ÉCOLE CENTRALE DE NANTES

MASTER CORO-IMARO
"CONTROL AND ROBOTICS"

2019 / 2020

Master Thesis Report

Presented by

MA JIE

January 2019

One strategy for implementing EM planner with limited computing capability

Jury

Evaluators:	Ina Taralova	Maitre de conferences(ECN)
	Olivier Kermorgant	Professor(LS2N,ECN)
Supervisor(s):	BAI YU	Engineer (Mei Tuan,china)

Abstract

An Internet company - Baidu is working on the autonomous driving, they have introduced a local path planning algorithm - expectation maximum planner(EM planner), this is a sampling-based algorithm. We hope to apply this algorithm to AGV cars to achieve the purpose of automatic delivery.

Compared with AGV cars, autonomous vehicles have higher requirements for various algorithms. Autonomous vehicles need to face more complex road environments, and they must be able to respond quickly when driving at high speeds. We can apply the algorithm to the AGV cars, but considering the cost of logistics robots, their motherboards have lower computing capability, if we use the exhaustive search approach (brute-forced search), the EM planner can always make an ideal choice based on the cost function. However, due to the limited computing capability of the motherboard, we need to adopt appropriate strategies to make the right behavior quickly and well.

The objective of this bibliography report is to present the state-of-the-art work on EM planner, with focus on details of algorithm implementation.

Acknowledgements

First of all, I would like to thank Prof. Olivier Kermorgant agreed to my request to return to China for internship.

I am thankful to the Meituan can give me a great opportunity to work on a topic of my interest.

Notations

Ω	The 3D space-time
S	Statement
$\mathcal{V}$	Vehicle
$\mathcal{R}$	3D Space
$\mathcal{O}$	Obstacle
τ	Collision Test

Abbreviations

EM Planner	Expectation Maximum Planner
AGV	Automated Guided Vehicle
MP	Motion Planning
DP	Dynamic Programming
E-step	SL and ST Mapping
M-step	Dynamic Programming of Speed and Path

List of Figures

1.1 Ilustration of demonstrated driving scenario consisting of multi-lane driving in presence of traffic lights. [1] 10
1.2 Obstacle created by vehicle which is speeding up and slowing down. [1] 12
1.3 Obstacles created by two traffic lights. [1] 13
1.4 Pseudocode using A* for horizon [2] 16
1.5 Pseudocode using A* for expand function [2] 17
1.6 Expanding parent node n to different child nodes n' by piecewise constant acceleration.[2] 18
1.7 Predicting movement of other vehicle - linearization.[1] 19

2.1 EM Framework.[12] 22
2.2 EM Iteration.[12] 23
2.3 SL projection with oncoming traffic example.[12] 24
2.4 ST projection with cut-in obstacle and obstacle behind ego care.[12] 25
2.5 Dynamic programming structure. [12] 25
2.6 Dynamic programming path optimizer sampling for default lane and change lane. [12] 26
2.7 Spline QP Path Example. [12] 27
2.8 Spline QP Path Example. [12] 27
2.9 Spline QP Path Constraint Linearization. [12] 28
2.10 DP Speed Optimizer. [12] 29
2.11 DP Speed Optimizer. [12] 29
2.12 Spline QP speed optimizer. [12] 30
2.13 Spline QP speed optimizer. [12] 31
2.14 Case Study - Nudge oncoming dynamic obstacle This figure shows how the EM planner manages to iteratively solve the update path and speed profile. [12] 33

4.1 Simulation result. 35

Contents

Introduction **8**

1 Seach-based Algorithm **10**
- 1.1 Problem of Statement . . . 10
  - 1.1.1 Driving In Dynamic Environment . . . 10
  - 1.1.2 Search Space . . . 11
  - 1.1.3 Obstacles . . . 11
  - 1.1.4 Vehicle Model . . . 14
  - 1.1.5 Cost Function . . . 15
- 1.2 Optimal Motion Planning Framework . . . 15
  - 1.2.1 Framework . . . 15
  - 1.2.2 Node Expansion . . . 16
  - 1.2.3 Heuristic Function . . . 18
  - 1.2.4 Search Horizons . . . 19
  - 1.2.5 Vehicle Motion Prediction . . . 19

2 EM Motion Planner **21**
- 2.1 Multilane Strategy . . . 21
  - 2.1.1 The Architecture of EM planner . . . 22
  - 2.1.2 The Lane-Level Optimizer . . . 22
- 2.2 EM Planner at Lane Level . . . 23
  - 2.2.1 SL and ST Mapping . . . 24
  - 2.2.2 M-Step DP Path . . . 25
  - 2.2.3 M-Step Spline QP Path . . . 27
  - 2.2.4 M-Step DP Speed Optimizer . . . 29
  - 2.2.5 M-Step QP Speed Optimizer . . . 30
  - 2.2.6 Notes on Solving Quadratic Programming Problems . . . 31
- 2.3 Case Study . . . 32

3 Working Outline **34**
- 3.1 Summary of Objectives . . . 34
- 3.2 Work Plan . . . 34

4 Actual Work **35**

Conclusion **36**

Bibliography **36**

Introduction

Autonomous driving research began in the 1980s and has significantly grown over the past ten years. Autonomous driving aims to reduce road fatalities, increase traffic efficiency and provide convenient travel. However, autonomous driving is a challenging task that requires accurately sensing the environment, a deep understanding of vehicle intentions and safe driving under different scenarios.

The HD map module provides a high-definition map that can be accessed by every online module. Perception and localization modules provide the necessary dynamic environment information, which can be further used to predict future environment status in the prediction module. The motion planning module considers all information to generate a safe and smooth trajectory to feed into the vehicle control module.

In motion planner, safety is always the top priority. We consider autonomous driving safety in, but not limited to, the following aspects: traffic regulations, range coverage, cycle time efficiency and emergency safety. Even in the area of logistics robots, we need also to consider this. All these aspects are critical. Traffic regulations are designed by governments for public transportation safety, and such regulations also apply to autonomous driving vehicles. An autonomous driving vehicle should follow traffic regulations at all times. For range coverage, we aim to provide a trajectory with at least an eight second or two hundred meter motion planning trajectory. The reason is to leave enough room to maintain safe driving within regular autonomous driving vehicle dynamics. The execution time of the motion planning algorithm is also important. In the case of an emergency, the system could react within 100 ms,compared with a 300 ms reaction time for a normal human driver. A safety emergency module is the last shell for protecting the safety of riders. For a level-4 motion planner, once upstream modules can no longer function normally, the safety module within the motion planner shall respond with an immediate emergency behavior and send warnings that interrupt humans. Moreover, the autonomous driving system shall have the ability to react to emergencies in a lower-level-like control module.Further safety design is beyond this manuscript's scope. For the logistics robots, We also need to face the different scenarios.

In addition to safety, passengers' ride experience is also important. The measurement of ride experience includes, but is not limited to, scenario coverage, traffic regulation and comfort. For scenario coverage, the motion planner should not only be able to handle simple driving scenarios (e.g., stop, nudge, yield and overtake) but also handle multilane driving, heavy traffic and other complicated on-road driving scenarios. Planning within traffic regulations is also important for ride experience. In addition to being a safety requirement, following the traffic regulations will also minimize the risk of accidents and reduce emergency reactions for autonomous driving. The comfort during autonomous driving is also important. In motion planning, comfort is generally measured by the smoothness of the provided autonomous driving trajectory.

This report is organized as follows: Chapter 1 is delicated to search-based algorithm. Chapter 2 is delicated to EM planner.

Chapter 1

Seach-based Algorithm

For level-4 on-road autonomous driving motion planning, a lane-change strategy is necessary. One common approach is to develop a search algorithm with a cost functional on all possible lanes and then select a final trajectory from all the candidates with the lowest cost(See Figure 1.1).

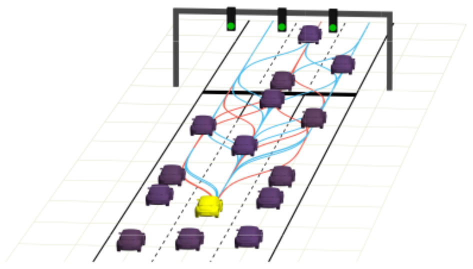


Figure 1.1: Ilustration of demonstrated driving scenario consisting of multilane driving in presence of traffic lights. [1]

1.1 Problem of Statement

In this section, we define the addressed problem. First, a general driving problem is defined, followed by the formulation of search space and obstacles. Finally, we provide the details about the vehicle model and the cost evaluation.

