Fakultät für Informatik
Lehrstuhl für Echtzeitsysteme und Robotik

Motion Planning for Nonholonomic Vehicles with Space Exploration Guided Heuristic Search

Dipl.-Ing. Univ. Chao Chen

Vollständiger Abdruck der von der Fakultät für Informatik der Technische Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Prof. Dr. Thomas Huckle
Prüfer der Dissertation:	1. Prof. Dr.-Ing. habil. Alois Christian Knoll
	2. Prof. Dr.-Ing. Rüdiger Dillmann

Die Dissertation wurde am 03.03.2016 bei der Technische Universität München eingereicht und durch die Fakultät für Informatik am 01.08.2016 angenommen.

Abstract

The development of the modern sensing, actuation, communication, and computation technology unfolds a promising future of mobile robots, especially for intelligent automobiles. Motion planning is one of the most important software components in these autonomous systems. It is responsible for a motion strategy or trajectory while considering the robot kinematic model, the prior knowledge of the environment, the real-time perception, and domain specific rules. Particularly, a motion planning problem for an autonomous vehicle is beyond a theoretical problem of finding an executable collision-free trajectory. An intelligent vehicle should behave rationally in traffic, which includes following a global route from the navigation, adapting local behaviors according to the circumstances, obeying traffic rules, and respecting the convenience of the human traffic participants. Furthermore, the driving tasks are highly diverse, so specific motion planning methods are required for different maneuver types. In this case, an intelligent agent should be able to process all kinds of information with domain knowledge, select the most efficient algorithm for a specific task, and integrate them seamlessly in a system. In this dissertation, a *Space Exploration Guided Heuristic Search* (SEHS) method is introduced as a base framework for mobile robot motion planning. Several extensions of SEHS such as *Orientation-Aware Space Exploration Guided Heuristic Search* (OSEHS) and *Space Time Exploration Guided Heuristic Search* (STEHS) are developed for specific autonomous driving scenarios. They can be integrated in a hierarchical planning architecture with a high-level task planning in order to process information in different layers and achieve online motion planning for mobile robots, especially autonomous vehicles.

The general idea of the *Space Exploration Guided Heuristic Search* approach is to boost the forward search in continuous state space with heuristics from the subspace properties such as topology and dimension. The SEHS framework consists of two sequential procedures: (1)*Space Exploration* investigates a low dimensional subspace with distance queries for a path corridor connecting the start and goal locations, (2) *Heuristic Search* propagates robot states with primitive motions following the path corridor for a trajectory from the start state to the goal state. The space knowledge obtained in the exploration phase enables the search algorithm to adapt the step size of motion primitives and the state resolution regarding the free-space dimension. Furthermore, as a generic exploration and search framework, the SEHS method can flexibly scale the subspace to robot orientation for maneuvering in narrow places as *Orientation-Aware Space Exploration Guided Heuristic Search*, or include time domain for dynamic scenarios as *Space Time Exploration Guided Heuristic Search*.

A *Motion Planning Engine* (MPE) is developed as an online planning agent for autonomous driving. Based on the hybrid planning schema, task planning is introduced as the middle layer between route planning and motion planning. It evaluates the symbolic states of the system with traffic domain knowledge, so that a sequence of driving tasks is selected to accomplish the driving mission. Each task can be

handled as a motion planning problem, which has an appropriate context in a moderate scope, so that it can be efficiently solved with a suitable planning algorithm in the motion planning layer. Furthermore, the planning results are continuously verified in the real-time situation, and re-planned when necessary. Thus, the MPE interfaces the inputs such as traffic rules, road-map, object-list, and vehicle state, plans motion with steering and acceleration commands, and guarantees safety in the whole duration of driving.

In addition, a C++ library, named *AutoDrive* for autonomous driving, is developed. The library consists of packages with general math utilities, vehicle kinematic models, environment models, and various planning algorithms. A simulation environment is also provided to visualize planning procedures and results in different traffic scenarios. The *AutoDrive* library provides a compact and essential software base for education, research, and prototype development for mobile robots and autonomous driving applications.

