Udacity Self-Driving Car Capstone Project

David Muk (muksiukei@gmail.com), Denise James (denisetoo@gmail.com), Daliso Zuze (daliso.zuze@gmail.com), Amogh Mahapatra (lucky5devilhere@gmail.com), Kris Harikrishnan (kharikri@yahoo.com)

Udacity SDC Capstone Project

Introduction

In this final capstone project of the Self-Driving Car Nanodegree course we write code to help drive Carla (an actual car) autonomously although at a maximum speed of 10mph for safety reasons. This project brings together several aspects of the SDC course such as Perception, Planning and Control. We first test the code on the simulator and then run it on Carla. The code is written using the ROS (Robotics Operating System) framework which works both on the simulator as well as on Carla.

Architecture

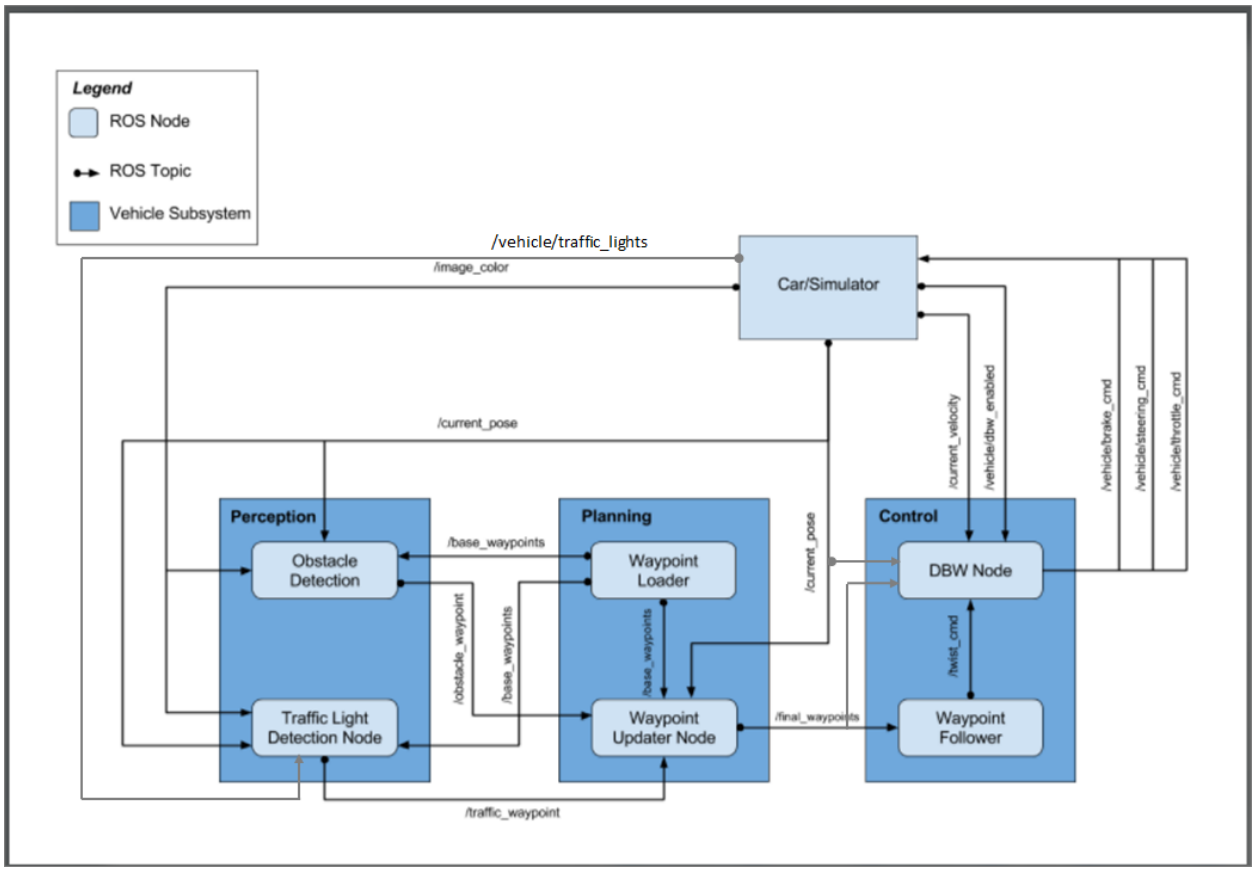

Figure 1 – SDC System Architecture within ROS Framework

The system architecture as shown above is comprised of three subsystems - Perception, Planning and Control. This architecture picture also shows the ROS nodes and ROS topics. For example we subscribe to the topic */vehicle/traffic_lights* from the simulator to acquire an accurate ground truth data of location and current color state of

all traffic lights in the simulator for the traffic light classifier. The implementation is described in more detail in the next section.

Implementation

The ROS (Robotic Operating System) framework is used to implement the above mentioned subsystems: Perception, Planning and Control. The ROS framework provides a powerful mechanism for ROS nodes to communicate between each other. Each of the ROS nodes typically performs a specific functionality. The above architecture picture shows six ROS nodes and we will implement three of them: **Traffic Light Detector node**, **Waypoint Updater node**, and **DBW node**. The ROS framework also allows nodes to publish (send) and subscribe (receive) messages called topics. In addition to the three nodes, we will also implement appropriate publishers and subscribers for the nodes to communicate between each other. See Appendix for a complete list of ROS nodes and topics used in this project.

