Synthesis project

DTU Electrical Engineering
Department of Electrical Engineering

Navigation System for the Autonomous Ecocar

Local map system and trajectory planner

Haotian Gao(s192232)

Kongens Lyngby 2020

Automation and Control
DTU Electrical Engineering
Technical University of Denmark

Elektrovej
Building 326
2800 Kongens Lyngby, Denmark
Phone +45 4525 3576
studieadministrationen@elektro.dtu.dk

Project period: September 10th,2020 - December 22th,2020

ECTS: 10

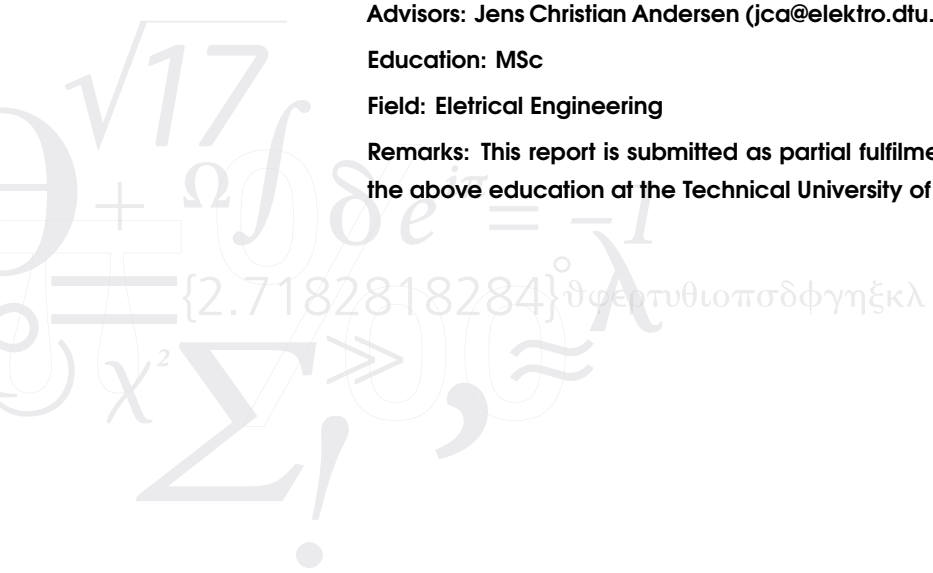
Author: Haotian Gao (s192232@student.dtu.dk)

Advisors: Jens Christian Andersen (jca@elektro.dtu.dk), Henning Si Høj (hsih@elektro.dtu.dk)

Education: MSc

Field: Eletrical Engineering

Remarks: This report is submitted as partial fulfilment of the requirements for graduation in the above education at the Technical University of Denmark.



Summary

To make Ecocar Dynamo challenge autonomous driving tasks, the DTU roadrunner team has established a software system, including lidar-based perception, central path extraction, steering and braking automation, and so on. However, in order to achieve a planning path with a fuel-efficient cost optimization and better smoothness, there is still a lot of work to be done in the navigation part.

This report contains three parts. The first part specifically describes the construction of a real-time updated local cost map system. The local map system is based on the basic framework of the *costmap_2d* software package of ROS. According to the characteristics of the track environment, a new map layer, *"disfill layer"* is used, and the function of changing the size of the local map as the track mode changes is added. These functions not only reduce the amount of calculation to a certain extent, but also ensure robustness, and provide convenient cost information for the back-end planner.

The second part specifically introduces a new local trajectory planner. The planner firstly predicts the forward generation of trajectory clusters based on the vehicle model at a certain frequency. Next, planner scores these trajectories based on the cost information of the local map and the offline optimized global path (In cooperation with *Liguang He*'s work), and executes the corresponding input for the track with the best score at the current time.

The third part is about the analysis of the test results of the local map system and the local planner. The real-time performance and effectiveness of the local map and local estimation planner have been verified in the simulation environment of *challange1_London* competition in Gazebo, and it is expected to be applied in actual competitions.

Preface

This report is the result of a 10 ECTS point synthesis course conducted at the the department of Electrical Engineering in the Technical University of Denmark. The project is carried out as a member of DTU Roadrunners.

December 21, 2020

Contents

Summary i

Preface iii

Contents v

1 Introduction 1
- 1.1 DTU Roadrunners . . . 1
- 1.2 Motivation . . . 2
- 1.3 Code list . . . 4

2 Local map system 5
- 2.1 Overview . . . 5
- 2.2 Costmap_2d . . . 6
- 2.3 New map layer . . . 9
- 2.4 Dynamic change of map size . . . 13

3 Local planner 15
- 3.1 Overview . . . 15
- 3.2 Work flow . . . 17
- 3.3 Dynamic model of the vehicle . . . 18
- 3.4 Control sampling selection . . . 20
- 3.5 Collision detection and scoring . . . 22
- 3.6 Perform steering input for optimal trajectory . . . 24

4 Test results and analysis 25

5 Conclusion 33

Bibliography 35

CHAPTER 1

Introduction

1.1 DTU Roadrunners

DTU Roadrunners is a student driven project at the Technical University of Denmark with the goal of constructing fuel-efficient ecocar and participating in annual Shell Eco-marathon[17]. DTU Roadrunners started participating in Shell Eco-Marathon in 2004, and have won many class victories in fuel consumption competitions.

As a race car of the Urban Concept class, DTU Dynamo won 10 championships in 11 races after its debut in Shell Eco-Marathon, and has been maintained and upgraded by the DTU Roadrunner team. The figure1.1 is the DTU Dynamo 2020 at Roskilde Racing Center.

Figure 1.1: DTU Dynamo 2020[Mad20].

To prepare for the future autonomous driving competitions, DTU roadrunner began to design and modify the ecocar in 2017, equipped with sensors such as lidar and cameras, realizing the automation of the steering wheel and brakes, and building an autonomous driving software system. Two autopilot-related awards are won in the 2018 Shell Eco-Marathon.

1.2 Motivation

In order to achieve better fuel-efficiency, a proper, smooth and safe execution path is necessary. Generally speaking, the navigation module of the automatic driving system receives the environmental information processed by the perception module, computing and planning an executable path, and inputs it to the control module for execution.

The members of the DTU roadrunner automation team have been updating and perfecting DTU Dynamo's autonomous driving system in recent years, including extracting point clouds on both sides of the track with or without barriers and the point cloud of obstacles on the track based on the lidar-based perception module[Mad20]. There is also a navigation module based on the voronoi algorithm. The center line of the track is extracted and smoothed through the point clouds on both sides of the track output by the perception module, which is input to the control module as a reference point[Lar18].

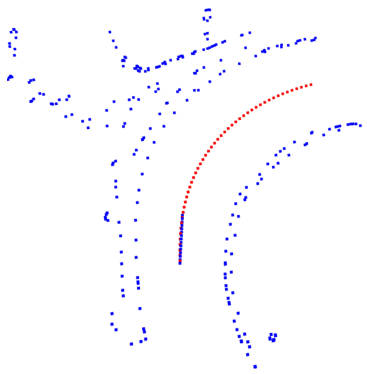
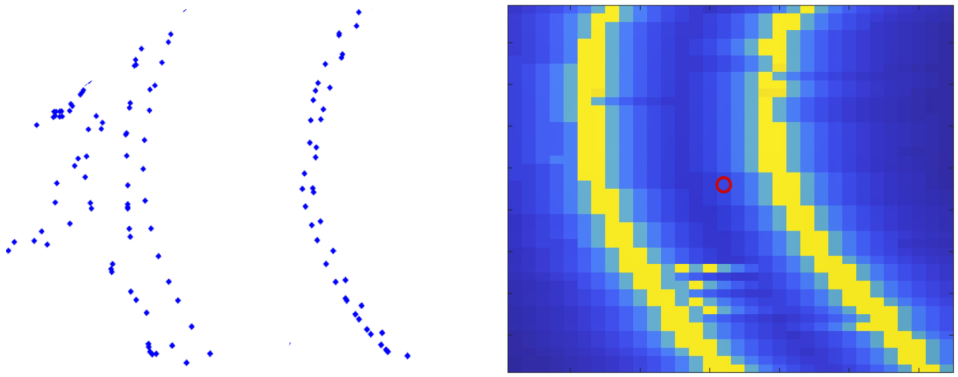


Figure 1.2: The center line of the track extracted by the voronoi algorithm[Lar18].

For the voronoi algorithm used in the current navigation module, although it has the advantages of low computational cost and suitable for simple lane environments, the center line extracted by the voronoi algorithm as the execution path only ensures a certain degree of safety, which can not perform well in terms of fuel-efficiency, smoothness and easy control. And it is difficult to deal with more complicated road conditions.

In addition, because of the path planning, speed planning based on the voronoi algorithm are separated, and the generated path does not consider the car's kinematic model, the driving efficiency is very dependent on the performance of the control module.



Point cloud map 2D cost grid map

Figure 1.3: Comparison of different maps.

Therefore, a local planner, which can plan an executable trajectory based on real-time environmental information and comprehensive consideration of safety, fuel efficiency, and ease of control is necessary. First of all, the environmental information processed by the perception module is given to the navigation module based on the type of point cloud, however, the planning of a 3D map based on the point cloud is very difficult. To achieve the goal that the planner can use the map environmental information conveniently and efficiently, for example, only $O(1)$ is required for search, real-time updated 2D local cost map is carried out, which is also the first part of the project.

