Project Summary

# Team Membership

I am submitting this project individually. I do not wish to have my code evaluated on Carla and have not joined a team.

My contact email is: [mark@gloudemans.net](mailto:mark@gloudemans.net)

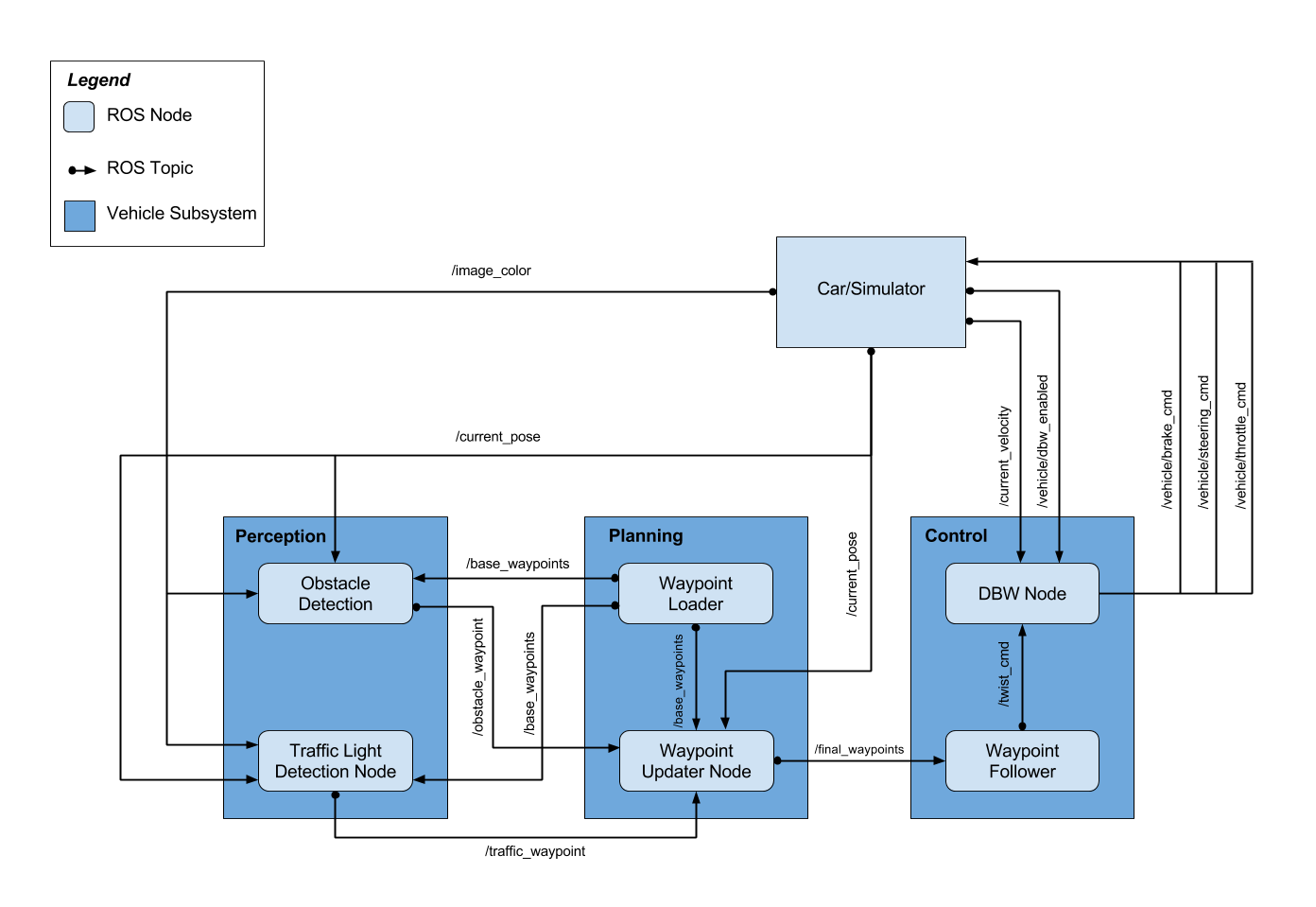
# Submission Checklist and Requirements

My project:

* Launches correctly using launch/styx.launch.
* Smoothly follows waypoints in the simulator.
* Respects the target top speed set for the waypoints' twist.twist.linear.x in waypoint\_loader.py.
* Stops at traffic lights when needed.
* Stops and restarts PID controllers depending on the state of /vehicle/dbw\_enabled.
* Publishes throttle, steering, and brake commands at 50 Hz.