Zusammenfassung

Die Entwicklung der modernen Sensorik, Aktorik, Kommunikations- und Rechentechnik lässt eine vielversprechende Zukunft von mobilen Robotern erwarten, vor allem für intelligente Fahrzeuge. Bewegungsplanung ist eine der wichtigsten Software-Komponenten in diesen autonomen Systemen. Sie ist für eine Bewegungsstrategie oder eine Trajektorie verantwortlich. Die gilt vor allem in Bezug auf das kinematische Modell des Roboters, das Vorwissen über die Umgebung, die Echtzeit-Wahrnehmung und die domänenspezifische Vorschriften. Insbesondere ist ein Bewegungsplanungsproblem für ein autonomes Fahrzeug mehr als das theoretische Problem, eine ausführbare kollisionsfreie Trajektorie zu finden. Ein intelligentes Fahrzeug sollte sich auch im Straßenverkehr rational verhalten. Das umfasst das Folgen eines globalen Pfads einer übergeordneten Navigationskomponente, die Anpassung des lokalen Verhaltens an äußere Umstände, sowie die Befolgung von Verkehrsregeln. Dies alles, unter Berücksichtigung des Komforts menschlicher Verkehrsteilnehmer. Darüber hinaus sind die Fahraufgaben sehr vielfältig, was in der Regel spezifisch optimierte Bewegungsplanungsmethoden für die unterschiedliche Manövertypen erfordert. In diesem Fall sollte ein intelligenter Agent in der Lage sein, alle möglichen Informationen mit Domänenkenntnissen zu bearbeiten, effizienteste Algorithmen für bestimmte Aufgaben zu wählen und alle Funktionen nahtlos in ein System zu integrieren. In dieser Dissertation wird eine *Space Exploration Guided Heuristic Search* (SEHS) Methode als Basis für die Bewegungsplanung mobiler Roboter eingeführt. Mehrere Erweiterungen von SEHS wie *Orientation-Aware Space Exploration Guided Heuristic Search* (OSEHS) und *Space Time Exploration Guided Heuristic Search* (STEHS) sind für bestimmte autonome Fahrszenarien entwickelt worden. Sie können in einer hierarchischen Planungsarchitektur mit einer High-Level-Aufgabenplanung integriert werden, um Daten in verschiedenen Schichten zu verarbeiten und eine Online-Bewegungsplanung für mobile Roboter, insbesondere für autonome Fahrzeuge, zu realisieren.

Die Grundidee des *Space Exploration Guided Heuristic Search* Ansatzes ist es, die Vorwärtssuche in kontinuierlichem Zustandsraum mit Heuristiken von der Eigenschaften des Unterraums zu unterstützen, wie etwa dem Wissen über Topologie und Abmessung des Raumes. Das SEHS Framework besteht aus zwei aufeinanderfolgenden Verfahren: (1) *Space Exploration* untersucht einen niedrigdimensionalen Unterraum mit Abfragen über Abstände für einen Pfadkorridor zwischen Start- und Zielpose, (2) *Heuristic Search* propagiert Zustände des Roboters mit Bewegungsprimitiven zum Korridorweg für eine Trajektorie vom Startzustand zum Zielzustand. Die Kenntnisse über den Raum, die in der Explorationsphase erhalten werden, ermöglichen es dem Suchalgorithmus, die Schrittweite der Bewegungsprimitiven und die Auflösung der Zustände zur Freiraumdimension anzupassen. Ferner lässt sich mit dem SEHS Verfahren als generisches Explorations- und Such-Framework der Unterraum flexibel skalieren, um etwa die Ausrichtung des Roboters für Manöver in engen Stellen bei *Orientation-Aware Space Exploration Guided Heuristic Search*, oder mit

der Zeitdomäne für dynamische Szenarien bei *Space Time Exploration Guided Heuristic Search*.