In the second part, the construction of the local planner will be combined with the master's thesis that *Liguang he* is working on. The global path optimized offline by his work will be used in the local planner construction. The specific method of the local planner will be explained in the following chapters.

1.3 Code list

In order to be clear and convenient to distinguish from the original work, the codes of the local map system and the local planner are placed in the *costmap_2d* package.

System	Path
Local map system	costmap_2d/include/costmap_2d/disFill_layer.h
Local map system	costmap_2d/plugins/disFill_layer.cpp
Local map system	costmap_2d/cfg/disFillPlugin.cfg
Local map system	costmap_2d/src/costmap_2d_ros.cpp
Local planner	costmap_2d/src/lattice_planner_node.cpp
Local planner	costmap_2d/src/steering_publisher.cpp

Table 1.1: Code list.

CHAPTER 2

Local map system

2.1 Overview

The use of the real-time local grid maps in the navigation module poses a great challenge to the computational ability of the CPU. How to update the local map as fast and accurate as possible within the limited computational cost is the key problem. The *costmap_2d* package of ROS provides an implementation of a *costmap_2d* that takes in sensor data from the world[Nic18] and works by separating the processing of cost-map data into semantically-separated layers[LHS14]. The plugin-based framework of *costmap_2d* has good scalability. According to the special environment of the track, the new carried-out map layer can be added to realize the construction of cost map, and more functions can be added to achieve the purpose of reducing the computational cost.

The 2d local cost map system in this project is based on the framework of *costmap_2d* package of ROS. The original software package of the *"obstacle layer"* is only used, the added functions are shown below.

- Insert the *"disfill layer"* map layer suitable for the track environment to replace the "inflated layer" commonly used in the original software package to fill in the cost information of each grid.
- Added the ability to adjust the size of the local map as the shape of the track changes.

2.2 Costmap_2d

In this section the characteristics of *costmap_2d* package, the composition of the framework and how it works are briefly introduced.

First of all, unlike traditional binary or further grid maps that represent occupancy probability, *costmap_2d* uses passing costs to represent the cost information of each grid. This means that each grid will have an unsigned char value between 0-255 shown in Figure 2.1. There is an original definition of the passing cost in the software package, however, due to the reason that this method is not used, it will not be elaborated in this report.

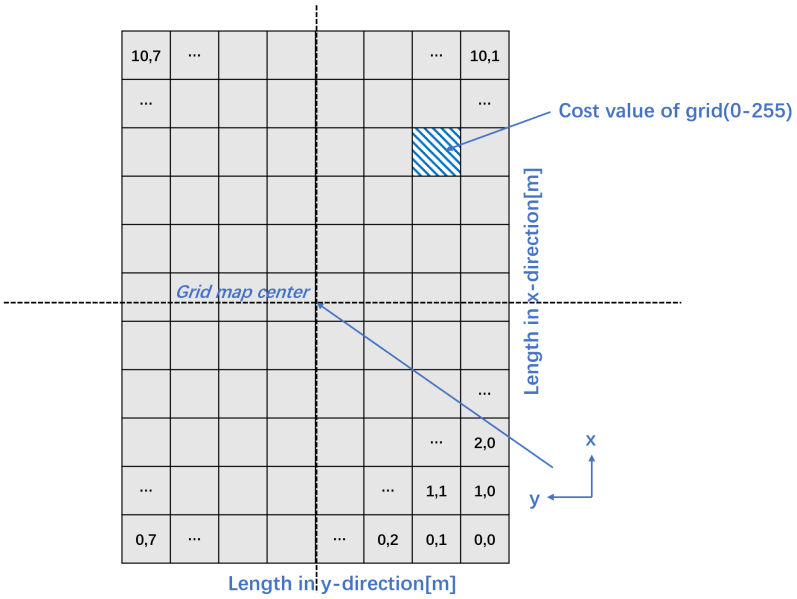


Figure 2.1: local grid map overview.

Figure 2.2 shows the main classes in the *costmap_2d* package and their inheritance relationship. The ROS function interface provided by *costmap_2d* is mainly *costmap_2d::Costmap2DROS*, which uses *costmap_2d::LayeredCostmap* to track each layer. Each layer is instantiated in *Costmap2DROS* as a plugin and added to the *LayeredCostmap*.

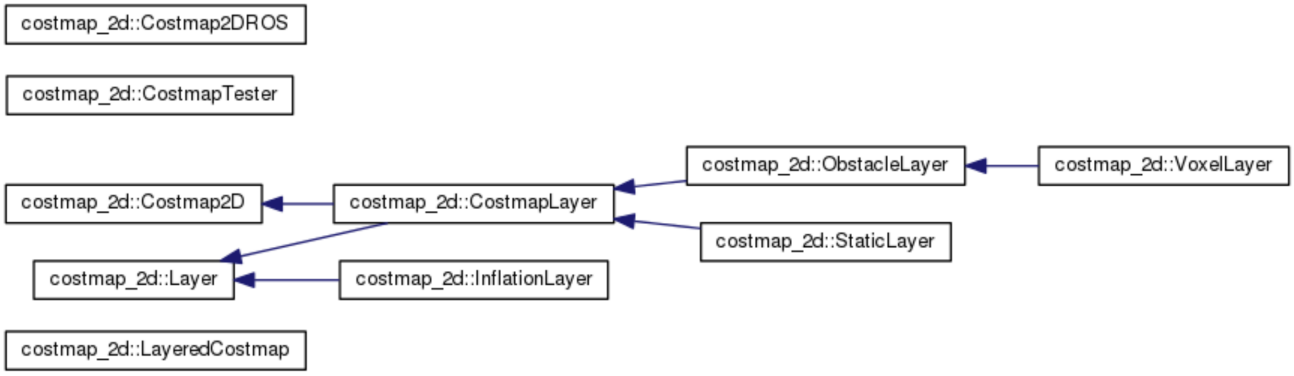


Figure 2.2: Main Class Hierarchy of *costmap_2d*[Eit19].

Each layer can be compiled independently, and the *C++* interface can be used to modify the cost map layer, that is, *LayerdCostmap* provides *Costmap2DROS* (user interface) with a plug-in mechanism for loading the map layer. Each plug-in (map layer) is of the Layer type. The *costmap_2d:: Costmap2D* class implements the basic data structure used to store and access the 2D cost map.

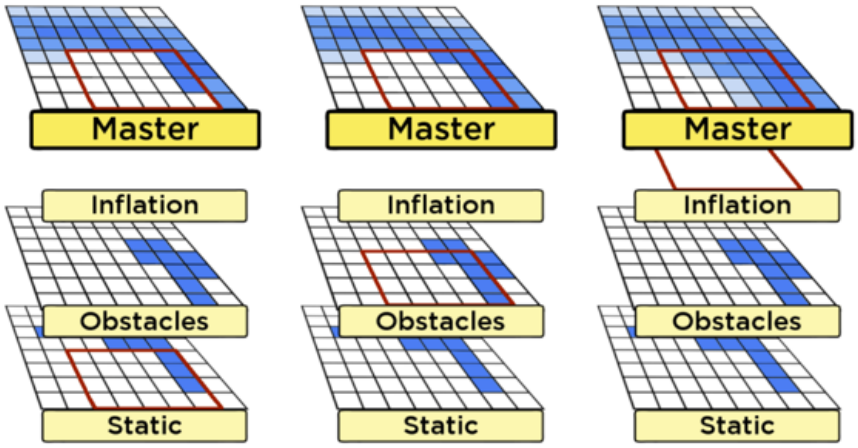


Figure 2.3: Update of multiple layers to the master map[LHS14].

Figure2.3 is a schematic diagram of how the various map layers defined in *costmap_2d* are updated and integrated into the master map. There are many examples of map layers in the figure that exist as plug-ins. The *costmap_2d::LayeredCostmap* mentioned before will start from bottom to top. The layers are updated and written into the master map. Some map layers such as *"obstacle layer"* need to input sensor information and therefore have their own maps, while some map layers such as *"inflation layer"* only need to operate on the existing map without having to own map layer to save space.

2.3 New map layer

In the original software package, if there is no global map, the *"static layer"* cannot be used. The local master map is generally constructed by first using *"obstacle layer"* to receive the obstacle point cloud through the function, and write into the corresponding grid with a cost value of 254. Then the "inflation layer" is used to fill in the surrounding grids of each grid with a cost value of 254. Although the filling radius parameter of inflation can be directly increased to complete the cost information of all grids, however, this will undoubtedly bring a lot of calculation costs. At the same time, because of the processing of the perception module, the point clouds at the two edges of the track are relatively sparse, and directly filling in the case of relatively low resolution will cause large errors.