1.1.1 Driving In Dynamic Environment

Driving is a complex task consisting of continuous planning and execution in order to achieve desired goals and avoid collisions with other participants, obey traffic rules, comply with vehicle dynamics and factors like comfort, safety and efficiency. To fully automate driving, the vehicle has to be able to autonomously make decisions and plan its motion, while considering all mentioned requirements. The environment is usually highly dynamic, with speeds that may reach 50m/s or above. Moreover, the environment is complex, including many different participants, traffic rules,traffic control devices, etc. The mentioned conditions impose many different constraints on the driving trajectory.

The mayor challenges can be stated as: i) vehicle dynamics influence feasibility of the plans, ii) dynamic constraints are not known during initial planning, iii) the real motion of other participants deviates from the predicted one, iv) planning for long horizons with dynamic constraints is computationally expensive, v) long horizon planning is necessary to achieve long-term benefits such as energy efficiency, vi) conservative assumptions narrow down the search space, which can cause the loss of solution even in the case it exists. Therefore, a frequent replanning with long horizons is necessary while considering the vehicle model and environment as well.

1.1.2 Search Space

To tackle dynamic obstacles and avoid the risk of losing quality solutions, we use the 3D space-time Ω as a search space via Cartesian product of 2D configuration space and time dimension:

$$\Omega = \{q \equiv [t, s, l]^T \mid t \in \mathcal{R}^+, s \in \mathcal{R}^+, l \in [1, N_l]\} \tag{1.1}$$

Here, t is time, s is the longitudinal position along the road and l is the lateral position on a road. Dimension l is defined such that the middle of the rightmost lane has value 1 while the middle of the left-most lane has value N_l. The value 1.5 means that vehicle is halfway between lane 1 and 2.

1.1.3 Obstacles

We consider several types of constraints/obstacles, such as constraints imposed by other vehicles, traffic lights, speed limits and forbidden lane-change.

1)Vehicle obstacle: Other vehicles on the road represent obstacles for ego vehicle and constrain it's motion. The violation of these constraints can be manifested not only as a direct collision, but sometimes also as a violation of the driving rules, such as overtaking from right side or slow overtaking from the left side. For a certain vehicle $\mathcal{V}_k$, the trajectory of its center is described with $s_k(t)$ and $l_k(t)$, while suitable lower and upper bounds can be defined as:

$$L_s = L_k/2 + L_{ego}/2 \tag{1.2}$$

$$\underline{s}_k(t) = s_k(t) - L_s \tag{1.3}$$

$$\overline{s}_k(t) = s_k(t) + L_s \tag{1.4}$$

where L_k is the length of vehicle $\mathcal{V}_k$ and L_{ego} is the length of ego vehicle. Thus, theL_{ego} is practically incorporated within the obstacle so ego vehicle can be considered as a point. Based on this, the corresponding obstacle can be defined as:

$$\mathcal{O}_k^V = \{q \in \Omega \,|\, s(t) \in [\underline{s}_k(t), \overline{s}_k(t)]\} \tag{1.5}$$

A collision check for a given q, or the condition for which collision occurs can be validated by:

$$\tau(\exists q \in \mathcal{O}_k^V \,|\, l(t) \in (l_k(t) - 1, l_k(t) + 1)) \tag{1.6}$$

Where $\tau(S)$ denotes the logical value (1 or 0) of the statement S. The assumption made here is that each vehicle occupies the whole lane, so if ego vehicle center deviates from the middle of the lane, it is colliding with vehicle in the adjacent lane. It is important to note that this is different from driving in a lane when executing the plan, as the control is not ideal and the vehicle can deviate from the middle of the lane. Figure 1.2 shows an example of geometric representation of the vehicle obstacle within a defined search space Ω. The presented vehicle speeds up and then slows down.

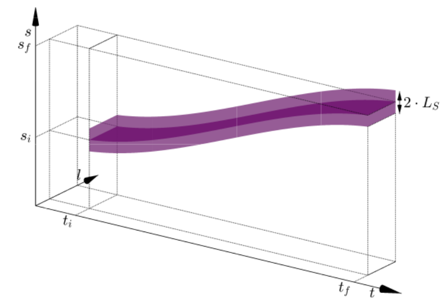


Figure 1.2: Obstacle created by vehicle which is speeding up and slowing down. [1]

Beside collision, it is sometimes forbidden to overtake the vehicle from the right side. This can be expressed by a collision-test given in (1.7). This is modeled by prohibiting velocities greater than the velocity of a vehicle on the left. Beside using the velocity limit, position is used so that in the case when a vehicle tries to overtake the ego vehicle, and ceases overtaking for some reason, the ego vehicle does not slow down too. The corresponding collision test is formulated via:

$$\tau(\exists q \in \mathcal{O}_k^V \,|\, l(t) \in [1, l_k(t) - 1], \frac{\partial s(t)}{t} \geq \frac{\partial s_k(t)}{t}, s_k(t) \geq s(t)) \tag{1.7}$$

Practically, overtaking a vehicle requires only a velocity greater than the velocity of a vehicle. However, to limit the time of the overtaking maneuver, in several countries (e.g. Austria) there is also a limit on the minimum velocity difference Δv_{ov}, when overtaking other vehicles. The corresponding collision test can be formulated as:

$$\tau(\exists q \in \mathcal{O}_k^V \mid l(t) \in [l_k(t)+1, N_l], \frac{\partial s(t)}{t} \leq \frac{\partial s_k(t)}{t} + \Delta v_{ov}) \tag{1.8}$$

In multilane urban driving scenarios, rules for overtaking are not applicable.

2)Traffic light obstacle: The traffic light is a traffic control device which prohibits passing the defined line, during specific periods in time on certain lane. Figure 1.3 shows obstacles created by two traffic lights across all lanes although they can be active on a single or several lanes. The obstacle is defined as:

$$\mathcal{O}_{ki}^{TL} = \{q \in \Omega \mid s = s_k, l = l_k, t \in [t_{ki}, t_{ki} + \Delta t_{ki}]\} \tag{1.9}$$

Each traffic light represents infinitely many obstacles, periodic in t, with the

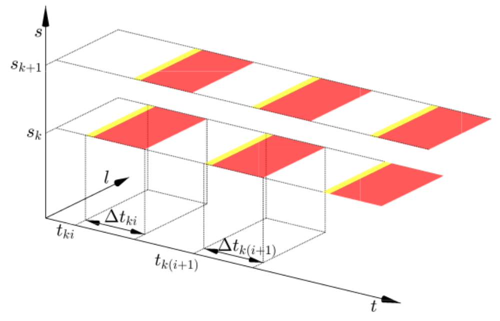


Figure 1.3: Obstacles created by two traffic lights. [1]

constant or variable period depending on the traffic light control system. The collision check is performed by evaluating:

$$\tau(q \cap \mathcal{O}_{ki}^{TL} \neq \emptyset) \tag{1.10}$$

The vehicle trajectory should not pass trough the region of the traffic light at the time when the red light is on.

3)Speed limit: Speed limits may originate from speed limit signs, road curvature or some other factors. They are defined on certain segment of the road and active in the following region of Ω:

$$\mathcal{O}_k^{TL} = \{q \in \Omega \mid s \in [s_k, s_k + \Delta s_k]\} \tag{1.11}$$

The collision check is validated by:

$$\tau(\exists q \in \mathcal{O}_k^{TL} \mid \frac{\partial s(t)}{t} \geq v_{k_{MAX}}) \tag{1.12}$$

On this segment, the vehicle velocity, represented by a gradient in direction of t, must not exceed the defined value.

4)Forbidden lane-change, solid line: Lane-change prohibition can be also defined on certain segments of the road. It is usually marked with the solid lane line. The obstacle representation is given as:

$$\mathcal{O}_{ki}^{LC} = \{q \in \Omega \,|\, s \in [s_k, s_k + \Delta s_k], l(t) \in (l_i, l_i + 1)\} \tag{1.13}$$

Prohibition may be applicable to both directions, where the collision check is performed by:

$$\tau(q \cap \mathcal{O}_{ki}^{LC} \neq \emptyset) \tag{1.14}$$

Alternatively, the prohibition can hold for single direction. Left-wise lane change prohibition is defined via collision test (1.15), while the right-wise is defined via negative partial derivative.

$$\tau(\exists q \in \mathcal{O}_{ki}^{LC} \,|\, \frac{l(t)}{t} > 0) \tag{1.15}$$

Obstacles formulated above are the most common constraints in everyday driving, and the majority of situations can be described by the combination of these. Clearly, multiple obstacles can be active at the same time. It is worth pointing out that the collision checking with respect to such defined obstacles appears to be rather trivial, since it usually reduces to closed-form analysis whether some elementary, analytically defined curves intersect or not, or if the gradient of these curves attain certain values.

