Introduction to Robot Operating Systems (ROS)

Autonomous Driving Project

Team 8

Documentation

1.ROS Nodes

- generatePointCloud
- traffic_light_detector
- Move_base_new
- Map_expander
- Octomap_server
- waypoint_publisher
- smach
- controller_node

2. Description of each node and its functionality

2.1 generatePointCloud
The node named “generatePointCloud_node” uses the ROS image transport interface to subscribe to image topics from specified RGB and depth cameras. The received RGB image is used to extract color information, while the depth image is used to generate the three-dimensional coordinates of each pixel. By processing these images, the node calculates the 3D coordinates and corresponding color values for each point, combining them into a point cloud. This point cloud is first organized in the “`PointCloud`” format and then converted to the “`PointCloud2`” format, before being published to two different ROS topics for use or further processing by other nodes.

2.2 Traffic light detector
“traffic_light_detector_node” is a node that detects traffic lights by subscribing to image data streams from RGB and semantic cameras. Initially, the node uses “`cv_bridge`” to convert ROS image messages into OpenCV image format. It processes two types of images: one is the RGB image used for final color detection, and the other is the semantic image used to identify the location of traffic lights by specific colors.

Upon receiving new image data, the node first checks if these images are complete. If one of the images is missing, it defaults to publishing a message indicating a green light, suggesting that if no traffic light is detected, the default action is to proceed. If both images are valid, it uses color thresholds in the semantic image to identify the approximate areas of traffic lights and extracts these areas from the RGB image. These areas are then used to determine the state of the traffic light (red, yellow, or green) through color recognition.

Through OpenCV's color space conversion and thresholding operations, the program can identify and count the number of pixels for each color. Based on the color with the highest pixel count, the node determines the current state of the traffic light and publishes this status through the ROS messaging system.

2.3 Move_base_new

2.3.1 The Octomap node

It was used to model the arbitrary environment, and was used in this project to generate an occupancy grid through which the car has to navigate. Subsequently, the occupancy grid was used to create a costmap. This provides more detailed information than a simple occupancy grid by assigning different costs to different areas based on obstacles, free space, and other considerations.

2.3.2 Global Plan

The Dijkstra algorithm is a graph search algorithm that solves the single-source shortest path problem for a graph with non-negative edge weights. It was used in this project for the global path planning of the environment.

2.3.3 Local Plan

TrajectoryPlannerROS is a local planner implementation in ROS (Robot Operating System) that is part of the base_local_planner package. It is responsible for generating velocity commands to move a robot from its current position to a target position, while avoiding obstacles and adhering to the robot's kinematic constraints. It evaluates multiple possible trajectories and selects the best one based on certain criteria, such as distance to obstacles, alignment with the global plan, and velocity. The TrajectoryPlannerROS was used in this project for the local planning of the environment.

2.4 Map_expander

It can expand the size of OccupancyGrid by filling the whole map with -1 in a none-detected area, so that the move_base can plan the path to the point out of the map.

2.5 Waypoint_publisher

The waypoint_publisher node was used to publish a series of Poses that the vehicle should follow. These waypoints serve as navigation goals for the vehicle and are consumed by other nodes, such as the path planning and control nodes, to guide the vehicle's movement.

2.6 Octomap_server

Generate the 3d-Octomap and the projected 2d OccupancyGrid as the map-input of the move_base_new.

2.7 State machine

This node implements a CAR-based **Finite State Machine** (FSM) to control a car's behavior at traffic lights. The FSM consists of two states: "**DRIVE**" and "**STOP**", which govern the car's movement based on traffic light signals. The solution is designed with clarity, modularity, and ROS best practices in mind.

2.7.1 State Machine Design

- The state machine is constructed using the ***smach*** library, with states inheriting from ***smach.State***.
- The **DriveState** and **StopState** classes manage the car's behavior in response to traffic light signals.

2.7.2 Traffic Light Detection

- A **TrafficLight** class subscribes to the ***traffic_light_state*** topic, processing messages to determine the current signal color.
- The signal's stability is ensured by counting consecutive detections, preventing erroneous state changes.
- Coordinate with the **Traffic Light Detector** to control when the vehicle stops and starts.