Perception

The Perception subsystem takes sensor data such as camera images and produces meaningful information such as whether traffic light is red or green.

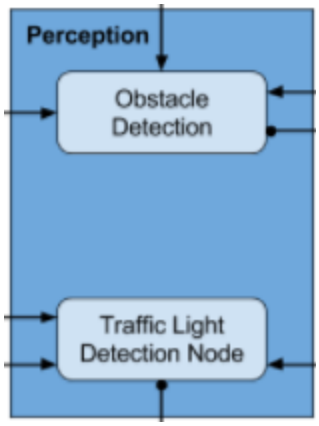

Figure 2 – Perception Subsystem

As depicted above the Perception subsystem has two ROS nodes: Obstacle Detection node and Traffic Light Detection node. The Obstacle Detection node is not required for this project. The Traffic Light Detection node has two parts to 1) to detect the traffic

lights within a camera image and 2) to classify the traffic lights whether they are red, green or yellow.

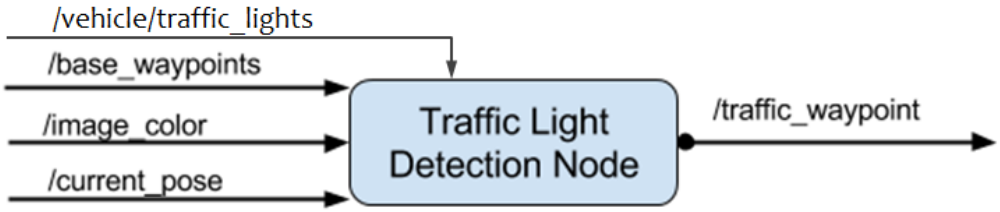

Figure 3 – Traffic Light Detection ROS Node

The Traffic Light Detection Node takes the inputs shown in the above picture to first detect traffic lights in the incoming camera image (*/image_color*) and then classify whether the traffic lights are red, green or yellow. Classification is not required for simulation as the color state information is obtained from the simulator (*/vehicle/traffic_lights*). However we need to classify traffic lights when operating on Carla. We used the following approach for traffic light classification:

1. Training data generation: As recommended in the course we first focussed on driving the car around without focussing on traffic lights. Once the car was reasonably stable in the simulator, we drove the car around and extracted the images while the car was driving. We then labelled the data using the following open source package: https://github.com/tzutalin/labelImg. We labelled the data by drawing bounding boxes around each image of interest. The labels were either Red/Yellow/Green.
2. Inference: The class deep_detector.py contains the code. It invokes a stored model, submits a given image and compares the predicted probability with a threshold to decide upon the image class it belongs to.
3. Model training: In the course of choosing the right models we researched many available models which could potentially solve this problem. The following approaches were considered in our literature review:
    a. Deep Residual Learning for Image Recognition (https://arxiv.org/abs/1512.03385)
    b. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks (https://arxiv.org/abs/1506.01497)
    c. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift (https://arxiv.org/abs/1502.03167)

d. Speed/accuracy trade-offs for modern convolutional object detectors (https://arxiv.org/abs/1611.10012)
e. Object Detection in Video using Faster R-CNN (https://courses.engr.illinois.edu/cs445/fa2015/projects/final/object-detection-video.pdf)
f. SSD: Single Shot MultiBox Detector (https://arxiv.org/abs/1512.02325)

After the analysis of these models, SSD with Inception V2 was chosen for its extremely fast inference response time (average inference time: 0.16 seconds with Nvidia GTX 960 equipped).

Once the classification is done as described above, we publish those waypoints of the stop line closest to red traffic lights on */traffic_waypoint* topic. Later on, the */traffic_waypoint* topic will be used by the Waypoint Updater node to set the traffic to 0 mph to bring the vehicle to a stop gracefully.

Planning

The Planning subsystem takes the output from the file data in the simulator mode or radar and lidar sensor data in the real road application as an input. Perception subsystem outputs the traffic_waypoint and obstacle_waypoint information. The obstacle_waypoint functionality is not covered in this document.