1.1.4 Vehicle Model

To model the vehicle motion for planning purposes, longitudinal and lateral dynamics should be derived. Assuming that the vehicle orientation does not deviate much from the road direction, the longitudinal motion is given by:

$$v(t) = \frac{\partial s(t)}{t} \tag{1.16}$$

$$a(t) = \frac{\partial v(t)}{t} = \frac{F_m(t) - F_r(t)}{m} \tag{1.17}$$

where v(t) and a(t) are velocity and acceleration along s, and $F_m(t)\, and\, F_r(t)$are forces generated by the motor and resistive force respectively. The rest of the vehicle model is presented in [2] where resistive forces and powertrain losses are modeled in detail. The vehicle model is used for computing the cost of a transition between certain states,$costtrans(v_i, v_f, t_t)$.

Since planning includes lane changes as well, modeling the lateral motion is of particular importance. This is not straightforward because of the vehicle kinematics and dynamics. For planning purposes, it is important that the model is conservative so that resulting trajectory is feasible, yet not too conservative to disregard many feasible trajectories. Therefore, the lateral motion is modeled as linear in time such that the vehicle needs a specific time T_{LC} to execute the full lane change. This simplification limits the use of a lane change on smaller velocities, which is acceptable as it can be considered a parking maneuver. Alternatively, the clearance for the lane change on smaller velocities can be provided with by increasing safety buffer around the obstacles.

1.1.5 Cost Function

A cost function is used to evaluate quality of a given trajectory. It can reflect multiple goals such as: short travel time [3], comfort [4], safety [5], energy efficiency [2], traffic rules (driving on the rightmost lane [6]) or a combination of these [7]. The design of the cost function is particularly important as it influences vehicle behavior and defines the optimal solution. Cost function used in this work is reflecting energy efficiency and is explained in details in [2], [8] with the additional constant cost for each lane change. The cost function is kept simple while other desired behaviors are implicitly defined through the obstacle formulation.

1.2 Optimal Motion Planning Framework

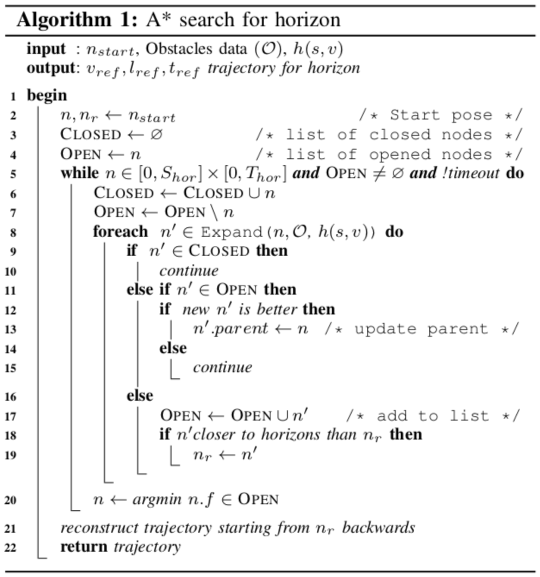
In this section, we elaborate on the proposed optimal MP framework. First, some general aspects of the framework are described, followed by the clarification of individual features.

1.2.1 Framework

The proposed framework is based on A* search method [9], guided by an exact cost-to-go map from a relaxed problem in an MPC-like replanning scheme. Each T_rep seconds, replanning is triggered with the current measurement of positions and velocities of other vehicles, traffic lights timing data together with the map data. Based on measurements, the motion of other vehicles is predicted and collision-free trajectory for a defined horizon is generated.

The trajectory is generated by a grid-like searching using A*. The grid is constructed by the discretization of t, s and l from the original continuous search space definition. Velocity v is additionally used to provide completeness because of the longitudinal dynamics of the model. Starting from the initial configuration, defined as the initial node, chosen as the first current node, all neighbors are determined by expanding the current node. The resulting child nodes are added to the OPEN list. If the child node is already in the OPEN list, and new child node has a lower cost, the parent of that node is updated, otherwise it is ignored. From the OPEN list, the node with the lowest cost is chosen to be the next current node and the procedure is repeated until horizon is reached, the whole graph is explored or the computation time limit for planning is reached. Finally, the node closest to the horizons is used to reconstruct the trajectory. The pseudocode for this procedure is presented in Figure 1.4.

To avoid rounding errors, as the expansion of node creates multiple transitions which in general do not end at gridpoints, the hybrid A* approach [10] is



```
Algorithm 1: A* search for horizon
input : $n_{start}$, Obstacles data ($\mathcal{O}$), $h(s,v)$
output: $v_{ref}, l_{ref}, t_{ref}$ trajectory for horizon
begin
    $n, n_r \leftarrow n_{start}$                    /* Start pose */
    CLOSED $\leftarrow \varnothing$          /* list of closed nodes */
    OPEN $\leftarrow n$          /* list of opened nodes */
    while $n \in [0, S_{hor}] \times [0, T_{hor}]$ and OPEN $\neq \varnothing$ and !timeout do
        CLOSED $\leftarrow$ CLOSED $\cup\, n$
        OPEN $\leftarrow$ OPEN $\setminus n$
        foreach $n' \in$ Expand($n, \mathcal{O}, h(s,v)$) do
            if $n' \in$ CLOSED then
                continue
            else if $n' \in$ OPEN then
                if new $n'$ is better then
                    $n'.parent \leftarrow n$  /* update parent */
                else
                    continue
            else
                OPEN $\leftarrow$ OPEN $\cup\, n'$     /* add to list */
                if $n'$closer to horizons than $n_r$ then
                    $n_r \leftarrow n'$
        $n \leftarrow$ argmin $n.f \in$ OPEN
    reconstruct trajectory starting from $n_r$ backwards
    return trajectory
```

Figure 1.4: Pseudocode using A* for horizon [2]

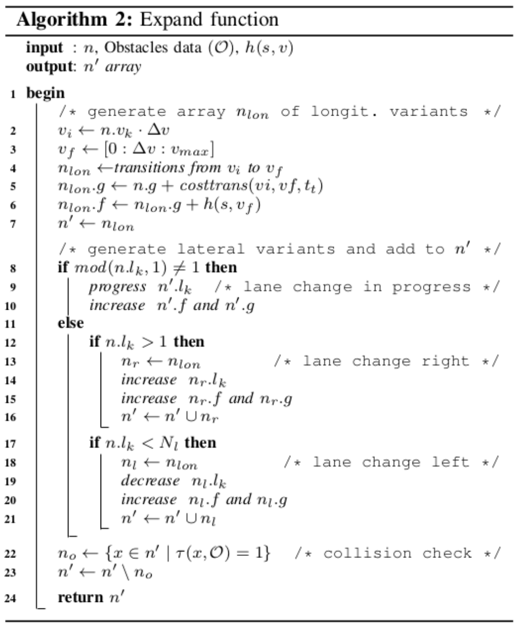
used for planning. Hybrid A* also uses the grid, but keeps continuous values for the next expansion without rounding it to the grid, thus preventing the accumulation of rounding errors.

As v belongs to the discrete set of values as defined in the expansion (Figure 1.5), the hybrid A* approach is used only for t, s, l. Therefore, each node n contains 14 values: four indices for $v, t, s, l, (n.v_k, n.t_k, n.s_k, n.l_k)$, four indices for the parent node (to reconstruct trajectory), three remainders from the discretization of t, s and$l(n.t_r, n.s_r, n.l_r)$, the direction of the lane-change $n.l_dir$, the exact cost-to-come to the node ($n.g$), and the estimated total cost of traveling from the initial node to the goal region ($n.f$). The value $n.f$ is computed as $n.g + h(n)$, where $h(n)$ is the heuristic function.

The planning clearly requires processing time. The compensation of the planning time can be achieved by introducing T_{plan} , a guaranteed upper bound on planning time. The planning is then initiated from a position where the vehicle would be after the T_{plan} . The old trajectory is executed while the new one is being processed. Thus, the new trajectory is already planed when T_{plan} arrives.

