Team 15 Cars: Autonomous Driving Project Report

Introduction to ROS SS23

Prof. Markus Ryll

Autonomous Aerial Systems

TUM School of Engineering and Design

Technical University of Munich

Submitted by:

Jididu Li, Peishi Liu,

Upendra Bevinamara Arun,

Xing Hu, Yuan Zhao

02.08.2023

Table of Contents

Table of Figures

1. Introduction

This project was done as part of the Introduction to ROS course in Summer Semester 2023 at TUM offered at the chair of autonomous aerial systems by Prof. Markus Ryll. A final project on Autonomous driving system was to be developed where in a car must autonomously drive through an urban environment in Unity simulation without leaving the road, crashing into other vehicles, and obeying traffic lights.

2. Overview of the pipeline

The pipeline broadly consists of three sub-packages. Starting with the perception pipeline that includes point cloud generation from the depth camera feed, which is then later used to calculate a 3d-voxel grid, thereby mapping the static environment. In the next step, the voxel grid is projected on a 2D plane to get an occupancy map. Once the occupancy map is obtained, the global path planning is done by setting up the waypoints through which the car should pass through from start to finish. In the local planner, the path between the waypoints is then calculated based on the position, obstacles, and the traffic light. The commands are then sent to the car through the controller package. Here the desired velocity and true velocity is compared to obtain a velocity control of the vehicle.

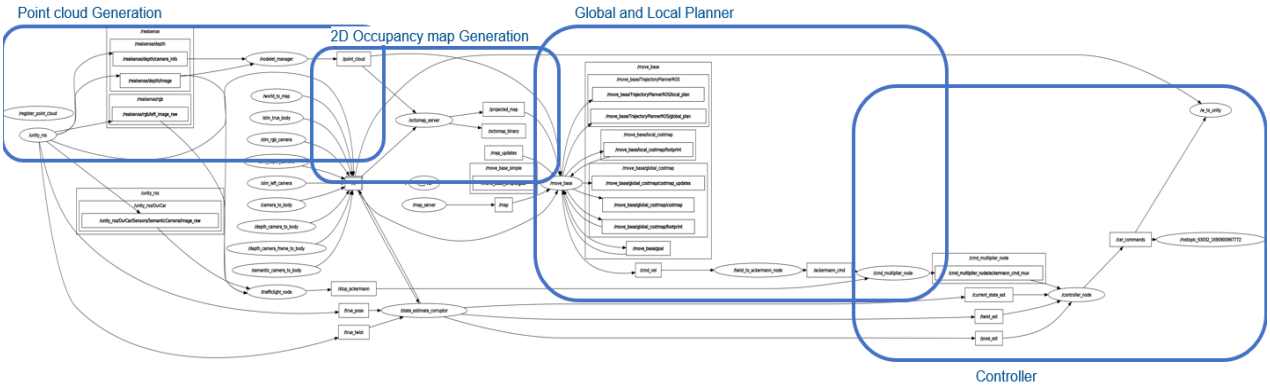

Figure 1: rqt_graph of the pipeline

3. Perception

3.1 Point cloud and 2D occupancy map

The main task of the perception pipeline is to generate a point cloud, 2D occupancy map and subsequently while driving, detect the traffic lights. Initially the coordinate systems had to be correctly setup so that all the sensor readings appear in the correct directions in the rviz visualization and also be helpful in further calculations. Firstly, the package depth_image_proc [1] was used to convert the depth camera info and depth camera feed into the topic '/point_cloud'. This topic is then subscribed by the node 'octomap server' node to get a 2D occupancy map which is under the topic of '/projected map'. This voxel grid to 2D occupancy map was generated with the help of octomap server [2] package. This occupancy grid also called octree-based occupancy grid involves probabilistic occupancy estimation. Parameters in the octomap server such as the sensor model range, hit/miss, ground filter was tuned so that the ground would not be detected as an obstacle and other static obstacles such as roadside pavements, buildings etc. are appropriately modelled.