2.7.3 Velocity Control

- Global variables ***v*** and ***omega*** represent the linear and angular velocities.
- These velocities are published to the ***target_linear_velocity*** and ***target_angular_velocity*** topics, respectively, to control the car's movement.
- A callback function ***target_twist_callback*** updates these velocities based on incoming **Twist** messages from the ***cmd_vel*** topic.

2.7.4 Robust Communication

- The system utilizes ROS publishers and subscribers to handle inter-node communication effectively.
- The ***target_v_pub*** and ***target_omega_pub*** publishers ensure that the car's velocity commands are continuously updated based on the FSM state..

2.7.5 ROS Integration

- The node adheres to ROS conventions, including node initialization, topic subscription/publishing, and using the ***rospy*** library for managing node lifecycle and communication.
- The state machine can be visualized using the ***smach_ros.IntrospectionServer*** for debugging and monitoring purposes.

2.7.6 Handle unexpected situations

- E.G. The vehicle unexpectedly missed the detection range of the traffic light when it is stopping.
- The solution is: Maintain the **Drive State** when the **Traffic Light Detector** does not detect a signal or detects an unknown signal.

2.8 Controller_node
The controller node controls the acceleration, deceleration, and turning of the car, enabling it to reach the target linear and angular velocities.

2.8.1 Subscribed Topics

- Current state from Unity
- Target state from the State_machine_node.

2.8.2 Feedforward Control

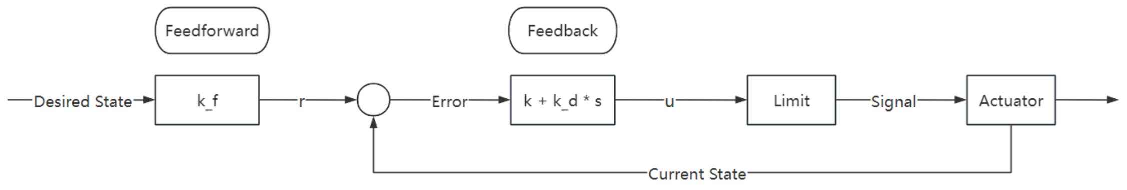
The feedforward-control is used to compensate for steady-state error. It can only reach 90% of the desired state without feedforward control.

2.8.3 Feedback Control

PD control is used to obtain the control signal, allowing the car to reach the target linear and angular velocities.

