CHAPTER 6

Decision, Planning, and Control

Abstract

*In this chapter, we continue with our discussion on the general planning and control modules by elaborating behavior **decision**, motion **planning**, and feedback **control**. Decision, planning, and control are the modules that compute how the autonomous vehicle should maneuver itself. These modules constitute the traditional narrow concept of planning and control. They all solve the same problem of how the autonomous vehicle should handle itself, however at different levels of the problem abstraction.*

6.1 BEHAVIORAL DECISIONS

The *behavior decision* module acts as the "co-driver" in the general autonomous vehicle planning and control modules. It is the module where most of the data sources get consumed and processed. Data sources fed to the decision module includes, but not limited to, information about the autonomous vehicle itself including location, speed, velocity, acceleration, heading, current lane info, and any surrounding perceptual objects information within a certain radius. The mission of behavioral decision module is to compute the behavioral level decision given all these various input data sources. These input data sources may include the following.

1. **The routing output:** A sequence of lanes along with the desired starting and ending position (where to enter and leave along the lane).
2. **The attributes about the autonomous vehicle itself:** Current GPS position, current lane, current relative position given the lane, speed, heading, as well as what is the current target lane given the autonomous vehicle location.
3. **The historical information about the autonomous vehicle:** In the previous frame or cycle of behavioral decision, what is the decision output? Is it to follow, stop, turn, or switch lanes?
4. **Obstacle information around the autonomous vehicle:** All the objects within a range radius of the autonomous vehicle. Each perceived object contains attributes

like located lane, speed, heading as well as their potential intention and predicted trajectories. The object and its attribute information mainly come from the *perception* and *prediction* module output.

5. **Traffic and map objects information:** The lanes and their relationships as defined by the HD map. For example, Lane 1 and Lane 2 are adjacent, and it is legal to make a lane switch. Then what is the legal range for switching lanes? Another example is, when we finished a straight lane and need to enter into a left-turn lane, is there a traffic light or stop-sign or pedestrian cross-walk at the connection of these two lanes? This kind of information comes from the mapping module as well as from the perceived dynamic traffic signs (e.g., traffic light green or red).

6. **Local traffic rules:** For example, the city speed limit or if it is legal to make a right-turn at the red light.

The goal of behavior *decision* module is to leverage all these pieces of information and make effective and safe decisions. It is easy to see that the decision module is where all the data sources get considered. Due to the heterogeneous characteristic of these data sources, and especially the diversified local traffic laws, it is very difficult to formulate the behavioral decision problem and solve it with a uniformed mathematical model. It is more suitable to use advanced software engineering concepts and design a traffic rule based system to solve this problem. In fact, an advanced rule based behavior decision system has been found in many successful autonomous driving systems. In the DARPA challenge, Stanford's autonomous driving system "Junior" [1] leverages Finite-State-Machine (FSM) with cost functions to deterministically compute autonomous vehicle trajectory and behaviors. Similarly, CMU autonomous driving system "Boss" [2] computes the space gaps between lanes, and utilizes such information together with pre-encoded rules to trigger the behavior of lane switching. Other systems such as Odin and Virginia Tech [3] also used rule-based engines to determine the behaviors of autonomous vehicle. With more and more research of autonomous driving decision and planning systems, Bayesian models are becoming more and more popular in modeling the autonomous vehicle behavior and have been applied in recent research works [4, 5]. Among the Bayesian models, Markov Decision Process (MDP) and Partially Observable Markov Decision Process (POMDP) are the widely applied methods in modeling autonomous driving behavior.

Even though academia prefers non-deterministic Bayesian model approaches, we do believe that, in practical industrial systems, rule-based deterministic decision systems still have a key role and we will demonstrate a typical scenarios and rule-based approach in this section.

The rule-based approach we introduced is based on the idea of *Divide and Conquer* to decompose the surrounding environment into layered scenes and solve them individually. In fact, we believe that in an actual autonomous driving production system, a rule-based system might even

be safer and more reliable given its simplicity. Imagine how human drivers drive from point A to point B via a fixed route. The traffic rules are always fixed. More importantly, given the surrounding environment including nearby vehicles, pedestrians and traffic signs, if we apply traffic rules together with our intention of where to go, the behavioral output, or how the human driver should re-act, is usually constrained within a very limited number of behavioral choices, or even clearly specified by traffic rules. For example, in California, if a vehicle wants to cross a four-way stop sign junction, it should first stop for 3 s, yield to any other vehicle that has right of way, and then proceed. The whole series of actions is determined clearly with the consideration of the surrounding objects, and it could be naturally modeled in a deterministic fashion. Even though there might be unexpected conditions which may lead to violation of certain traffic rules, the rule of *safety first* to avoid collision could also be deterministically enforced. Therefore, we present a rule-based decision module implementation with details in this chapter since rule-based decision implementations are still main-stream solutions in industry practice.

6.1.1 MARKOV DECISION PROCESS APPROACH

A Markov Decision Process (MDP) is defined by the following five element tuple (S, A, P_a, R_a, γ), where:

1. S represents the state space of the autonomous vehicle. The division of state space should consider both location of the autonomous vehicle and map elements. With the dimension of the location, one can divide the surrounding square of the autonomous vehicle into grids with fixed length and width. Considering different road map objects, we could also create spaces concerning different combination of map objects, such as the current and adjacent lanes where the autonomous vehicle is located at;

2. A represents the behavioral decision output space, which is a fixed set of all possible behavioral actions to take: exemplar decision state could be to *Follow* a vehicle on the current lane, *Switch Lane* to an adjacent parallel lane, *Turn Left/Right*, *Yield, or Overtake* a crossing vehicle at a junction, or *Stop* for traffic lights and pedestrians;

3. $P_a(s, s') = P(s' \mid s, a)$ is the conditional probability, which represents the probability to reach state s', given that the autonomous vehicle is currently at state s and takes action a;

4. $R_a(s, s')$ is the reward function, which represents the reward of transforming from state s to state s' by taking action a. The reward is synthetized measure of how we evaluate such state transformation. Factors that should be considered and represented in the reward include: safety, comfortableness, reaching the destination, and the difficulty for the downstream motion planning to execute;

5. γ is the decay factor for reward. The reward at the current time has a factor of 1 and the reward will for the next time frame will be discounted by a factor of γ. And, accordingly, the reward for t time frames in the future will be deemed as γ^t for the time being. The decaying factor guarantees that the same amount of reward will always be more valuable for the time being than in the future;

With the formal MDP setting, the problem that behavioral decision needs to solve, is to find an optimal *policy*, denotes as π:$S \rightarrow A$. Given any state s, the policy accordingly computes a behavioral decision $a = \pi(s)$. When the policy has been determined, the whole MDP could be viewed as a Markov Chain. The behavioral decision policy π is to optimize the accumulated rewards from current time to the future. Note that if the reward is not deterministic but a random variable, then the policy will optimize the expected accumulated rewards. Mathematically, the accumulated reward to maximize is written as:

$$\sum_{t=0}^{\infty} \gamma^t R_{at}(s_t, s_{t+1}),$$

where action a is the policy output $a = \pi(s)$. The method to find such policy is usually based on *Dynamic Programming*. Assume that the state transition probability matrix P and reward distribution matrix R are known, the optimal policy solution could be obtained by iterating on the computing and storing the following two state arrays:

$$\pi(s_t) \leftarrow \underset{a}{\operatorname{argmax}} \left\{ \sum_{s_{t+1}} P_a(s_t, s_{t+1}) \left(R_a(s_t, s_{t+1}) + \gamma V(s_{t+1})\right) \right\}$$

$$V(s_t) \leftarrow \sum_{s_{t+1}} P_{\pi(s_t)}(s_t, s_{t+1}) \left(R_{\pi(s_t)}(s_t, s_{t+1}) + \gamma V(s_{t+1})\right) \quad .$$

$V(s_t)$ represents the accumulated future rewards discounted at current time, and $\pi(s_t)$ represents the policy that we want to search for. The solution is based on repeated iteration between possible state pairs (s, s'), until the above two state arrays converge [6, 7]. Furthermore, in Bellman's value iteration algorithm, there is no need to explicitly compute $\pi(s_t)$. Instead, $\pi(s_t)$ related computation could be incorporated into computation of $V(s_t)$, which leads to the following single step "Value Interation":

$$V_{i+1}(s) \leftarrow \max_{a} \left\{ \sum_{s'} P_a(s, s') \left(R_a(s, s') + \gamma V_i(s')\right) \right\} \quad ,$$

where i is the iteration step. In step $i = 0$, we start with an initial guess of $V_0(s)$ to kick off the iteration. $V(s)$ gets updated in each step until convergence. There are various methods for applying MDP on autonomous vehicle, and we will not dive into different MDP autonomous driving decision implementation details here in this book. Interested readers could refer to [6, 7] to get an idea of how to design state spaces, action spaces, state transitions, and the reward function implementation.

Here we want to emphasis a few factors to consider when designing the reward function $R_a(s, s')$, since it is the critical element in building a working MDP decision system. A good reward function in the MDP based decision module include the following aspects.

1. **Reach the destination:** *Encourage* the autonomous vehicle to follow the routing module output route to reach the destination. In more details, if the action chosen by the policy, i.e. $a = \pi(s)$, makes the autonomous vehicle diverting from the route, punishment should be given. And vice versa, reward should be issued to actions following the route.

2. **Safety and collision free:** If the state is based on $N \times N$ equal squared grids centered around the autonomous vehicle, then any decision to move to a grid where collision might occur should be punished. Movements to grids with lower collision possibility or larger distance to collision likely grids will be rewarded.

3. **Comfortableness and smoothness:** These two factors are coherent. A comfortable journey mostly indicates few or no sharp maneuvers. And a limited amount of abrupt maneuver facilitates that the downstream modules could smoothly execute most of the decisions. For example, action leading transition from a speed to a similar speed should have higher rewards than sharp accelerating or decelerating in this category of rewards.