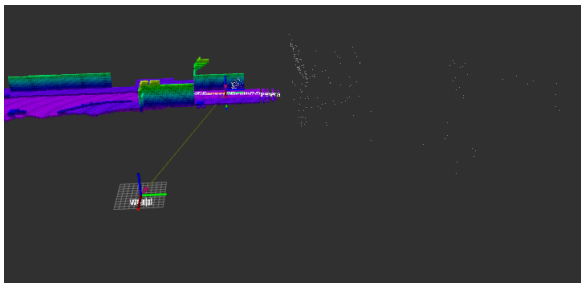

Figure 2: Point cloud and voxel grid generation

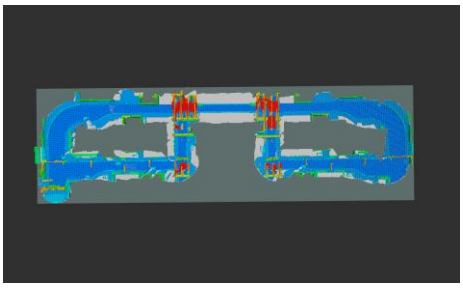

Figure 3: 3D Occupancy grid and projected map

3.2 Perception: Traffic light detection

A node called 'trafficlight_node' was created to detect traffic signals and make the adequate response in an appropriate range. This node subscribes to images from RGB, depth, and semantic cameras, all transformed into the 'cv::Mat' type. Following road construction conventions, only 2/3 of the area on the right-hand side is processed.

The traffic lights' positions in the scene are extracted from the masked semantic image. Simultaneously, the distance and color of the traffic lights are identified in the depth and RGB images, respectively. If the vehicle is close enough to a traffic light and a red signal is detected, the 'trafficlight_node' publishes a message in the format of 'ackermann_msgs' to the '/stop_ackermann' topic. This message contains factors set to zero, effectively stopping the vehicle.

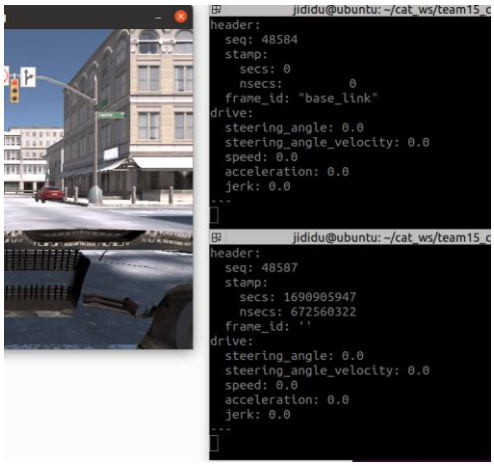

Figure 5: /Car command Velocity is 0 when the traffic light is red

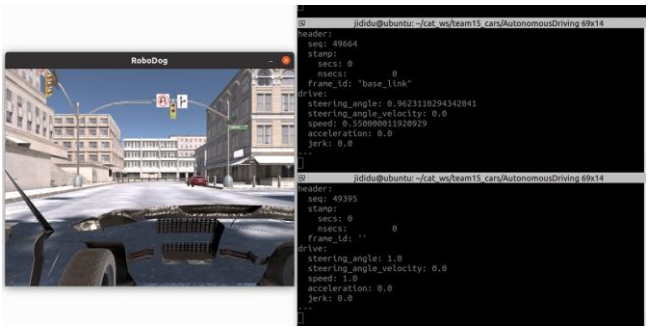

Figure 4: /Car command Velocity is transmitted when traffic light is green

4. Path Planning: Global Planner

The path planning is divided into two parts. The global planning and the local planning. The global planning includes identifying waypoints at certain instances of the road through which the car must pass through. 20 waypoints were chosen based on the number of turns (8turns x 2points) present in the provided environment and the straight road (2points) and the start and finish points (2 points). Additionally, having these many points helps in smoothening the trajectory followed by the car. The backbone of the path planning is 'move base'[3] package. The global planner was initialized such that global planning functions according to the A-star approach. Additionally, a global and cost map was initialized such way that the global costmap is taken from the map which was generated from the perception pipeline and the local cost map considers the point cloud feed to detect if there are any obstacles in the path. The waypoints are sent to the move base package via the '/goal_publisher' node. It reads the list of waypoints from a yaml file and then sends it to move base successively. The local planner then calculates the path between the successive waypoints. The next part describes in detail the local planner and trajectory planning developed.

