number of NxN blocks and slide NxN blacks over image with stride of M pixels .

C) Select a first half and left portion of image as ROI to detect traffic light

D) Threshold pixels in each blocks with in red , green and yellow color ranges to get three seperate output 64x64 bitmap for three colors. (For more details on range can refer this: https://en.wikipedia.org/wiki/HSL_and_HSV)

E) Apply GaussianBlur to remove unwanted noise in all three output 64x64 bitmap.

F) Apply Hough circle detection on this bit map to get circles in all bit bamps seperatly.

G) count number of circles detected in each individual bitmap. Find total count of circles in selected ROI.

H) Take decision on red , green , yellow based on number of circles detected in each bit map.

I) Hough circle min and max radius is configured based on distance from traffic light . we can expect circle of small radius when vehicle is away from traffic light and can expect circle increase as it moves towards traffic light

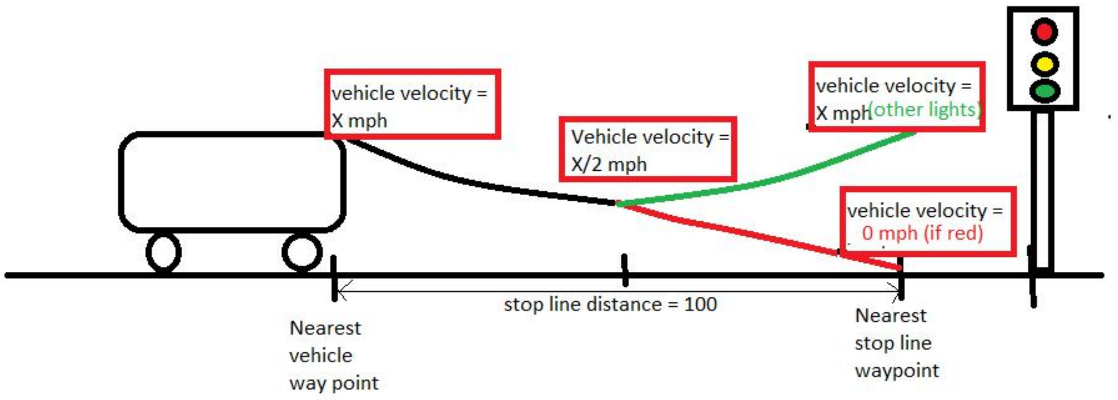
J) We also added extra logic to verify this increase in radius temporally to verify detected circle is traffic light .

3) Detection of other shapes like rectangle and detection of presence of circle with in specific detected rectangle to confirm presence of traffic light . This method was proposed and implemented by our teammate Sergio Valero.

4) Train a CNN with TF or using Just use SVM classifier with some images from the simulator and also some reals images with ground truth downloaded from public network. This method was proposed by our teammate Sergio Valero. We couldn't able to achieve better performance on simulate after traning on real data . Our teammate sergio is currently extracting images from simulator for training .

We decided to use simple hough circle based traffic light detection for initial submission. We will continue improving our method to make it work for both simulator and real data .

Waypoint Update :


vehicle velocity = X mph
Vehicle velocity = X/2 mph
vehicle velocity = X mph(other lights)
vehicle velocity = 0 mph (if red)
stop line distance = 100
Nearest vehicle way point
Nearest stop line waypoint

This module was designed and implemented by our teammates **Venkat and Swaroop** .Waypoint Updater module is responsible for taking the inputs from Base waypoints, Traffic light waypoints and current position of the car and it publishes final way points to the Waypoint Follower.

It is also responsible to set the appropriate velocity to the Waypoints in such way that car de accelerate and stops smoothly in case of RED traffic light detects.

In design this module is using adaptive number of look ahead Waypoints (LOOKAHEAD_WPS) based on velocity and stop line distance.

Maximum look ahead Waypoints is 200

Minimum look ahead Waypoints is 20

Based on stop line distance Waypoint Updater module is start to slow down when stop line distance close to 100 and start reducing velocity gradually till stop distance close to 50 irrespective of lights , this is done to anticipate for sudden change in the light to red.

This proposal solution continues to reduce velocity gradually from 50 to zero if red light is detected; else it will continue with maximum velocity.

Smooth Velocity Increment :
In the following cases velocity increments gradually in step wise, in each step 0.5 increment until car reaches to target velcity

Case 1: In case of car stops at red signal and moving after signal turns into green.

Case 2: In case of traffic signal anticipation irrespective of signal light car slows down to 3/4th of the target velocity, after that if signal is green car accelerates to target velocity gradually steps wise 0.5 in each step until car reaches to target velocity.

Drive by wire node (DBW) :

This module was primarily implemented by our teammates **Giuseppe Bruno**. Later , our teammates **venkat and swaroop** were consistently tuning to improve performance of module .

Once messages are being published to /final_waypoints, the vehicle's waypoint follower will publish twist commands to the /twist_cmd topic. The goal for this part of the project is to implement the drive-by-wire node (dbw_node.py) 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

Since a safety driver may take control of the car during testing, you should not assume that the car is always following your commands. If a safety driver does take over, your PID controller will mistakenly accumulate error, so you will need to be mindful of DBW status. The DBW status can be found by subscribing to /vehicle/dbw_enabled.

When operating the simulator please check DBW status and ensure that it is in the desired state. DBW can be toggled by clicking "Manual" in the simulator GUI.

All code necessary to implement the drive-by-wire node can be found in the package:

(path_to_project_repo)/ros/src/twist_controller

Simulator Braking: twist_controller.py

We investigated torque based braking in the simulator for drive by wire . In this approach we observed either under estimation or overestimation of which leds to vehicle stopping way before or some time not stopping at all based on traffic light and way point target velocity feedback . we are requesting target velocity correctly , vehicle is unable to stop exactly at stopline as target velocity was never reached . we started tuning PID control gain factors and lowpass filter time constants . But , we were not able to achieve expected results .

We start using brake_deadband when calculating brake torque. If the brake deceleration is smaller than the brake deadband , there is no need to apply the brakes , since , vehicle is suppose to de accelerate on its own after reaching that limit due to engine and drag .

We tried couple of boundary condition thresholding based on deadband like ,

- handling boundary conditions for brake based on deadband at deadstop condition . Checking for target linear velocity less than minimum velocity limit and brake value is less than deadband , we are keeping brake constant at deadband value .

- When deadbank is less than minimum threshold , we tried varying steering based on target angular velocity and steer_ratio . This was improving steering at turns .

- When throttle brake is less than negative of deadband , we were making brake signal positive of throttle brake

All the above mentioned logic helped us to smooth our braking at stop signal .

Later , we tried using percentage braking instead of torque braking (BrakeCmd.CMD_PERCENT) . This also help us to sharp stopping of vehicle when there is sudden change in light close to stop distance . It has almost similar performance to torque based braking

dbw_node.py:

This python file implements the dbw_node publishers and subscribers. You will need to write ROS subscribers for the /current_velocity, /twist_cmd, and /vehicle/dbw_enabled topics. This file also imports the Controller class from twist_controller.py which will be used for implementing the necessary controllers. The function used to publish throttle, brake, and steering is publish.

Note that throttle values passed to publish should be in the range 0 to 1, although a throttle of 1 means the vehicle throttle will be fully engaged. Brake values passed to publish should be in units of torque (N*m). The correct values for brake can be computed using the desired acceleration, weight of the vehicle, and wheel radius.