With the consideration of state space design, action space design, transition probability matrix, and the reward function, readers could sense that it is a very delicate job to build a working MDP based decision system.

6.1.2 SCENARIO-BASED DIVIDE AND CONQUER APPROACH

The key idea is to apply the notion of *Divide and Conquer* to decompose the autonomous vehicle surroundings into scenarios. In each scenario, the corresponding rule will be applied individually to the objects or elements in the scenarios to compute an *Individual Decision* for each object, and then a *Synthetic Decision* for the autonomous vehicle itself is computed by consolidating all the individual decisions.

Synthetic Decision

Synthetic Decision	Parametric Data
Cruise	➢ Current lane ➢ Speed limit of the current lane
Follow	➢ Current lane ➢ *id* for the vehicle to follow ➢ Speed to reach minimum of current lane speed limit and speed of the vehicle to follow ➢ Not exceeding 3 m behind the vehicle in front
Turn	➢ Current lane ➢ Target lane ➢ Left or right turn ➢ Speed limit for turning
Change Lane	➢ Current lane ➢ Target lane ➢ Change lane by overtaking and speed up to 10 m/s ➢ Change lane by yielding and speed down to 2 m/s
Stop	➢ Current lane ➢ *id* for any object to stop, if any ➢ Stop by 1 m behind the object to stop

Figure 6.1: Synthetic decision with its parameters in behavioral decision.

The notion of *Synthetic Decision* is regarding how the autonomous vehicle itself should behave. It is the top-level behavioral decision. Example synthetic decisions include: keep the current lane to follow a vehicle, switch lane to an adjacent parallel lane, or stop by a certain stop-line as per specified by a traffic sign or light. As the top-level decision behavior, its possible output space, along with its definitions, must be consistent and shared with the downstream motion planning module. In addition, to help motion-planning to come out with the planned trajectory, the synthetic decision is always companied with parameters. Figure 6.1 lists a few synthetic decision definitions as well as their possible parameters. Consider when the synthetic decision at the current time frame is to *Follow*. The output command to the motion planning module is not only the behavioral follow command, but also parameters as: the *id* of the vehicle to follow on the current lane, suggested speed to follow (which is usually the lesser of the front vehicle speed and the lane speed limit), and suggested distance to keep while following (for example 3 m behind the rear of the front vehicle). In this way, the downstream motion planning could utilize these parameters as constraints, such that a smooth and collision-free trajectory could be computed.

Individual Decision

In comparison with the synthetic decision are the individual decisions. As we mentioned, the synthetic decision is a comprehensive consolidated decision for the autonomous vehicle itself after considering all the information including all the road objects. Hereby we propose to explicitly produce an *individual decision* for each element in our surrounding world. The object accompanied by an individual decision could be an actual perceived obstacle on the road, or just a logical map object such as the stop-line corresponding to a traffic light or pedestrian cross-walk. Actually, in our design as shown in Figure 6.4, the logic of scenario division takes place first. Individual decisions for objects are then computed and associated with each object in all the scenarios. Only after all the objects have been computed of individual decisions, then comes the final synthetic decision, which is a consolidation of the individual ones. Like synthetic decision, the individual decisions also come with parameters. These individual decisions are not only necessary pre-requisites to compute the synthetic final decision, but also transmitted to downstream motion planning module to facilitate trajectory planning. Readers might wonder why these individual decisions are also sent to downstream module. Isn't it just enough to convey the final synthetic decision given that we only plan the action for the autonomous vehicle itself?

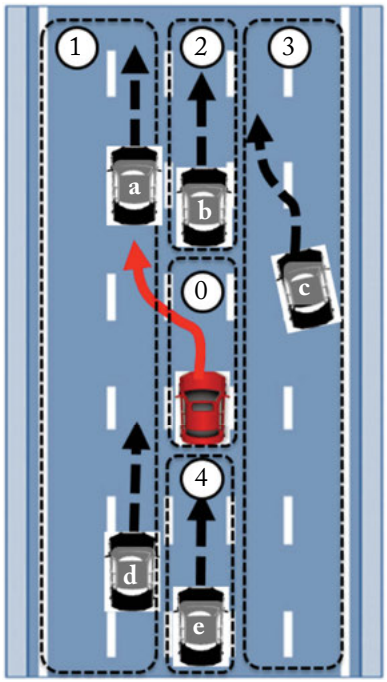
From industrial experiences, sending both the synthetic final decision and its supporting individual decisions will be significantly beneficial for the downstream motion-planning task. Since these individual decisions serve as projections of the synthetic decision in a consistent way, motion planning will have much more reasonable and explicit constraints, and hence the optimization problems of motion-planning could be much better formalized with the furnish of individual decisions. In addition, debugging efficiency will be greatly improved with individual decisions. Figure 6.2 lists a few typical individual decisions and their parameters. For example, if the individual decision for an object X is to *overtake*, then the parameters associated with this overtake decision will possibly include the time and distance to keep when overtaking the object X. The distance parameter indicates the minimum distance to be ahead of the object X's front, and the time parameter is the minimum time corresponding to how long the gap of overtaking should be existent given speeds of the autonomous vehicle and object X. Note that such overtake or yield individual decisions only exist when an object's predicted trajectory intersects with the planned trajectory of the autonomous vehicle. Typical examples of yielding/overtaking an object include scenarios at junctions. We will use an example at junction to illustrate how exactly we divide the surroundings into layered scenarios, apply specific rules to obtain individual decisions and finally consolidate them as synthetic decision output.

Individual Decision		Parametric Data
Vehicle	Follow	➢ *id* for the vehicle to follow ➢ Speed to reach for following the vehicle ➢ Distance to keep for following the vehicle
	Stop	➢ *id* for the vehicle to stop ➢ Distance to stop behind the vehicle
	Attention	➢ *id* for the vehicle to stop ➢ Minimum distance to keep while paying attention to the vehicle
	Overtake	➢ *id* for the vehicle to overtake ➢ Minimum distance to keep for overtaking ➢ Minimum time gap to keep for overtaking
	Yield	➢ *id* for the vehicle to yield ➢ Minimum distance to keep for yielding ➢ Minimum time gap to keep for yielding
Pedestrian	Stop	➢ *id* for the pedestrian to stop ➢ Minimum distance to stop by the pedestrian
	Swerve	➢ *id* for the pedestrian to swerve ➢ Minimum distance to keep while swerving around

Figure 6.2: Individual decisions with parameters in behavior decision module.

Scenario Construction and System Design

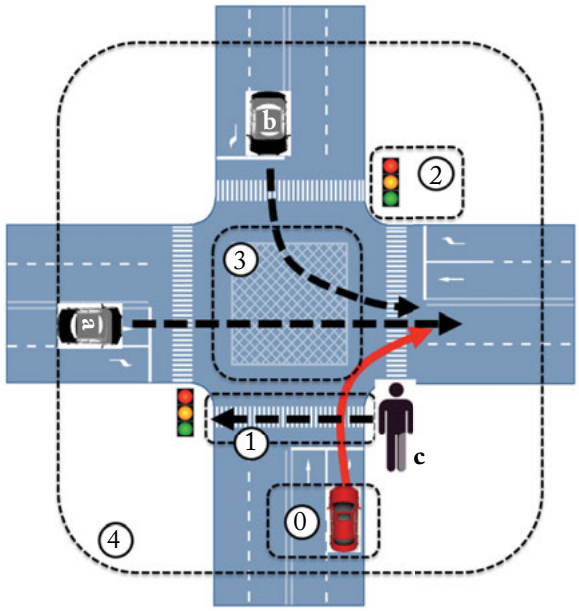
The computation of individual decisions is dependent on the construction of *scenarios*. Here one could simply think of the scenarios as a series of relatively independent divisions for the surrounding world of the autonomous vehicle. In fact, the way we divide the surrounding world is in a layered structured fashion. Scenarios belong to different layers, and the scenarios within each layer is independent. A deeper layer of scenarios could leverage any computation result or information of the shallower layers. Objects usually belong to only one scenario. The idea behind this structured layered scenario division of the world is *divide and conquer*. We first aim to focus on independent small *worlds*, i.e., scenarios and solve the problem of computing the decisions to make within that small world. While computing the individual decisions in each independent scenario in the same layer, the routing intention (where the autonomous vehicle wants to go) and the previous layered computation results are shared. After obtaining the individual decisions, synthetic decision is then consolidated with a set of rules. Figures 6.3a and 6.3b show two examples of how we divide into scenarios and compute behavioral decisions.



Synthetic Decision:
Switch lane from current lane to the left lane: yield vehicle **a**, overtake vehicle **d**, and attention to vehicle **b** at current lane.

Scenarios and Individual Decisions:
0. Master Vehicle
1. Left Lane(s); Overtake **d** and yield **a**
2. Front Vehicle(s): Attention **b**
3. Right Lane(s): Ignore **c**
4. Rear Vehicle(s): Ignore **e**

Figure 6.3: (a) Layered scenarios while doing lane switch.



Synthetic Decision:
Stop by the crosswalk stop-line and wait for pedestrian **c** to cross

Scenarios and Individual Decisions:
First Layer Scenarios
0. Master Vehicle
1. Crosswalk: Stop for pedestrian **c**
2. Traffic Light: Red light turn right, yield any through/turn traffic
3. Keep Clear Zone Ignore

Second Layer Scenarios
4. Junction Scenarios: Based on Scenarios 1, 2, 3

Figure 6.3: (b) Layered scenarios while at junction.