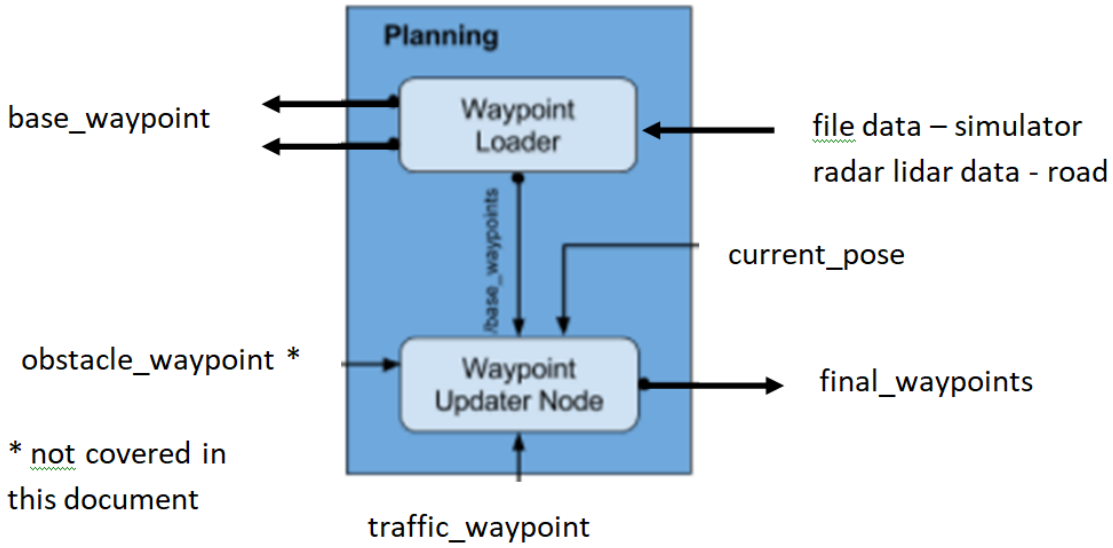

Figure 4 – Planning Subsystem

The Waypoint Loader reads in the waypoint values, position *x,y,z* and *yaw* from a file or sensor data. The *yaw* is transformed from euler to a quaternion value. This quaternion is used to transform the position orientation. The global speed limit from the waypoint_loader.launch file and is used to set every waypoint to have the speed limit as velocity. Then, it decreases the target speed gradually near the end of the track. In the simulator mode all of the base_waypoints are sent to the Waypoint Updater Node in one published message.

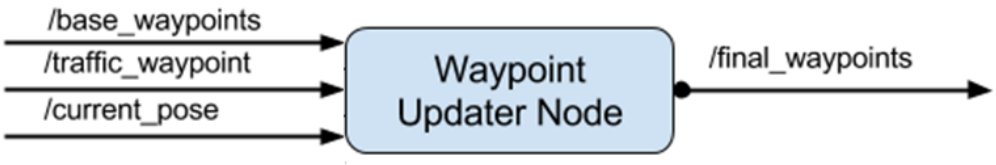

Figure 5 – Waypoint Updater ROS Node

The Waypoint Updater node shown above has two functions. First it takes the base_waypoint and determines the next point the vehicle will proceed to. Second it takes traffic_waypoint information from the Traffic Light Detection node (and the obstacle detection obstacle_waypoint, not covered in this document) to assign correct target velocities for waypoints near traffic lights. Then the Waypoint Updater node publishes final_waypoints depending on traffic lights (and obstacle when defined).

The Waypoint Updater node publishes 30 look ahead final waypoints at a time, based on the other additional inputs as traffic_waypoint, current_pose, and velocity changes if a traffic light is detected.

The target velocity is modified in the Waypoint_Updater in case there is a traffic light detected in the Waypoint_Updater. In the Waypoint_Updater, the velocity is first set to 0 at the waypoint for stop line, and 4 waypoints before that to provide a buffer. The velocity of every waypoint before that will be modified according to its next waypoint's velocity, by the formula:

$$v = min(\sqrt{u^2 + 2as})$$

where *a* is the deceleration rate, *s* is the distance between this and next waypoint, *u* is the velocity of next waypoint, *v* is the velocity of the current waypoint.

Control

The Control subsystem takes the path planned by the Planning subsystem and issues throttle (acceleration), brake, and steering commands to the car through a drive-by-wire system. Drive-by-wire means the throttle, brake, and steering of the car can be controlled electronically through software.

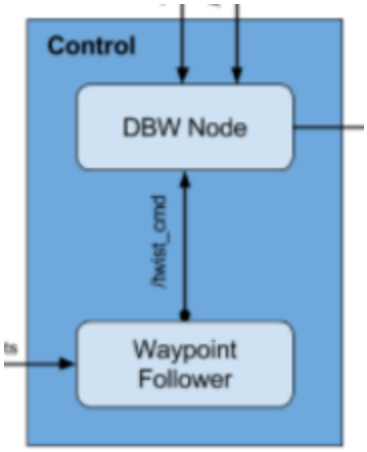

Figure 6 – Control Subsystem

As depicted above the Control subsystem has two ROS nodes: Waypoint Follower node and DBW node. The Waypoint follower node is already implemented for us. It takes the final waypoints determined by the Planning subsystem as input and produces the targeted linear and angular velocities accounting for obstacles like traffic lights.

Within the DBW node, we use these targeted linear and angular velocities (*/twist_cmd*) along with the current velocity (*/current_velocity*), current location (*/current_pose*) of the vehicle and the final waypoints (*/final_waypoints*) as inputs to a standard PID controller described below. The PID controller produces actuation values for throttle, brake and steering to drive the vehicle through drive-by-wire. As seen from the following picture, besides current velocity and twist_cmd we also pass whether drive-by-wire is enabled. Drive-by-wire is always enabled in the simulator however on Carla it may or may not be enabled.

