UDACITY

‹ Return to Classroom

DISCUSS ON STUDENT HUB

Programming a Real Self-Driving Car

审阅

代码审阅 5

HISTORY

▼ ros/src/waypoint_updater/waypoint_updater.py 4

```
1  #!/usr/bin/env python
2
3  import rospy
4  from geometry_msgs.msg import PoseStamped
5  from styx_msgs.msg import Lane, Waypoint
6  from scipy.spatial import KDTree
7  import numpy as np
8  from std_msgs.msg import Int32
9
10 import math
11
12 '''
13 This node will publish waypoints from the car's current position to some `x` distance ahe
14
15 As mentioned in the doc, you should ideally first implement a version which does not care
16 about traffic lights or obstacles.
17
18 Once you have created dbw_node, you will update this node to use the status of traffic li
19
20 Please note that our simulator also provides the exact location of traffic lights and the
21 current status in `/vehicle/traffic_lights` message. You can use this message to build th
22 as well as to verify your TL classifier.
23
24 TODO (for Yousuf and Aaron): Stopline location for each traffic light.
25 '''
26
27 LOOKAHEAD_WPS = 100 # Number of waypoints we will publish. You can change this number
28 MAX_DECEL = 1.25
29
```

```
30
31 class WaypointUpdater(object):
32     def __init__(self):
33         rospy.init_node('waypoint_updater')
34
35         rospy.Subscriber('/current_pose', PoseStamped, self.pose_cb)
36         rospy.Subscriber('/base_waypoints', Lane, self.waypoints_cb)
37
38         # TODO: Add a subscriber for /traffic_waypoint and /obstacle_waypoint below
39         rospy.Subscriber('/traffic_waypoint', Int32, self.traffic_cb)
40
41         self.final_waypoints_pub = rospy.Publisher('final_waypoints', Lane, queue_size=1)
42
43         # TODO: Add other member variables you need below
44         self.pose = None
45         self.base_waypoints = None
46         self.waypoints_2d = None
47         self.waypoint_tree = None
48         self.stopline_wp_idx = -1
49
50         self.loop()
51
52     def loop(self):
53         rate = rospy.Rate(50)
54         while not rospy.is_shutdown():
55             if self.pose and self.base_waypoints:
56                 # get closest waypoint
57                 closest_waypoint_idx = self.get_closest_waypoint_idx()
58                 self.publish_waypoints()
59             rate.sleep()
60
61     def get_closest_waypoint_idx(self):
62         x = self.pose.pose.position.x
63         y = self.pose.pose.position.y
64         closest_idx = self.waypoint_tree.query([x,y], 1)[1]
65
66         # check if closest is ahead or behind vehicle
67         closest_coord = self.waypoints_2d[closest_idx]
68         prev_coord = self.waypoints_2d[closest_idx-1]
69
70         # equation for hyperplane through closest_coords (vector)
71         cl_vect = np.array(closest_coord)
72         prev_vect = np.array(prev_coord)
73         pos_vect = np.array([x,y])
74
75         val = np.dot(cl_vect-prev_vect, pos_vect-cl_vect)
```

棒极了

Awesome, confirming if the closest waypoints are indeed in the front of the car.

```
76         # if the waypoint is behind the car(val>0), take the next one
77         if val>0:
78             closest_idx = (closest_idx + 1) %len(self.waypoints_2d)
79         return closest_idx
80
81     def publish_waypoints(self):
82         final_lane = self.generate_lane()
83         self.final_waypoints_pub.publish(final_lane)
84
85     def generate_lane(self):
86         lane = Lane()
```

```
        closest_idx = self.get_closest_waypoint_idx()
        farthest_idx = closest_idx + LOOKAHEAD_WPS
        base_waypoints = self.base_waypoints.waypoints[closest_idx:farthest_idx]

        if self.stopline_wp_idx == -1 or (self.stopline_wp_idx >= farthest_idx):
            lane.waypoints = base_waypoints
        else:
            # slow down the vehicle when encounter the red light until stop
            lane.waypoints = self.decelerate_waypoints(base_waypoints, closest_idx)
```

棒极了

Well done checking if the traffic light is not within the farthest waypoint and calling `decelerate_wayp`

```

        return lane

    def decelerate_waypoints(self, waypoints, closest_idx):
        temp = []
        for i, wp in enumerate(waypoints):
            p = Waypoint()
            p.pose = wp.pose

            # two waypoints back from line so front part of car stops at stopline
            stop_idx = max(self.stopline_wp_idx - closest_idx -4, 0)
            dist = self.distance(waypoints, i, stop_idx)
            # change the velocity according the distance to the red light
            vel = math.sqrt(2 * MAX_DECEL * dist)
```