In Figure 6.3(a), there are two vehicles **a** and **d** in the scenario of "Left Lane(s)". The intention of the autonomous vehicle is to switch from the current lane to its left adjacent lane as specified by the routing output. Considering the relative position and speed of the autonomous vehicle regarding vehicle a and d, the computational result of the *Left Lane(s)* scenarios is to yield vehicle a and overtake vehicle d, which means to switch lane between these two vehicles; meanwhile, another scenario of *Front Vehicle(s)* depicts the small world of things in front of the autonomous vehicle itself, and this scenario is independent of the *Left Lane(s)* scenario. We should pay attention that even though the intention of autonomous vehicle is to switch to the left lane, it is still important not to ignore anything in front of us at the current lane. Therefore, individual decision for vehicle b in the *Front Vehicle(s)* scenarios is to pay attention and keep an appropriate distance at vehicle b. We also have the *Rear Vehicle(s)* and *Right Lane Vehicle(s)* scenarios. However, given that the predicted trajectories of objects in these scenarios do not have conflicts with our planned trajectory, we will be able to safely ignore them

Scenarios in Figure 6.3(a) do not dependent on each other much, except for the status of the *Master Vehicle* information which is shared among all these scenarios. In Figure 6.3(b), we show a complicated case where more layers of scenarios are shown. The *Master Vehicle* scenario is a special one whose information will be shared and utilized in other scenarios. The first layer of scenarios includes *Front/Rear Vehicle(s)*, *Left/Right Lane Vehicle(s)*, and traffic sign area related scenarios such as Traffic Light and *Crosswalk*. More complicated *composite* scenarios could be built on top of the first layer scenarios by using them as elements. As shown in Figure 6.3(b), the four-way intersection scenario is based on the scenarios of *Crosswalk*, *Traffic Light*, and *Master Vehicle*. Besides these membership scenarios, vehicle a and b belong to the four-way intersection scenario itself as they reside in lanes under the concept coverage of the *junction*. Suppose the routing intention is to turn right, and we currently see a red light as well as a pedestrian crossing the road. Traffic rules allow a red light right turn but the autonomous vehicle must firstly yield and wait for any pedestrians. Individual decision for the crossing pedestrian will be stop while the individual decisions for both vehicle a and b will be yield. Consolidating these individual decisions, the synthetized decision for our autonomous vehicle itself will be to just stop in front of the crosswalk defined stop-line.

As described above, each individual scenario focuses on its own business logic to compute the individual decisions for elemental objects within itself. Then, the behavioral decision module considers all the individual decisions for every object and comes up with a final synthetic decision for the autonomous vehicle itself by consolidating individual decisions. Here a natural question is what if there are different or even conflicting individual decisions for the same object. For example, a vehicle gets two different individual decisions in two separate scenarios, one being *yield* and the other one being *overtake*? In general, the way we divide the surrounding world into scenarios will naturally assign objects, either actual perceived objects or conceptual logical objects into distinctive scenarios that they belong to. In most cases, an object will not likely appear in more than one sce-

narios. However, we cannot completely rule out such possibilities. In fact, some of the scenarios do cover a small overlapped map area with robustness consideration. When such low probability cases do happen, there is a layer which handles merging individual decisions for safety and coherence check in the behavioral decision system (Figure 6.4). Imagine a vehicle on the same lane behind us is in the process of changing from the current lane and to the left lane, leading to its existence in both the *Rear Vehicle(s)* and *Left Lane(s)*. And let's assume that our autonomous vehicle also has intention to switch to the left lane. The *Rear Vehicle(s)* scenario gives *attention* individual decision while the *Left Vehicle(s)* scenarios decides that we should *yield* the vehicle. The individual decision merging layer will review these different individual decisions for each object, and re-computes a merged final individual decision considering both safety and our autonomous vehicle intention. In this case, since our autonomous vehicle is also trying to switch to the left lane, we will then obtain *yield* to the vehicle if we have already started the switching lane motion, or keep the *attention* if we have not started switching lane yet.

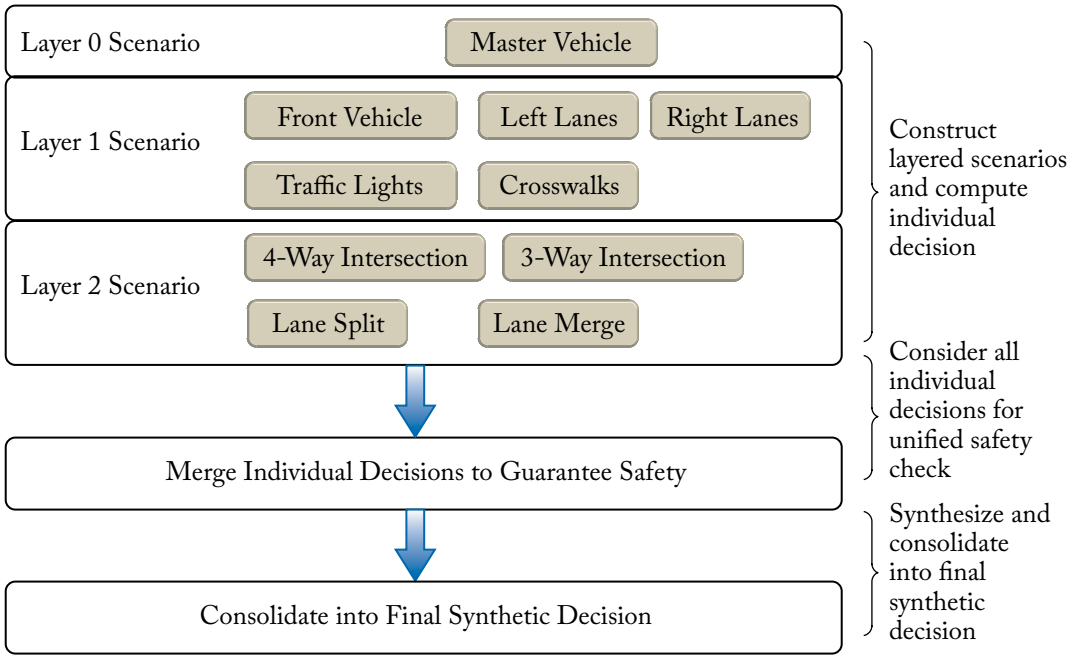


Figure 6.4: Architecture of a rule-based behavior decision system containing layered scenarios.

In conclusion, the system framework and logic process is shown as Figure 6.4. On the top are layers focusing on layered scenario construction, where information regarding our autonomous vehicle's intension, mapping and localization, and perceived surrounding world, are all utilized to building layers of independent scenarios. Within each layer's independent scenario, its own busi-

ness logic and the shared autonomous vehicle routing intention will determine the computation of individual decisions for all the objects in the scenarios. After all the layered scenarios have finished individual decision computation, the merging layer will double-check all the individual decisions and solves any possible conflicts or inconsistence for any object. And, finally, at the bottom, the final synthetic decision for the autonomous vehicle itself is computed by consolidating the merged and coherent individual decisions. This synthetic decision, along with the merged individual decisions, will be sent to downstream motion planning module, where a spatial-temporal trajectory for the autonomous vehicle will be planned for physical execution.

6.2 MOTION PLANNING

The direct downstream module of behavioral decision is the motion *planning* module. The task of motion planning is to generate a trajectory and send it to the feedback control for physical vehicle control execution. The planned trajectory is usually specified and represented as a sequence of planned *trajectory points*. Each of these points contains attributes like location, time, speed, curvature, etc. The problem of autonomous vehicle motion planning could be viewed as a special case of general motion planning in robotics. In some sense, the motion planning problem for autonomous vehicle on road, is even easier than general motion planning in robotics, since cars mostly follow the pre-existing road graph and move on a 2D plane. With the only control signal being throttled, brake and wheel, the class of possible trajectories naturally exhibits certain characteristics such as smoothness and curvature constraints, and is therefore easier to optimize compared with planning a trajectory in higher dimensions with more constraints (e.g., motion planning in 3D space for a flying drone).

Since the DARPA urban challenge, motion planning in autonomous vehicle has been gradually developing as a relatively independent module. [4, 8] attempted to solve the problem of motion planning under certain conditions of urban driving and parking, and there are also works on solving special motion planning problems such as [7]. [5, 9] list recent works on motion planning in various aspects, which readers could refer to. With all these research works, the problem for motion planning module in the context of autonomous driving is becoming clear: most motion planning work proposes to solve the problem of optimizing a spatial-temporal trajectory within certain spatial-temporal constraints. As we have mentioned before, a *spatial-temporal* trajectory consists of *trajectory points*. The attributes of each point, including but not limited to, position, time, speed, acceleration, curvature, and even higher-order derivative of attributes like curvature. These attributes are essential constraints since the costs associated with these points constitute the optimization goal. Since vehicle control is not a harmonious system, the actual vehicle trajectory exhibits properties of spline trajectories. Therefore, motion planning could be formalized as an optimization problem for trajectories with certain common properties/constraints on the 2D plane.

The two key elements in this optimization problem are *Optimization Object* and *Constraints*. Here the *Optimization Object* is usually represented as *costs* associated with different candidate solutions, and the goal of optimization is to search for the one with minimum cost. The function that computes cost is based on the following two key factors. First, as the direct downstream module of behavioral decision, the cost function must obey the upstream behavioral decision output. For example, individual decision for a vehicle in front might be to follow and stay within a distance range relative to the rear of the front vehicle, then the planned trajectory must reach but not exceed the designated area specified the individual decision. Also, the planned trajectory should be collision free, which means a minimum distance to any physical object has to be kept while computing the trajectory. Second, since we focus on the autonomous driving on urban roads, the planned trajectory should be consistent with road shape, and this requirement generally could be interpreted that our autonomous vehicle should follow natural roads to move. All these aspects are represented in the design of cost functions. While the cost function design put emphasis on obeying the upstream behavioral decision output and following routing direction, the constraints in the motion planning optimization problem is more about the constraints such that the downstream feedback control could comfortably execute. For example, the curvature and second order derivative of the curvature have constraints given the steering wheel control. Similarly, with throttle as the control method for accelerating, rate of how acceleration changes is also limited.