Eine *Motion Planning Engine* (MPE) wird als Online-Planungsagent für autonomes Fahren entwickelt. Auf Basis eines Hybridplanungsschemas wird Aufgabenplanung als eine Mittelschicht zwischen Routenplanung und Bewegungsplanung eingeführt. Diese evaluiert die symbolischen Zustände des Systems mit domänenspezifischem Wissen über den Verkehr, damit eine Folge von Fahraufgaben ausgewählt werden kann, um die Fahrmission auszuführen. Jede Aufgabe kann als ein Bewegungsplanungsproblem dargestellt werden, das mit einem geeigneten Kontext in angemessenem Umfang beschränkt wird, so dass es mit einem passenden Planungsalgorithmus in der Bewegungsplanungsebne effizient gelöst werden kann. Außerdem werden die Planungsergebnisse kontinuierlich mit der Echtzeitsituation verifiziert und falls nötig neu geplant. Somit verbindet die MPE die Inputs wie Verkehrsregeln, Fahrplan, Objekt-Liste und Fahrzeugzustand, plant Bewegungen mit Lenk- und Beschleunigungsbefehlen und gewährleistet die Sicherheit während des gesamten Fahrvorgangs.

Darüber hinaus wurde eine C++ Bibliothek mit dem Namen *AutoDrive* für autonomes Fahren entwickelt. Die Bibliothek besteht aus Paketen mit Funktionen für allgemeine Mathematik, Fahrzeugkinematik, Umwelt-Modelle und verschiedenen Planungsalgorithmen. Eine Simulationsumgebung wurde ebenfalls bereitgestellt, um Planungsverfahren und Ergebnisse in verschiedenen Verkehrsszenarien zu visualisieren. Die *AutoDrive* Bibliothek bietet eine kompakte und essenzielle Software-Basis für Bildung, Forschung und die Prototypentwicklung von mobilen Robotern oder autonomen Fahranwendungen.

This thesis is dedicated to my grandparents.

Acknowledgements

The major part of the research work in this thesis is done in fortiss from 2012 to 2015, where is a nice place to work as it provides close links to the university as well as the practical world. The mission of transferring the innovative technology from academic to industry gives me a great chance to apply my research result.

First of all, I would like to thank Prof. Dr. Alois Knoll for the precious opportunity to study the exciting topic of robot motion planning. His vision gave much valuable advice to my research work. He also provided me with the great chance to share my knowledge with the students by teaching the autonomous driving lecture at TUM. I would also like to thank Prof. Dr. Rüdiger Dillmann for the time to know my work, and the inspirational discussion during my defense.

Special thanks to Dr. Markus Rickert, who led me into the fascinating research field of robotics and supported me wholeheartedly in many projects, research work, software development, and publications. Many thanks to Prof. Oussama Khatib, Sebastian Thrun, Ph.D., and Peter Norvig, Ph.D. for the interesting online lectures, that taught me the fundamental knowledge about robotics, artificial intelligence, and autonomous driving.

As a member of the autonomous driving research team in fortiss, it is grateful to work with many smart and kind colleagues, David Lenz, Pascal Minnerup, Dominik Bauch, Martin Büchel, Tobias Kessler, and also Erwin Roth from AEV together. Thanks to the people in the robotic group, especially Dr. Manuel Giuliani and Dr. Andre Gaschler. Thanks to Dr. Harald Ruess, Dr. Christian Buckl and all the other people for the great time at fortiss. Thanks to Dr. Georg von Wichert and Thomas Woesch from Siemens for the nice cooperation in RACE project, which is the first time to put my algorithm on a real demonstrator.

I would also like to thank John Dolan, Ph.D. and Tianyu Gu for the kind invitation to Carnegie Mellon University. It is a great pleasure to meet them and have fruitful conversations at many conferences.