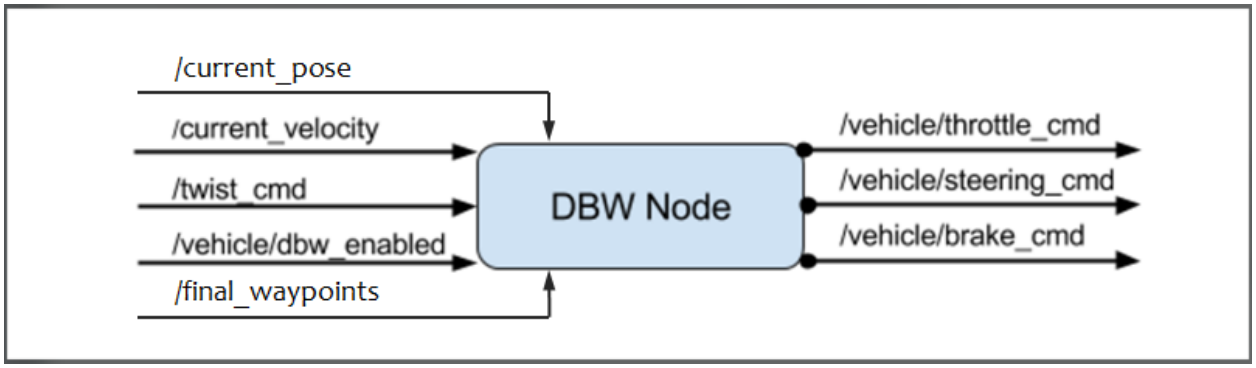

Figure 7 – Drive-By-Wire ROS Node

PID Controller

The P, I, and D parameters play different roles in the control system as described below:

- P generates a steering correction proportional to the cross track error (*cte*). Cross-track error (described below) is the lateral distance between the vehicle and the reference trajectory. A large P results in the vehicle overshooting the track leading to oscillations. A small P results in fewer oscillations but also means it is slower to respond to corrections
- I controls the drift in the vehicle. A misalignment in the car leads to drift which makes the car drive at an angle instead of in a straight line. To fix the effects of drift the I parameter collects the *cte* over time
- The D parameter dampens the overshooting caused by P. D is proportional to the changing rate of the cte

For us the simulator does not have drift so we keep the I parameter close to zero (0.003). Essentially this becomes a PD controller which is straightforward to tune manually. We vary the P value between 0.1 and 0.9. At high P values as expected the vehicle is wobbly. We get the best results at P = 0.3. We varied D between 1.5 to 3.0. At D = 1.3 the wobbles are dampened quite a bit. At D = 2.5 we have minimal amount of wobbles and the vehicle stays inside the track. Our final PID values are P = 0.3, I = 0.003 and D = 2.5.

Cross-Track Error (cte)

The computation of cross-track error involves three steps:

1. Coordinate System Transformation
2. Polynomial Fitting
3. Iterative Error Computation

Coordinate System Transformation

For the cross-track error computation, first eight waypoints $w_1, \ldots, w_8$ from */final_waypoints* are used. These waypoints, together with the current position *p* of the vehicle, is first transformed to the coordinate system with *p* the origin and x-axis in the direction $u = w_8 - w_1$. This is done by applying the following transformation:

$$\begin{pmatrix} x_T \\ y_T \end{pmatrix} = \begin{pmatrix} \cos(-\psi) & -\sin(-\psi) \\ \sin(-\psi) & \cos(-\psi) \end{pmatrix} \begin{pmatrix} x - x_p \\ y - y_p \end{pmatrix}$$

where ψ is the angle between u and the world x-axis.

The direction $w_8 - w_1$ is chosen as the new x-axis rather than the current vehicle orientation because it is guaranteed to point to the forward direction of the route closely.

Polynomial Fitting

The resulting reference points are then used to fit a degree-3 (or lower if less points are available) polynomial, *f* (line 164 in dbw_node.py), by solving the least-square problem

$$\min \|Az - y\|^2$$

where

- *A* is an n-by-4 matrix, in which each row a_i is the vector $[1, x_i, x_i^2, x_i^3]$ for each reference waypoint (x_i, y_i),
- $z = (z_0, z_1, z_2, z_3)$ is the vector of coefficients of the resulting polynomial $\sum_{k=0}^{3} z_k x^k$.

Iterative Error Computation

The cross-track error *cte* is then computed by applying Newton's method (lines 169-174 in dbw_node.py) with 5 iterations:

$$x_{n+1} = x_n - \frac{f(x_n)f'(x_n) + x_n}{f'(x_n)^2 + f(x_n)f''(x_n) + 1}$$

We use the x-coordinate of the vehicle's transformed current position as an initial guess as it is generally close to the optimal solution. This approach computes the x-coordinate of the point on the curve such that its distance from the vehicle is minimized. The magnitude of *cte* is then computed by

$$cte = \sqrt{x_{sol}^2 + f(x_{sol})^2}$$

The sign of the *cte* is determined by the sign of the constant term of *f* as it indicates on which side the vehicle is in w.r.t. the path.

Conclusions

We have implemented a system that supports simple autonomous driving that is able to follow waypoints and respond to the state of traffic lights. This system comprises three sub-systems, namely Perception, Planning, and Control. The Perception sub-system perceives the world through camera images from which it detects traffic light state via a detection neural network. The Single Shot Multibox Detector architecture with Inception V2 as its feature extractor is chosen due to its reasonable accuracy and high efficiency as demanded by real-time applications. The stop line information is then determined using the traffic light state, and passed to the Planning sub-system. The Planning sub-system publishes the coming 30 waypoints that the vehicle should follow to */final_waypoints*, with the target velocities set to either the speed limit, or speed which facilitates a graceful deceleration that stops the vehicle right before the stop line if a RED or YELLOW traffic light is detected. The Control sub-system receives waypoints from */final_waypoints* and publishes throttle or brake, together with steer controls to the vehicle to achieve desired behavior. A PID filter is used to assist lane-keeping correction steer, and several low-pass filters are used to smoothen the change between consecutive signals. The optimal values for the PID controller is found to be 0.25, 0.003, 3.0 for k_p, k_i, k_d respectively after intensive experiments. Here is the video of the vehicle in the simulator.

