Individual Report

Cyber-Physical Systems Lab: Autonomous Applications
Name: Nicolas Helio Cunha Nabrink - 03783400

Introduction

The Cyber-Physical Systems Lab: Autonomous Applications course is designed to immerse students in the fundamental concepts and practical applications of the Robot Operating System (ROS). Throughout the lab, students are introduced to the intricacies of autonomous driving systems through the implementation of various strategies and safety mechanisms, culminating in the development of a comprehensive software solution. This report details the process and challenges encountered in completing three core tasks: the "Follow the Gap" strategy, Input Multiplexing, and Automatic Emergency Braking (AEB).

The first task focuses on implementing the "Follow the Gap" (FTG) strategy, an intuitive method where the vehicle steers towards the direction with the most available space, effectively navigating through complex environments. This section discusses the development and tuning of this strategy to ensure the car can successfully complete laps at varying speeds without collisions.

The second task involves Input Multiplexing, a critical feature in autonomous driving that allows seamless switching between manual and autonomous control modes. The task required the integration of a ROS architecture enabling real-time mode selection, allowing the user to alternate between manual keyboard input and the "Follow the Gap" strategy during operation.

Finally, the third task addresses the implementation of an Automatic Emergency Braking (AEB) system, a vital safety feature that prevents collisions by halting the vehicle when a potential crash is detected. The task involved creating a node to compute the Time to Collision (TTC) and integrate it into the existing architecture, ensuring the car could autonomously prevent accidents while allowing manual control to resume post-braking.

This report will elaborate on the methods and strategies employed to complete these tasks, followed by a discussion of the performance and integration of the final software solution.

1 Definitions

Symbol	Hyperparameter	Value	Unit
i_s	FTG input softner	0.3	-
c_t	FTG collision threshold	0.5	m
c_i	FTG collision input multiplier	-0.17	-
ttc_{lt}	AEB lower threshold for the TTC	0.3	s
ttc_{ut}	AEB upper threshold for the TTC	0.45	s

Table 1: Definitions of Hyperparameters used.

2 Task 1: Follow the Gap Strategy

2.1 Objective

The "Follow the Gap" strategy is a navigation technique for autonomous vehicles that directs the car towards the largest open space detected in its environment. The goal of this task was to develop a controller that could effectively implement this strategy, enabling the car to complete a full lap of the course without collisions. Additionally, the final objective was to optimize the controller to achieve the highest possible speed while maintaining stability and control throughout the lap.

2.2 Methodology

The initial implementation employed a straightforward "Follow the Gap" approach. The LIDAR signal was cropped to focus solely on the front-facing vision of the car. An index in the signal vector was then transformed into an angle that could be input into the vehicle's steering system. The basic concept was for the car to steer towards the farthest point detected in front of it, aiming for the largest gap.

However, this initial approach was unsuccessful, as the car frequently collided with curves. To mitigate this issue, a parameter (i_s) ranging from 0 to 1 was introduced and multiplied by the input angle. This modification ensured that the car would not aim directly at the farthest distance but slightly before it, thereby reducing sharp steering and improving control.

Although this adjustment led to better results, the car continued to occasionally hit the walls. To further enhance the strategy, a collision check algorithm was implemented. This algorithm calculated the minimum distance detected by the LIDAR and, if it fell below a certain threshold (c_t), the input angle was adjusted. Specifically, the input angle was reduced by a percentage (c_i) of the angle corresponding to the minimum distance. This adjustment caused the car to steer away from the wall when a collision was imminent, improving its ability to navigate tight turns without crashing.

2.3 Results

The car successfully completed a full lap using the implemented "Follow the Gap" strategy. The maximum speed achieved was 4 m/s. At this speed, the car completed the lap in approximately 22.35 seconds.

The collision detection algorithm proved effective, particularly in navigating curves, as it prevented the car from crashing. However, the car's trajectory was not ideal, as the collision algorithm was activated in every curve, leading to suboptimal path-following. This issue suggests that reducing the vehicle's speed could help in achieving a smoother trajectory, minimizing the need for frequent collision avoidance maneuvers

3 Task 2: Input Multiplexing

3.1 Objective

The objective of the input multiplexing task was to develop a system that allows seamless switching between different driving modes for the autonomous vehicle. Specifically, the goal was to enable the user to switch between manual control via a keyboard and the autonomous "Follow the Gap" strategy at runtime. This functionality is crucial for integrating human

supervision with autonomous driving, ensuring that the user can easily regain control of the vehicle when necessary.

3.2 Methodology

The architecture of the software was developed as suggested by the exercise in figure 1. The system was designed with a keyboard node, a "Follow the Gap" node, and a multiplexing (mux) node to manage the input sources.

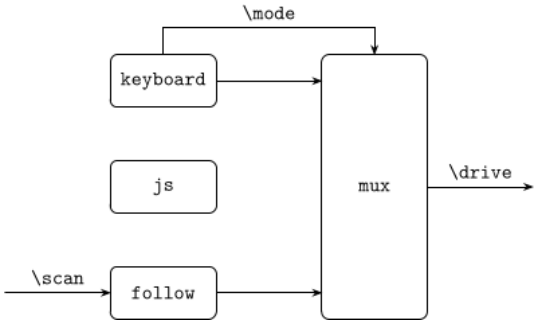

Figure 1: Architecture used in the ROS system.

The keyboard node publishes driving messages on the `/keyboard_drive` topic and the mode on the `/mode` topic. In this setup, mode `0` corresponds to keyboard control, and mode `1` corresponds to the "Follow the Gap" strategy. The keyboard node switches to mode `0` when any key is pressed, and to mode `1` when the `f` key is pressed.