As we mentioned in the very beginning of the planning and control chapter, we are not going to iterate and describe all existing motion planning solutions in a throughout fashion. Instead, we present two typical approaches, both very successful and proven to work. The first proposed approach for motion planning is based on the idea in [9], but is a simpler version. Here, the problem of planning a spatial-temporal trajectory is divided into two problems to be tackled sequentially: *path planning* and *speed planning*. Path planning only solves the problem of computing trajectory shape on the 2D plane, given the behavior decision output and the cost function definition. The generated paths do not have any speed information, and are merely spines with various shapes and lengths. Speed planning is based on the results of path planning, and solves the problem of how the autonomous should follow a given trajectory. Compared with the proposal in [9] which simultaneously solves the problem of optimizing the spatial-temporal trajectory, our solution represents a clearer divide and formalization of the problem. Even though the proposal here might not necessarily find the optimal solution, industrial practices teach us that such separate divide and conquer fashion of motion planning is effective. Instead of dividing the motion planning problem into path planning and speed planning, the second approach tackles the motion planning problem by considering the motion planning on two orthogonal directions. The two orthogonal directions are the *longitudinal* (s-direction) and *lateral* (l-direction) as per the *SL-coordinate* system, which we will describe in this section. The advantage of this approach over the first one is that the shape of the planned trajectory naturally takes into account the speed, while the first approach may lead to

a chosen trajectory shape inappropriate for the desired speed profile due to separate optimization of trajectory and speed. While the first approach of separately optimizing path and speed works well in urban low-speed autonomous driving, the second approach is more suitable for higher speed scenarios such as highways.

6.2.1 VEHICLE MODEL, ROAD MODEL, AND SL-COORDINATION SYSTEM

We bring the mathematical concept of vehicle pose and road-based *SL-coordination* system. A vehicle's pose is determined by: $\bar{x} = (x, y, \theta, \kappa, v)$ where (x, y) represents the position on the 2D surface, θ represents the direction, κ is the curvature (the rate θ changes), v represents the speed tangential to trajectory. These pose variables satisfy the following relationship:

$$\dot{x} = v\cos\theta$$
$$\dot{y} = v\sin\theta$$
$$\dot{\theta} = v\kappa \ ,$$

where κ's constraint is input to the system. Considering a continuous path generated by the vehicle, we will define the direction along the path as the s-direction, and the pose variables' relationship with the s-direction satisfy the following derivative equations:

$$dx / ds = \cos(\theta(s))$$
$$\mathrm{dy} / \mathrm{ds} = \sin(\theta(s))$$
$$\mathrm{d}\theta / \mathrm{ds} = \kappa(s) \ .$$

Note that here we haven't put any constraints on the relationship between κ and θ, meaning that the vehicle could change its curvature κ at any direction θ. However, in the actual control model, relationship of curvature κ and direction θ is restrained, but this constraint is trivial in terms of the speed limit for urban road driving, and therefore does not have significant impact on the practice and feasibility of the proposed motion planning algorithm.

The path planning part of our proposed motion planning algorithm heavily depends on the high-definition-map (HD-map), and especially the *Center Line* of lanes in the map, which we refer to as *reference line*. Here a road lane is defined by its sampling function $r(s) = [r_x(s), r_y(s), r\theta(s), r_\kappa(s)]$, where s represents the distance along the tangential direction of the path, and we refer to this *longitudinal* distance as s-distance. Correspondingly, there is the *lateral* distance, and it represents the distance perpendicular to the s-direction, which we will refer to as l distance. Consider a pose p under the (s, l) coordinate system and the corresponding pose under the (x, y) world coordinate system, the pose in the world coordinate system $p(s, l) = [x_r(s, l), y_r(s, l), \theta_r(s, l), \kappa_r(s, l)]$ satisfies the following relationships with the (s, l) coordinate pose:

$$x_r(s, l) = r_x(s) + l\cos(r_\theta(s) + \pi/2)$$
$$y_r(s, l) = r_y(s) + l\sin(r_\theta(s) + \pi/2)$$
$$\theta_r(s, l) = r_\theta(s)$$

$$\kappa_r(s, l) = r_\kappa(s)^{-1} - l)^{-1} ,$$

where κ_r is defined to increase towards the inner side of a turn, and to decrease towards the outside (meaning that κ_r increases with l at the same s). As shown in Figure 6.5, near the x-axis at the origin, the lateral l increases with y. Assume for a certain Lane(k), its width keeps as a constant, then the whole lane could be represented as a set of points along the longitudinal direction following the central reference line: $\{p(s, l) : \in R^+\}$. We call such a lane coordinate system formally as the *SL-coordinate* system.

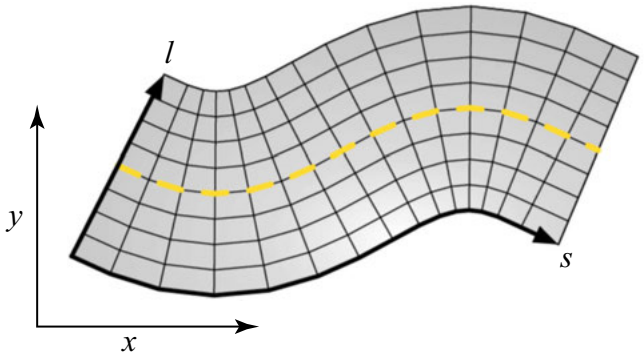


Figure 6.5: Lane-based SL-coordinate system under *XY* plane [9], used with permission.

6.2.2 MOTION PLANNING WITH PATH PLANNING AND SPEED PLANNING

Given the previously described road-based SL-coordinate system, we now discuss how we can solve the motion planning problem by first doing path planning and then speed planning. We define the vehicle *path* as a continuous mapping $\rho : [0,1] \rightarrow C$ from range [0,1] to set of vehicle poses $C = \{\vec{x}\}$. The initial pose of a planning cycle or frame is known as $\rho_1(0) = \rho_2(0) = q_{init}$ for path ρ_1 and ρ_1, which end separately at $\rho_1(1) = q_{end1}$ and $\rho_2(1) = q_{end2}$, as shown in Figure 6.6. The goal of path planning is to find a path, which starts from the initial pose, reaches a desired end pose, and satisfies certain constraints with minimum cost.

The way we search for the optimal cost path is similar to the way we used for computing routing, which is to place *sampling points* towards potential areas where the path might traverse. In Figure 6.6, we uniformly divide the lane into *segments* with equal s and l distance. Consider the central point in each divided small (s_i,l_j) grid, we refer to this point as a sampled trajectory point. Hence, a candidate path is a smooth spline connecting the sampled trajectory points along the increasing s direction. With the segmentation and trajectory point sampling in Figure 6.6, there are 16 trajectory points (4 in the s direction, and 4 in the l direction), and we only consider the splines which connect trajectory points along the increasing s direction since backward driving is not con-

sidered in normal urban road driving conditions and will be treated specially. The total number of candidate paths is $4^4 = 256$, among which the path planning problem is to search for the optimal one with minimal cost with certain constraints.

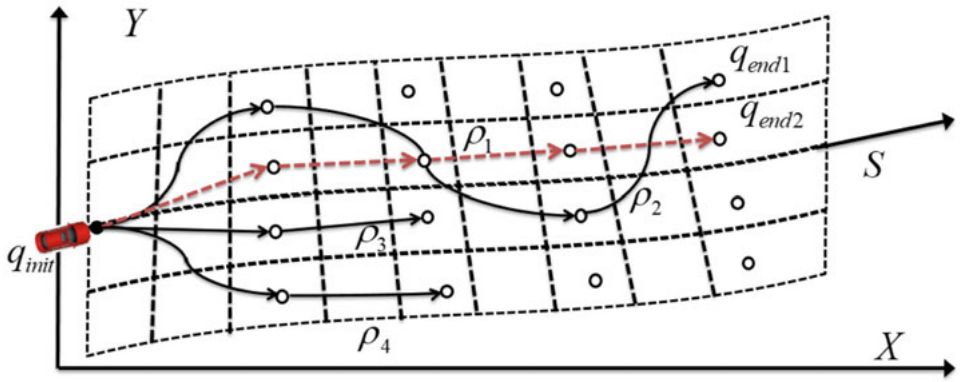


Figure 6.6: Possible candidates in path planning with divided road grids and sampling points in lane-based SL-coordinate system.

We use polynomial spirals to connect sampled trajectory points. Polynomial spirals represent a cluster of curves whose curvature could be represented as polynomial functions of the arc length (corresponding to the s direction). The degree of polynomial spiral is not essential, and we use cubit or quintic spirals, whose arc length s and curvature κ satisfy:

$$\kappa(s) = \kappa_0 + \kappa_1 s + \kappa_2 s^2 + \kappa_3 s^3 \text{ or } \kappa(s) = \kappa_0 + \kappa_1 s + \kappa_2 s^2 + \kappa_3 s^3 + \kappa_4 s^4 + \kappa_5 s^5 \ .$$

The only significant difference between cubic and quintic spirals is that on satisfying boundary constraints: the second order derivative of curvature $d\kappa^2 / ds^2$, which corresponds to wheel rotating speed, is not continuous in cubic spirals; while quintic spirals could make both $d\kappa / ds$ and $d\kappa^2 / ds^2$ continuous. When speed is low, the discontinuity introduced by cubic spirals is not a significant for downstream feedback control. However, such discontinuity could not be safely ignored in high speeds.

The parameters of proposed polynomial spirals connecting sampled trajectory points could be effectively obtained by gradient descent fashioned search algorithm. For example, consider cubic spirals ($\kappa(s) = \kappa_0 + \kappa_1 s + \kappa_2 s^2 + \kappa_3 s^3$) connecting initial pose $q_{\text{init}} = (x_I + x_I + \theta_I + \kappa_I)$ to destination pose $q_{\text{goal}} = (x_G + x_G + \theta_G + \kappa_G)$ with continuous curvature. When at the path start s, the first- and second-order derivative of curvature need to satisfy the initial constraints as follows:

$$\kappa_0 = \kappa_1$$
$$\kappa_1 = d\kappa(0) / ds$$

$$\kappa_2 = d^2\kappa(0) / ds^2 \ .$$

This makes the actual unknown parameters to be two (κ_3, s_G), which could be quickly found by gradient descent algorithm.