# Code Summary

Starting with the cloned project repository, I updated the code in the files listed below files to implement the required functionality. My updates generally followed the approach outlined in the class walkthrough videos.



## waypoint\_updater.py

This file implements the ROS waypoint updater node. The node subscribes to the following topics:

* /base\_waypoints – waypoints defining the route
* /current\_pose – current position of the vehicle
* /traffic\_waypoint – index of red light at which to stop, or -1 if no red light

The node publishes the following topic:

* /final\_waypoints – list of waypoints to traverse and proposed velocities for the vehicle

When the node receives a base waypoint message, it stores the list of waypoints defining the track. When the node receives a current pose message, it stores the current pose. When the node receives a traffic waypoint message, it stores the traffic waypoint.

At a 50 Hz rate, the node publishes a list of the next 50 waypoints defining the desired trajectory for the vehicle. The node computes these waypoints by finding waypoints that lie immediately ahead of the vehicle position defined by the current pose. The node sets the desired maximum vehicle velocities to the speed limit unless a red light is detected, in which case it computes velocities to uniformly decelerate the car to a stop just prior to the traffic light location.

## dbw\_node.py

This file implements the ROS dbw (drive by wire) node. The node is responsible for converting proposed linear and angular velocities to throttle, brake, and steering commands. This translation is vehicle specific, making use of various physical properties of the car.

The node subscribes to the following topics:

* /current\_velocity – current vehicle velocity
* /twist\_cmd – proposed vehicle velocity
* /vehicle/dbw\_enabled – drive by wire enable state

When drive by wire is enabled, the node publishes the following topics:

* /vehicle/steering\_cmd – steering angle (radians)
* /vehicle/throttle\_cmd – throttle command (0 to 1)
* /vehicle/brake\_cmd – brake torque (N/m)

This object acts principally as a rospy interface wrapper for a Controller object that performs the business logic.

## twist\_controller.py

The twist controller receives inputs describing the proposed vehicle linear velocity and angular velocity and the current linear velocity and is responsible for computing throttle, brake torque, and steering angle values to approximate the proposed state.

The yaw controller is implemented as a separate component and outputs a steering angle as a function of commanded vehicle velocity and angular velocity.

## yaw\_controller.py

Given a desired rate of turn and the vehicle longitudinal velocity, the yaw controller computes the required steering angle. The yaw controller constrains the steering angle such that it remains within mechanical limits and such that lateral acceleration limits are not exceeded. When the vehicle is stopped, the controller should returns steering angle of zero because no rate of turn can be achieved.

## tl\_detector.py

This file implement the traffic light detection ROS node. The node subscribes to the following topics:

* /current\_pose – current vehicle twist state
* /base\_waypoints – desired vehicle route
* /vehicle/traffic\_lights – traffic light positions along route
* /image\_color – camera images

When drive by wire is enabled, the node publishes the following topics:

* /vehicle/traffic\_waypoint – index of upcoming red traffic lights in range

When the node receives current pose, base waypoints, or traffic light messages it saves their contents for reference. When the node receives an image message, it classifies the image as either containing a red traffic light or not. If the image contains a red traffic light, the node determines the index of the traffic light and publishes that index on the traffic waypoint topic. If the image does not contain a red light the node published the sentinel value -1. The traffic light classifier is implemented as a separate component.

## tl\_classifier.py

This file implements a traffic light classifier object. The object constructor loads a frozen TensorFlow inference graph, and the object provides a method that applies that inference graph to the specified image. If a traffic light is detected, the classifier returns its most probable state, otherwise it returns unknown.

# Submission

The submission includes:

* All code for your project (including code used to train the traffic light detector), submitted via a zipped file.
* This ReadMe.pdf file.
* Blank “notes to the reviewer” field during submission.