Appendix

ROS Nodes, Topics and Logging

This final Udacity self driving car project designs, debugs and tests Robot Operating System, ROS, to operate the autonomous vehicle. There are two types of final code, code to run on a simulator and code to run in the Udacity Carla autonomous vehicle. After cloning the Udacity github for this project into either Linux or the Udacity Ubuntu Virtual Machine the follow directory structure is shown in figure 8.

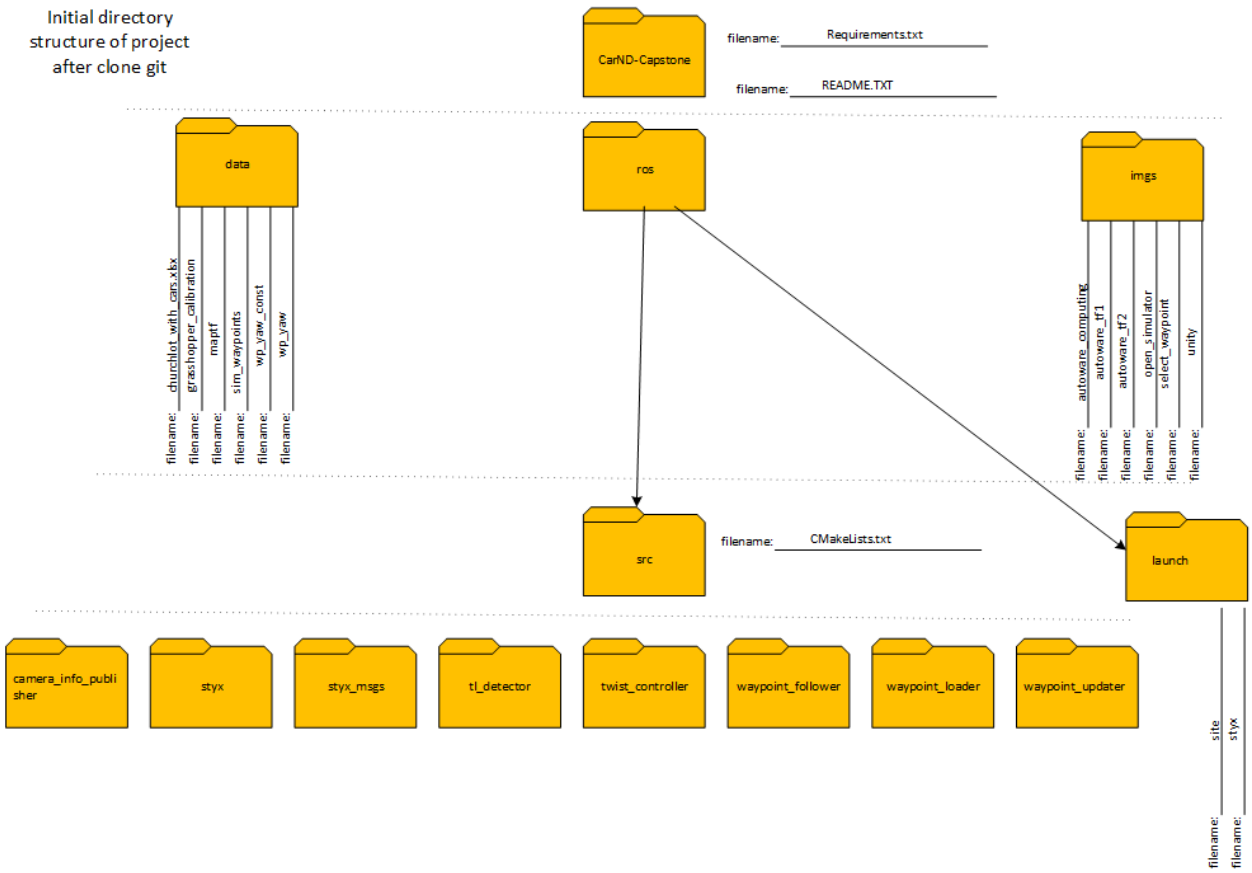

Figure 8 – Directory Structure After Git Clone

The eight ROS packages, camera_info_publisher, styx, styx_msgs, tl_detector, twist_controller, waypoint_follower, waypoint_loader, and waypoint_updater and a CMakeLists file that will be used to build with the source files. First we update the ROS python requirements in the new repo direction with “pip install -r requirements.txt”. Next we build in the ros directory with , “catkin_make”. The corresponding eight package directories are created in the new build directory as shown in figure 9.