棒极了

The car slows down successfully based on the distance with the stop waypoint and also stops 4 wayp well implemented. 👏🏽

```
            if vel < 1.:
                vel = 0
            p.twist.twist.linear.x = min(vel, wp.twist.twist.linear.x)
            temp.append(p)
        return temp


    def pose_cb(self, msg):
        # TODO: Implement
        self.pose = msg

    def waypoints_cb(self, waypoints):
        # TODO: Implement
        self.base_waypoints = waypoints
        if not self.waypoints_2d:
            self.waypoints_2d = [[waypoint.pose.pose.position.x, waypoint.pose.pose.posit
            self.waypoint_tree = KDTree(self.waypoints_2d)
```

棒极了

Good job using KDTree to get the closest waypoints from the car efficiently.

```

    def traffic_cb(self, msg):
```

```
130         # TODO: Callback for /traffic_waypoint message. Implement
131         self.stopline_wp_idx = msg.data
132
133     def obstacle_cb(self, msg):
134         # TODO: Callback for /obstacle_waypoint message. We will implement it later
135         pass
136
137     def get_waypoint_velocity(self, waypoint):
138         return waypoint.twist.twist.linear.x
139
140     def set_waypoint_velocity(self, waypoints, waypoint, velocity):
141         waypoints[waypoint].twist.twist.linear.x = velocity
142
143     def distance(self, waypoints, wp1, wp2):
144         dist = 0
145         dl = lambda a, b: math.sqrt((a.x-b.x)**2 + (a.y-b.y)**2  + (a.z-b.z)**2)
146         for i in range(wp1, wp2+1):
147             dist += dl(waypoints[wp1].pose.pose.position, waypoints[i].pose.pose.position)
148             wp1 = i
149         return dist
150
151
152 if __name__ == '__main__':
153     try:
154         WaypointUpdater()
155     except rospy.ROSInterruptException:
156         rospy.logerr('Could not start waypoint updater node.')
157
```

▶ ros/src/twist_controller/twist_controller.py 1

▶ ros/src/waypoint_updater/package.xml

▶ ros/src/waypoint_updater/CMakeLists.txt

▶ ros/src/waypoint_loader/waypoint_loader.py

▶ ros/src/waypoint_loader/package.xml

▶ ros/src/waypoint_loader/CMakeLists.txt

▶ ros/src/waypoint_follower/src/pure_pursuit_core.cpp

▶ ros/src/waypoint_follower/src/pure_pursuit.cpp

▶ ros/src/waypoint_follower/package.xml

▶ ros/src/waypoint_follower/lib/libwaypoint_follower.cpp

▶ ros/src/waypoint_follower/include/pure_pursuit_core.h

- ros/src/waypoint_follower/include/libwaypoint_follower.h
- ros/src/waypoint_follower/CMakeLists.txt
- ros/src/twist_controller/yaw_controller.py
- ros/src/twist_controller/pid.py
- ros/src/twist_controller/package.xml
- ros/src/twist_controller/lowpass.py
- ros/src/twist_controller/dbw_test.py
- ros/src/twist_controller/dbw_node.py
- ros/src/twist_controller/CMakeLists.txt
- ros/src/tl_detector/tl_detector.py
- ros/src/tl_detector/site_traffic_light_config.yaml
- ros/src/tl_detector/sim_traffic_light_config.yaml
- ros/src/tl_detector/package.xml
- ros/src/tl_detector/light_publisher.py
- ros/src/tl_detector/light_classification/tl_classifier.py
- ros/src/tl_detector/light_classification/__init__.py
- ros/src/tl_detector/CMakeLists.txt
- ros/src/styx_msgs/package.xml
- ros/src/styx_msgs/CMakeLists.txt
- ros/src/styx/server.py
- ros/src/styx/package.xml
- ros/src/styx/conf.py

- ros/src/styx/bridge.py
- ros/src/styx/CMakeLists.txt
- ros/src/dbw_mkz_msgs/package.xml
- ros/src/dbw_mkz_msgs/bmr/README.md
- ros/src/dbw_mkz_msgs/CMakeLists.txt
- ros/src/dbw_mkz_msgs/CHANGELOG.rst
- ros/src/camera_info_publisher/yaml_to_camera_info_publisher.py
- ros/src/camera_info_publisher/package.xml
- ros/src/camera_info_publisher/CMakeLists.txt
- requirements.txt
- data/grasshopper_calibration.yml
- README.md

返回 PATH

给这次审阅打分

开始