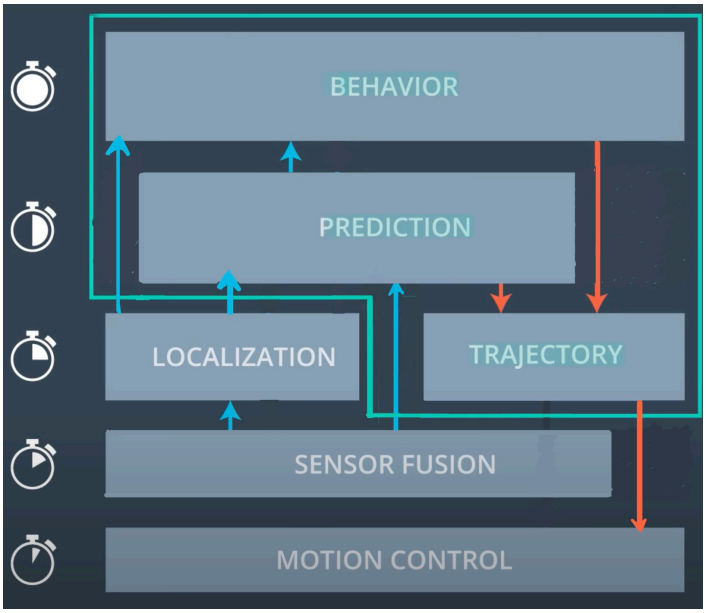
# System Integration - ROS

## Goal:

Design a software that will calculate the trajectory of an autonomous vehicle . The speed of the vehicle speed has to be updated as the traffic lights change color.

## Path Planing :

**Data flow :**




**Autonomous Vehicle Data pipeline**

Path planning was addressed in extenso in a previous project ( Path Planning High Way Driving) .

We'll only focus on some parts of the pipeline components :

- sensor fusion: detection of the traffic lights location and their states ( Green, Yellow, Red)
- localization : vehicle pose
- behavior : adapting the vehicle speed to obey the relevant traffic lights
- trajectory : calculate the vehicle trajectory (pose) and adapt its speed to its behavior

**Robot Operating System (ROS):**

Information circulates from one module to the next ones at different frequencies : a specialized operating system (**ROS**) passes messages between each components .

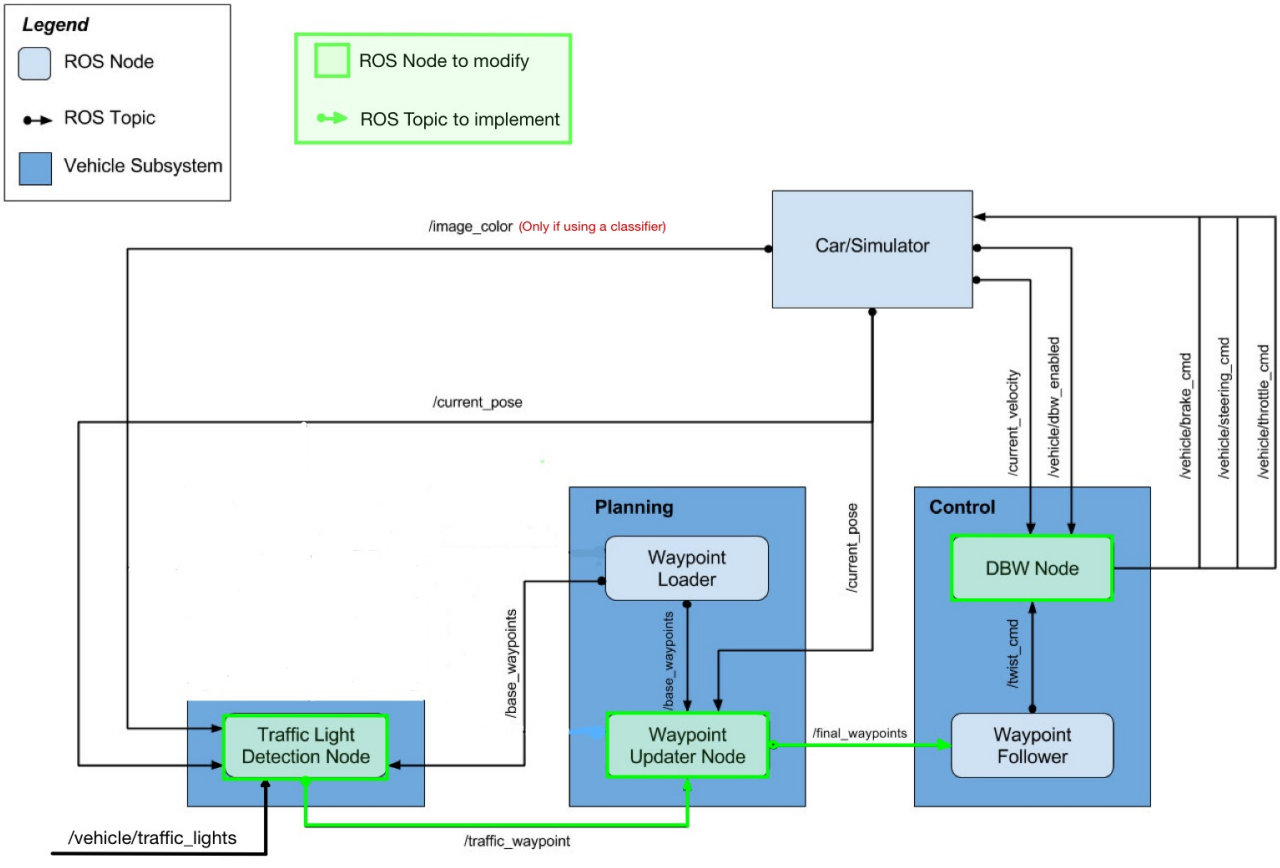
ROS quick over view:

- nodes are libraries of functions
- nodes communicate with each others through 2 types messages: topics or services

| | topic | service |
|---|---|---|
| **type** | Bus | on demand |
| **mode** | publish /subscribe | request /response |
| **frequency** | event based | as needed |

- **topics** :
  - nodes **publish topics** to **output** some of their functions results .
  - nodes that need **input** from other nodes **subscribe** to the corresponding **topics**
  - topics frequency is set in each nodes (ie asynchronous messaging)
- **services** :
  - services are messages exchange between specific nodes on request (ie synchronous messaging)

This project will only feature topic messages as per the following layout.



## Technical Set Up Overview:

**Drive Simulator :**

A provided simulator emulates the movements of an autonomous vehicle in its surrounding environment:

- the **current_pose** is published the vehicle localization at 50hz
- **base_waypoint**s publishes the entire original trajectory
- **/vehicle/traffic_lights** publishes the location of all traffic lights
- **/traffic_waypoint** publishes the location of the closest light to the moving vehicle
- **final_waypoint** only publishes a limited portion of the trajectory ahead of the car . The speed along this section of trajectory is updated to obey the relevant traffic light.
- the Waypoint Follower subscribes to **final_waypoint,** to calculate and publish twist commands ( **twist_cmd**). This topic is then processed by the DBW node (drive by wire) to publish brake, steering and throttle commands. The simulator subscribes to these topics to simulate the driving.

NB: The topics and nodes to implement or modify within this project appear in green in the previous chart and list .

**Data format and programing environment :**

All nodes are implemented in Python .

## Modules methodology :

### 1) Trajectory Planning :

A trajectory is a series of points the autonomous vehicle will drive on sequentially . Each waypoint features the vehicle intended location , orientation, velocity and more.

Once planned, the entire sequence of waypoints for the complete trajectory (origin to destination) is published by the **Waypoint Loader** node (topic **/base_waypoints**) .


entire trajectory : all waypoints
**behind and ahead** of the vehicle

The Waypoint Updater narrows down on a window of 50 waypoints only located right ahead of the moving vehicle ( using the vehicle pose from topic **/current_pose**) .


waypoint lane pruned to **50 points ahead** of the vehicle, **no trailing waypoints**

These 50 waypoints are then edited to decrease the velocity to a complete stop from the vehicle pose to the **closest detected red-light** within this range. ( light location and state from topic **/vehicle/traffic_light**)

When the light turns back (or is) green (or yellow) these 50 waypoints are then taken unaltered straight from **/base_waypoints**

These 50 waypoints are then published to **/final_waypoints**.

## 2) Traffic Light detection :

All traffic light location are available and published by the topic **/vehicle/traffic_lights**. Although the state (Green,Yellow, Red) could be inferred by a classifier, we'll use the state also provided in this topic.

The Traffic Light detection node :

- spots the closest traffic light ahead of the vehicle
- check if the light color remains steady before publishing its location .

This persistence check is to filter out classifications error.

Consequently the topic **/traffic_waypoint** publishes :

- the index of the closest waypoint to the traffic line stop for a red light
- -1 for green or yellow

Note that traffic lights located beyond 50 waypoints (corresponding roughly to a driver lookahead) are out of scope and therefore not considered to manage the vehicle velocity.

## 3) Programming Highlights :

*get_closest_waypoint_idx*:

The provided KDTree outputs a list of waypoints sorted by their distance to a specific location.

**1) closest waypoints to the vehicle:**

we pick the closest waypoint **ahead** of the vehicle by using the sign of the projection of the 2 closest waypoints vector on the vector "vehicle_location to closest waypoint " .

**2) closest waypoint from the traffic stop-line :**

We use closest point straight from the KDTree function.

*decelerate_waypoints*:

The velocity decreases for each waypoint between the car and the stop line as a function of the distance of each waypoint to the stop-line.

## Results :

See a trajectory a video of a trajectory through traffic lights:

See video of an edge case: light change to yellow as the vehicle is about to cross the stop line:


## Conclusion :

Although the project touch upon detecting traffic lights and managing the vehicle trajectory accordingly, the significant take away is to learn how the functional modules of the autonomous vehicle pipeline communicate with each others through ROS.

A developing version of this project code featuring debugging comments (log warn) can help to understand the challenges encountered while using internodes topic messages .

The final version thoroughly commented (but purged of any debugging lines for clarity) can be found here.

Like most AI products, Autonomous Vehicle driving platform will also have to be localized to comply to the countries driving rules . For example:

- in Germany traffic lights turn from Red to Yellow ( prepare to start ), to Green.
- all over Europe the stop line is at the traffic light, as all lights are actually before the intersection but not across (as opposed to the USA) .