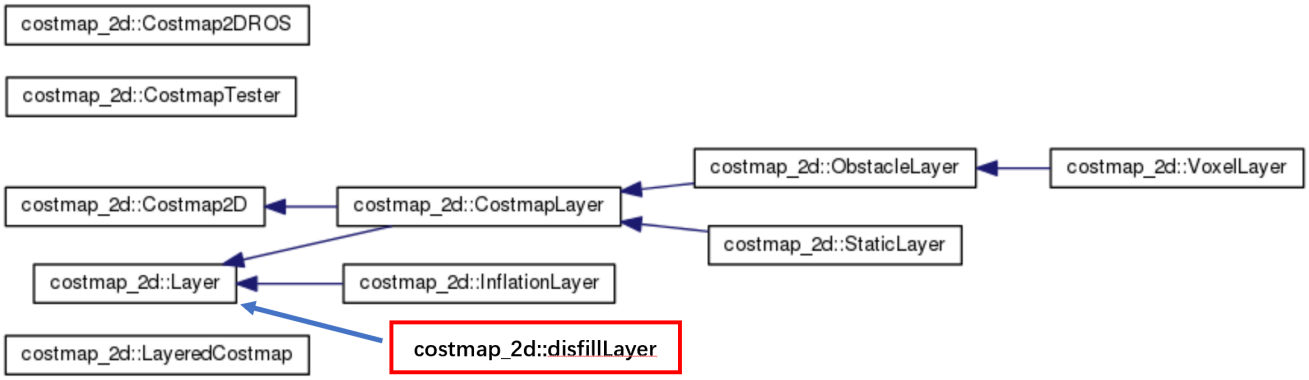


Figure 2.4: Main Class Hierarchy of *costmap_2d* with new layer.

So instead of using the *"inflation layer"* of the original software package, a *"disfill layer"* is created to complete the filling of all grid cost value after the *"obsatcle layer"*. The workflow of this map layer is listed below.

- After the *"obsatcle layer"* writes the grid corresponding to the sparse track edge with a cost value of 254, the *"disfill layer"* completes the unfilled grid of the incomplete track edge with a cost value of 254 to make track edge complete.

- After track edge is complete, we traverse each line of the map (because of another function we added, the size and direction of map will change, so the line here is usually the line along the y-axis of the car coordinate system) and fill in the cost value based on the distance between each grid and the grid of track edge. The cost value is between 0-253.

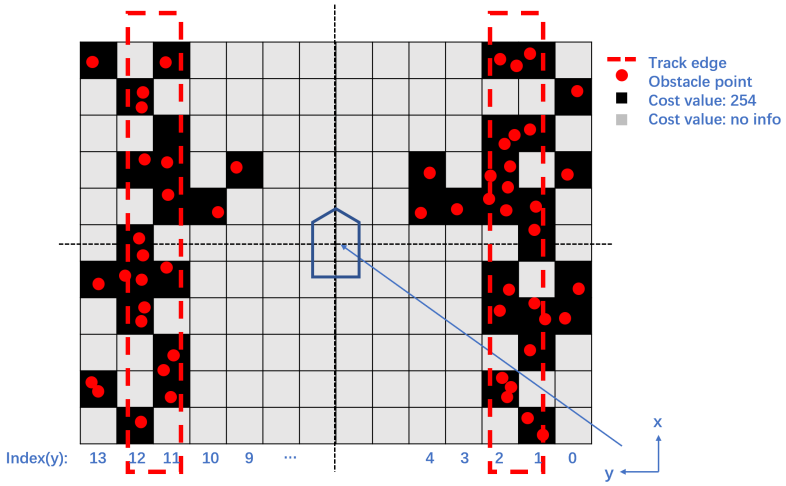


Figure 2.5: Incomplete track edge.

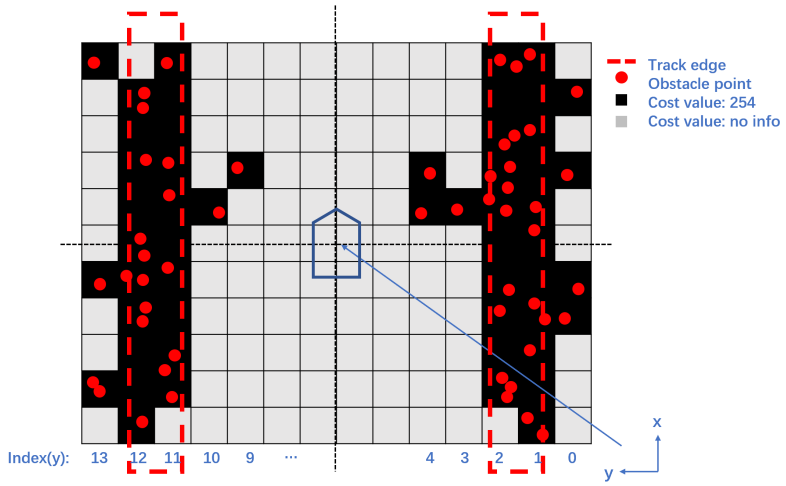


Figure 2.6: complete track edge.

How the *"disfill layer"* program works is specifically introduced in the following section. The first is about how to complete the grid at track edges. In order to reduce the computational cost as much as possible, reduce and speed up operations such as traversal, sorting, and search, we use the *"multimap"* in *STL*(Standard Template Library). Take the figure2.5 as an example, y coordinate of the grid with a cost value of 254 is used as the key and the x coordinate as the value. After traversing all keys, the number of corresponding values is compared with a threshold. If the number is large enough, it can be regarded as a track edge, writing down the minimum and maximum value (x coordinate) as the start and end points of the track edge, and write the cost value 254 into the grid one by one. Figure 2.7 is a schematic diagram of building a *multimap*.

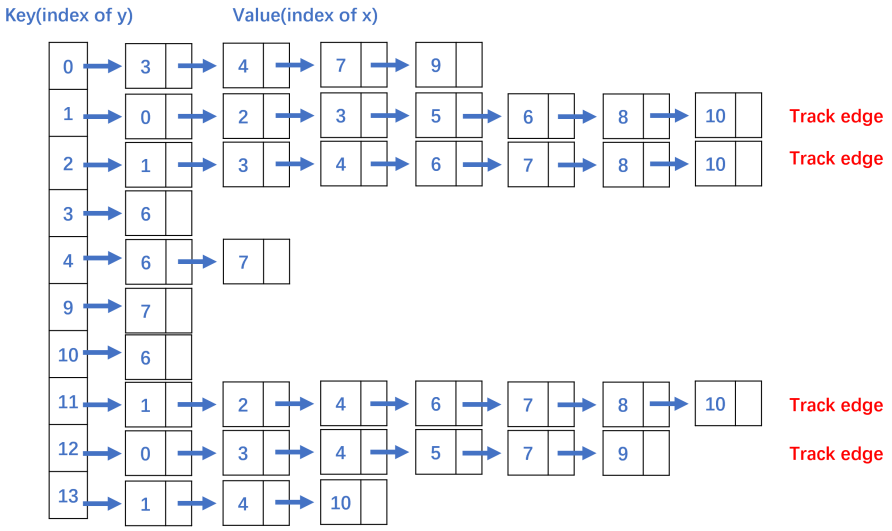


Figure 2.7: *Multimap* for completing track edge.

After getting the complete track edge, the *"disfill layer"* can easily find the edge grid line by line and calculate the closest distance to the edge grid, and fill in each grid as cost information. The cost value is between 0-253.

Here a rough comparison of the calculation cost with the *"inflation layer"* method of the original software package is made, assuming that the resolution is 0.5[m/cell], the map size is $22[m] \times 16[m]$, and the 7-meter-wide track requires an expansion radius of at least 3.5[m]. As a result an obstacle grid

needs to expand the grid as $14 \times 14 = 196$, if the result of N obstacle points is obtained from the perception module, then the grid that needs to be operated is $196 \times N$. And a large number of grids are repeated operations. The completion track of the "disfill layer" only needs to operate $22/0.5 \times 4 = 176$ grids, and traversing the map grid requires $22 \times 16/0.5 = 704$ grids. It can be found that the calculation cost of *"disfill layer"* will not increase with the increase of obstacle points.

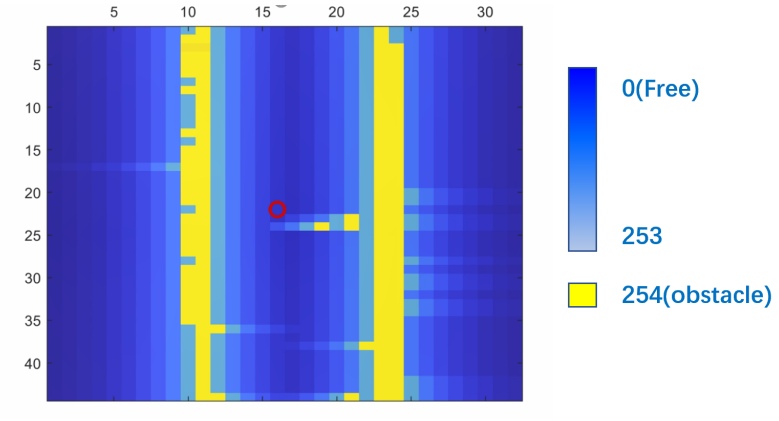


Figure 2.8: Cost value description.

