Introduction to ROS Project Report Group 16: Autonomous Driving

Wenjie Xie, Lingyue Zhao, Jingyi He, Xinhua Wang, Dian Yu

***Abstract*— This project report presents the development and implementation of an autonomous driving system utilizing the Robot Operating System (ROS). The system focuses on three core components: perception, planning, and control. The perception module employs a depth camera to generate 3D point clouds and a lightweight deep neural network for traffic light detection using RGB image data. The planning module is divided into global and local planning, using Dijkstra's algorithm for global trajectory planning and the Timed Elastic Band (TEB) Local Planner for local path optimization. The control module integrates a state estimator and a PID control pipeline to manage vehicle dynamics and state transitions. Experimental results show that the designed autonomous driving system is capable of safely and effectively controlling the vehicle, demonstrating basic competencies in safe and efficient driving.**

I. Perception

A. Overview

Perception is a critical component in autonomous driving systems, enabling vehicles to understand and interpret their surroundings. In this project, the perception system focuses on building a pipeline for environment recognition and traffic light detection. Utilizing sensor data and advanced image processing techniques, the perception system provides essential information for safe navigation.

In this project, a depth camera provided with the simulator is used that captures depth image data of the environment. Using the `depth_image_proc` package this data can be converted into a 3D point cloud representation. The processed point cloud can be used to create maps in conjunction with the `octomap` package as shown in Fig.1.

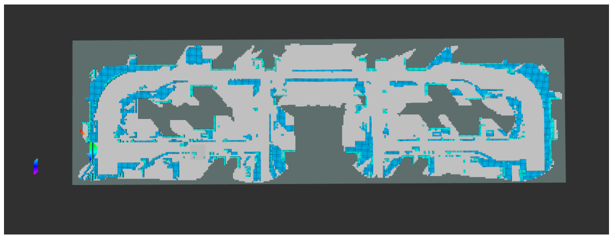


Fig. 1: Mapping Visualization

After this process is complete, self-driving cars can effectively navigate and plan routes based on depth data in a simulated urban environment.

Wenjie Xie: ge63bip@mytum.de
Lingyue Zhao: ge23tal@mytum.de
Jingyi He: holly.he@tum.de
Xinhua Wang: xinhua.wang@tum.de
Dian Yu: go69joc@mytum.de

After completing the basic obstacle avoidance and trajectory planning functions assisted by the depth camera, the perception module of the self-driving car also needs to be able to recognize other necessary visual background information in the road traffic environment, such as the red light condition. In this project, a lightweight deep neural network of independent design structure is used to segment the onboard RGB camera, combine with RGB detection to judge the red light condition in front of the car, establish a complete recognition pipeline and synchronize the results to the control algorithm. This approach avoids the use of semantically segmented cameras in simulation environments, while being closer to real-world autonomous driving algorithms.

B. Point Cloud Generation

Our project employs a depth camera to capture depth image data, which is then processed to generate a 3D point cloud representation of the environment. The `depth_image_proc` package is utilized to convert depth images into point clouds. This conversion is crucial for subsequent perception tasks such as object detection, localization, and obstacle avoidance. The depth camera captures the depth image data and, using the `depth_image_proc` package, transforms this data into a meaningful point cloud representation. As presented in Fig.2, each point in the cloud corresponds to a three-dimensional coordinate in the real-world space, providing an accurate spatial mapping of the environment. The generated point cloud is published to the `/DepthCamera/pointcloud` topic, serving as a fundamental input for further processing and visualization in `RViz`.

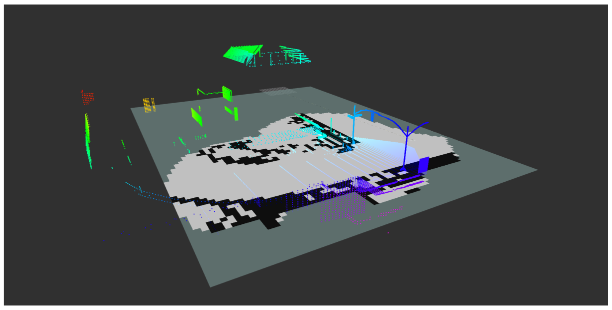


Fig. 2: Generated Point Cloud

C. Traffic Light Detection

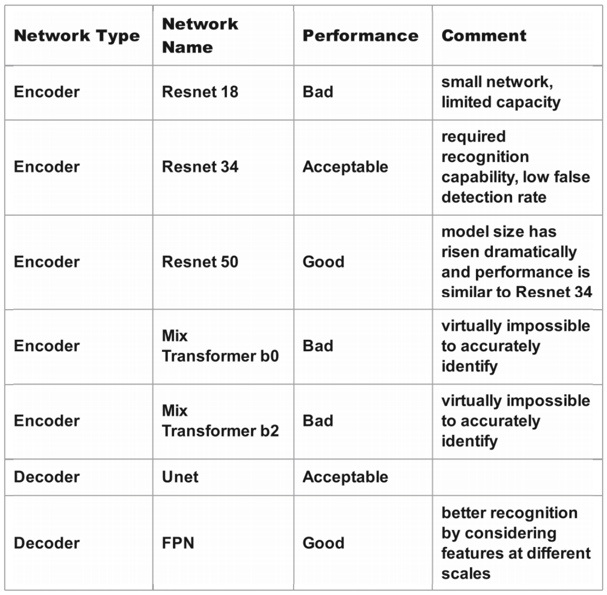
1) Visual Segmentation Network:

a) Network Design: In autonomous driving tasks, the vehicle needs to have the ability to recognize the surrounding environment to assist in driving decisions. In real-world applications, autonomous vehicles often use multi-modal sensor solutions, such as LiDAR + camera or millimeter-wave radar + camera. Radar is often used to perceive the surrounding object environment and generate depth data, while road traffic background information related to color (such as traffic lights, traffic signs, lane markings, special markings, etc.) relies heavily on camera recognition results. Cameras typically provide RGB images that clearly represent color differences, providing essential information for autonomous driving algorithms.

In this project, the expected decision input for the autonomous vehicle is whether there is a red light ahead. This is a typical feature extraction and recognition problem, and deep learning often performs well in this task. Therefore, a segmentation network based on convolutional neural networks is used. In terms of the neural network architecture, a pre-trained network with non-frozen weights is used as the encoder to complete feature extraction, and a self-trained FPN architecture convolutional layer is used as the decoder to complete segmentation. Pre-trained networks (such as ResNet, VGG, etc.) are fully trained on large-scale datasets (such as ImageNet), capable of extracting rich visual features, including edges, textures, and high-level semantic features. This allows the model to quickly adapt and provide high-quality feature representations when handling new tasks. Using a pre-trained encoder greatly reduces training time and data requirements, as these networks have already been trained on large amounts of data and learned general features. FPN builds a feature pyramid, integrating features from different layers, allowing the model to use both high-resolution low-level features and low-resolution high-level features. This is very important for object detection and semantic segmentation, as it can capture objects and background information of different scales. FPN improves the resolution and quality of feature maps through upsampling and skip connections, thereby improving segmentation accuracy. To maximize the performance of the neural network, the segmentation network's categories are set to the simple and straightforward "red light" and "non-red light" (background), and in the output mask, only pixels considered to be red lights will be non-zero.

As shown in Fig.3, a series of tests and experiments were conducted, balancing factors such as recognition performance and model size, ultimately deciding to use Resnet 34 combined with FPN as the network structure as despited in Fig.4. The network accepts 240 x 320 RGB images and outputs 240 x 320 segmentation results.

b) Training Data Preparation: To enable the model to recognize and segment red lights, we used `labelme` to annotate video frames captured by the vehicle's camera as despited in Fig.5. The total dataset contains 3039 images, including images with red lights, images with other colored lights, and images without any traffic lights. To improve the model's generalization ability, especially to reduce the false positive rate of red lights, we strictly controlled the number of images with red lights to avoid data imbalance.



Network Type	Network Name	Performance	Comment
Encoder	Resnet 18	Bad	small network, limited capacity
Encoder	Resnet 34	Acceptable	required recognition capability, low false detection rate
Encoder	Resnet 50	Good	model size has risen dramatically and performance is similar to Resnet 34
Encoder	Mix Transformer b0	Bad	virtually impossible to accurately identify
Encoder	Mix Transformer b2	Bad	virtually impossible to accurately identify
Decoder	Unet	Acceptable	
Decoder	FPN	Good	better recognition by considering features at different scales

Fig. 3: Network Performance Comparison

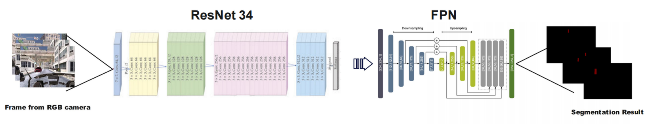


Fig. 4: Framework of Segmentation Network.

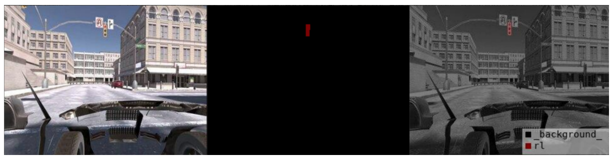


Fig. 5: Data annotation example using `labelme`.

To improve the network's generalization ability and avoid overfitting to specific intersection features, data augmentation methods such as vertical flipping and random Gaussian noise were used. The addition of Gaussian noise has two levels: weak and strong. These attempts expanded the total dataset to 4000 frames. The effect of data augmentation is shown in Fig.6.