1.2.2 Node Expansion

To build trajectories iteratively, nodes are expanded and child nodes are generated, progressing toward the goal. From each node n, only dynamically feasible and collision-free child nodes n'should be generated. A single child node is generated for each possible longitudinal and lateral motion variant.



```
Algorithm 2: Expand function
input  : $n$, Obstacles data ($\mathcal{O}$), $h(s,v)$
output: $n'$ array
begin
    /* generate array $n_{lon}$ of longit. variants */
    $v_i \leftarrow n.v_k \cdot \Delta v$
    $v_f \leftarrow [0 : \Delta v : v_{max}]$
    $n_{lon} \leftarrow$ transitions from $v_i$ to $v_f$
    $n_{lon}.g \leftarrow n.g + costtrans(vi, vf, t_t)$
    $n_{lon}.f \leftarrow n_{lon}.g + h(s, v_f)$
    $n' \leftarrow n_{lon}$
    /* generate lateral variants and add to $n'$ */
    if $mod(n.l_k, 1) \neq 1$ then
        progress $n'.l_k$   /* lane change in progress */
        increase $n'.f$ and $n'.g$
    else
        if $n.l_k > 1$ then
            $n_r \leftarrow n_{lon}$          /* lane change right */
            increase $n_r.l_k$
            increase $n_r.f$ and $n_r.g$
            $n' \leftarrow n' \cup n_r$
        if $n.l_k < N_l$ then
            $n_l \leftarrow n_{lon}$          /* lane change left */
            decrease $n_l.l_k$
            increase $n_l.f$ and $n_l.g$
            $n' \leftarrow n' \cup n_l$
    $n_o \leftarrow \{x \in n' \mid \tau(x, \mathcal{O}) = 1\}$   /* collision check */
    $n' \leftarrow n' \setminus n_o$
    return $n'$
```

Figure 1.5: Pseudocode using A* for expand function [2]

The longitudinal motion variants are generated by assuming uniform accelerations from the inherited parent velocity, so that the discrete final velocities (represented by the array v_f) are reached at expansion limits. Expansion limits are defined by Δs_{exp} for distance and $\Delta t_{exp}$for time (Figure 1.6). Since trajectories reflect the motion with the uniform acceleration, the average velocity of a specific motion variant equals $v = (v_i + v_f)/2$. If $v < \Delta s_{exp}/\Delta t_{exp}$, the trajectory will end on time expansion limit $\Delta t = \Delta t_{exp}$. Otherwise, it will end on the distance expansion limit $\Delta s = \Delta s_{exp}$. Distance and time values (Δs and Δt) for each variant are summed with the parent node's remainders $n.s_r$ and $n.t_r$. Resulting sums are used to generate child nodes by increasing parent node's indices ($n.s_k$ and $n.t_k$) with the quotient of division of sums with discretization steps of the grid (Δs grid and Δt grid), and computing new remainders. For each child node from the array n', costs are computed as well. Cost-to-come is inherited from the parent and increased by the cost of transition. Cost-to-go is provided by the heuristic function explained in the following subsection. The compliance with the vehicle's internal constraints (e.g.maximum acceleration) is checked and the nodes that violate these constraints are removed.

Generated longitudinal motion variants are then used for lateral motion expansion. If the parent node is in the middle of the lane (l is the integer), variants for possible lane change right and left, beside staying in the lane are generated. Generated variants are tripled, one set of longitudinal variants for each. The values $n'.l_k$ and $n'.l_r$ are increased or decreased for the lane changes

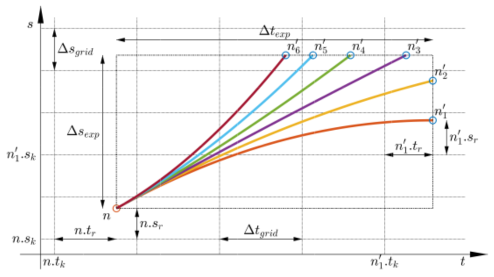


Figure 1.6: Expanding parent node n to different child nodes n' by piecewise constant acceleration.[2]

left or right respectively. They are modified by the value $\Delta t/T_{LC}$ based on the travel time of the particular variant, and the defined lane change time T_{LC} . If the parent node is already in the process of lane change, lane change is progressed without generating other lateral motion variants. Finally, compliance with obstacles such as other vehicles, traffic lights, etc. is checked and all child nodes and motion variants that are not collision-free are removed.

1.2.3 Heuristic Function

The heuristic function $h(n)$ is used to estimate the cost needed to travel from some node n to the goal state (cost-to-go). As it is shown in [9], if the heuristic function is underestimating the exact cost to go, A* search provides the optimal trajectory. For the shortest path search, the usual heuristic function is the Euclidian distance. To find the energy optimal velocity trajectory, the heuristic function must underestimate the energy needed to drive to the goal. In this framework, the cost-to-go map resulting from the backward search for the relaxed problem is used as heuristic function. The cost-to-go computation phase is executed only once at the beginning of the trip, or if the goal is changed. The computation is performed by using backward dynamic programming (DP), starting from the goal state (s and v), backward in s, as it was shown in [2].

In this phase, only time invariant constraints are considered (e.g. speed limits) with topological road profile and the vehicle model without considering time-varying constraints. The resulting cost-to-go map is an admissible heuristic, as other vehicles can prohibit certain regions of the state space, which may only increase the cost to travel from the initial state to the goal region. This is valid if platooning effects are neglected, as platooning can potentially reduce the airdrag effect (which is considered in the initial computation), and decrease the cost of travel, but for compact vehicles, this effect is usually negligible. Heuristic function $h(s, v)$ depends only on s and v. The resulting heuristic function is applied based on child node's s and v values, while neglecting t and l values.

1.2.4 Search Horizons

The planing is performed until any of the trajectories reaches time (T_{hor}) or distance (S_{hor}) search horizon. Slower driving trajectories will reach the time, while faster trajectories will reach the distance horizon. Thus, the unnecessary planning can be avoided. If only one is chosen (e.g., T_{hor}) other one could adopt a large even infinite value. The search horizons should not be confused with local expansion limits, which uses a similar principle, but represents atomic motion segments when building the whole trajectory.

1.2.5 Vehicle Motion Prediction

Though it is required for prediction of potential collisions, the perfect knowledge of the future motion of other vehicles is not available in principle. A naive way to predict the motion is to assume that the vehicles will continue to drive with their current velocity and stay in the current lane. On the other hand, a motion planning framework should provide collision-free plans even if trajectories deviate from the predicted one and the environment perception system introduces estimation errors. Therefore, safety buffers are used to increase obstacle regions, and frequent replanning is executed. The approach introduced in this framework is that the most intuitive prediction of driving (constant velocity) is used for finding the optimal trajectory, but an additional safety mechanism ensures a collision-free plan even for the worst case error regarding the relative distance estimation. This is provided by adding a step-like safety buffer to the obstacle. The lower and upper bound of the vehicle obstacle can be defined as:

$$\underline{\hat{s}}_k(t) = \hat{s}_k(t_0) + \hat{v}_k(t_0)(t - t_0) - L_s - s_b(t) \tag{1.18}$$

$$\hat{\bar{s}}_k(t) = \hat{s}_k(t_0) + \hat{v}_k(t_0)(t - t_0) + L_s + s_b(t) \tag{1.19}$$

$$\begin{cases} \Delta s_{max}, & t_0 \le t \le t_0 + T_{rep} \\ 3\Delta s_{max}, & t_0 + T_{rep} \le t \le t_0 + T_{hor} \end{cases} \tag{1.20}$$

where Δs_{max} is the maximum error of the vehicle relative distance estimation.

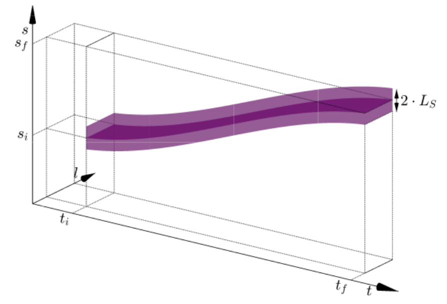


Figure 1.7: Predicting movement of other vehicle - linearization.[1]

The safety buffer s_b is increased after T_{rep} (the next replanning instance) to maintain robustness, so that in the next replanning instance, the vehicle always starts from the position that is collision-free according to a new safety buffer. This is visualized in the Figure 1.7, showing the worst case scenario. The estimation error is such that in the first planning instance, Δs_{max} is positive, while in the second instance, it is negative. It can be seen that the safety buffer from the first planning instance ensures that the trajectory is outside of obstacle area for the time interval $[t_0, t_0 + T_{rep}]$ and the trajectory is outside of the safety buffer from the second planning instance for the time interval $[t_0 + T_{rep}, t_0 + 2T_{rep}]$. This safety buffer provides a partial robustness for deviations from the predicted trajectory as well, but no guarantees can be provided.

Chapter 2

EM Motion Planner

This manuscript presents the Apollo EM planner, which is based on an EM-type iterative algorithm([11]). This planner targets safety and ride experience with a multilane, path-speed iterative, traffic rule and decision combined design.