Find the Minimum Cost Path by Dynamic Programming

As we described, path planning has been formalized into a search problem to find the minimum cost path among the $|l_{total}/\Delta|^{|stotal/\Delta s|}$ candidate paths connecting the trajectory points along s-direction among $|l_{total}/\Delta l|*|s_{total}/\Delta s$ sampled trajectory points. Let's assume that these sampled trajectory points constitute a graph $G = (V,E)$, where each trajectory point is a node in this graph: $v \in V, v = (x,y,s,l)$. For any two points $v, u \in V$, if their s coordinates satisfy $s_v < s_u$, we use $e(v,u) \in E$ to denote the cubic or quintic spiral from v to u. And the optimal path planning problem could be converted to the problem of searching for a lowest-cost path (shortest path) on a directed weighted graph. What is special here is that the shortest path not only incudes costs accumulated along the currently expanded path, but could also incorporate a potential cost associated with the newly expanded path if the expansion of a node is established. Consider the path τ connecting $n_0, n_1, \ldots n_k$, where the initial point is n_0 and the end point is n_k, the cost of this established trajectory could be written as:

$$\Omega(\tau) = c(\tau) + \Phi(\tau) \ ,$$

where $c(\tau)$ represents the cumulated cost by following along the path; And $\Phi(\tau)$ is the cost introduced if we conclude by ending the planned path at the end point n_k. If we write the $\Phi(\tau)$ function as the incremental cost introduced by ending at n_k, we get:

$$\Omega(\tau(n_0,n_1,\ldots n_k)) = g(n_k) + \Phi_c(\tau(n_{k-1}, n_k)) \ ,$$

among which we define $g(n)$ as the minimum cost for reaching the node n. Note that cost incurred by following along the spiral does not include the additional cost introduced by ending the path at n. Consider all the paths with n_{k-1} as the second to last trajectory point, we would need to find the last trajectory point n_k which minimizes total cost. More specifically, the node n_k satisfies the following properties.

1. There exists a directed edge $e(n_{k-1},n_k)$, also denote as $\tau(n_{k-1}, n_k)$ that connects node n_{k-1} and n_k.

2. For the set of nodes which n_{k-1} could reach (edge $e(n_{k-1},\tilde{n}_k)$ exists), denoted as $\{\tilde{n}_k\}$, the trajectory which ends at n_k has the lowest total cost: $n_k \leftarrow \underset{\tilde{n}_k}{\text{argmin}}\ g(n_{k-1}) + c(\tau(n_{k-1}, \tilde{n}_k)) + \Phi_c(\tau(n_{k-1}, \tilde{n}_k))$ where c is the cost of the spiral connecting trajectory points $n_{k-1}, \tilde{n}_k$.

And hence we could update $g(n_{k-1})$ as: $g(n_k) \leftarrow g(n_{k-1}) + c(e(n_{k-1}, n_k))$.

We could use the Dynamic Programming algorithm shown in Figure 6.7 to compute the optimal path with minimum cost, which starts from the initial node n_0, connecting the trajectory points along increasing longitudinal s-direction. Note that in the algorithm shown in Figure 6.7, connections between two trajectory points are computed in an ad-hoc fashion while searching for the optimal path on the graph. The $g(n)$ represents the cost of merely reaching the node n, and $\phi(n)$ represents the current path cost to node n, which includes both the cost of merely reaching node n and the additional cost of ending the path with node n. The former term $g(n)$ measures the additional cost incurred while choosing nodes to expand from current node to successor nodes (Figure 6.7, line 13). And the latter term $\phi(n)$ is the criteria when considering which predecessor nodes expand to current node (Figure 6.7, line 11). When all the computations of $g(n)$ and $\phi(n)$ have been finished, it is trivial to traverse the predecessor node map *prev_node* to construct the trajectory point with optimal(minimum) cost. Since candidate paths end at different trajectory points, we create a virtual node n_f and construct virtual edges connecting the last trajectory points to n_f, then our task becomes to find a path connecting node n_f to node n_f with minimum cost. As shown in the algorithm, after the computation of $g(n)$ and $\phi(n)$, the last actual trajectory point could be found by the algorithm in Figure 6.7.

```
1  function Search_DP(TrajectoryPointMatrix(V,E), {s}, {l})
2      Initialize map g : ∀n∈V, g(n) ← inf
3      Initialize map prev_node : ∀n∈V, g(n) ← null
4      for each sampled s_i ∈ {s} :
5          ∀n∈V s.t. s(n) = s_i : φ(n) ← inf
6          for each lateral direction Trajectory Point n = [s_i, l_j] :
7              if g(n) ≠ inf :
8                  Form the vehicle pose vector x̂_n = [x(n), y(n), θ(n), κ(n)]
9                  for each outgoing edge ẽ = (n, n')
10                     Form the polynominal spiral τ(ẽ(n, n'))
11                     if g(n) + Φ_C(n) < φ(n') :
12                         φ(n') ← g(n) + Φ_C(n)
13                         g(n') ← g(n) + c(τ)
14                         prev_node(n') ← n
15                     end if
16                 end for
17             end if
18         end for
19     end for
```

Figure 6.7: Finding the minimum cost path connecting trajectory points with Dynamic Programming.

The cost function in Figure 6.7, line 13 would require certain design considerations. Consider trajectory cost $\Omega(\tau) = c(\tau) + \Phi(\tau)$, where $c(\tau)$ represents the cost of the spiral connecting two s-direction adjacent trajectory points, the following factors need to be taken into consideration while designing the cost function.

- **Road-map related aspect:** We want the planned spiral paths to be close to the central reference line of lanes. For example, when behavioral decision is to follow on a straight lane, the planned path will have a larger cost when the planned path exhibits larger lateral distance (not approaching the central reference line).
- **Obstacle related aspect:** The planned path will have to be collision free. For example, with the grid division on the SL-coordinate lanes (Figure 6.6), any grid that has been occupied any obstacle, along with their proximate grids, should be assigned with extremely high costs to guarantee safety Note that the collision freeness is mostly about static obstacles in path planning, which is only in the spatial space. How to guarantee safety, especially for avoiding non-static obstacles in the spatial-temporal dimensions, will be addressed in speed planning.
- **Comfortableness and control feasibility:** Shape of the planned spiral paths should be smooth enough. This usually indicates smooth curvature change and slow change of derivatives of the curvature. In addition, not only the individual slices of planned spiral paths should be smooth, but also the connections between two spirals.

Regarding the $\Phi(\tau)$ cost which is more about the trajectory in general, we could only consider the longitudinal s-distance since speed planning will further address the problem. One way to design the cost function $\Phi(\tau)$ is from [9]:

$$\Phi(\tau) = -\alpha s_f(\tau) + h_d(s_f(\tau)),$$

$$h_d(s) = \begin{cases} -\beta \text{ if } s \geq s_{\text{threshold}} \\ 0 \quad \text{otherwise.} \end{cases}$$

The first term $-\alpha s_f(\tau)$ is the linear cost that prefers longer s by giving a discount, and the second term is a nonlinear cost which only gets triggered if s is larger than a threshold.

Speed Planning with ST-Graph

After a path has been determined by the path planning, motion planning module will compute how fast the autonomous vehicle will traverse along this path, which we refer as the *speed planning* problem. The inputs for speed planning are a few candidate paths, as well as the upstream behavior

decisions. And the constraints for speed planning are usually imposed by physical limits and comfortableness concerns such as rate of acceleration and wheel direction change. Since input paths are represented as sequences of point, the output of speed planning is to associate these points with desired speed information such as velocity, acceleration, curvature, and even their higher-order derivatives. As discussed previously in path planning, static obstacle information such as road shape is already taken into consideration while doing path planning. And in this section, we will introduce the concept of *ST-graph* to show how we formalize the speed planning problem into a search optimization problem on the *ST-Graph*.

In a typical *ST-Graph* (see Figure 6.8), two dimensions are usually considered, where T is the time axis and the S axis represents the tangential or longitudinal distance traveled along a given path. It is important to remember that when we talk about an *ST-Graph,* there is always an underlying pre-determined path. Any object whose predicted trajectories have intervals with the pre-determined ST-graph path will *cover* an area on the ST-graph. In addition, the *ST-graph* does not necessarily have to be 2D, but could also be a 3D *SLT-Graph*, where the additional dimension is the lateral distance perpendicular to the trajectory, denoted as the L dimension. In Figure 6.8, we use a detailed example to demonstrate how speed planning is done via a 2D ST-graph.

Consider a trajectory that represents a lane switch by trajectory planning, and there are two obstacle vehicles (**a** and **b**) at the destination lane (the left adjacent lane of the autonomous vehicle). Without loss of generality, let's consider that traffic prediction results of these two vehicles are both to follow their current lane with constant speed. These prediction results for **a** and **b** will have their perspective area covered as shown by the shadow areas in the *ST-graph* in Figure 6.8. We can see that, at any moment, projections of vehicles **a** and **b** onto the *ST-graph* are always line segments parallel to the S-axis, and these S-axis paralleled line segments are dragged along the t-axis as vehicles **a** and **b** move along the *ST-graph* trajectory, leading to the shadowed area (shadowed quadrilaterals) in Figure 6.8. Similar to path planning, we also divide the *ST-graph* area uniformly into small *Lattice Grid* and associate each grid with a cost. Then speed planning could be formalized into a minimal cost path search problem on the *ST-graph* lattice grids. At $t = 0$ our autonomous vehicle is at position $s = 0$, and it needs to reach $s = s_{end}$ eventually following a *ST-graph* path whose cost is minimal.