The dataset was divided into 10% for the test set, 10% for the validation set, and 80% for the training set. The network was trained using the Adam optimizer, random initial weights, a batch size of 16, and a maximum of 200 epochs with a learning rate of 0.002. Intersection over Union (IoU) was used as the segmentation accuracy metric. The IoU is defined as:

$$\mathrm{IoU} = \frac{|\text{Prediction} \cap \text{Ground Truth}|}{|\text{Prediction} \cup \text{Ground Truth}|} \tag{1}$$

We used `wandb` to record training parameters as presented in Fig.7. Data analysis showed that the model reached over

Fig. 6: Original image and image with Gaussian noise added.

60% final accuracy within 10 epochs and converged by 50 epochs. The final model stopped training at 80 epochs, achieving an average IoU of over 0.9 on the training set and over 0.85 on the test and validation sets. The trained model was placed into the red light recognition pipeline as a model weight file (.pth), directly receiving RGB camera images and outputting segmented red light results. If no red lights were detected in the current frame, the output `npy` file was all zeros.

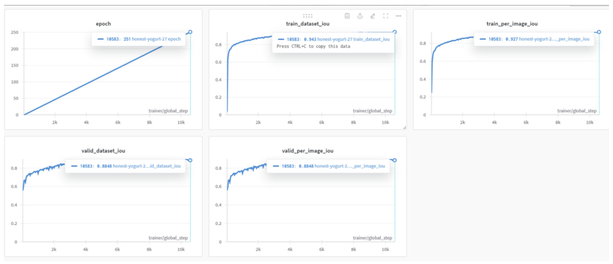


Fig. 7: Training curve recorded by `wandb`.

After adjusting hyperparameters and determining the model structure, we used all the data (validate set leak) to train the final model to maximize model performance with a limited dataset.

2) RGB Detection Pipeline: In this project, the segmentation network results are used to achieve the goal of recognizing red lights through a series of steps. The macro logic of the pipeline can be found in `red_light_detection_node.cpp`. In this recognition pipeline shown in Fig.8, the deep neural network helps filter out red light objects, followed by RGB detection and related auxiliary algorithms to determine.

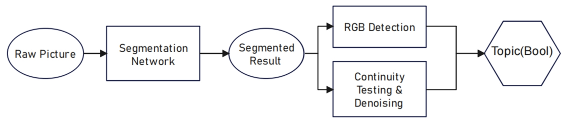


Fig. 8: Overview of RGB Detection Pipeline.

a) System Initialization and Subscriber/Publisher Configuration: First, the `RedLightDetector` class is initialized, and ROS subscribers and publishers are configured.

Subscriber:

- `semantic_sub_`:
  Subscribes to the `/semantic_image` topic, which publishes segmentation images generated by the semantic segmentation network.
- `rgb_sub_`:
  Subscribes to the `RGBCameraLeft/image_raw` topic, which publishes raw images captured by the RGB camera.

Publisher:

- `alert_pub_`:
  Publishes traffic light status messages to the `/traffic_light_state` topic.

b) Semantic Image Processing (`semanticCallback`): The `semanticCallback` function processes subscribed semantic segmentation images, only processing if a red light has not been detected. First, ROS image messages are converted to OpenCV format. Then, the `countNonBlackPixels` function is called to count the number of non-black pixels in the image. If the number of non-black pixels exceeds the set threshold, `current_consecutive_` is incremented. If consecutive frames exceed a threshold, red light detection mode is activated, and the counter is reset.

c) RGB Image Processing (`rgbCallback`): When red light detection mode is activated, the `rgbCallback` function processes RGB images. ROS image messages are converted to OpenCV format, and the middle 3/5 of the image is extracted to reduce processing range and speed up processing. The `check_red_light` function checks the extracted image for red lights. If red lights are detected, a stop signal is set and a traffic light status message is published.

II. Planning

For the planning part in our project, it is divided into two parts, the global planning and the local planning.

A. Global Planning

The `global_planner` package within the `move_base` framework is tasked with generating a viable trajectory for a robot to achieve a specified destination within a given environment. As depicted in Figure 9, this package utilizes the `global_costmap` and the goal published in the `move_base_simple/goal` topic to formulate a path from the robot's current location to the intended target.

Additionally, global planning involves identifying waypoints at specific points along the route that the car must pass through. In this context, 38 waypoints were selected, taking into account the number of turns in the environment, as well as the straight sections of the road and the start and end points. The choice of this number of waypoints aids in ensuring a smooth trajectory for the car to follow. With the build-in global planner, which is used Dijkstra's algorithm, the car can find an optimal path from the initial position to the target, as shown in Figure 10.