2.4 Dynamic change of map size

The size of the local map in the original software package is generally set to a square, because the *Navigation* module in *ROS* is generally suitable for mobile robots in an open environment, so the surrounding information is necessary. But for the track environment, the width of the track is known, and the car pays more attention to the information ahead, so the map size in the y-axis direction of the car coordinate system can be reduced to a certain extent, for example, the x direction of the car coordinate system is 22 meters, the y-axis is taken as 16 meters. The calculation cost can be reduced.

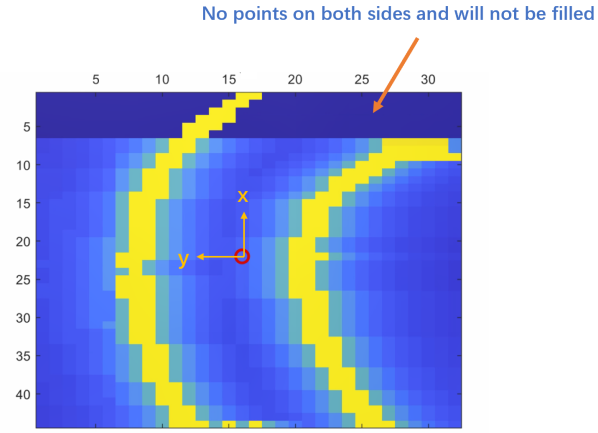


Figure 2.9: Situation 1 in different directions.

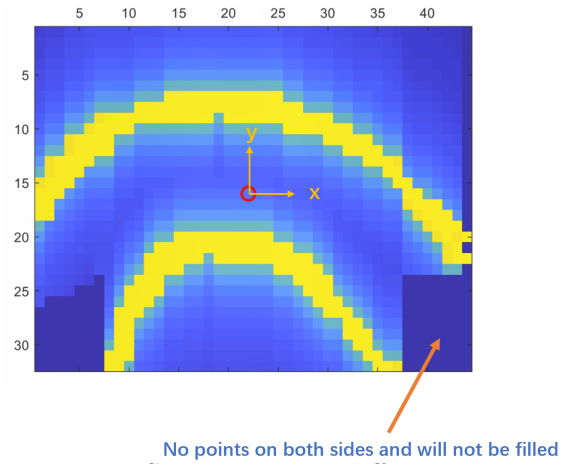


Figure 2.10: Situation 2 in different directions.

Because the curve and direction of the track are constantly changing, it is necessary for the map size to change with the shape of the track for the completeness of the *"disfill layer"* function. But it is always expected to activate this function when the track really changes drastically. So it is necessary to extract accurate signals from environmental information. A function in the *"disfill layer"* is added for assisting in judging this conversion signal. After the track edges is completed, the grid of each row will be traversed. Normally, there are grids with a cost of 254 at both ends of each row. However, when undergoing a sharp turn, the curvatures of the two edges differs a lot. Generally, only one end has a grid with a cost of 254. In this project, it is decided not to fill in the cost information in this row, let them keep the default 0 cost value. Figure 2.9 and Figure 2.10 show two of these situations.

Therefore, the conversion signal can determine the direction of the map by the steering angle of the vehicle, and then select the corresponding row to find how many grids with a cost value of 0, and compare with the set threshold to determine whether the map size should be changed. At the same time, in order to prevent the uncertainty of perception from causing instability of the conversion signal, we set the interval for the map size change, that is, it is not allowed to continuously change the map size in a very short time.

CHAPTER 3

Local planner

3.1 Overview

In order to comprehensively consider safety, fuel efficiency, easy control and other indicators to plan an executable trajectory. The method based on online trajectory optimization can ensure the optimality of the computed trajectory. Trajectory optimization can be defined as an optimal control problem. Although there are many numerical solutions (linear/non-linear solvers) that can speed up the solution of the problem, local minimums or no solutions often occur, and the uncertain computational cost is very high. It is difficult to guarantee real-time performance.

In addition, DTU Dynamo's engine is a single cylinder modified moped engine. It works in one way only: coast and burn. When turned the engine "burns" to accelerate. When turned off the car coasts until burning again[Lar18]. This places great limitations on the trajectory optimization problem, and it is difficult to find the optimal solution.

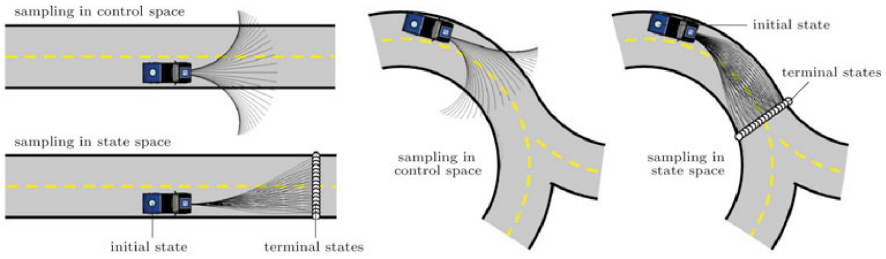


Figure 3.1: Control sampling and state sampling of lattice planning[How+08].

It is considered to use another trajectory planning method-lattice planning to get a sub-optimal solution, but to ensure real-time and stability. Lattice planning is divided into control space sampling and state space sampling according to the difference in sampling space. Since state space sampling still needs to optimize the parameter curve, it is also not suitable for the engine mode. Therefore, the control space sampling method is adopted. And the lattice planner is very flexible and can generate multi-layer trajectories[Fig3.2], but considering the track environment and computational cost, one-layer trajectory will be only generated.

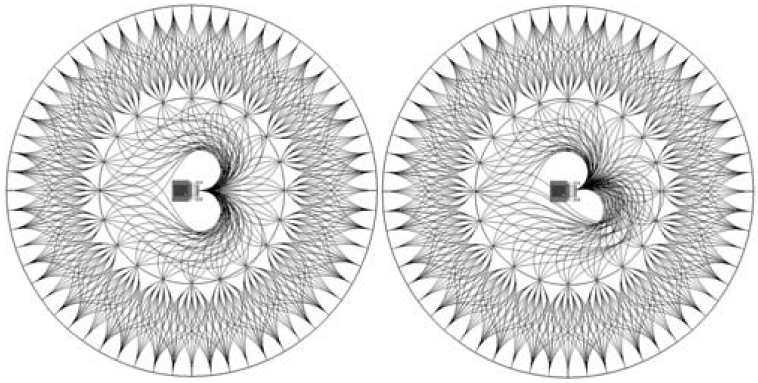


Figure 3.2: Multi-layer trajectories generated by lattice[HK07].

3.2 Work flow

The workflow of lattice planner is divided into:

- Generate trajectory clusters based on the model
- Collision detection and scoring
- Choose the best path and Perform the corresponding input

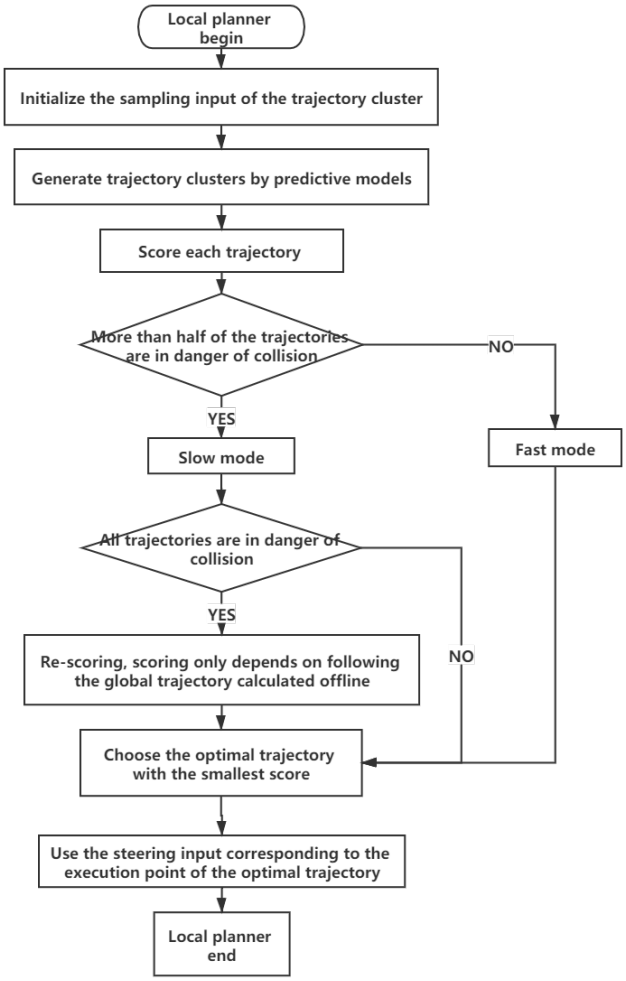


Figure 3.3: Flow chart of local planner.

3.3 Dynamic model of the vehicle

One advantage of lattice planner is that the output trajectory takes into account the vehicle model, because the executable trajectory depends on whether the kinematics constraints of the car body are satisfied. At the same time, considering the particularity of the engine, the construction of the vehicle model is very important.

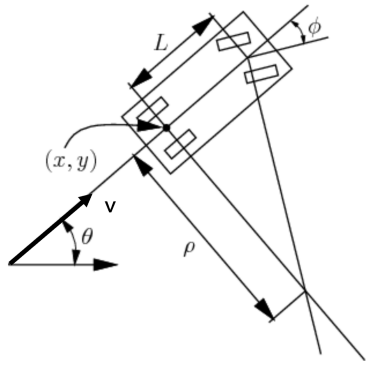


Figure 3.4: Schematic diagram of vehicle kinematics model[LaV06].