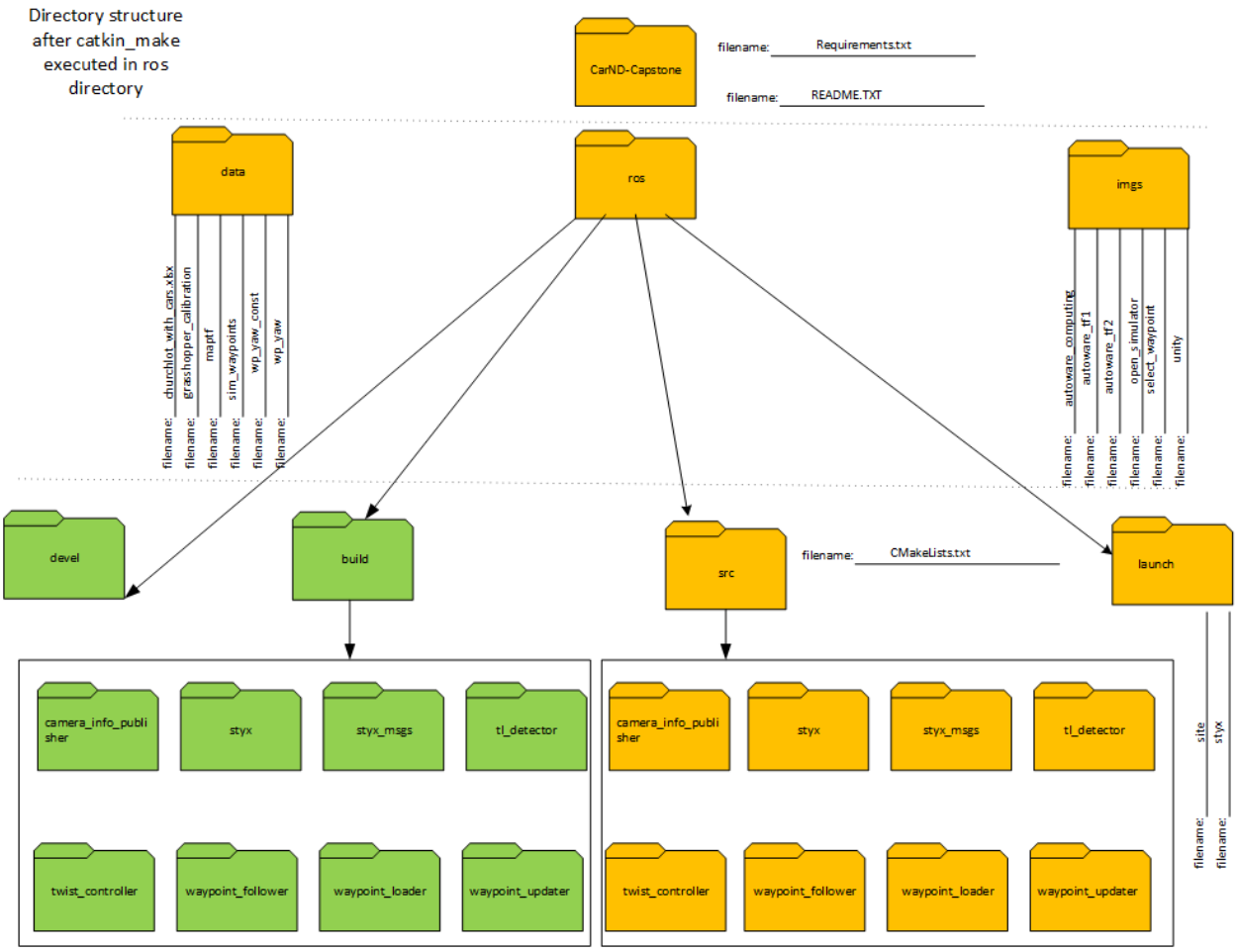

Figure 9 – Directory Structure Build

The newly created devel directory has the setup information to source the ROS environment with this command:

$ source devel/setup.sh

Verify the VM is connected to the simulator as port forwarding is described in the Udacity lecture notes by using the roslaunch command:

$ roslaunch launch/styx.launch

After the connection between the Udacity Virtual Ubuntu Machine and the Udacity Simulator is established a partial waypoint updater that subscribes to /base_waypoints and /current_pose and publishes to /final_waypoints is required to get the car moving in the simulator.

Udacity SDC Capstone Project

Nodes

$ rosnode list

Gives the following ros nodes

Node Name	Function of Node
dbw_node	Drive by wire controller interface to the vehicle.
port pure_pursuit (waypoint_follower)	
rosout	For subscribing, logging, and republishing the messages.
styx_server	Car or simulator
tl_detector	Provides way point information to the Waypoint Updater node
waypoint_loader	Provides the planned road trajectory points
waypoint_updater	Updates trajectory points based on traffic light

dbw_node

Publications: (outputs from the dbw_node)

* /vehicle/steering_cmd [dbw_mkz_msgs/SteeringCmd]

* /vehicle/brake_cmd [dbw_mkz_msgs/BrakeCmd]

* /rosout [rosgraph_msgs/Log] – logs are not shown in the overall drawing.

* /vehicle/throttle_cmd [dbw_mkz_msgs/ThrottleCmd]

Subscriptions: (Inputs to the dbw_node)

* /final_waypoints [styx_msgs/Lane]

* /current_pose [geometry_msgs/PoseStamped]

* /current_velocity [geometry_msgs/TwistStamped]

* /twist_cmd [geometry_msgs/TwistStamped]

* /vehicle/dbw_enabled [std_msgs/Bool]

Services: (not shown for logging purposes)

* /dbw_node/set_logger_level

* /dbw_node/get_loggers

Pid: 3165

Connections:

* topic: /vehicle/throttle_cmd
    * to: /styx_server node
    * direction: outbound
    * transport: TCPROS
* topic: /rosout - used for logging not shown in overall drawing
    * to: /rosout node
    * direction: outbound
    * transport: TCPROS
* topic: /vehicle/steering_cmd
    * to: /styx_server node
    * direction: outbound
    * transport: TCPROS
* topic: /vehicle/brake_cmd
    * to: /styx_server node
    * direction: outbound
    * transport: TCPROS
* topic: /current_velocity
    * to: /styx_server node
    * direction: inbound
    * transport: TCPROS
* topic: /twist_cmd
    * to: /pure_pursuit node
    * direction: inbound
    * transport: TCPROS