B. Local Planning

In this part, we choose `teb_local_planner` for the local planning task. The Timed Elastic Band (TEB) Local

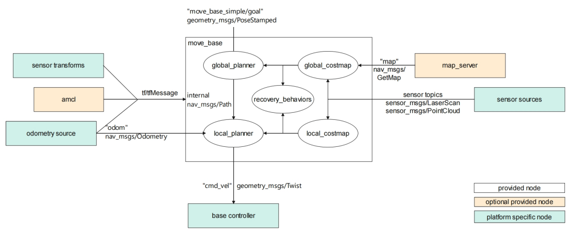


Fig. 9: Overview of move_base package

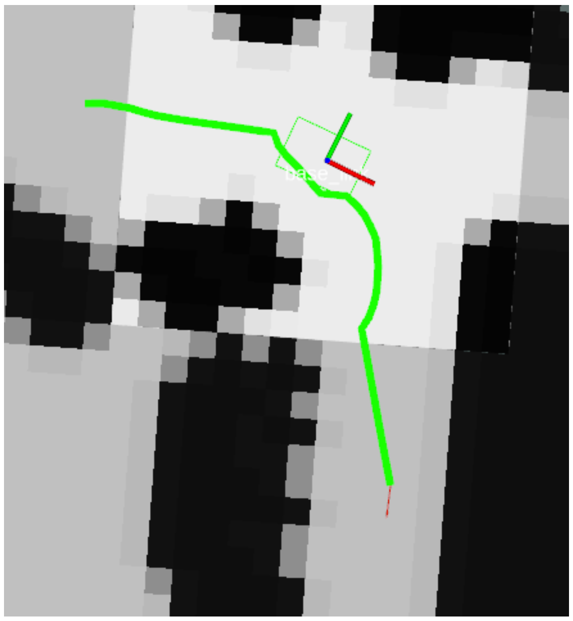


Fig. 10: Global path generated by build-in global planner

Planner, part of the ROS navigation stack, is an advanced local planner designed to optimize path planning for wheeled robots, particularly in dynamic environments. Unlike the build-in planner, the TEB Local Planner excels at handling non-holonomic constraints and is adept at generating feasible paths for car-like robots with Ackermann steering. This is achieved by leveraging a time-parameterized elastic band model, which considers both spatial and temporal aspects to ensure smooth and collision-free trajectories.

III. Control

A. State Estimator

`State_estimator_node` processes and filters state estimation data from robot sensors and then publishes the filtered state information. The StateEstimator class is designed to subscribe to the current state data from the `current_state_est` topic, buffer and filter this data using a sliding window approach, and then publish the filtered state information to the `forward_state` topic. Additionally, it broadcasts the filtered state as a `tf` transform. The `onCurrentState` callback function receives `nav_msgs::Odometry` messages from the `current_state_est` topic, extracts the current position, velocity, quaternion (orientation), and angular velocity, and stores these values in their respective buffers. It then calls the `publishFilteredState` method. The `publishFilteredState` method computes the average of all data in the buffers to obtain the filtered state information. It then broadcasts a `tf` transform and publishes a `nav_msgs::Odometry` message with the filtered position, velocity, and orientation to the `forward_state` topic.

B. PID Control Pipeline

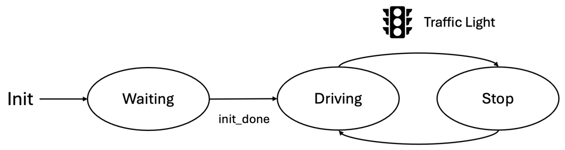


Fig. 11: State Machine Illustration

1) State Machine: As depicted in Fig. 11 , the state machine consists of three states: the initial waiting state, the driving state, and the stop state. In the waiting state, the recognition model is initialized. Upon receiving `init_done=True` signal from Perception, the state machine transitions from the waiting state to the operational state, where it alternates between the driving and stop states based on the traffic light status.

The `state_machine_node` subscribes to the `cmd_vel` topic, which conveys the desired linear and angular velocities. This node dissects and publishes these velocities in accordance with the traffic light state. A global flag is used to employ a singular signal as the switcher across both states. An external subscriber is responsible for updating the switcher. This subscriber acquires the current traffic light state and implements a counter which acts as a low-pass filter. If the current state remains consistent with the previous one, the counter increments; if the state differs, the counter resets, thereby reducing fluctuation due to noises.

Given the different frequencies of the planning unit and the PID control unit, the `state_machine_node` updates and publishes the desired velocities at distinct rates. When the traffic light is red, the state machine enters the stop state and publishes zero values for both two velocities. When the traffic light is any color other than red, it transitions to the drive state, broadcasting updated velocities as computed by the planning unit. This ensures that the autonomous vehicle responds appropriately to the traffic light states, maintaining safety and operational efficiency.

IV. Task Responsibility

Perception: Wenjie Xie, Dian Yu
Planning: Lingyue Zhao, Xinhua Wang
Control: Lingyue Zhao, Jingyi He