2.1 Multilane Strategy

In the chapter1, we talk about one common approach is to develop a search algorithm with a cost functional on all possible lanes, This approach has some difficulties. First, the search space is expanded across multiple lanes, which causes the algorithm to be computationally expensive. Second, traffic regulations (e.g., right of the road, traffic lights) are different across lanes, and it is not easy to apply traffic regulations under the same frame. Furthermore, trajectory stability that avoids sudden changes between cycles should be taken into consideration. It is important to follow consistent on-road driving behavior to inform other drivers of the intention of the autonomous driving vehicle.

Typically, a multilane strategy should cover both nonpassive and passive lane-change scenarios. In EM planner, a nonpassive lane-change is a request triggered by the routing module for the purpose of reaching the final destination. A passive lane change is defined as an ego car maneuver when the default lane is blocked by the dynamic environment. In both passive and nonpassive lane changes, we aim to deliver a safe and smooth lane change strategy with a high success rate. Thus, we propose a parallel framework to handle both passive and nonpassive lane changes. For candidate lanes, all obstacles and environment information are projected on lane-based Frenet frames. Then, the traffic regulations are bound with the given lane level strategy. Under this framework, each candidate lane will generate a best-possible trajectory based on the lane-level optimizer. Finally, a crosslane trajectory decider will determine which lane to choose based on both the cost functional and safety rules.

2.1.1 The Architecture of EM planner

Figure 2.1 presents an overview of EM planner. On top of the planner, all sources of information are collected and synced at the data center module. After data collection, the reference line generator will produce some candidate lane-level reference lines along with information about traffic regulations and obstacles. This process is based on the high-definition map and navigation information from the routing module. During lane-level motion planning, we first construct a Frenet frame based on a specified reference line. The relation between the ego car and its surrounding environment is evaluated in the Frenet frame constructed by the reference line, as well as traffic regulations. Furthermore, restructured information passes to the lane-level optimizer.

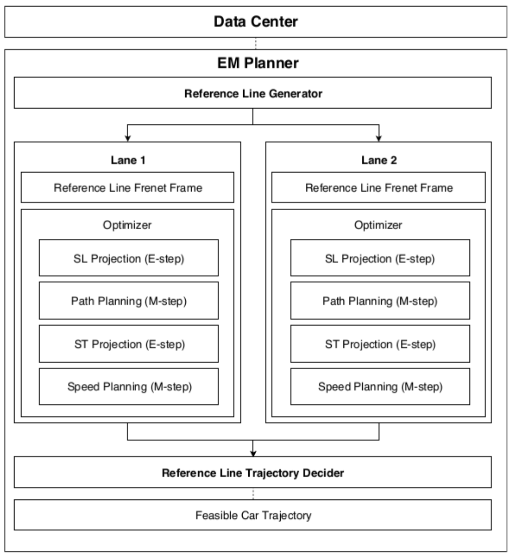


Figure 2.1: EM Framework.[12]

2.1.2 The Lane-Level Optimizer

The lane-level optimizer module performs path optimization and speed optimization. During path optimization, information about the surroundings is projected on the Frenet frame(E-step).Based on the information projected in the Frenet frame, a smooth path is generated (M-step). Similarly, during speed optimization, once a smooth path is generated by the path optimizer, obstacles are projected on the station-time graph (E-step). Then, the speed optimizer will generate a smooth speed profile (M-step). Combining path and speed profiles, we will obtain a smooth trajectory for the specified lane. In the last step, all lane-level best trajectories are sent to the reference line trajectory decider.

Based on the current car status, regulations and the cost of each trajectory, the trajectory decider will decide a best trajectory for the ego car maneuver.

2.2 EM Planner at Lane Level

In this section, we discuss the lane-level optimization problem. Figure 2.1 shows the path-speed EM iteration inside lane-level planning. The iteration includes two E-steps and two M-steps in one planning cycle. The trajectory information will iterate between planning cycles. We explain the submodules as follows.

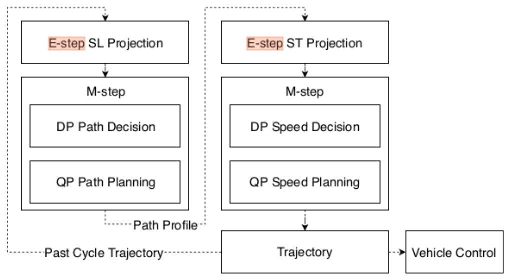


Figure 2.2: EM Iteration.[12]

In the first E-step, obstacles are projected on the lane Frenet frame. This projection includes both static obstacle projection and dynamic obstacle projection. Static obstacles will be projected directly based on a Cartesian-Frenet frame transformation. In the Apollo framework, the intentions of dynamic obstacles are described with an obstacle moving trajectory. Considering the previous cycle planning trajectory, we can evaluate the estimated dynamic obstacle and ego car positions at each time point. Then, the overlap of dynamic obstacles and the ego car at each time point will be mapped in the Frenet frame. In addition, the appearance of dynamic obstacles during path optimization will eventually lead to nudging. Thus, for safety considerations, the SL projection of dynamic obstacles will only consider low-speed traffic and oncoming obstacles. For high-speed traffic, EM planner's parallel lane-change strategy will cover the scenario. In the second E-step, all obstacles, including high-speed, low-speed and oncoming obstacles, are evaluated on the station-time frame based on the generated path profile. If the obstacle trajectory has overlap with the planned path, then a corresponding region in the station-time frame will be generated.

In two M-steps, path and speed profiles are generated by a combination of dynamic programming and quadratic programming. Although we projected obstacles on SL and ST frames, the optimal path and speed solution still lies in a non-convex space. Thus, we use dynamic programming to first obtain a rough solution; meanwhile, this solution can provide obstacle decisions such as nudge, yield and overtake. We use the rough decision to determine a convex hull for the quadratic-programming-based spline optimizer. Then, the optimizer can find solutions within the convex hull. We will cover the modules in the following.

2.2.1 SL and ST Mapping

The SL projection is based on contin- uous curvature derivative smooth reference line. In Cartesian space, obstacles and the ego car status are described with location and heading $(x, y,)$, as well as curvature and the derivative of curvature (κ, d_κ) for the ego car. Then, these are mapped to the Frenet frame coordinates $(s, l, dl, ddl, dddl)$, which represent station, lateral, and lateral derivatives. Since the positions of static obstacles are time invariant, the mapping is straight forward. For dynamic obstacles, we mapped the obstacles with the help of the last cycle trajectory of the ego car. The last cycle's moving trajectory is projected on the Frenet frame to extract the station direction speed profile. This will provide an estimate of the ego car's station coordinates given a specific time. The estimated ego car station coordinates will help to evaluate the dynamic obstacle interactions.

Once an ego car's station coordinates have interacted with an obstacle trajectory point with the same time, a shaded area on the SL map will be marked as the estimated interaction with the dynamic obstacle. Here, the interaction is defined as the ego car and obstacle bounding box overlapping. For example, as shown in Figure 2.3, an oncoming dynamic obstacle and corresponding

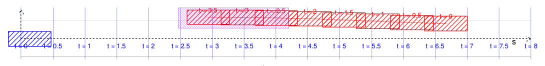


Figure 2.3: SL projection with oncoming traffic example.[12]

trajectory estimated from the prediction module are marked in red. The ego car is marked in blue. The trajectory of the oncoming dynamic obstacle is first discretized into several trajectory points with time, and then the points are projected to the Frenet frame. Once we find that the ego car's station coordinates have an interaction with the projected obstacle points, the overlap region (shown in purple in the figure 2.3) will be marked in the Frenet frame.

ST projection helps us evaluate the ego car's speed profile. After the path optimizer generates a smooth path profile in the Frenet Frame, both static obstacle and dynamic obstacle trajectories are projected on the path if there are any interactions.An interaction is also defined as the bounding boxes overlapping. In Figure 2.4, one obstacle cut into the ego driving path at 2 seconds and 40 meters ahead, as marked in red, and one obstacle behind the ego car is marked in green. The remaining region is the speed profile feasible region. The speed optimization M-step will attempt to find a feasible smooth solution in this region.