Finally, I would like to address my greatest thanks to my family for their selfless support to my study and living in Germany, especially my wife Liwen Zhao and my children Yuhong and Yuqiu, who gave me infinite courage and energy to pursuit my research goals and finish my thesis.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Motion Planning Problem . . . 3
- 1.3 State of the Art . . . 4
  - 1.3.1 Potential Field Methods . . . 5
  - 1.3.2 Random Sampling Methods . . . 5
  - 1.3.3 Heuristic Search Methods . . . 6
  - 1.3.4 Planning with Space Decomposition . . . 7
  - 1.3.5 Planning with Dynamic Environments . . . 8
  - 1.3.6 Task Planning . . . 9
  - 1.3.7 Advanced Driver Assistant Systems . . . 9
- 1.4 Contributions . . . 10
  - 1.4.1 Space Exploration Guided Heuristic Search Algorithms . . . 10
  - 1.4.2 Real-Time Planning Engine Combining Task Planning and Motion Planning 11
  - 1.4.3 AutoDrive Library . . . 12
- 1.5 Outline . . . 12

2 Space Exploration Guided Heuristic Search **13**
- 2.1 General Idea . . . 13
- 2.2 Space Exploration . . . 14
  - 2.2.1 Problem Definition . . . 14
  - 2.2.2 Circle-Based Space Exploration Algorithm . . . 15
    - 2.2.2.1 Euclidean Distance Heuristic Estimation . . . 17
    - 2.2.2.2 Circle-Based Vertex Expansion . . . 18
    - 2.2.2.3 Redundant Vertices with Circle Overlapping . . . 21
  - 2.2.3 Circle-Path Optimization . . . 22
  - 2.2.4 Multi-Directional Exploration . . . 22
- 2.3 Heuristic Search . . . 23
  - 2.3.1 Problem Definition . . . 24
  - 2.3.2 Vehicle Kinematic Models . . . 24
    - 2.3.2.1 Constant Curvature Model . . . 25
    - 2.3.2.2 Continuous Curvature Model . . . 25
    - 2.3.2.3 Kinodynamic Curvature Model . . . 26
    - 2.3.2.4 Single Track Model . . . 26
  - 2.3.3 Primitive Motion Based Heuristic Search Algorithm . . . 26
    - 2.3.3.1 Circle-Path States Clustering and Heuristic Estimation . . . 29
    - 2.3.3.2 Search Step-Size and Resolution Adaptation . . . 30

2.3.3.3 Motion Cost Calculation . . . 31
2.4 Experiments . . . 32
2.4.1 Local Minimum Scenario . . . 34
2.4.2 Narrow Passage Scenario . . . 35
2.4.3 Simple Navigation Scenario . . . 37
2.4.4 Large Labyrinth Scenario . . . 37
2.5 Summary . . . 40

3 Orientation-Aware Space Exploration Guided Heuristic Search 41
3.1 General Idea . . . 41
3.2 Orientation-Aware Space Exploration . . . 42
3.2.1 Cost Function for Orientation Change . . . 42
3.2.2 Directed Circle Expansion . . . 43
3.2.3 Directed Circle Overlapping . . . 44
3.2.4 Example Scenario . . . 45
3.3 Orientation-Aware Heuristic Search . . . 45
3.3.1 Directed Circle-Path Heuristic . . . 46
3.3.2 Motion Cost Adaptation . . . 48
3.3.3 Motion Direction Adaptation . . . 49
3.4 Experiments . . . 49
3.4.1 Cross Parking Scenario . . . 50
3.4.2 Parallel Parking Scenario . . . 51
3.4.3 Maneuvering Through Narrow Space . . . 52
3.5 Summary . . . 54

4 Space-Time Exploration Guided Heuristic Search 55
4.1 General Idea . . . 55
4.2 Cylinder-Based Space-Time Exploration . . . 56
4.2.1 Heuristic Time Cost . . . 56
4.2.2 Cylinder Expansion and Overlapping . . . 57
4.2.3 Example Scenario . . . 58
4.3 Cylinder Path Guided Heuristic Search . . . 59
4.3.1 Cylinder-Path Heuristic . . . 59
4.3.2 State Expansion and Resolution . . . 59
4.3.3 Example Scenario . . . 60
4.4 Experiments . . . 61
4.4.1 Overtaking Scenario . . . 61
4.4.2 Intersection Scenario . . . 62
4.5 Summary . . . 64