The generation of trajectory clusters requires the current state and sampling control input to perform forward integration of the model. As shown in Figure 3.4, the state of the model includes the position, linear velocity and direction of the vehicle, which can be obtained from sensors, and the perception module provides related interfaces.

$$s = [x, y, v, \theta]$$

The input of the model includes engine switch, brake level and steering angle. Considering the particularity of the engine, when the speed exceeds a certain speed, the speed ratio will switch to continue driving with a set of smaller accelerations. Therefore, the input of the engine is set to three options of 0, 0.5 and 1, which respectively represent turning off the engine, low-speed engine and high-speed engine. The brake level is constrained to be an integer between 1 and 34, and the unit is bar. The steering angle is

constrained to be between -15 degrees and +15 degrees.

$$\begin{aligned} \text{Engine input} &: u_e = [0, 0.5, 1] \\ \text{Brake input} &: u_b = [1, 2, ...33, 34] \\ \text{Steering input} &: -15 \leq u_s \leq 15 \\ u &= [u_e, u_b, u_s] \end{aligned}$$

In this model, resistance F_w is also considered,

$$F_w = -\frac{1}{2}\rho A C_w v^2$$

where ρ is the density of air, A is the cross-sectional area, C_w is the coefficient of air resistance and v is the relative velocity between the air and vehicle. Below is our continuous system state equation

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= \frac{v}{L}\tan\theta \\ \dot{v} &= g_{acc}\left(u_e\right) + g_{dec}\left(u_b\right) + g_w\left(F_w\right) \end{aligned}$$

Where g_{acc} is a linear function of acceleration and engine switch, g_{dec} is a linear function of deceleration and braking, and g_w is a linear function of resistance. The function coefficients of acceleration and deceleration are estimated from previous test data.

In the actual trajectory generation, the continuous system state equation needs to be discretized, so as to effectively generate the trajectory for a given control input sequence. The zero-order hold discrete method is used.

$$\begin{aligned} x_n &= \sum_{i=0}^{n-1} v_i \cos\left(\theta_i\right)\Delta t = x_{n-1} + v_{n-1}\cos\left(\theta_{n-1}\right)\Delta t \\ y_n &= \sum_{i=0}^{n-1} v_i \sin\left(\theta_i\right)\Delta t = y_{n-1} + v_{n-1}\sin\left(\theta_{n-1}\right)\Delta t \\ \theta_n &= \sum_{i=0}^{n-1} \frac{v_i \tan\left(u_s^i\right)}{L}\Delta t = \theta_{n-1} + \frac{v_{n-1}\tan\left(u_s^{n-1}\right)}{L}\Delta t \\ v_n &= \sum_{i=0}^{n-10} \left(g_{acc}\left(u_e^i\right) + g_{dec}\left(u_b^i\right) + g_w\left(F_w^i\right)\right)\Delta t \\ &= v_{n-1} + \left(g_{acc}\left(u_e^{n-1}\right) + g_{dec}\left(u_b^{n-1}\right) + g_w\left(F_w^{n-1}\right)\right)\Delta t \end{aligned}$$

3.4 Control sampling selection

The generation of trajectory clusters over a period of time through model prediction means that each trajectory corresponds to a fixed input in a fixed time range. These fixed inputs can be uniformly sampled within the range of available input values to generate various potential candidate trajectories. In previous section, three inputs of the model are introduced, which are engine switch, brake and steering angle. First, based on a large number of previous tests, the vehicle's engine mode is designed for fuel-efficiency, so the speed control mode is certain. The brake input is always set to 0, so the sampling dimension of the control can be reduced to two inputs. The control mode of the engine will select the three inputs 0, 0.5 and 1 according to the judgment of real-time speed to ensure that the trajectory cluster is within a reasonable judgment range. For example, if the engine input is still 1 at high speed, the generated trajectory will be too long to be judged. On the contrary, when the engine is turned off at low speed, the generated trajectory is too short and has no reference significance.

$$\begin{aligned} u_e &= 1 \qquad && v < v_{slow} \\ u_e &= 0.5 \qquad && v_{slow} < v < v_{fast} \\ u_e &= 0 \qquad && v > v_{fast} \end{aligned}$$

Regarding the sampling of steering angle input, if a small range of input is used, calculation time will be shortened. However, some potential candidate trajectories may be missed in the calculation, which may reduce the quality of the generated trajectories. However, sampling too many candidate trajectories means that additional computational costs is added at each step. At the same time, the limit change speed of the steering angle should also be considered. For example, planner cannot give actuator a steering angle of -15 degrees in the last 0.1 second, and let actuator turn to 15 degrees in the next 0.1 second.

In each planned unit time, the steering angle within the range of $[-15, 15]$ will be sampled and planner generates 13 trajectories, respectively [-15, -12.5, -10, -7.5, -5, -2.5, 0, 2.5, 5, 7.5, 12.5, 15]. However, in order to consider the limit change speed of the steering angle, a constant steering angle input is not used in the process of generating a trajectory, but a steering angle input that is gradually linearly approximated.

when $u_{goal} - u_{last} > 0$

$$u_s^k = u_{last} + kw_{max}\Delta t \qquad u_s^k < u_{goal}$$
$$u_s^k = u_{goal} \qquad u_s^k \geq u_{goal}$$
$$k = [0, 1, 2, ...N-1, N] \qquad N = \frac{T_{final}}{\Delta t}$$

when $u_{goal} - u_{last} < 0$

$$u_s^k = u_{last} - kw_{max}\Delta t \qquad u_s^k > u_{goal}$$
$$u_s^k = u_{goal} \qquad u_s^k \leq u_{goal}$$
$$k = [0, 1, 2, ...N-1, N] \qquad N = \frac{T_{final}}{\Delta t}$$

Where T_{final} is the length of the trajectory prediction time, Δt is the sampling time of the trajectory prediction, u_{goal} is the sampling target value selected ([-15, -12.5...12.5, 15]), u_{last} is the steering angle input executed during the last planning, and w_{max} is limit change speed of the steering angle.

In order to better illustrate the input selection of the steering angle, an example is shown in Figure3.5. Suppose that in a certain unit time of the planning task, the steering angle input obtained in the last planning is $5[degree]$, the limit change speed of the steering angle is $20[degree/s]$, and Sampling time to generate trajectory is $0.1[s]$, The selection of steering angle input for 13 trajectories after calculation is shown in Figure3.5.

Trajectory generation timeline

Different sampling target value

	k=0	k=1	k=2	k=3	k=4	k=5	k=6	k=7	k=8	k=9	k=10	k=11	···	k=N
-15	5	3	1	-1	-3	-5	-7	-9	-11	-13	-15	-15	···	-15
-12.5	5	3	1	-1	-3	-5	-7	-9	-11	-12.5	-12.5	-12.5	···	-12.5
-10	5	3	1	-1	-3	-5	-7	-9	-10	-10	-10	-10	···	-10
-7.5	5	3	1	-1	-3	-5	-7	-7.5	-7.5	-7.5	-7.5	-7.5	···	-7.5
-5	5	3	1	-1	-3	-5	-5	-5	-5	-5	-5	-5	···	-5
-2.5	5	3	1	-1	-2.5	-2.5	-2.5	-2.5	-2.5	-2.5	-2.5	-2.5	···	-2.5
0	5	3	1	0	0	0	0	0	0	0	0	0	···	0
2.5	5	3	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5	2.5	···	2.5
5	5	5	5	5	5	5	5	5	5	5	5	5	···	5
7.5	5	7	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	7.5	···	7.5
10	5	7	9	10	10	10	10	10	10	10	10	10	···	10
12.5	5	7	9	11	12.5	12.5	12.5	12.5	12.5	12.5	12.5	12.5	···	12.5
15	5	7	9	11	13	15	15	15	15	15	15	15	···	15

Unit: degree

Figure 3.5: Selection of steering angle input.

3.5 Collision detection and scoring

After the trajectory cluster is generated, each trajectory needs collision detection and scoring. In the local map system built in the first part, the *"disfill layer"* has written a cost value for each grid, so collision detection can be achieved simply by checking whether the cost value of the grid where the track point is located is 254. If a collision is detected, then there is no need to score this trajectory.