The "Follow the Gap" node operates independently and publishes its driving messages on the `/follow_drive` topic.

The mux node subscribes to all of the mentioned topics (`/mode`, `/keyboard_drive`, and `/follow_drive`) and publishes directly to the `/drive` topic. The `/mode` callback updates a mode variable, which is set to `0` or `1`. The keyboard and follow-the-gap callbacks each update their own variables related to speed and angle. In the main function of the mux node, if `mode == 0`, the keyboard speed and angle variables are used to update the message, while if `mode == 1`, the follow-the-gap speed and angle variables are used to update the message that is published to the `/drive` topic.

3.3 Results

The system successfully switches between keyboard control and the "Follow the Gap" mode based on the user's input. If any key is pressed, the system changes to keyboard mode, while pressing the `f` key switches it to the "Follow the Gap" mode.

The simulation begins with the keyboard node active, and the car remains stationary due to the lack of input. From this initial state, the user can choose how to proceed by either controlling the car manually or activating the autonomous driving mode. The configurations used in the "Follow the Gap" mode are the same as those implemented in the previous task.

4 Task 3: Automatic Emergency Braking (AEB)

4.1 Objective

The goal of the Automatic Emergency Braking (AEB) task was to implement an AEB system specifically for the keyboard control mode, as the "Follow the Gap" mode already provided sufficient collision avoidance. The primary objective was to ensure that the car could detect a potential collision, apply the brakes, and stop safely when under manual control. Additionally, the task aimed to implement an algorithm that, after braking, would automatically reverse the car to a safe position and then wait for further keyboard input from the user.

4.2 Methodology

A new node, `aeb`, was created to handle the Automatic Emergency Braking (AEB) functionality. This node publishes to the `/aeb` topic and receives LIDAR data from the `/scan` topic.

The ranges vector from the LIDAR data was utilized in its entirety. The same algorithm used in previous tasks for calculating angles based on the index was applied here. An algorithm was developed to iterate through each distance, dividing it by the cosine of its corresponding angle, and determining the minimum value across the entire vector. Consequently, the `aeb` node published the minimum `distance/cos(angle)` value to the `/aeb` topic, without considering speed in this calculation, shown in equation (1).

$$ttc = \frac{d}{v \cdot cos(\alpha)} \tag{1}$$

The `mux` node then subscribed to the `/aeb` topic, read this information, and divided it by the speed being published to the `/drive` topic, thus obtaining the final equation for the Time-to-Collision (TTC).

In the main function, the current TTC value was compared against a lower threshold (`ttc_lt`). If the value fell below this threshold, and the mode was in keyboard mode, the initial approach was to set the speed and angle to zero and switch the mode to keyboard control (`mode = 0`), which successfully halted the car. To add the functionality of reversing to a safe position, the speed was set to `-1.8 m/s`, causing the car to reverse, and a `ttc_enable` flag was set to 1.

As the car began reversing, the TTC absolute value increased. Once the TTC reached an upper threshold (`ttc_ut`) and the `ttc_enable` flag was still set (indicating that the car was in the AEB process and had reached a safe position), the speed was set to zero, the mode was set to `0` (keyboard mode), and the `ttc_enable` flag was reset to 0. The car then stopped and awaited further input from the keyboard to resume movement.

4.3 Results

The AEB algorithm successfully functions in most scenarios, effectively preventing the car from crashing while in keyboard control mode. The reversing algorithm also operates as intended, although it occasionally encounters difficulties in resetting the car's movement without triggering another AEB event.

Initially, the approach involved using the minimum distance from the LIDAR vector, applying the angle and speed, which led to issues, particularly in quasi-parallel collisions. This

method was inaccurate and caused the car to fail in some situations. By adjusting the algorithm to use the minimum distance divided by the cosine of the angle, the system provided a more accurate assessment. With this corrected approach, the car did not experience unintended collisions during simulations.

The upper threshold used to set the safe position was based on the TTC of collision, considering the angle of the minimum distance to the wall. As a result, if a collision occurs almost parallel to the wall, the car will reverse to a position closer to the wall compared to a full frontal collision. While this did not present any significant problems, it is an important factor to consider for refining the algorithm.

However, the algorithm has a limitation: it only activates the AEB system for frontal or lateral collisions. Reversing collisions can still occur due to the way angles are calculated in the system, indicating that further refinements are needed to handle all types of collisions effectively.

5 Conclusion

In this laboratory course, the project achieved its educational objective of demonstrating a comprehensive understanding of the Robot Operating System (ROS), the simulation platform, and various autonomous driving functions.

The "Follow the Gap" strategy was successfully implemented, allowing the car to consistently complete the laps at speed of 4 m/s. Although the initial approach faced challenges with collision detection and handling curves, subsequent adjustments, including collision avoidance and steering smoothing techniques, significantly improved performance.

The input multiplexing task effectively allowed for seamless switching between manual keyboard control and the autonomous "Follow the Gap" mode, enabling flexible control over the vehicle. The system's architecture and node interactions were well-integrated, ensuring that mode transitions functioned smoothly.

The Automatic Emergency Braking (AEB) system was implemented for keyboard mode, successfully preventing collisions and allowing the car to reverse to a safe position when necessary. While the reversing algorithm demonstrated effectiveness, it occasionally struggled with resetting the car's movement without reactivating the AEB system. Additionally, the system showed limitations in handling certain collision scenarios, specifically reversing collisions due to angle calculations.

Overall, the final system developed was considered successful. It effectively met the objectives of the laboratory course, demonstrating a solid grasp of ROS concepts and autonomous driving functionalities. The project not only fulfilled its educational goals but also provided practical insights into the complexities of developing autonomous vehicle systems.