As shown in the figure, we compare three candidate speed planning paths on the *ST-graph*

- **Speed Plane 1:** The first one (Speed Plan 1) represents a path always behind vehicles **a** and **b** at any moment. The slope of Speed Plan 1 represents the speed of our autonomous vehicle, and the path will eventually reach $s = s_{end}$ position even though it is not explicitly drawn on the figure. Speed Plan 1 represents that our autonomous vehicle will *yield* both vehicle **a** and **b** by only entering the adjacent left lane after both **a** and **b** have passed.

- **Speed Plan 2:** The second trajectory that also starts from the origin. However, the slope of the path keeps increasing until our autonomous vehicle reaches a certain speed. Also the corresponding *s*-direction distance will surpass vehicle **a** at some point, but never exceeds the *s* distance of vehicle **b** at any moment. In real-world execution, the speed planning results of Speed Plan 2 is a typical movement of *yield* the leading vehicle **b** and *overtake* the trailing vehicle **a** by entering the gap between them.
- **Speed Plan 3:** The autonomous will accelerate until it surpasses both vehicle **a** and **b**. For any time, its *s* distance is always larger than vehicle **a** and **b**.

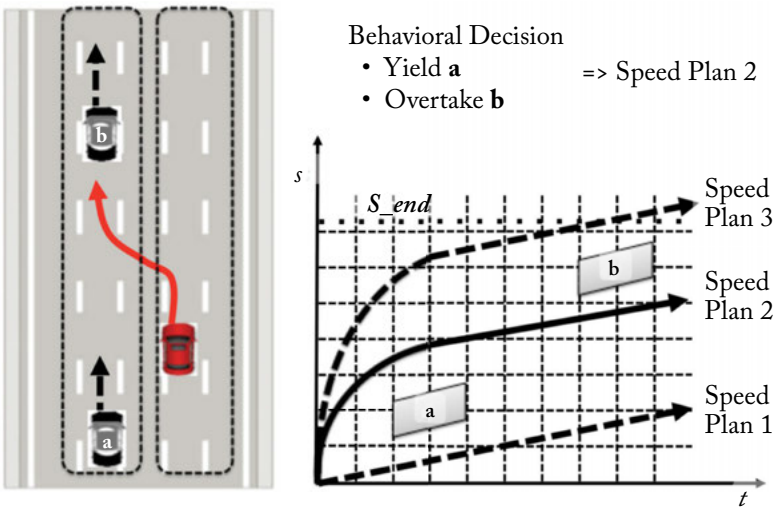


Figure 6.8: Speed planning via an ST graph.

Let's assume that the behavioral decision output for vehicle **a** is to "overtake" and to "yield" for **b**. Such behavioral decisions will assign lower costs to grids above the covered area of vehicle **a**, encouraging the algorithm to favor paths on the graph above **a**. Costs of grids below **b** will also be put lower to set algorithmic preferences over paths below **b**. Meanwhile, costs of grids below **a** or above **b** will get higher such that these areas could be avoided by the search algorithm.

Grids that are close enough to any obstacle will have very high costs to guarantee collision free. In addition, the speed-planning curve in general will have its costs associated with accelerations. For example, if the connection between two points on the speed planning curve is too steep, which represents a very large or even discontinuous acceleration, then the cost associated with the acceleration needed for these two speed planning points will be very prohibitively high. In fact, sharp speed increase will lead to control failure due to the large acceleration. And sometimes it is important to adjust the locations of speed-planning points inside a chosen grid to optimize

the whole speed planning trajectory cost. The speed planning trajectory on the *ST-graph* could be computed via graph search algorithms like A* or Dijkstra, given the cost profile of each grid and the grid definitions. After we have computed the speed planning curve on the *ST-graph*, we could easily retrieve speed as slope and acceleration as the derivative of slope for each trajectory point we want to output, and hence the motion planning phase is completed. With the path planning and speed planning module described above, we now have computed the behavior decisions based on our destination and surrounding environments, into a concrete trajectory with spatial and temporal information. We uniformly sample points from this spatial-temporal trajectory and send these sampled trajectory points with speed, acceleration, curvature, etc. to the downstream feedback control module. The feedback control module will output the actual control signals to physically manipulate the vehicle.

What worth mention here is that the *ST-Graph* is not only a very intuitive method in solving speed planning, it is also an important concept in the simultaneous longitudinal and lateral planning method which we are going to describe in the next subsection.

6.2.3 MOTION PLANNING WITH LONGITUDINAL PLANNING AND LATERAL PLANNING

Instead of doing path planning and then speed planning, [10] proposed the idea of doing longitudinal and lateral planning. The longitudinal and lateral dimensions naturally fit into the SL-coordinate system describe in Section 6.3.1, with s corresponding to the longitudinal direction and l to the lateral direction. In both the longitudinal and lateral dimension, the planning problem occurs in a space very similar with the speed-planning problem in the *ST-graph*. In fact, a simple but straightforward approach could be leverage the *ST-graph* solution and perform two graph searches on the *s-t* dimension and *l-t* dimension. However, the *ST-graph* fashioned approach we discussed above is more suitable for obstacle avoidance and obeying upstream behavioral decision, but not a perfect method for determining the optimal desired spatial-temporal trajectory. A desired motion planning trajectory should be smooth which is usually represented mathematically in continuity of pose with its derivatives in a dimension (e.g., *position*, *speed*, and *acceleration*). And ease and comfort can be best measured by *jerk*, which is the change rate of acceleration. Consider the jerk-optimal trajectory connecting a start state $P_0 = [p_0, \dot{p}_0, \ddot{p}_0]$ and an end state $P_1 = [p_1, \dot{p}_1, \ddot{p}_1$, where the pose P could be either the longitudinal s-pose or lateral l-pose, in a time frame of $T := t_1 - t_0$. Let's denote the integral of the squared *jerk* to be: $J_t(p_t) := \int^{t_1} \dddot{p}^2(\tau)\, d\tau$. An important proposition we have is that, for any cost function with the form:

$$C = K_j J_t + K_t g(T) + K_p h(p_1),$$

where g and h are arbitrary functions and $K_j, K_t, K_p > 0$, the optimal solution for minimizing the above cost function is a quintic polynomial. The proof from an optimal control's perspective is clear since the end point costs $g(T)$ and $h(p_1)$ do not change the Euler-Lagrange equation, and the formal proof could be illustrated in [10]. Intuitively, the cost function penalizes high jerk integral J_t along the trajectory, and also considers the time T and end pose $h(p_1)$.

Lateral Planning

We start the 1D planning with the lateral l-dimension. Then the start state P_0 becomes $D_0 = [d_0, \dot{d}_0, \ddot{d}_0]$, and we set this start state according to the actual end state of the previous planning frame, such that continuities are maintained. The end state P_1 are chosen from a bunch of feasible candidate lateral offset d's with $\dot{d}_1 = \ddot{d}_1 = 0$ since we prefer to move parallel to the central reference line direction (s-direction). Functions g and h are chosen with $g(T) = T$ and $h(d_1) = d_1^2$. We can see that slow convergence is penalized as well as any lateral difference with $d = 0$ at the end state. With the optimal solution taking the form of a quintic polynomial: $l_{optimal}(t) = a_5 t^5 + a_4 t^4 + a_3 t^3 + a_2 t^2 + a_1 t + a_0$ which minimizes the cost function of:

$$C_l = K_j J_t (l(t)) + K_t T + K_l d_1^2 .$$

The coefficients $a_5, a_4, a_3, a_2, a_1, a_0$ could be calculated with boundary conditions at $D_0 = [d_0, \dot{d}_0, \ddot{d}_0]$ and $D_0 = [d_1, \dot{d}_1, \ddot{d}_1]$. We could choose a set of candidate d_i's and compute a set of best candidate one-dimensional trajectories as $Solution_{i,j}$ from $[d_1, \dot{d}_1, \ddot{d}_1, T]_{i,j} = [d_1, 0, 0, T_j]$. Each candidate solution will have a cost. For each candidate, we check if this solution will be consistent with upstream behavioral decision outputs and make this solution in valid if any violations/collisions is found. The remaining valid trajectories make a candidate set of lateral dimension, and will be utilized in computing the planned trajectory on the 2D dimension.

The method described above works well for high-speed trajectories where longitudinal move xxx and lateral move xxx could be chosen independently. However, this assumption does not remain valid at low speed. At extreme low speeds, many of the generated trajectories in the lateral dimension will probably be invalid because of the non-holonomic property of vehicle control. When speed is low, a different mode of lateral dimension motion planning could be triggered. In this mode, the lateral trajectory is dependent on the longitudinal trajectory, and is denoted as $l_{low\text{-}speed}(t) = l(s(t))$. Since the lateral movement is dependent on the longitudinal movement, we treat the lateral movement as function of longitudinal position s. With the cost function being the same format but dependent on s the arc length rather than t, we modify the cost function into:

$$C_l = K_j J_t (l(s)) + K_t S + K_l d_1^2 ,$$

with $S = s_1 - s_0$ and all the derivatives are taken with respect to s instead of t. The optimal solution to this s-based cost function is a quintic function over s, and could be computed similarly with chosen T_j and end state d_i's.

Longitudinal Planning

The longitudinal dimensional planning is similar except that the end state that we want to track sometimes depends on a moving target. Let us use $s_{target}(t)$ as the trajectory we would like to track. The start state is $S_0 = [s_0, \dot{s}_0, \ddot{s}_0]$. The candidate trajectory set could be chosen with different Δs_i (either positive or negative) and T_j:

$$S_{i,j} = [s_1, \dot{s}_1, \ddot{s}_1, T_j] = [(s_{target}(T_j) + \Delta s_i), \dot{s}_{target}(T_j), \ddot{s}_{\text{target}}(T_j), T_j].$$

And we will discuss cost function settings with respect to different behavioral decision scenarios.