As mentioned in the introduction, *liguang he*'s work offline calculated the optimal route based on the global track, and the interval of the path points is $1[m]$. Therefore, in scoring the trajectory, local planner comprehensively considers following the global optimal route calculated offline and the safety of obstacle avoidance.

$$cost_{traj} = cost_{global} + cost_{safety}$$

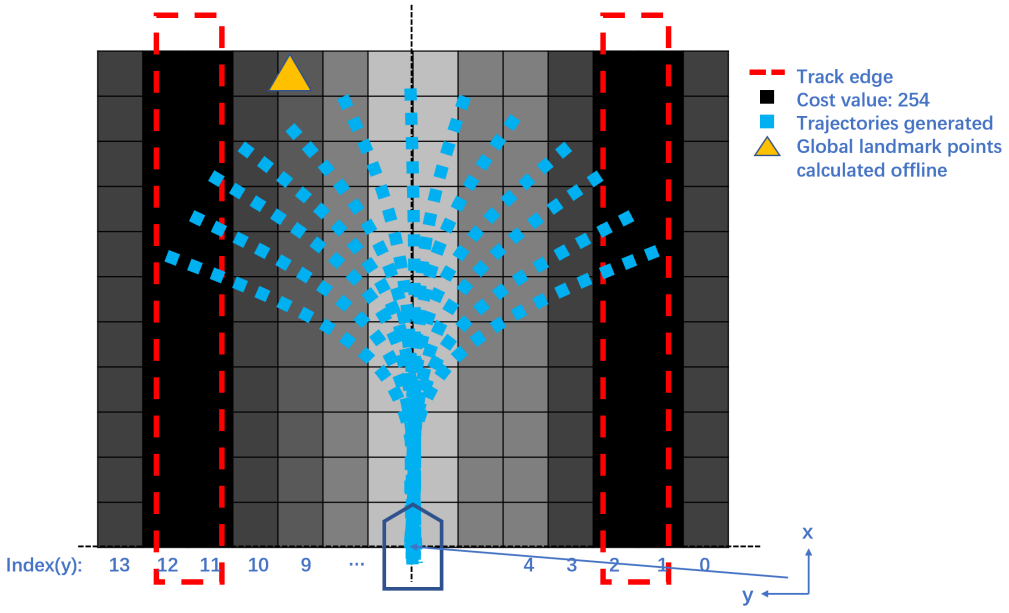


Figure 3.6: Schematic diagram of scoring.

With regard to following the global optimal path, in each planning, planner will find the appropriate global road landmark in the global optimal path according to the distance traveled and increase a certain distance. Then for each trajectory, planner calculates the distance between all trajectory points and this landmark point and sum them to get the $cost_{global}$.

About the safety of obstacle avoidance, the purpose of our local planner is to drive the optimal path as much as possible, so in order to cooperate with following the global optimal path, the safety requirements are desired not to be so strict while ensuring that the obstacles are not hit. Compared to the simple sum of the cost values of the grid corresponding to the trajectory points to get $cost_{safety}$, exponential function is used to process each cost value and then sum to get $cost_{safety}$. The advantage of exponential function is that when the cost value of the trajectory point is large and close to the obstacle, its value will be exponentially magnified, but when the cost value is small, the impact on $cost_{safety}$ will be small.

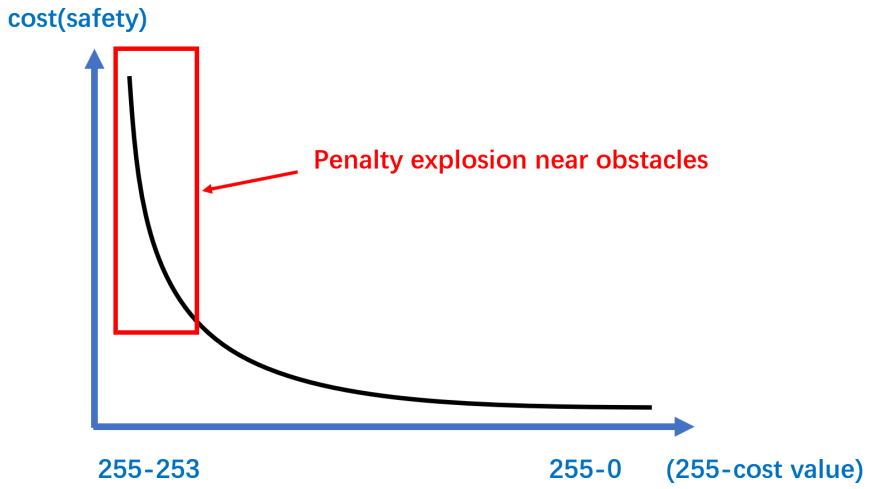


Figure 3.7: Exponential function for cost value in local map.

3.6 Perform steering input for optimal trajectory

When the optimal trajectory is selected, planner will execute the corresponding steering input, but in order to be as close to the optimal as possible, the planner will execute as fast as possible, and only the the steering input corresponding to the first point in the optimal trajectory will be executed after each plan. However, considering the car is still driving in the planning calculation, instead of the first trajectory point, the steering input corresponding to the trajectory point one meter forward will be selected for execution.

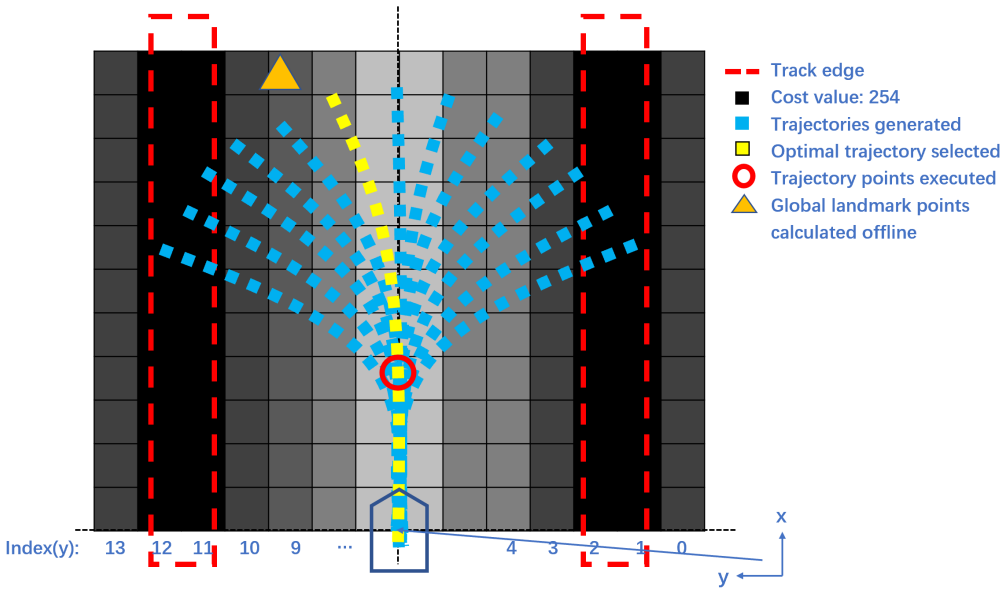


Figure 3.8: Perform steering input for optimal trajectory.

CHAPTER 4

Test results and analysis

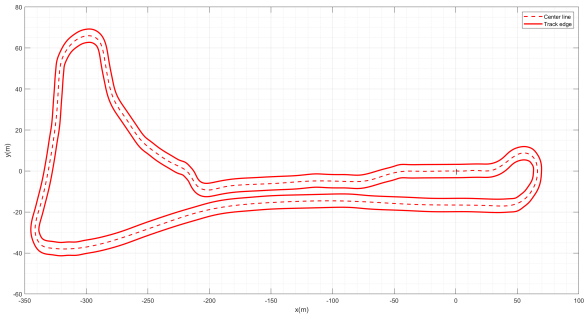


Figure 4.1: Track layout of ”chanllange1_London”.

The gazebo simulation environment tested is based on the Shell Eco-marathon 2018 in London. The local map system and local planner were tested in the simulation environment ”chanllange1_London” of Gazebo[Jen18]. Because the Gazebo environment is very slow and the time factor is 0.05, meaning that it takes 20s to simulate 1s. Part of the simulation process is recorded by video. Video1 is a normal speed video, and Video2 is an accelerated video.

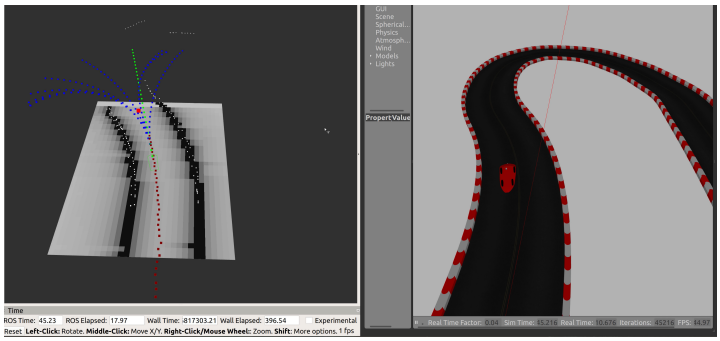


Figure 4.2: Gazebo simulation environment of ”chanllange1_London”.