5. Trajectory Planning

5.1 Local Planner

To generate the desired velocity and steering angle, we utilize the 'teb_local_planner,'[4] a time-optimal planner that supports holonomic robots [5].

The planner calculates the desired command values, primarily tailored for car-like robots using Ackermann geometry. These command values are then processed by the '/twist_to_ackermann_node' and published on the '/Ackermann_cmd' topic.

5.2 Command fusion

The vehicle must adhere to traffic rules and factor in various considerations for safe operation. To achieve this, the '/cmd_multiplier_node' subscribes to information from topics '/stop_ackermann' and '/Ackermann_cmd.' It multiplies the factor obtained from the '/stop_ackermann' topic with the desired velocity generated by the local planner. The resultant message facilitates traffic signal compliance and is published on the '/cmd_multiplier_node/Ackermann_cmd_mux'[6] topic. This flexible design allows for the integration of additional factors, accommodating diverse scenarios and requirements.

Figure 6: Command fusion representation

6. Controller: Controller and Drive

6.1 Velocity and turning angle

Control the speed and steering angle of the car by comparing the difference between the current state (speed, steering angle) and the ideal state. The current state was obtained through the Odometry of the

car, and the ideal state is calculated through the previous part. The speed value and acceleration value calculated by the controller will be transmitted to the power unit of the car, and the minimum and maximum values of each value will be limited, so that the car can run as fast as possible.

6.2 PID Controller

The PID algorithm contains the main information of the past, present, and future in the dynamic control process, and its configuration is almost optimal. Among them, the ratio (P) represents the current information, which plays a role in correcting deviations and makes the process respond quickly. Differentiation (D) has an advanced control function when the signal changes, representing future information. Force the process to proceed at the beginning of the process, reduce overshoot at the end of the process, overcome oscillation, improve system stability, and speed up the transition process of the system. The integral (I) represents the information accumulated in the past, which can eliminate static errors and improve the static characteristics of the system. The proper coordination of these three functions can make the dynamic process fast, stable, and accurate, and obtain good results.

7. Result and challenges

Based on the tests that we conducted by initially giving a single goalpoint to movebase and listening to the rostopic echo of the '/cmd_multiplier_node/Ackermann_cmd_mux' and simultaneously comparing the data from the topic '/car_command', (refer Figure 4 and Figure 5) it was ascertained that traffic light detection was functioning as required. When the car was manually moved with help of keyboard and visualized through rviz, the global and local planner updated the path, thereby concluding that there was a problem with the controller package. Additionally, the controller package and the costmap didn't work as intended in the responsible team member's machine and it consumed a considerable amount of time in debugging the package.

8. Task Responsibility

- Setup of transformations, point cloud generation, occupancy map and tuning – Yuan Zhao and Upendra B A
- Global planner and goal publisher node – Upendra B A
- Local planner, Traffic light detection and command fusion – Jididu Li
- Controller – Xing Hu, Peishi Liu

9. References

[1] Point cloud from Depth Image: http://wiki.ros.org/depth_image_proc (last accessed on 02.08.2023)

[2] Point cloud to Voxel Grid and Octomap: http://wiki.ros.org/octomap_server (last accessed on 02.08.2023)

[3] Move base: http://wiki.ros.org/move_base (last accessed on 02.08.2023)

[4] Teb local planner: http://wiki.ros.org/teb_local_planner (last accessed on 02.08.2023)

[5] Integrated online trajectory planning and optimization in distinctive topologies, Robotics and Autonomous Systems. C. Rösmann; F. Hoffmann; T. Bertram. 2017, Robotics and Autonomous Systems, pp. 74-79.

[6] Command fusion with mux: http://wiki.ros.org/twist_mux (last accessed on 02.08.2023)