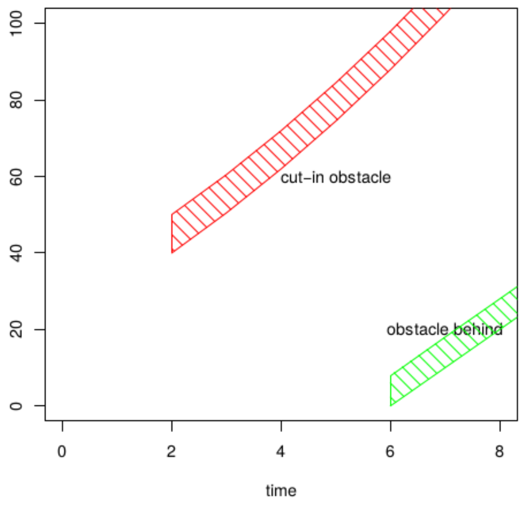


Figure 2.4: ST projection with cut-in obstacle and obstacle behind ego care.[12]

2.2.2 M-Step DP Path

The M-step path optimizer optimizes the path profile in the Frenet frame. This is represented as finding an optimal function of lateral coordinate $l = f(s)$ w.r.t. station coordinate in nonconvex SL space (e.g., nudging from left and right might be two local optima). Thus, the path optimizer includes two steps: dynamic-programming-based path decision and spline-based path planning. The dynamic programming path step provides a rough path profile with feasible tunnels and obstacle nudge decisions. As shown in Figure 2.5, the step includes a lattice sampler, cost function and dynamic programming search.

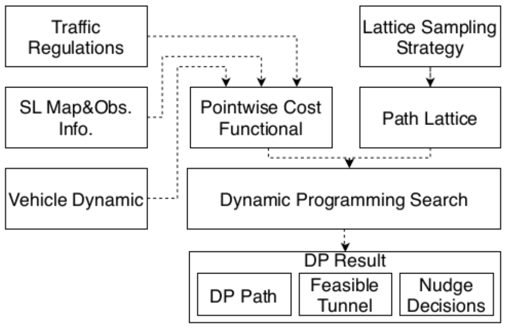


Figure 2.5: Dynamic programming structure. [12]

The lattice sampler is based on a Frenet frame. As shown in Figure 2.6, multiple rows of points are first sampled ahead of the ego vehicle. Points between different rows are smoothly connected by quintic polynomial edges. The interval distance between rows of points depends on the speed, road structure, lane change and so forth. The framework allows customizing the sampling strategy based on application scenarios. For example, a lane change might need a longer sampling interval than current lane driving. In addition, the lattice total station distance will cover at least 8 seconds or 200 meters for safety considerations.

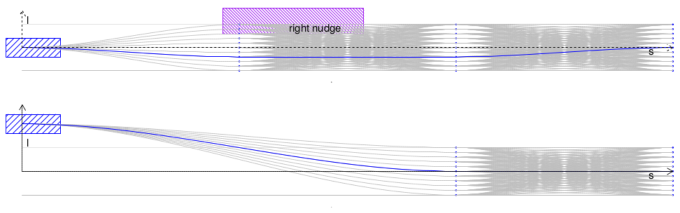


Figure 2.6: Dynamic programming path optimizer sampling for default lane and change lane. [12]

After the lattice is constructed, each graph edge is evaluated by the summation of cost functionals. We use information from the SL projection, traffic regulations and vehicle dynamics to construct the functional. The total edge cost functional is a linear combination of smoothness, obstacle avoidance and lane cost functionals.

$$C_{total}(f(s)) = C_{smooth}(f) + C_{obs}(f) + C_{guidence}(f) \tag{2.1}$$

The smoothness functional for a given path is measured by:

$$C_{smooth}(f) = w_1 \int (f'(s))^2 ds + w_2 \int (f^{''}(s))^2 ds + w_3 \int (f^{'''}(s))^2 ds \tag{2.2}$$

In the smoothness cost functional, $f'(s)$ represents the heading difference between the lane and ego car, $f^{''}(s)$ is related to the curvature of the path, and $f^{'''}(s)$ is related to the derivative of the curvature of the ego car. With the form of polynomials, the above cost can also be evaluated analytically.

The obstacle cost given an edge is evalu- ated at a sequence of fixed station coordinates $\{s_0, s_1, ..., s_n\}$ with all obstacles. The obstacle cost functional is based on the bounding box distance between the obstacle and ego car. Denote the distance as d. The form of individual cost is given by:

$$C_{obs}(d) = \begin{cases} 0, & d > d_n \\ C_{nudge}(d - d_c), & d_c < d < d_n \\ C_{collision}, & d < d_c \end{cases} \tag{2.3}$$

where C_{nudge} is defined as a monotonically decreasing function. d_c is set to leave a buffer for safety considerations. The nudge range d n is negotiable based on the scenario. $C_{collision}$ is the collision cost, which has a large value that helps to detect infeasible paths.

The lane cost includes two parts: guidance line cost and on-road cost. The guidance line is defined as an ideal driving path when there are no surrounding obstacles. This line is generally extracted as the centerline of the path. Define the guidance line function as $g(s)$. Then, it is measured as:

$$C_{guidence}(f) = \int (f(s) - g(s))^2 ds \tag{2.4}$$

The on-road cost is typically determined by the road boundary. Path points that are outside the road will have a high penalty.

The final path cost is a combination of all these smooth, obstacle and lane costs. Then, edge costs are used to select a candidate path with the lowest cost through a dynamic programming search. The candidate path will also determine the obstacle decisions. For example, in Figure 2.6, the obstacle is marked as a nudge from the right side.

2.2.3 M-Step Spline QP Path

The spline QP path step is a refinement of the dynamic programming path step. In a dynamic programming path, a feasible tunnel is generated based on the selected path. Then, the spline-based QP step will generate a smooth path within this feasible tunnel, as shown in Figure 2.7.

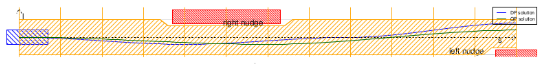


Figure 2.7: Spline QP Path Example. [12]

The spline QP path is generated by optimizing an objective function with a linearized constraint through the QP spline solver. Figure 2.8 shows the pipeline of the QP path step.

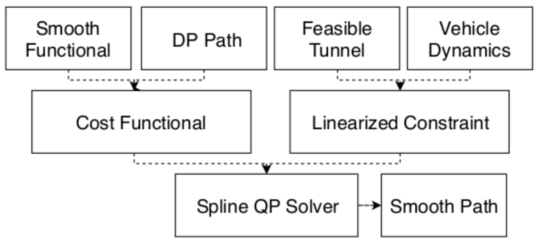


Figure 2.8: Spline QP Path Example. [12]

The objective function of the QP path is a linear combination of smoothness costs and guidance line cost. The guidance line in this step is the DP path. The guidance line provides an estimate of the obstacle nudging distance. Mathematically, the QP path step optimizes the following functional:

$$C_s(f) = w_1 \int (f'(s))^2 ds + w_2 \int (f^{''}(s))^2 ds + w_3 \int (f^{'''}(s))^2 ds + w_4 \int (f(s) - g(s))^2 ds \tag{2.5}$$

where $g(s)$ is the DP path result. $f'(s)$, $f^{''}(s)$ and $f^{'''}(s)$ are related to the heading, curvature and derivative of curvature. The objective function describes the balance between nudging obstacles and smoothness.

The constraints in the QP path include boundary constraints and dynamic feasibility. These constraints are applied on $f(s)$, $f'(s)$ and $f^{''}(s)$ at a sequence of station coordinates s_0 , s_1 , ..., s_n. To extract boundary constraints, the feasible ranges at station points are extracted. The feasible range at each point is described as $(l_{low,i} \leq l_{high,i})$. In EM planner, the ego vehicle is considered under the bicycle model. Thus, simply providing a range for $l = f(s)$ is not sufficient since the heading of the ego car also matters.

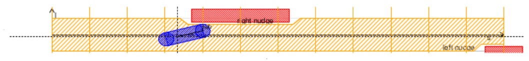


Figure 2.9: Spline QP Path Constraint Linearization. [12]

As shown in Figure 2.9, to keep the boundary constraint convex and linear, we add two half circles on the front and rear ends of the ego car. Denote the front-to-rear wheel center distance as l_f and the vehicle width as w. Then, the lateral position of the left-front corner is given by

$$l_{left\,front\,corner} = f(s) + \sin\theta l_f + w/2 \tag{2.6}$$

where θ is the heading difference between the ego car and road station direction. The constraint can be further linearized using the following inequality approximation:

$$f(s) + \sin\theta l_f + w/2 \leq f(s) + f'(s) l_r + w/2 \tag{2.7}$$

Similarly, the linearization can be applied on the remaining three corners. The linearized constraints are good enough since   is generally small. For $\theta < \pi/12$, the estimation will be less than 2 - 3cm conservative on the lateral direction compared to the constraint without linearization.