5 SEHS Motion Planning Engine with Task Planning 65
5.1 System Architecture . . . 65
5.2 Driving Task Planning . . . 67
5.2.1 Domain Definition . . . 68
5.2.1.1 World Model . . . 68
5.2.1.2 Predicates . . . 69
5.2.1.3 Tasks . . . 70
5.2.2 Task Planning . . . 72
5.2.3 Example Scenarios . . . 73
5.2.3.1 Overtaking on the Motorway . . . 73

5.2.4 Round-About with Blockage . . . 75
5.3 SEHS Motion Planning Engine . . . 77
5.3.1 Apply Traffic Knowledge in SEHS . . . 77
5.3.1.1 Roadmap Knowledge . . . 77
5.3.1.2 Driving Behavior Knowledge . . . 78
5.3.2 Incremental Re-planning with Safety Motion . . . 80
5.3.2.1 Operation Modes and Planning Period . . . 80
5.3.2.2 Validation and Re-planning . . . 81
5.4 Applications . . . 81
5.4.1 Precise Parking onto an Inductive Charging Station . . . 82
5.4.2 Transit and Parking in Semi-Structured Environments . . . 84
5.4.3 Automatic Driving through Intersections . . . 85
5.5 Summary . . . 92

6 AutoDrive Library **93**
6.1 Structure Overview . . . 93
6.2 Mathematical Tools . . . 94
6.2.1 Real Number Arithmetic . . . 94
6.2.2 Geometric Models and Operations . . . 95
6.2.3 Random Distributions and Generators . . . 96
6.2.4 Matrix and Grid . . . 96
6.3 Vehicle Models . . . 97
6.3.1 States and Trajectory . . . 97
6.3.2 Segments and Path . . . 97
6.3.3 Vehicle Motions . . . 98
6.3.4 Configuration Metrics . . . 98
6.4 Environment Models . . . 99
6.4.1 Object List . . . 99
6.4.2 Grid Map . . . 99
6.5 Motion Planning . . . 100
6.5.1 Planner Abstraction . . . 100
6.5.2 Heuristics . . . 100
6.5.3 Space Explorers . . . 101
6.5.4 Heuristic Search Planners . . . 102
6.5.5 Sampling Based Planners . . . 102
6.5.6 Trajectory Generator . . . 103
6.5.7 Task Planner . . . 103
6.6 Demo in 3D Scene . . . 103
6.7 Summary . . . 104

7 Conclusion **105**

A Vehicle Kinematic Models **107**
A.1 Constant Curvature Model . . . 107
A.2 Continuous Curvature Model . . . 108
A.3 Kinodynamic Curvature Model . . . 110
A.4 Single Track Model . . . 111

List of Figures

1.1 Motion Planning Engine system architecture 11

2.1 Space exploration in a labyrinth 17
2.2 Comparing space explorations with greedy search and heuristic search. 18
2.3 Circle expansion with uniform interpolation 19
2.4 Comparing random and uniform circle expansions. 20
2.5 Comparing number of vertices in each expansion step. 20
2.6 Circle overlapping 21
2.7 Circle-path optimization 22
2.8 Comparing bidirectional and single directional space explorations. 23
2.9 Heuristic search with the guidance of space exploration in a labyrinth 28
2.10 Circle based states clustering for heuristic search 29
2.11 Circle-path heuristic distance estimation 30
2.12 Comparing constant step size and adaptive step size in heuristic search 31
2.13 Comparing heuristic search with simple cost and complex cost functions 32
2.14 Experiment scenarios 33
2.15 Local minimum scenario 34
2.16 Narrow passage scenario 36
2.17 Narrow navigation scenario 37
2.18 Large labyrinth scenario 38