Ecocar can safely and effectively follow the global optimal path calculated offline. At the same time, the real-time performance of the local map system and local planner has also been verified. The local map update frequency can reach 20hz; the planning frequency of the local planner can reach 5hz, and it ensures that the local map information can be read accurately and quickly in every planning. The data in the test is recorded by Rosbag, and the subsequent analysis is based on the data in the test. Local map updates during driving are recorded by Video3

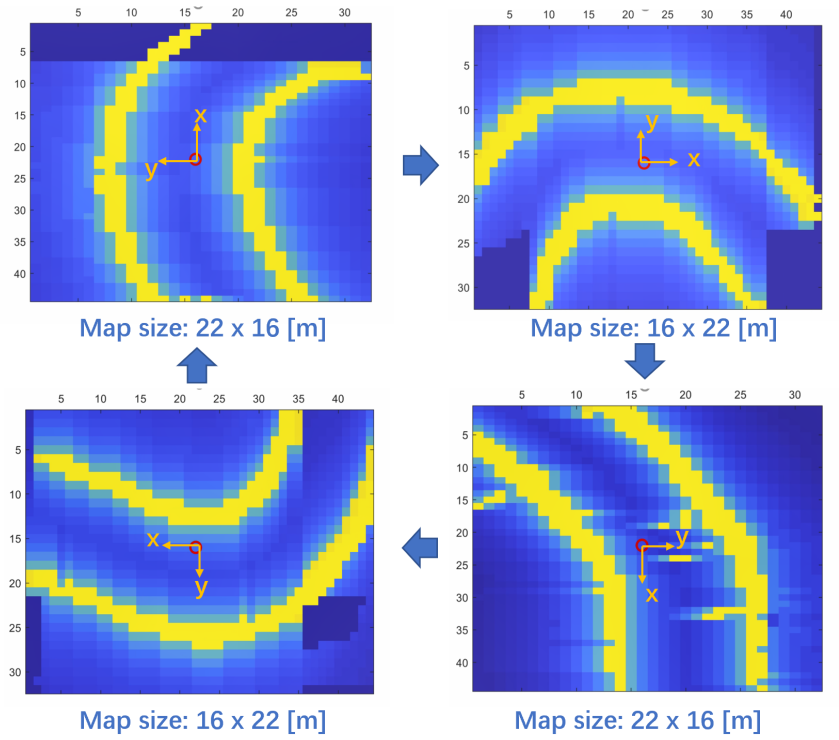


Figure 4.3: Local map update during test driving.

Figure 4.3 shows the update of the local map when driving on straights and curves in the test. It can be seen that the "disfill layer" has worked well in the completion of the track edge and the writing of grid information. Figure 4.3 also shows the automatic adjustment of the size and direction of the local map due to changes in the track during the test.

The difficulty of the task is to follow the curves, in order to achieve fuel efficiency and time saving, the offline optimization of the global path will tend to be more aggressive for too severe curves, but the uncertainty and errors during the operation will cause Unsafe driving. Therefore, The part of the data that contains the most severe curves in the track is selected for analysis.

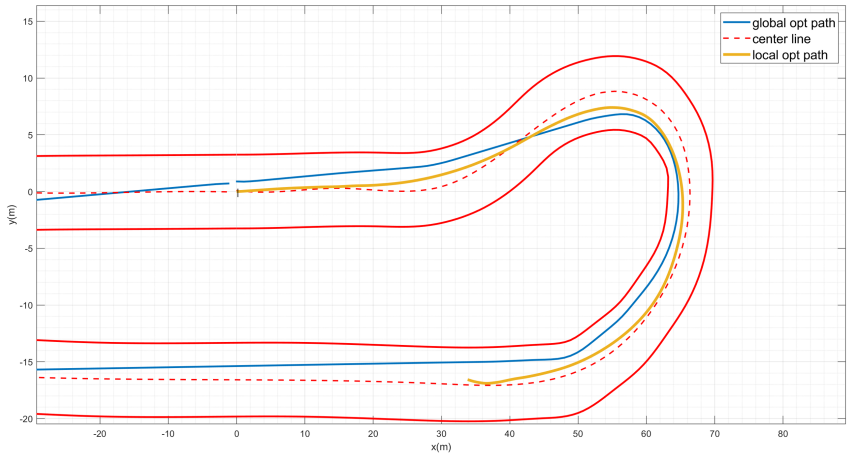


Figure 4.4: Driving trajectory in the test when using linear scoring.

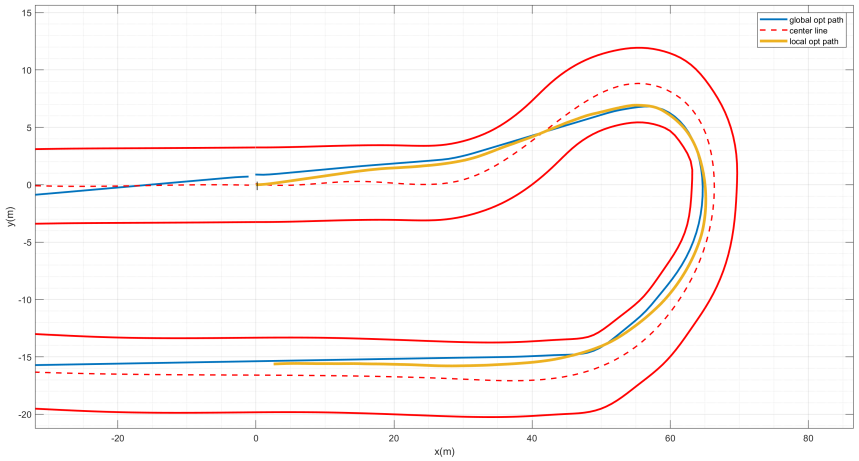


Figure 4.5: Driving trajectory in the test when using exponential scoring.

The first curve is the most severe curve in the track. Figures 4.4 and 4.5 show the driving trajectory at the first curve and the global trajectory calculated offline. Their difference lies in the scoring function for safety mentioned in the section of the local planner.

Figure 4.4 is the trajectory when the linear function of each grid cost value is directly added as the scoring function, and Figure 4.5 is the trajectory when the exponential function sum of each grid cost value is used as the scoring function. Obviously it can be seen that the driving trajectory in Figure 4.5 is closer to the global trajectory calculated offline, because the exponential function allows the planner to select a threshold. When the cost of the grid on which the trajectory is located is above this threshold, the trajectory will be punished a lot. On the contrary, when the value is below the threshold, the cost value will be infinitely reduced, which will make the scoring more dependent on following the global trajectory of offline calculation.

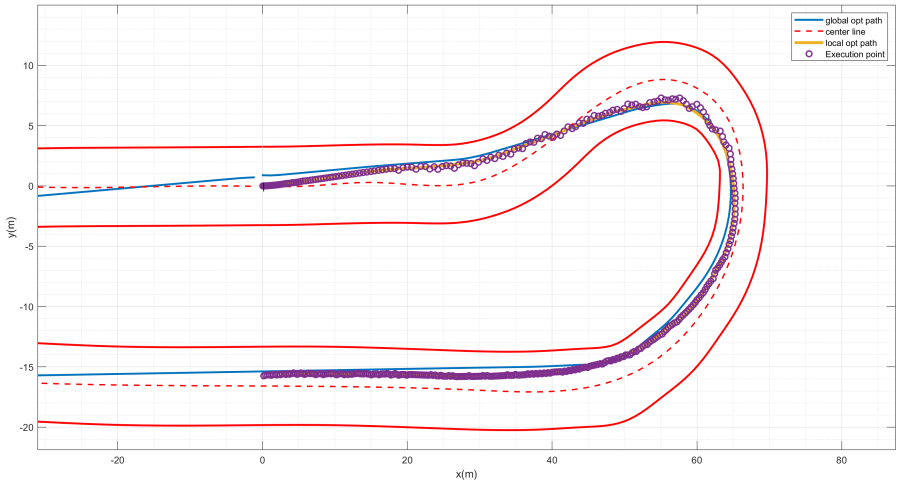


Figure 4.6: Driving trajectory and execution point in the test.

Figure 4.6 shows the execution point after each plan in the driving process. The steering angle input corresponding to the execution point will be executed in each planning cycle. It can be seen that the driving trajectory and the execution point almost fit, indicating that the execution point conforms to the vehicle model and is easy to execute.

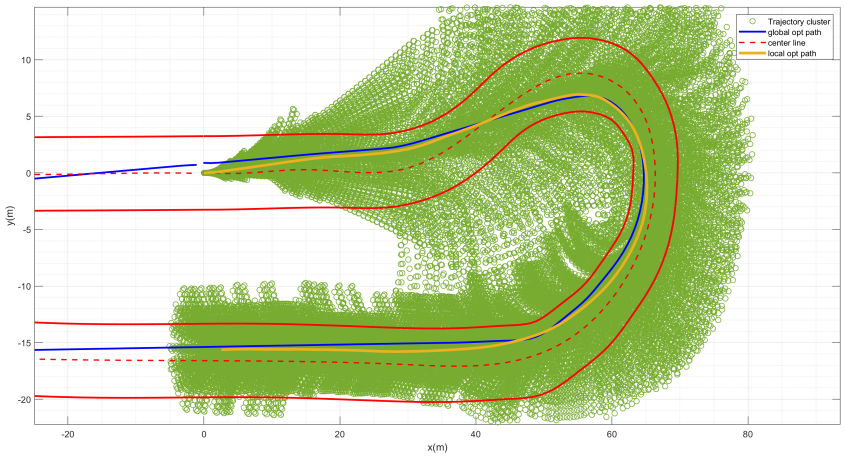


Figure 4.7: The driving trajectory and all generated trajectory cluster in the test.