The range constraints of $f^{''}(s)$ and $f_{'''}(s)$ can also be used as dynamic feasibility since they are related to curvature and the curvature derivative. In addition to the boundary constraint, the generated path shall match the ego car's initial lateral position and derivatives $(f(s_0), f'(s_0), f^{''}(s_0))$. Since all constraints are linear with respect to spline parameters, a quadratic programming solver can be used to solve the problem very fast.

2.2.4 M-Step DP Speed Optimizer

The speed optimizer generates a speed profile in the ST graph, which is represented as a station function with respect to time $S(t)$. Similar to in the path optimizer, finding a best speed profile on the ST graph is a non-convex optimization problem. We use dynamic programming combined with spline quadratic programming to find a smooth speed profile on the ST graph. In Figure 2.10, the DP speed step includes a cost functional, ST graph grids and dynamic programming search. The generated result includes a piecewise linear speed profile, a feasible tunnel and obstacle speed decisions, as shown in Figure 2.11. The speed profile will be used in the spline QP speed step as a guidance line, and the feasible tunnel will be used to generate a convex region.

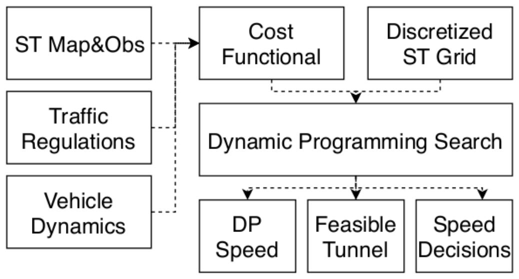


Figure 2.10: DP Speed Optimizer. [12]

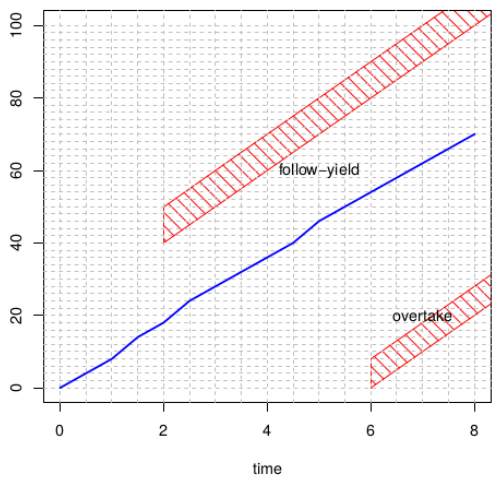


Figure 2.11: DP Speed Optimizer. [12]

In detail, obstacle information is first discretized into grids on the ST graph.

Denote $(t_0, t_1, ..., t_n)$ as equally spaced evaluated points on the time axis with interval dt. A piecewise linear speed profile function is represented as $S = (s_0, s_1, ..., s_n)$ on the grids. Furthermore, the derivatives are approximated by the finite difference method.

The goal is to optimize a cost functional in the ST graph within the constraints. In detail, the cost for the DP speed optimizer is represented as follows:

$$C_{total}(S) = w_1 \int_{t_0}^{t_n} g(S' - V_{ref})dt + w_2 \int_{t_0}^{t_n} (S^{''})^2 dt + w_3 \int_{t_0}^{t_n} (S^{'''})^2 dt + w_4 C_{obs}(S) \tag{2.8}$$

The first term is the velocity keeping cost. This term indicates that the vehicle shall follow the designated speed when there are no obstacles or traffic light restrictions present. V_{ref} describes the reference speed, which is determined by the road speed limits, curvature and other traffic regulations. The g function is designed to have different penalties for values that are less or greater than V_{ref}. The acceleration and jerk square integral describes the smoothness of the speed profile. The last term, C_{obs} , describes the total obstacle cost. The distances of the ego car to all obstacles are evaluated to determine the total obstacle costs.

The dynamic programming search space is also within the vehicle dynamic constraints. The dynamic constraints include acceleration, jerk limits and a monotonicity constraint since we require that the generated trajectories do not perform backing maneuvers when driving on the road. Backing can only be performed under parking or other specified scenarios. The search algorithm is straightforward; some necessary pruning based on vehicle dynamic constraints is also applied to accelerate the process. We will not discuss the search part in detail.

2.2.5 M-Step QP Speed Optimizer

Since the piecewise linear speed profile cannot satisfy dynamic requirements, the spline QP step is needed to fill this gap. In Figure 2.12, the spline QP speed step includes three parts: cost functional, linearized constraint and spline QP solver.

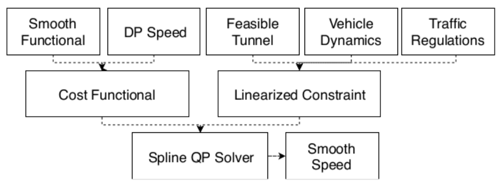


Figure 2.12: Spline QP speed optimizer. [12]

The cost functional is described as follows:

$$C_{total}(S) = w_1 \int_{t_0}^{t_n} (S - S_{ref})^2 dt + w_2 \int_{t_0}^{t_n} (S^{''})^2 dt + w_3 \int_{t_0}^{t_n} (S^{'''})^2 dt \tag{2.9}$$

The first term measures the distance between the DP speed guidance profile S_{ref} and generated path S. The acceleration and jerk terms are measures of the speed profile smoothness. Thus, the objective function is a balance between following the guidance line and smoothness.

The spline optimization is within the linearized constraint. The constraints in QP speed optimization include the following boundary constraints:

$$\begin{gathered} S(t_i) < S_(t_{i+1}),\ i = 0, 1, 2, ..., n-1, \\ S_{l,t_i} \leq S(t_i) \leq S_{u,t_i}, \\ S'(t_i) \leq V_{upper} \end{gathered} \tag{2.10}$$

as well as constraints for matching the initial velocity and acceleration. The first constraint is monotonicity evaluated at designated points. The second, third, and fourth constraints are requirements from traffic regulations and vehicle dynamic constraints. After wrapping up the cost objective and constraints, the spline solver will generate a smooth feasible speed profile as in Figure 2.13. Combined with the path profile, EM planner will generate a smooth trajectory for the control module.

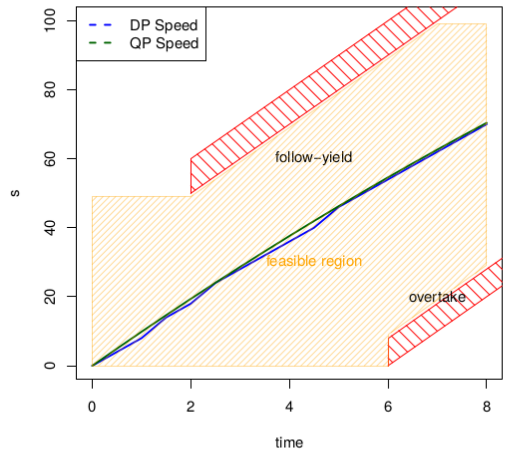


Figure 2.13: Spline QP speed optimizer. [12]

2.2.6 Notes on Solving Quadratic Programming Problems

For safety considerations, we evaluate the path and speed at approximately one-hundred different locations or time points. The number of constraints is greater than six hundred. For both the path and speed optimizers, we find that piecewise quintic polynomials are good enough. The spline generally contains 3 to 5 polynomials with approximately 30 parameters. Thus, the quadratic programming problem has a relatively small objective function but large number of constraints. Consequently, an active set QP solver is good for solving the problem. In addition to accelerating the quadratic programming, we use the result calculated in the last cycle as a hot start. The QP problem can be solved within 3 ms on average, which satisfies our time consumption requirement.

2.3 Case Study

Although most state-of-the-art planning algorithms are based on heavy decisions, EM planner is a light-decision-based planner. It is true that a heavy-decision-based algorithm, or heavily rule-based algorithm, is easily understood and explained. The disadvantages are also clear: it may be trapped in corner cases (while its frequency is closely related to the complexity and magnitude of the number of rules) and not always be optimal. In this section, we will illustrate the benefits of the light-decision-based planning algorithm by presenting several case studies. These cases were exposed during the intense daily test routines in Baidu's heavy-decision planning modules and solved by the latest light-decision planning module.

Figure 2.14 is a hands-on example of how EM planner iterates within and between planning cycles to achieve the optimal trajectory. In this case study, we demonstrate how the trajectory is generated while an obstacle enters our path. Assuming that the master vehicle has a speed of 10 meters per second and there is a dynamic obstacle that is moving toward us in the opposite direction with a speed that is also 10 meters per second, EM planner generates the path and speed profile iteratively with the steps below.