2.8.4 Limitations

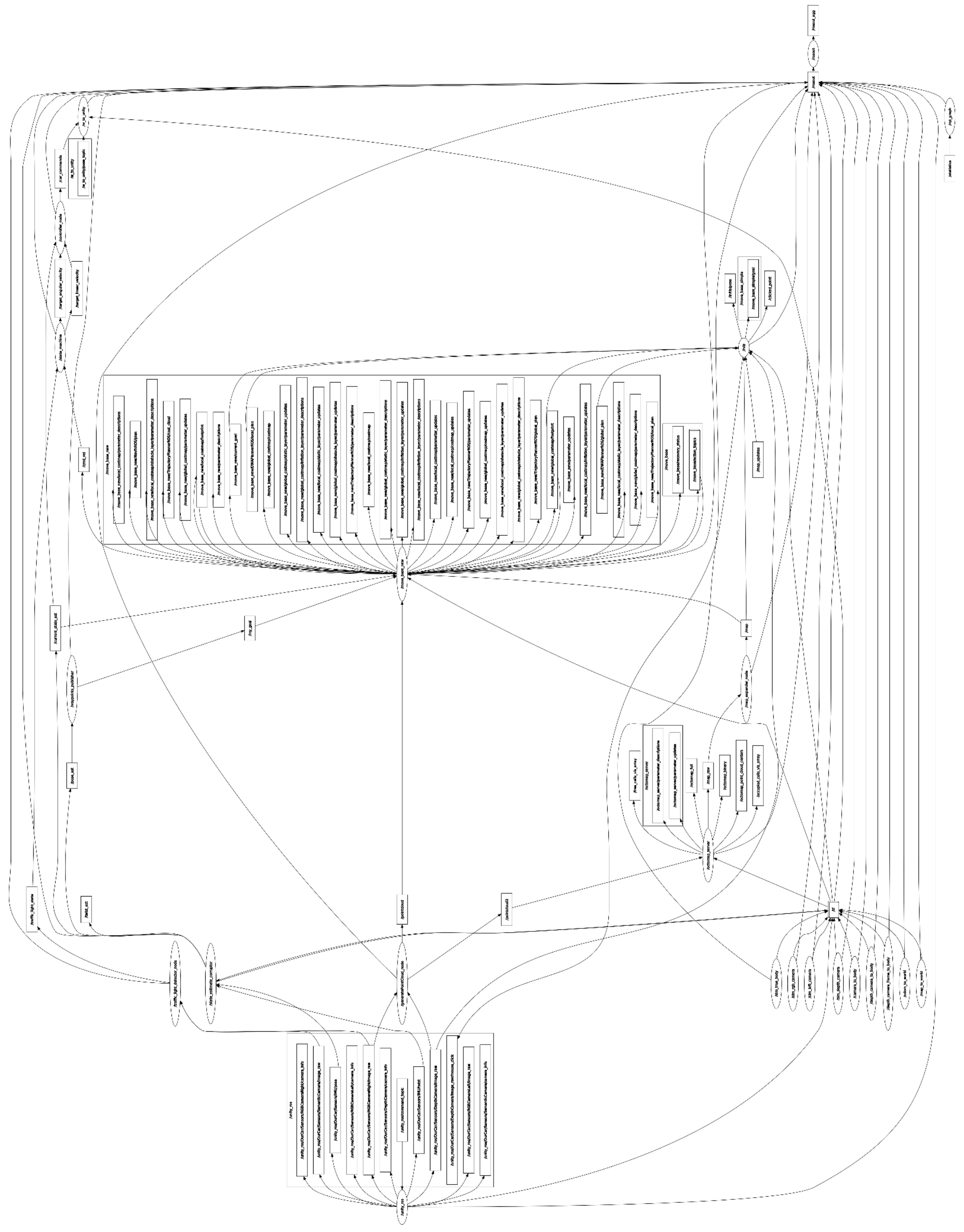
The signal is constrained to ensure it remains within a proper range.



Team member	Corresponding Responsible Nodes
Yijia Qian	traffic_light_detector_node, generatePointCloud_node
Ziou Hu	Move_base_new, Map_expander, waypoint publisher, Octomap_server
Penglin Li	State Machine, Waypoint set, Traffic-Light detect optimize
Zibo Xie	Controller_node, Documentation
Joshua Man	Octomap_server, Simulation, transform and launch

3. ROS graph

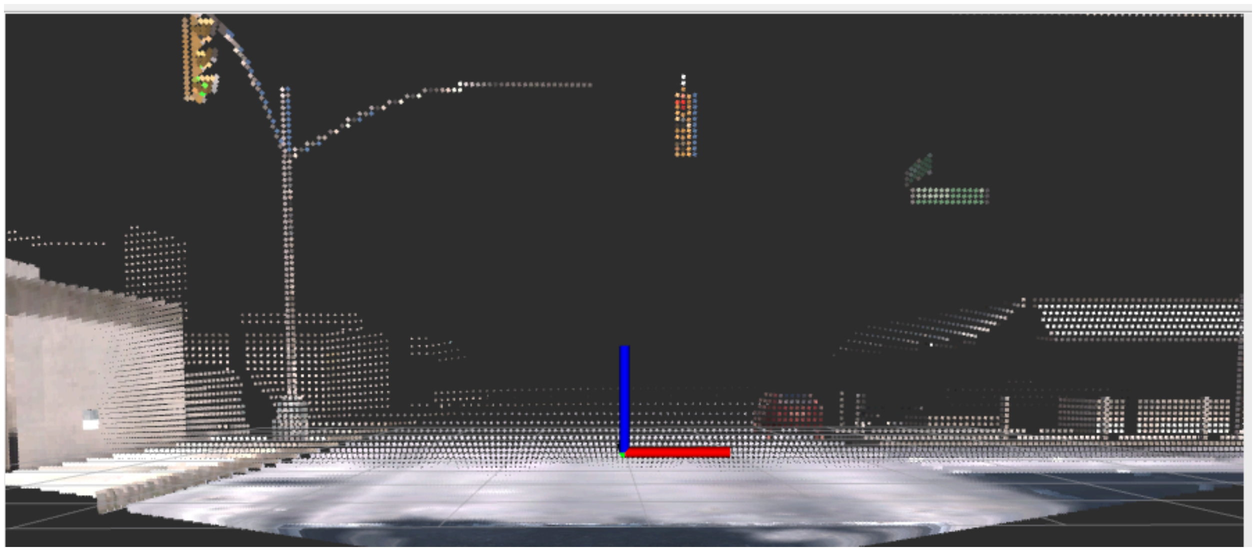
A clearer version of the ros_graph highlighting the active nodes can be found on the main branch in the repository.