Following

The desired longitudinal movement for the autonomous vehicle along the s direction to follow a front vehicle is to maintain a minimum distance as well as a time gap behind the front vehicle. Here we want to emphasis the importance of traffic *prediction* output, since $s_{target}(t)$ will be dependent on the front vehicle predicted trajectory $s_{front\text{-}vehicle}(t)$ at the planning time frame/cycle. The target movement becomes:

$$s_{target}(t) = s_{front\text{-}vehicle}(t) - D_{min} - \gamma\, \dot{s}_{front\text{-}vehicle}(t),$$

which implies at least a distance D_{min} and a constant time gap γ given the front vehicle speed $\dot{s}_{front\text{-}vehicle}(t)$ at the end time. Given the desired end state as $[s_{desire}, \dot{s}_{desire}, \ddot{s}_{desire}]$, the cost function hereby becomes:

$$C_s = K_j J_t + K_t T + K_p (s_1 - s_{desire})^2.$$

If we assume that the predicted trajectory has **const** acceleration: $\ddot{s}_{front\text{-}vehicle}(t) = \ddot{s}_{front\text{-}vehicle}(t_0)$, then the speed $\dot{s}_{front\text{-}vehicle}(t)$ and position $s_{front\text{-}vehicle}(t)$ could be integrated as:

$$\dot{s}_{front\text{-}vehicle}(t) + \ddot{s}_{front\text{-}vehicle}(t_0) + \ddot{s}_{front\text{-}vehicle}(t_0)(t - t_0)$$

$$s_{front\text{-}vehicle}(t) = s_{front\text{-}vehicle}(t_0) + \dot{s}_{front\text{-}vehicle}(t_0)(t - t_0) + \tfrac{1}{2}\, \ddot{s}_{front\text{-}vehicle}(t_0)(t - t_0)^2.$$

And the end state longitudinal direction speed is $\dot{s}_{target}(t) = \dot{s}_{front\text{-}vehicle}(t) - \gamma \ddot{s}_{front\text{-}vehicle}(t)$ and acceleration is $\ddot{s}_{target}(t) = \ddot{s}_{front\text{-}vehicle}(t_1)$. Sometimes the *following* does not have a specific object,

but rather indicates following on a current lane. In this scenario, s_{desire} is no longer important, but $\dot{s}_{desire}$ is what we want to keep and it is usually a const. Then the cost function becomes:

$$C_s = K_j J_t + K_t T + K_{\dot{p}}(\dot{s}_1 - \dot{s}_{desire})^2.$$

In this case of no object to follow but speed keeping, the optimal trajectory set will be quartic polynomials instead of quintic with the set of $\Delta\dot{s}_i$ and T_j.

Switching Lane by Yielding and/or Overtaking

If switching to an adjacent lane by yielding to a vehicle, then it is similar to follow a front vehicle in the longitudinal sense, which we have already described in the previous case. Let's denote the target longitudinal movement to be: $s_{target}\,(t)$.

- When switching lane by yielding a vehicle **b**, the target trajectory becomes:

$$s_{target}\,(t) = s_b\,(t) - D_{min\text{-}yield} - \gamma\dot{s}_b\,(t).$$

- When switching lane by overtaking a vehicle **a**, the target trajectory becomes:

$$s_{target}\,(t) = s_a\,(t) + D_{min\text{-}overtake} - \gamma\dot{s}_a\,(t).$$

- When merging into two vehicles **a** and **b** (for example, Figure 6.8), the target trajectory becomes:

$$s_{target}\,(t) = 1/2\,[s_a\,(t) + s_b\,(t)].$$

Stop

When the autonomous vehicle needs to stop at a pedestrian cross or traffic signal line. The same form of cost function could be used, and the target movement will become constant for the s-direction $s_{target}\,(t) = s_{stop}$ with first- and second-order derivatives being 0 ($\dot{s}_{target}\,(t) = 0$ and $\ddot{s}_{target}\,(t) = 0$).

After both longitudinal and lateral candidate trajectory sets have been computed, the optimal trajectory comes from $|Traj_{longitudinal}| \times |Traj_{lateral}|$ possible combinations. Each combined trajectory is checked against behavioral decision output for abeyance as well as collision free. In addition, trajectories which push toward feedback control limit will also be filtered out. The final trajectory will be chosen from the remaining valid trajectories with minimum weighted longitudinal and lateral cost.

6.3 FEEDBACK CONTROL

From the stand-alone controlling point of view, the feedback control module of autonomous driving has no essential difference with general mechanic control. Both autonomous vehicle control and general mechanic control are based on certain pre-defined trajectories, and track the difference between the actual pose and the pose on the pre-defined trajectory by continuous feedback. [11] listed a lot of works on autonomous vehicle feedback control, in which [8, 12] introduced additional obstacle avoidance and route optimization to the traditional feedback control system. With our proposed system architecture of autonomous planning and control, the feedback control module in our system could mostly leverage existing work of traditional vehicle pose feedback control. Since this part of work is relatively mature and traditional, it is not the focus of this autonomous driving book. To provide a basic knowledge of feedback control for autonomous driving, we will just introduce two important concepts: Vehicle Bicycle Model and PID Feedback Control System [13] [14]. Readers could refer to [12] for more detailed descriptions of other feedback control systems in autonomous driving.

6.3.1 BICYCLE MODEL

In Section 3.1, we briefly introduced vehicle model to better describe the trajectory generation algorithm in Motion Planning. Here we will describe in details a frequently used vehicle model in autonomous driving Feedback Control: *Bicycle Model*. The pose represented by the bicycle model is within a 2D plane. The vehicle pose could be fully described by the central position (x,y) and heading angle θ between the vehicle and the 2D plane's x-axis. Under this model, the vehicle is considered to be a rigid body with the front and rear wheels connected by a rigid axis. The front wheels could freely rotate within a certain angle range, while the rear wheels stay parallel to the vehicle body and could not rotate. The rotation of the front wheels corresponds to the degree position of the steering wheel. An important characteristic of the bicycle model is that vehicles cannot make lateral movements without moving forward (making longitudinal movements), and such characteristic is also addressed as nonholonomic constraint. Under this vehicle model, the nonholonomic constraint is usually expressed as differential derivative equations or inequations. We also ignored the inertial and slippery effect at the contact point between the tire and the ground surface. Such ignorance does not have any significant impact under low speed and brings very little error. However, when at high speed, the effect of inertial on feedback control is significant and could not be safely ignored. The physical vehicle model at high speed with inertial effect is more complicated and interested readers could refer to [12] for such vehicle models at high speeds.

The vehicle pose representation in the bicycle model is shown in Figure 6.9. We use a xy-based plane as the 2D plane, where $\hat{e}_x$ and $\hat{e}_y$ separately represent the unit vector in the x and y directions. p_r and p_f stand for the front wheel contact point and the rear wheel contact point. The

heading angle θ is the angle between the vehicle and the x-axis (angle between vector p_r and unit vector $\hat{e}_x$). The steering wheel rotation angle δ is defined as angle between the front wheel direction and the vehicle body, where the ground contact points of front and rear wheels (p_f and p_r) satisfy the following properties:

$$(\dot{p}_r \cdot \hat{e}_y)\cos(\theta) - (\dot{p}_r \cdot \hat{e}_x) = 0$$
$$(\dot{p}_f \cdot \hat{e}_y)\cos(\theta + \delta) - (\dot{p}_f \cdot \hat{e}_x)\sin(\theta + \delta) = 0\ ,$$

where $\dot{p}_f$ and $\dot{p}_r$ are the instant speed vector of front and rear wheel at their ground contact point. Consider the scalar projections of the rear wheel speed at the x-axis and y-axis: $x_r := p_r \cdot \hat{e}_x$ and $x_y := p_r \cdot \hat{e}_y$, along with the tangential speed at the rear wheel $v_r := \dot{p}_r \cdot (p_f - p_r)/\|p_f - p_r\|$, then the above constraints between p_f and p_r could also be written as:

$$\dot{x}_r = v_r \cos(\theta)$$
$$\dot{y}_r = v_r \sin(\theta)$$
$$\theta = v_r \tan(\delta)/l\ ,$$

where represents length of the vehicle (distance between front axis center and rear axis center). And similarly, the relationship among the front wheel variables could be written as:

$$\dot{x}_f = v_r \cos(\theta + \delta)$$
$$\dot{y}_f = v_r \sin(\theta + \delta)$$
$$\theta = v_f \sin(\delta)/l\ .$$

And here the scalar variables of front and rear wheel speeds satisfy: $v_r = v_f \cos(\delta)$.

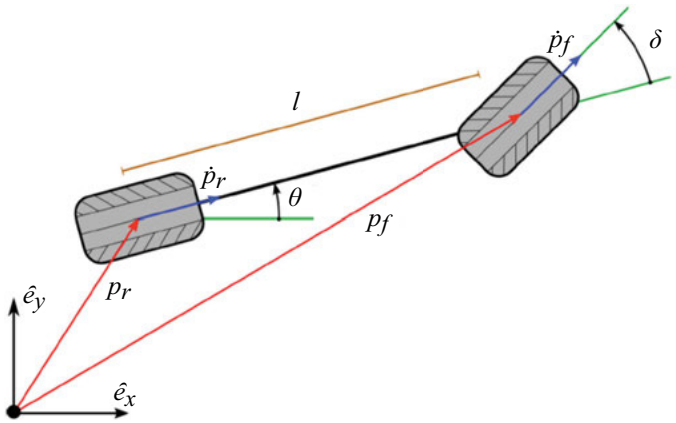


Figure 6.9: Bicycle model in feedback control [5], used with permission.