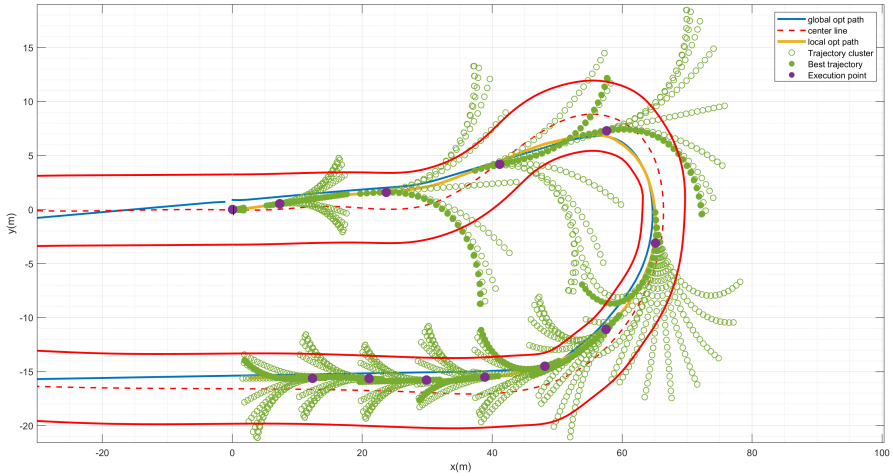


Figure 4.8: The driving trajectory , execution point, generated trajectory cluster and the trajectory with the best score in the test.

Since the planning frequency is 5hz, there are too many trajectory clusters generated during the whole driving process shown in Figure 4.7. Several sets of trajectory clusters is sampled and the corresponding optimal trajectories and execution points are shown in Figure 4.8. It can be seen that the

trajectory cluster is explored in the driving direction in each plan, and an optimal trajectory is selected and executed after scoring. As mentioned in the scoring chapter, when the vehicle is at a relatively high speed, sometimes the generated trajectory clusters may all be detected as collisions. At this time, the scoring will only rely on following the global trajectory calculated offline, but this situation does not happen often.

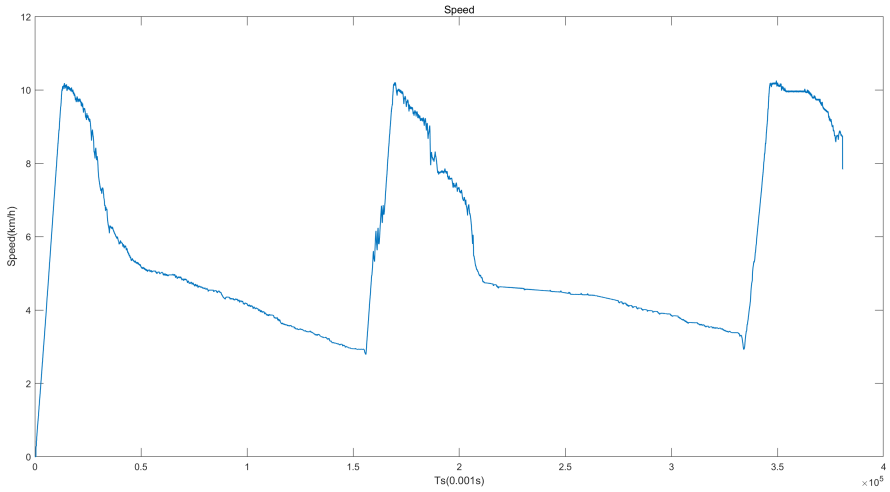


Figure 4.9: Speed in the test.

Figure 4.9 shows the speed during driving. It can be seen that the speed curve in the test is consistent with the engine mode: Turn on the engine to accelerate, accelerate to a certain speed and then turn off the engine to coast down to a certain speed and then accelerate again.

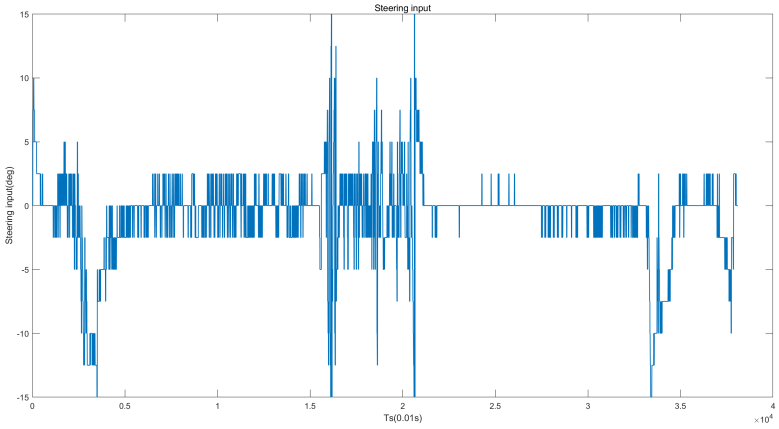


Figure 4.10: Steering input in the test.

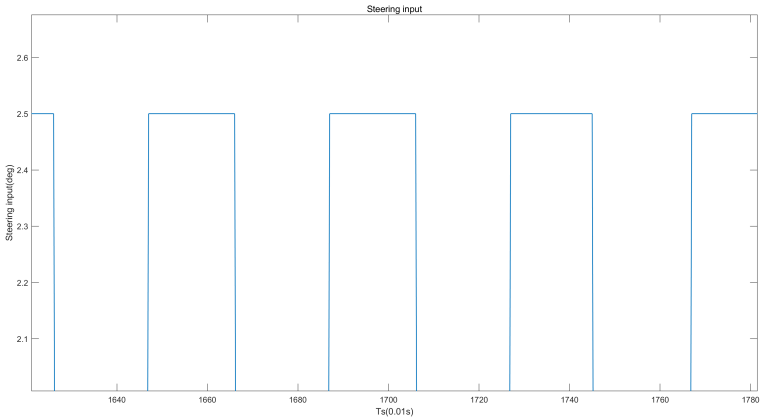


Figure 4.11: Part of steering input in the test.

Figure 4.10 shows the steering input during driving, and part of steering input during driving is shown in Figure 4.11. Since the input of the previous plan and the limit change speed of the steering are considered when constructing the steering input to generate the trajectory, the steering input performed in each plan is easy for the vehicle implemented. Furthermore, the steering angle input corresponding to the execution point will be maintained and executed during the planning cycle, so the vehicle has 0.2s to adjust (the planning frequency is 5hz), which increases the robustness of performing steering inputs while driving.

CHAPTER 5

Conclusion

In this report, the construction of the local map system and the local planner is described. The establishment of the local map system makes it more convenient for the navigation module to use the data processed by the perception module, and it also makes it possible to try more complex planning algorithms. The local planner comprehensively considers the safety of obstacle avoidance and following the global path calculated offline through the scoring method, so that the driving can be guaranteed to be effective in both aspects. At the same time, the scoring method has good scalability. For example, when more factors need to be considered in future driving, only the value function of the scoring needs to be changed.

In the test section, the ecocar successfully followed the global path calculated offline of challange london. Both local map system and local planner have proved its real-time and effectiveness. The local map update frequency is 20hz, and the planning frequency of the local planner is 5hz.

In future work, when the track environment is more complex (such as obstacles on the track), the local map system needs more complete functions to deal with, and this part also needs to cooperate with the point cloud processing of the perception module. In addition, because the local planner generates trajectories in the forward direction for searching, if the computational cost allows, the more trajectories generated, the greater the optimality of the selected trajectories. Finally, more accurate positioning (such as the integration of GPS and odometer) will also enhance the performance and robustness of the local planner.

Bibliography

[17] *About DTU Roadrunners*. Website. https://www.ecocar.mek.dtu.dk/english/about-dtu-roadrunners. 2017.

[Eit19] Dave Hershberger Eitan Marder-Eppstein David V. Lu. *Class Hierarchy*. http://docs.ros.org/en/indigo/api/costmap_2d/html/inherits.html. 2019.

[HK07] Thomas M. Howard and Alonzo Kelly. "Optimal rough terrain trajectory generation for wheeled mobile robots." In: *International Journal of Robotics Research* (2007), pages 141–166.

[How+08] Thomas M. Howard et al. "State Space Sampling of Feasible Motions for High-Performance Mobile Robot Navigation in Complex Environments." In: *J. Field Robot.* 25.6–7 (June 2008), pages 325–345. ISSN: 1556-4959.

[Jen18] Thomas Passer Jensen. *Pre-competition preparations for an autonomous Shell Eco-marathon car, Simulations, steering and braking*. 2018.

[Lar18] Jesper Martin Christensen Lars David Aktor. *Sensor Based Navigation for Autonomous Car*. 2018.

[LaV06] Steven M. LaValle. *Planning Algorithms*. USA: Cambridge University Press, 2006. ISBN: 0521862051.

[LHS14] D. V. Lu, D. Hershberger, and W. D. Smart. "Layered costmaps for context-sensitive navigation." In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. September 2014, pages 709–715. DOI: 10.1109/IROS.2014.6942636.

[Mad20] Tobias Stougaard Grønne Mads Dahl Larsen. *Optimal route planning and navigation for autonomous track driving*. 2020.

[Nic18] NickLamprianidis. *costmap$_2$d*. http://wiki.ros.org/costmap_2d. 2018.

$\sqrt{17}$
$\Omega \int \delta e^{i\pi} = -1$
{2.7182818284}
αβγδεζηθικλμνξοπρστυφχψω
χ^2
$\sum$