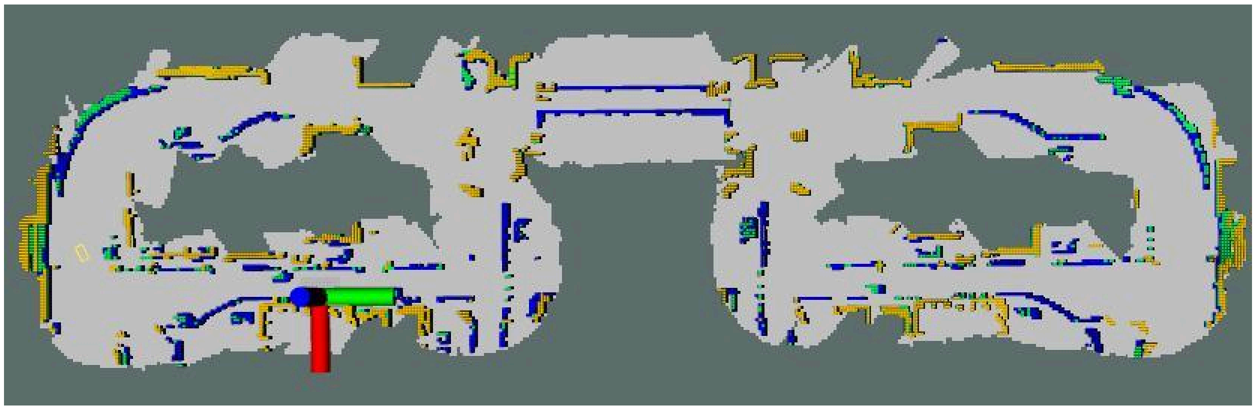
4. Figures and plots of results

4.1 Mapping

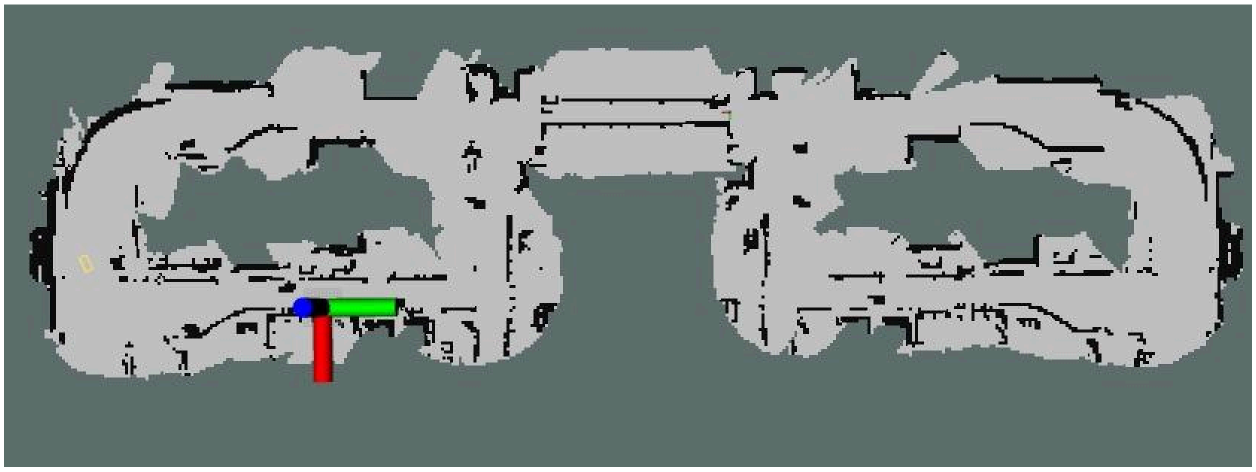
4.1.1 Pointcloud



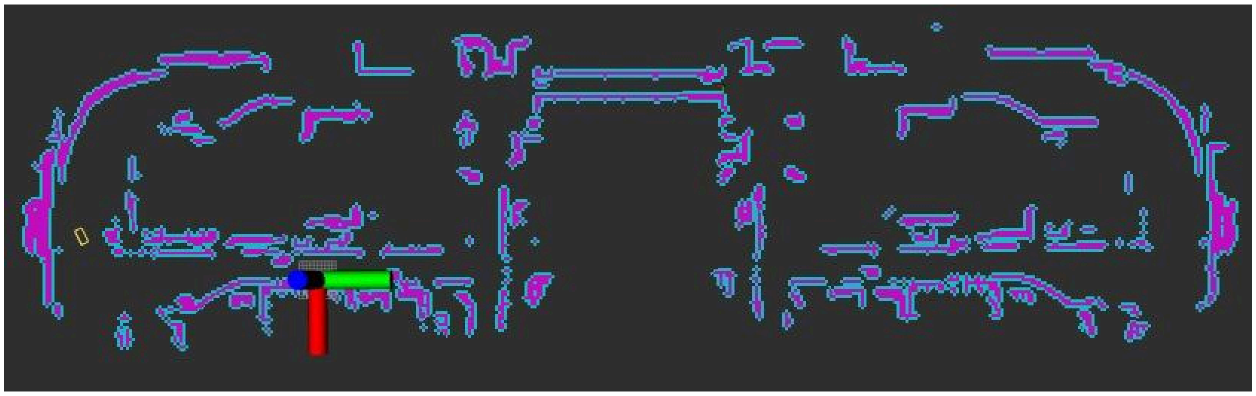
4.1.2 Octomap



4.1.3 Occupancy Grid

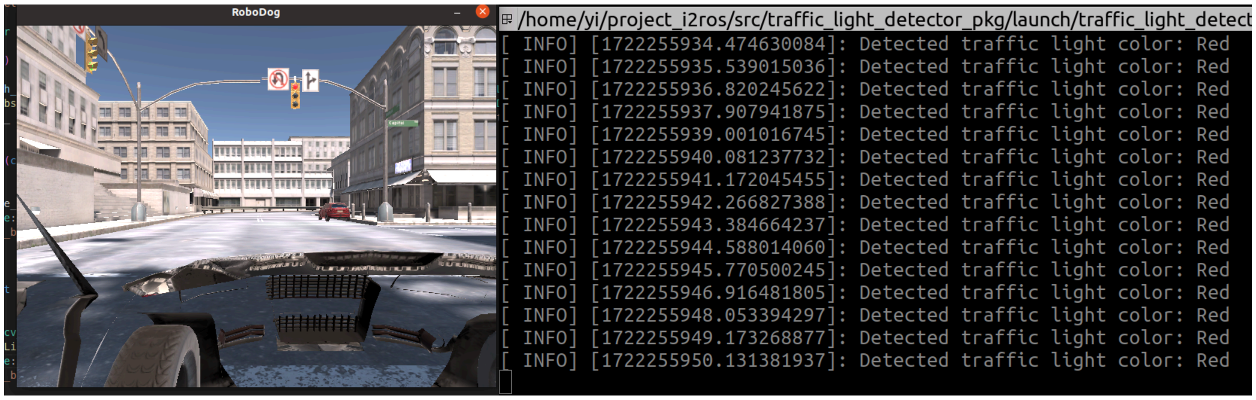


4.1.4 Costmap



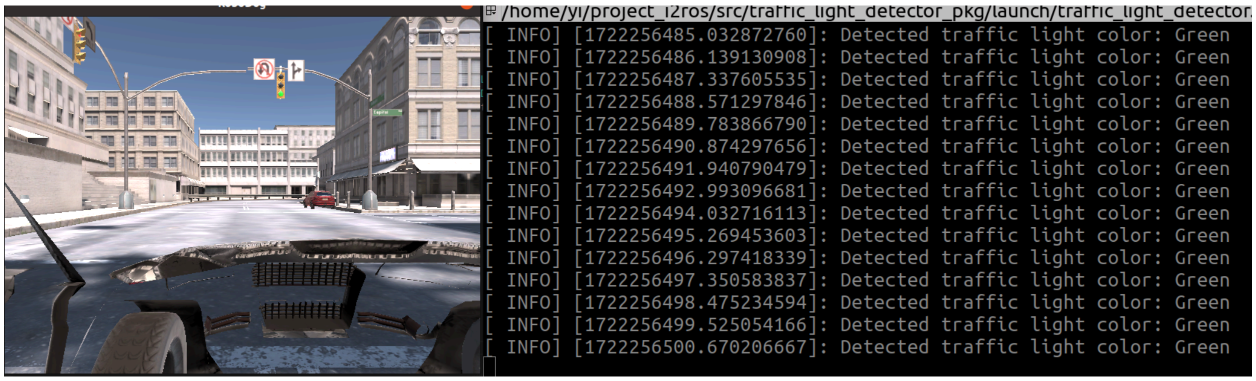
4.2 Traffic light detection

Red light:



```
/home/yi/project_i2ros/src/traffic_light_detector_pkg/launch/traffic_light_detect
[ INFO] [1722255934.474630084]: Detected traffic light color: Red
[ INFO] [1722255935.539015036]: Detected traffic light color: Red
[ INFO] [1722255936.820245622]: Detected traffic light color: Red
[ INFO] [1722255937.907941875]: Detected traffic light color: Red
[ INFO] [1722255939.001016745]: Detected traffic light color: Red
[ INFO] [1722255940.081237732]: Detected traffic light color: Red
[ INFO] [1722255941.172045455]: Detected traffic light color: Red
[ INFO] [1722255942.266827388]: Detected traffic light color: Red
[ INFO] [1722255943.384664237]: Detected traffic light color: Red
[ INFO] [1722255944.588014060]: Detected traffic light color: Red
[ INFO] [1722255945.770500245]: Detected traffic light color: Red