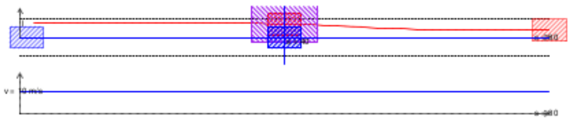
1) Historical Planning (Figure 2.14a). In the historical planning profile, i.e., before the dynamic obstacle enters, the master vehicle is moving forward straight with a constant speed of 10 meters per second.

2) Path Profile Iteration 1 (Figure 2.14b). During this step, the speed profile is cruising at 10m/s from the historical profile. Based on this cursing speed, the master vehicle and the dynamic obstacle will meet each other at position S = 40m. Consequently, the best way to avoid this obstacle is to nudge it from the right side at S = 40m.

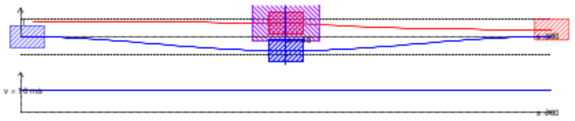
3) Speed Profile Iteration 1 (Figure 2.14c). Based on the path profile, which is nudge from the right, from step 1, the master vehicle adjusts its speed according to its interaction with the obstacle. Thus, the master vehicle will slow to 5 m/s when passing an obstacle with a slower speed, as passengers may expect.

4) Path Profile Iteration 2 (Figure 2.14d). Under the new speed profile, which is slower than the original one, the master vehicle no longer passes the dynamic obstacle at S = 40m but rather a new position at S = 30m. Thus, the path to nudge the obstacle should be updated to a new one to maximize the nudge distance at S = 30m.

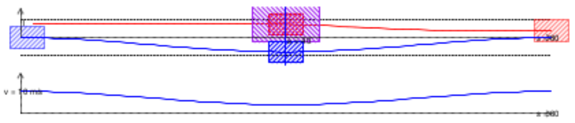
5) Speed Profile Iteration 2 (Figure 2.14e). Under the new path profile, where the nudge is performed at S = 30m, the slow down at S = 40m is no longer necessary. The new speed profile indicates that the master vehicle can accelerate at S = 40m and still generate a smooth pass at S = 30m.



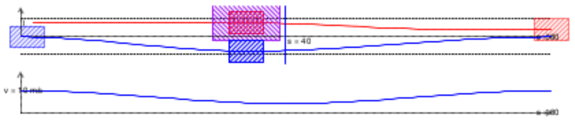
(a) Stage A: Historical Planning



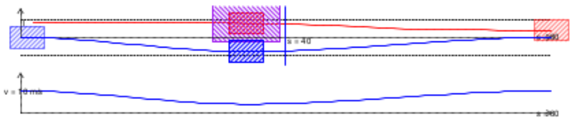
(b) Stage B: Path Planning Cycle 1



(c) Stage C: Speed Planning Cycle 1



(d) Stage D: Path Planning Cycle 2



(e) Stage E: Speed Planning Cycle 2

Figure 2.14: Case Study - Nudge oncoming dynamic obstacle This figure shows how the EM planner manages to iteratively solve the update path and speed profile. [12]

Thus, the final trajectory based on the four steps is to slow to nudge the obstacle at S=30 m and then accelerate after the master vehicle passes the obstacle, which is very likely how human drivers perform under this scenario.

Note that it is not necessary to always take exactly four steps to create the plan. It could take fewer or more steps depending on the scenario. In general, the more complicated the environment is, the more steps that may be required.

Chapter 3

Working Outline

3.1 Summary of Objectives

The aim of the thesis is to implement EM planner in the AGV cars. For ego cars and AGV cars, we need to face different scenarios, so we need to find a suitable strategy to do it. Especially, due to the huge workload, we can focus on the scene of the intersection, or the scene of the crowd, or the scene of the elevator door. Finally, the response to which scenario depends on the demands of the company.

3.2 Work Plan

Fortunately, Baidu's apollo project is open source We can directly read the source code of the EM planner section. This is the first step, then, the most impotant is how to apply it to the AGV cars.

Chapter 4

Actual Work

During the summer internship last year, I have done some related work. The main steps is:

1) Taking sufficient sampling points in the feasible area. The sampling points refer to the target points in each planning cycle. The more sampling points, the more paths, and the longer sampling points, the longer paths. Sampling points need to be set reasonably. If too many, it will increase the calculation, and the calculation of the car is not enough, it will affect the performance of the car.

2) Calculating the cost for each trajectory and seting a suitable cost function. The cost function needs to consider the feasibility and safety of the trajectory.

3) Cycle monitoring, each time the trajectory with the lowest cost is selected

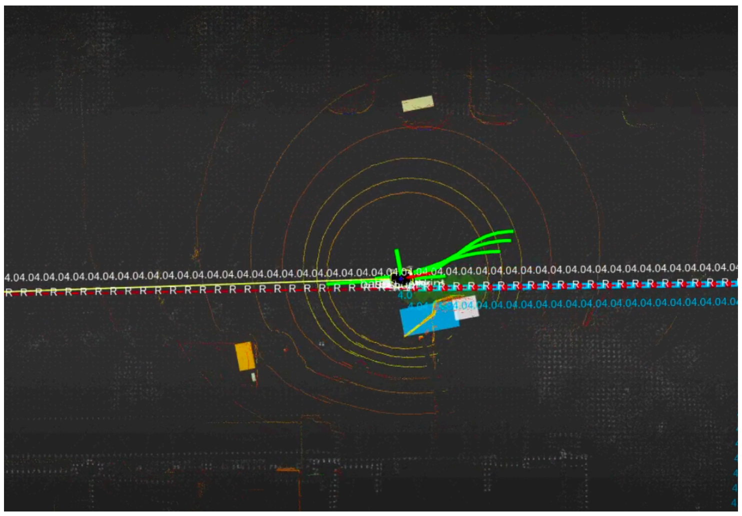


Figure 4.1: Simulation result.

for collision detection and physical limit detection. Physical limit detection refers to if the car can follow this trajectory physically. If the speed or angular velocity or acceleration of the car is limited, this trajectory can't be executed.

4) Returning the trajectory, then find the speed and angular velocity of this trajectory, and then give it to the control module for execution. The simulation result is shown in the figure 4.1.

Bibliography

[1] Z. Ajanovic, B. Lacevic, B. Shyrokau, M. Stolz, and M. Horn, "Search-based optimal motion planning for automated driving," *CoRR*, vol. abs/1803.04868, 2018.

[2] Z. Ajanović, M. Stolz, and M. Horn, *Energy Efficient Driving in Dynamic Environment: Considering Other Traffic Participants and Overtaking Possibility*, 05 2017, pp. 61–80.

[3] T. Fraichard, "Dynamic trajectory planning with dynamic constraints: A 'state-time space' approach," in *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '93)*, vol. 2, July 1993, pp. 1393–1400 vol.2.

[4] S. Anderson, S. Peters, T. Pilutti, and K. Iagnemma, "An optimal-control-based framework for trajectory planning, threat assessment, and semi-autonomous control of passenger vehicles in hazard avoidance scenarios," *International Journal of Vehicle Autonomous Systems*, vol. 8, pp. 190–216, 10 2010.

[5] S. Anderson, S. Peters, T. Pilutti, and K. Iagnemma, "An optimal-control-based framework for trajectory planning, threat assessment, and semi-autonomous control of passenger vehicles in hazard avoidance scenarios," *International Journal of Vehicle Autonomous Systems*, vol. 8, pp. 190–216, 10 2010.

[6] A. Rizaldi and M. Althoff, "Formalising traffic rules for accountability of autonomous vehicles," in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, Sep. 2015, pp. 1658–1665.

[7] M. Wang, S. Hoogendoorn, W. Daamen, B. Arem, and R. Happee, "Game theoretic approach for predictive lane-changing and car-following control," *Transportation Research Part C Emerging Technologies*, vol. 58, pp. 73–92, 07 2015.

[8] Z. Ajanovic, M. Stolz, and M. Horn, "A novel model-based heuristic for energy optimal motion planning for automated driving," in *Proceedings of the 15th IFAC Symposium on Control in Transportation Systems (CTS 2018)*, 6 2018.

[9] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, July 1968.

[10] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, and S. Thrun, "Junior: The stanford entry in the urban challenge," *Journal of Field Robotics*, vol. 25, pp. 569 − 597, 09 2008.

[11] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38, 1977. [Online]. Available: http://www.jstor.org/stable/2984875

[12] H. Fan, F. Zhu, C. Liu, L. Zhang, L. Zhuang, D. Li, W. Zhu, J. Hu, H. Li, and Q. Kong, "Baidu apollo EM motion planner," *CoRR*, vol. abs/1807.08048, 2018. [Online]. Available: http://arxiv.org/abs/1807.08048