* topic: /final_waypoints

    * to: /waypoint_updater node

    * direction: inbound

    * transport: TCPROS

* topic: /current_pose

    * to: /styx_server node

    * direction: inbound

    * transport: TCPROS

* topic: /vehicle/dbw_enabled

    * to: /styx_server node

    * direction: inbound

    * transport: TCPROS

Node pure_pursuit

Publications:

* /rosout [rosgraph_msgs/Log]

* /twist_cmd [geometry_msgs/TwistStamped]

Subscriptions:

* /final_waypoints [styx_msgs/Lane]

* /current_pose [geometry_msgs/PoseStamped]

* /current_velocity [geometry_msgs/TwistStamped]

Services: (not shown for logging purposes)

* /pure_pursuit/set_logger_level

* /pure_pursuit/get_loggers

Pid: 3169

Connections:

* topic: /rosout

    * to: /rosout node

    * direction: outbound

    * transport: TCPROS

* topic: /twist_cmd

    * to: /dbw_node

    * direction: outbound

    * transport: TCPROS

* topic: /final_waypoints node

    * to: /waypoint_updater

    * direction: inbound

    * transport: TCPROS

* topic: /current_pose

    * to: /styx_server

    * direction: inbound

    * transport: TCPROS

* topic: /current_velocity

    * to: /styx_server node

    * direction: inbound

    * transport: TCPROS

Node rosout

Publications:

* /rosout_agg [rosgraph_msgs/Log]

Subscriptions:

* /rosout [rosgraph_msgs/Log]

Services: (not shown for logging purposes)

* /rosout/get_loggers

* /rosout/set_logger_level

Pid: 3157

Connections:

* topic: /rosout

    * to: /pure_pursuit node

    * direction: inbound

    * transport: TCPROS

* topic: /rosout

    * to: /waypoint_updater

    * direction: inbound

    * transport: TCPROS

* topic: /rosout

    * to: /waypoint_loader

    * direction: inbound

    * transport: TCPROS

* topic: /rosout

    * to: /dbw_node

    * direction: inbound

    * transport: TCPROS

* topic: /rosout

    * to: /styx_server

    * direction: inbound

    * transport: TCPROS

* topic: /rosout

    * to: /tl_detector node

    * direction: inbound

      * transport: TCPROS

Node styx_server

Publications:

* /vehicle/obstacle_points [sensor_msgs/PointCloud2] - not covered in this document

* /vehicle/lidar [sensor_msgs/PointCloud2] - not covered in this document

* /vehicle/dbw_enabled [std_msgs/Bool]

* /rosout [rosgraph_msgs/Log] – used for logging

* /vehicle/steering_report [dbw_mkz_msgs/SteeringReport]

* /current_pose [geometry_msgs/PoseStamped]

* /current_velocity [geometry_msgs/TwistStamped]

* /vehicle/obstacle [geometry_msgs/PoseStamped] - not covered in this document

* /vehicle/traffic_lights [styx_msgs/TrafficLightArray]

* /vehicle/brake_report [std_msgs/Float32]

* /tf [tf2_msgs/TFMessage] - not described in this document

* /image_color [sensor_msgs/Image]

* /vehicle/throttle_report [std_msgs/Float32]

Subscriptions:

* /vehicle/steering_cmd [dbw_mkz_msgs/SteeringCmd]

* /vehicle/brake_cmd [dbw_mkz_msgs/BrakeCmd]

* /vehicle/throttle_cmd [dbw_mkz_msgs/ThrottleCmd]

Services:

* /styx_server/set_logger_level

* /styx_server/get_loggers

Pid: 3160

Connections:

* topic: /vehicle/dbw_enabled
    * to: /dbw_node
    * direction: outbound
    * transport: TCPROS
* topic: /rosout
    * to: /rosout
    * direction: outbound
    * transport: TCPROS
* topic: /current_velocity
    * to: /dbw_node
    * direction: outbound
    * transport: TCPROS
* topic: /current_velocity
    * to: /pure_pursuit
    * direction: outbound
    * transport: TCPROS
* topic: /vehicle/traffic_lights
    * to: /tl_detector
    * direction: outbound
    * transport: TCPROS
* topic: /current_pose
    * to: /waypoint_updater

        * direction: outbound

        * transport: TCPROS

* topic: /current_pose

        * to: /dbw_node

        * direction: outbound

        * transport: TCPROS

* topic: /current_pose

        * to: /pure_pursuit

        * direction: outbound

        * transport: TCPROS

* topic: /current_pose

        * to: /tl_detector

        * direction: outbound

        * transport: TCPROS

* topic: /image_color

        * to: /tl_detector

        * direction: outbound

        * transport: TCPROS

* topic: /vehicle/throttle_cmd

        * to: /dbw_node (http://denise-HP-Compaq-dc5800-Small-Form-Factor:36485/)

        * direction: inbound

        * transport: TCPROS

* topic: /vehicle/steering_cmd

        * to: /dbw_node (http://denise-HP-Compaq-dc5800-Small-Form-Factor:36485/)

        * direction: inbound

* transport: TCPROS

* topic: /vehicle/brake_cmd

* to: /dbw_node (http://denise-HP-Compaq-dc5800-Small-Form-Factor:36485/)

* direction: inbound

* transport: TCPROS

Node tl_detector

Publications:

* /rosout [rosgraph_msgs/Log]

* /traffic_waypoint [std_msgs/Int32]

Subscriptions:

* /current_pose [geometry_msgs/PoseStamped]

* /vehicle/traffic_lights [styx_msgs/TrafficLightArray]

* /image_color [sensor_msgs/Image]

Services:

* /tl_detector/set_logger_level

* /tl_detector/get_loggers

Pid: 3171

Connections:

* topic: /rosout

* to: /rosout

* direction: outbound

* transport: TCPROS

* topic: /traffic_waypoint

* to: /waypoint_updater

* direction: outbound

* transport: TCPROS

* topic: /vehicle/traffic_lights

* to: /styx_server (http://denise-HP-Compaq-dc5800-Small-Form-Factor:33425/)

* direction: inbound

* transport: TCPROS

* topic: /image_color

* to: /styx_server

* direction: inbound

* transport: TCPROS

* topic: /current_pose

* to: /styx_server

* direction: inbound

* transport: TCPROS

Node waypoint_loader

Publications:

* /rosout [rosgraph_msgs/Log]

* /base_waypoints [styx_msgs/Lane]

Subscriptions:

None

Services:

* /waypoint_loader/set_logger_level

* /waypoint_loader/get_loggers

Pid: 3166

Connections:

* topic: /rosout

* to: /rosout

* direction: outbound

* transport: TCPROS

Node waypoint_updater

Publications:

* /final_waypoints [styx_msgs/Lane]

* /rosout [rosgraph_msgs/Log]

Subscriptions:

* /current_pose [geometry_msgs/PoseStamped]

* /traffic_waypoint [std_msgs/Int32]

Services:

* /waypoint_updater/set_logger_level

* /waypoint_updater/get_loggers

Pid: 3170

Connections:

* topic: /final_waypoints

* to: /pure_pursuit

* direction: outbound

* transport: TCPROS

* topic: /final_waypoints

* to: /dbw_node

* direction: outbound

* transport: TCPROS

* topic: /rosout

* to: /rosout

* direction: outbound

* transport: TCPROS

* topic: /traffic_waypoint

* to: /tl_detector

* direction: inbound

* transport: TCPROS

* topic: /current_pose

* to: /styx_server

* direction: inbound

* transport: TCPROS

Topics

The topics are listed using the following ros command:

$ rostopics list #Red text indicates we are not using in this code release

base_waypoints

Type: styx_msgs/Lane

Publishers: waypoint_loader

Subscribers: None

current_pose

Type: geometry_msgs/PoseStamped

Publishers: styx_server

Subscribers: pure_pursuit

waypoint_updater

dbw_node

tl_detector

current_velocity

Type: geometry_msgs/TwistStamped

Publishers: styx_server

Subscribers: pure_pursuit

dbw_node

final_waypoints

Type: styx_msgs/Lane

Publishers: waypoint_updater

Subscribers: pure_pursuit

dbw_node

image_color

Type: sensor_msgs/Image

Publishers: styx_server

Subscribers: tl_detector

Type: rosgraph_msgs/Log

Publishers: pure_pursuit

waypoint_updater

waypoint_loader

dbw_node

styx_server

tl_detector

Subscribers: rosout

rosout_agg

Type: rosgraph_msgs/Log

Publishers: rosout

Subscribers: None

tf

Type: tf2_msgs/TFMessage

Publishers: styx_server

Subscribers: None

traffic_waypoint

Type: std_msgs/Int32

Publishers: tl_detector

Subscribers: waypoint_updater

twist_cmd

Type: geometry_msgs/TwistStamped

Publishers: pure_pursuit

Subscribers: dbw_node

vehicle/brake_cmd

Type: dbw_mkz_msgs/BrakeCmd

Publishers: dbw_node

Subscribers: styx_server

vehicle/dbw_enabled

Type: std_msgs/Bool

Publishers: styx_server

Subscribers: dbw_node

vehicle/lidar

Type: sensor_msgs/PointCloud2

Publishers: styx_server

Subscribers: None

vehicle/obstacle

Type: geometry_msgs/PoseStamped

Publishers: styx_server

Subscribers: None

vehicle/obstacle_points

Type: sensor_msgs/PointCloud2

Publishers: styx_server

Subscribers: None

vehicle/steering_cmd

Type: dbw_mkz_msgs/SteeringCmd

Publishers: dbw_node

Subscribers: styx_server

vehicle/throttle_cmd

Type: dbw_mkz_msgs/ThrottleCmd

Publishers: dbw_node

Subscribers: styx_server

vehicle/throttle_report

Type: std_msgs/Float32

Publishers: styx_server

Subscribers: None

vehicle/traffic_lights

Type: styx_msgs/TrafficLightArray

Publishers: styx_server

Subscribers: tl_detector

Logging

$ rosservice list

1. /dbw_node/get_loggers
2. /dbw_node/set_logger_level
3. /pure_pursuit/get_loggers
4. /pure_pursuit/set_logger_level
5. /rosout/get_loggers
6. /rosout/set_logger_level
7. /styx_server/get_loggers
8. /styx_server/set_logger_level
9. /tl_detector/get_loggers
10. /tl_detector/set_logger_level
11. /tl_detector/tf_frames
12. /waypoint_loader/get_loggers
13. /waypoint_loader/set_logger_level
14. /waypoint_updater/get_loggers

15. /waypoint_updater/set_logger_level