With the bicycle model described above, the goal of control is to find the steering wheel $\delta \in [\delta_{min}, \delta_{max}]$ and forward speed $v_r \in [\delta_{min}, \delta_{max}]$, which satisfy the physical pose constraints. In practice and for simplicity, the control outputs are steering-wheel angle change rate ω and the throttle/brake percentage instead of actual steering wheel or forward speed goals. The relationship between ω and δ is simplified as: $\tan(\delta)/l = \omega / v_r = \kappa$. And here the problem gets reduces to find the satisfying steering wheel change rate δ. Such simplification is called the *Unicycle Model*, with the characteristic that the forward speed has been simplified to be only dependent on the vehicle axis length and the steering angle change rate.

6.3.2 PID CONTROL

The most typical and widely used algorithm in autonomous vehicle feedback-control is the PID feedback control system as shown in Figure 6.10, where the term $e(t)$ represents the current *tracking error* between desired pose variable and actual pose variable. The variable to track could be the longitudinal/lateral difference along a trajectory, angle/curvature difference at various trajectory points, or even a comprehensive combination of these vehicle pose variables. In Figure 6.10, the P controller represents the feedback for the current tracking error, whose coefficient is governed by K_p ; I and D controllers represent the integral and differential part, whose coefficients are separately governed by K_I and K_D.

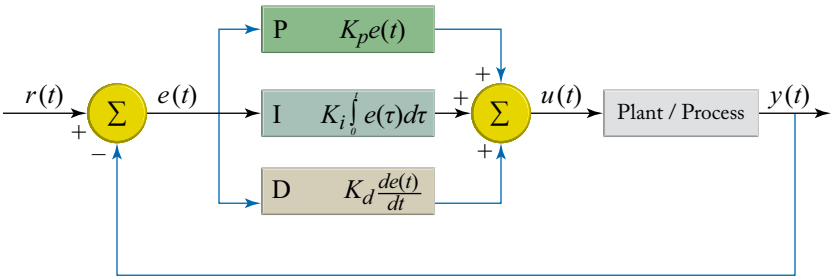


Figure 6.10: PID based feedback-control system (based on [13]).

As per to the feedback control module in autonomous vehicle, the task is to control the vehicle to follow the upstream motion planning output trajectory as closely as possible. Here we propose to use the methodology in [15], and leverage two PIC controllers to separately control the steering-wheel angle δ and the forward speed V_s. At a given time frame n, the PID controller for the steering-wheel angle is as follows:

$$\delta_n = K_1\theta_e + K_2 l_e / V_s + K_3 \dot{l}_e + K_4 \sum_{i=1}^{n} l_e \Delta t \,.$$

The variables θ_e and l_e are all tracking error terms between the actual pose and the desire pose on the motion-planning output trajectory point at this time frame n. For each time frame at this time frame n, the corresponding pose on the motion planning output trajectory point is addressed as the *reference point*. θ_e represents the angle difference between the vehicle pose heading and the reference point heading, and l_e tracks lateral difference between the vehicle actual lateral position and the reference point lateral position. V_s is the forward speed. We can see the coefficient K_1 and K_2 are serving for the P controller, while K_3 governs the differential part (D controller) and K_4 for integral part (I controller). Given this steering-angle controller serves for direction, the other PID controller is more about the forward speed V_s along the longitudinal direction (s-direction), and controls throttle/brake output. This controller considers the difference between the actual vehicle pose curvature xxx and the reference point curvature xxx. From these curvatures, we could design a function xxx to track the forward speed error. Then the longitudinal forwards speed goal becomes: xx. Given this desired forward speed to track and the actual forward speed xxx, the PID controller for the forward speed could be written as:

$$V_e = V_{desired} - V_s$$
$$U_V = K_p V_e + K_I \sum V_e \Delta t + K_D \Delta V_e / \Delta t \ ,$$

where K_p, K_I, and K_D separately represents the gain for the proportional, integral, and differential part, and U_V represents throttle/brake output for this given time frame n.

These two PID controllers discussed here are the most typical and basic implementation practices for the feedback *control* module in autonomous vehicles. To make a even better passenger experience in autonomous driving, more complicated feedback control systems will be necessary to further track and tune variables such as curvature and jerk. The problem of generating delicate and accurate control to enforce an object to follow a pre-defined trajectory is not an unique problem for autonomous driving, and there are ample existing solutions which interested readers could refer to [15].

6.4 CONCLUSIONS

With Chapter 5 and Chapter 6, we depicted the general architecture of autonomous vehicle planning and control, consisting of *routing*, behavioral *decision*, traffic *prediction*, motion *planning*, and feedback *control*. All these modules have existing and working proposals in both academia and industry. These existing proposals exhibit different strengths, either in solid theoretical foundation or practical industry practices. In fact, we believe the challenge of autonomous driving does not lie in solving the problem on any individual level or module, but rather in the philosophy behind how we partition the big autonomous vehicle planning and control problem into distinctive layers and

solve these divided problems in a coherent fashion. From this perspective, we do not aim to bring all possible solutions in a survey-like fashion. However, we do bring to the readers a consistent series of working solutions for each module. Our focus is to illustrate clearly what is the problem scope and definition for each module, and how they together solve the general problem following the data flow from abstractness to concreteness. In each module or layer, the problem is formalized and solved with practical industrial solutions. We also discussed how each downstream layer will utilize the upstream module output as input and further computes more concrete solutions toward the data flow to actual control signals. We hope that, by showing this idea of a *divide and conquer* approach, we could help the readers with their understanding of general autonomous vehicle planning and control.

6.5 REFERENCES

[1] Montemerlo, M., Becker, J., Bhat, S., Dahlkamp, H., Dolgov, D., Ettinger, S., Haehnel, D., Hilden, T., Hoffmann, G., Huhnke, B., Johnston, D., Klumpp, S., Langer, D., Levandowski, A., Levinson, J., Marcil, J., Orenstein, D., Paefgen, J., Penny, I., Petrovskaya, A.,Pflueger, M., Stanek, G., Stavens, D., Vogt, A., and Thrun, S. 2008. Junior: The Stanford entry in the urban challenge. *Journal of Field Robotics: Special Issue on the 2007 DARPA Urban Challenge* 25(9), pp. 569–597. 108

[2] Urmson,C. , Anhalt, J., Bagnell, D., Baker, C., Bittner, R., Clark, M. N., Dolan, J., Duggins, D., Galatali, T., Geyer, C., Gittleman, M., Harbaugh, S., Hebert, M., Howard,T. M., Kolski, S., Kelly, A., Likhachev, M. McNaughton, M., Miller, N., Peterson, K., Pilnick, B., Rajkumar, R., Rybski, P., Salesky, B., S, Y. W., Singh, S., Snider, J., Stentz, A., Whittaker, W., Wolkowicki, Z., Ziglar, J., Bae, H., Brown, T., Demitrish, D., Litkouhi, B., Nickolaou, J., Sadekar, V., Zhang, W., Struble, J., Taylor, M., Darms, M., and Ferguson, D. 2008. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics: Special Issue on the 2007 DARPA Urban Challenge* 25(9), pp. 425–466. 108

[3] Buehler, M., lagnemma, K., and Singh, S. (Eds.) 2009. *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*. Springer-Verlag Berlin Heidelberg. DOI: 10.1007/978-3-642-03991-1. 108

[4] Katrakazas, C., Quddus, M., Chen, W. H., and Deka, L. 2015. Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. Elsevier Transporation Research Park C: *Emerging Technologies* Vol. 60, pp. 416–442. DOI: 10.1016/j.trc.2015.09.011. 108, 118

[5] Paden, B., Cap, M., Yong, S. Z., Yershow, D., and E. Frazzolo. 2016. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles* 1(1), pp. 33–55. DOI: 10.1109/TIV.2016.2578706. 108, 118, 133

[6] Brechtel,S. and Dillmann, R. 2011. Probabilistic MDP-behavior planning for cars. *IEEE Conference on Intelligent Transportation Systems*. DOI: 10.1109/ITSC.2011.6082928. 110

[7] Ulbrich, S. and Maurer, M. 2013. Probabilistic online POMDP decision making for lane changes in fully automated driving. *16th International Conference on Intelligent Transportation Systems (ITSC)*. DOI: 10.1109/ITSC.2013.6728533. 110, 118

[8] Gu, T., Snider, J., M. Dolan, J. and Lee J. 2013. Focused trajectory planning for autonomous on-road driving. *IEEE Intelligent Vehicles Symposium (IV)*. DOI: 10.1109/IVS.2013.6629524. 118, 132

[9] Mcnaughton, M. 2011. Parallel algorithms for real-time motion planning. Doctoral Dissertation. Robotics Institute, Carnegie Mellon University. 118, 119, 121,125

[10] Werling, M., Ziegler, J., Kammel, S., and Thrun, S. 2010. Optimal trajectory generation for dynamic street scenarios in a frenet frame. In *2010 IEEE International Conference on Robotics and Automation (ICRA)*, (pp. 987–993). IEEE. DOI: 10.1109/ROBOT.2010.5509799. 129

[11] Zakaria, M. A., Zamzuri, H., and Mazlan, S. A. 2016. Dynamic curvature steering control for autonomous vehicle: Performance analysis. *IOP Conference series: Materials Science and Engineering 114,* 012149. DOI: 10.1088/1757-899X/114/1/012149. 132

[12] Connors, J. and Elkaim, G.H. 2008. Trajectory generation and control methodology for an ground autonomous vehicle. *AIAA Guidance, Navigation and Control Conference*. 132

[13] https://en.wikipedia.org/wiki/PID_controller. 132, 134

[14] National Instruments: http://www.ni.com/white-paper/3782/en/.132

[15] Zakaria, M.A., Zamzuri, H., and Mazlan, S.A. 2016. Dynamic curvature steering control for autonomous vehicle: Performance analysis. *IOP Conference series: Materials Science and Engineering 114,* 012149. DOI: 10.1088/1757-899X/114/1/012149. 134, 135