[ INFO] [1722255946.916481805]: Detected traffic light color: Red
[ INFO] [1722255948.053394297]: Detected traffic light color: Red
[ INFO] [1722255949.173268877]: Detected traffic light color: Red
[ INFO] [1722255950.131381937]: Detected traffic light color: Red
```

Green light:



```
/home/yi/project_i2ros/src/traffic_light_detector_pkg/launch/traffic_light_detector.
[ INFO] [1722256485.032872760]: Detected traffic light color: Green
[ INFO] [1722256486.139130908]: Detected traffic light color: Green
[ INFO] [1722256487.337605535]: Detected traffic light color: Green
[ INFO] [1722256488.571297846]: Detected traffic light color: Green
[ INFO] [1722256489.783866790]: Detected traffic light color: Green
[ INFO] [1722256490.874297656]: Detected traffic light color: Green
[ INFO] [1722256491.940790479]: Detected traffic light color: Green
[ INFO] [1722256492.993096681]: Detected traffic light color: Green
[ INFO] [1722256494.032716113]: Detected traffic light color: Green
[ INFO] [1722256495.269453603]: Detected traffic light color: Green
[ INFO] [1722256496.297418339]: Detected traffic light color: Green
[ INFO] [1722256497.350583837]: Detected traffic light color: Green
[ INFO] [1722256498.475234594]: Detected traffic light color: Green
[ INFO] [1722256499.525054166]: Detected traffic light color: Green
[ INFO] [1722256500.670206667]: Detected traffic light color: Green
```

5. Implemented and Unimplemented Features

5.1 Implemented Features

- Successfully working perception pipeline
- working path planning （Partial route completion）
- working trajectory planning (Partial route completion)
- Successfully avoiding other cars
- Successfully stopping/driving at street lights

5.2 Unimplemented Features

Unable to successfully complete the entire route as the turns towards the end of the route require very precise turns, and the interaction between the local plan and the controller is not completely optimised. There are a lot of bugs in the turns, and the green light is too short, and will turn back to red shortly after turning green, resulting in a very small window to move off the line.

6. External code used

- Move_base
- Octomap_server

7.Bibliography

[1] Ros Wiki navigation Tutorials Robot setup:
https://wiki.ros.org/navigation/Tutorials/RobotSetup
[2] Depth_image_proc: https://wiki.ros.org/depth_image_proc