3.1 Circle angle difference 43
3.2 Orientation-aware circle expansion 44
3.3 Directed circles overlapping 45
3.4 Space exploration of SEHS and OSEHS methods 46
3.5 Circle driving directions 47
3.6 Search with different heuristics of SEHS and OSEHS methods. 47
3.7 Search with different cost functions. 48
3.8 Search with adapted motion primitives (5329 vertices) 49
3.9 Cross parking scenario. 50
3.10 Parallel parking scenario. 51
3.11 Maneuvering with turn-around scenario. 53

4.1 Cylinder-based space-time exploration 57
4.2 Space-time exploration 58
4.3 Space-time exploration heuristic search 60
4.4 Overtaking scenario 61
4.5 Intersection scenario 63

LIST OF FIGURES

5.1 Hybrid planning system architecture . . . 66
5.2 Combined task planning and motion planning. . . . 67
5.3 Example of world model . . . 69
5.4 Overtaking scenario . . . 74
5.5 Task planning for the overtaking scenario . . . 74
5.6 Round-about scenario . . . 75
5.7 Task planning for round-about scenario. . . . 76
5.8 Roadmap based motion planning in a car park scenario . . . 77
5.9 Behavior-based steering motion primitives . . . 78
5.10 Overtaking with roadmap and driving behavior knowledge . . . 79
5.11 SEHS motion planning engine states diagram . . . 80
5.12 Autonomous charging scenario . . . 82
5.13 Precise parking onto an inductive charging station . . . 83
5.14 Automated parking scenario . . . 85
5.15 Transit and parking in a semi-structured parking area . . . 86
5.16 Collision avoidance during transit . . . 87
5.17 Simple intersection yield scenario . . . 87
5.18 Yield and driving through an intersection . . . 88
5.19 Complex left turn scenario . . . 89
5.20 Turning at an intersection with traffic light, incoming traffic, and pedestrians . . 91

6.1 AutoDrive Library . . . 94

A.1 Control space of a constant curvature model . . . 108
A.2 Control space of a continuous curvature model . . . 109
A.3 Control space of a kinodynamic curvature model . . . 110
A.4 Single track kinematics model . . . 111
A.5 Control space of a single track kinodynamic model. . . . 112

List of Tables

2.1 Results of the local minimum scenario 35
2.2 Results of the narrow passage scenario 36
2.3 Results of the simple navigation scenario 38
2.4 Results of the large labyrinth scenario 39

3.1 Results of the cross parking scenario 51
3.2 Results of the parallel parking scenario 52
3.3 Results of the maneuvering with turn-around scenario 53

4.1 Results of the overtaking scenario 62
4.2 Results of the intersection scenario 64

6.1 Arithmetic Functions 95
6.2 Geometric Operations 96
6.3 Environment Queries 99

Chapter 1

Introduction

Planning is a fundamental intelligent ability, which proposes an arrangement or actions to accomplish a certain mission. It has been studied from various aspects since the very beginning of science and engineering. The modern artificial intelligence research [1] has brought rapid developments and a wide range of applications to planning and problem-solving techniques, especially robot path or motion planning [2].

Generally, a robot motion planning problem is defined with a robot model including geometries and kinematics, a set of environment and task constraints, an initial state, and goal conditions. A planning algorithm solves the problem by looking for a path or motion from the start to a goal, which is described with a sequence of robot states or control commands. In this case, motion planning problems and approaches are highly diverse regarding different robot kinematics and environment models. Unlike the motion planning for a stationary robot-arm with many joints and links [3], which deals with a high dimensional configuration space and a small workspace, the motion planning for a mobile robot concerns about the motion of the entire robot in a relatively large area. A car-like mobile robot subjects to the nonholonomic constraints such as a minimum turning radius [4]. Furthermore, the differential constraints such as bounded velocity, acceleration, and steering speed should be considered as well, especially in kinodynamic motion planning [5]. The specific motion planning problem for nonholonomic vehicles is studied in this thesis with the focus on applications in automotive domain.

In this chapter, the motivation of the motion planning research for autonomous driving is presented at first. Then, a general problem definition of mobile robot motion planning is formulated with the common terminologies and notations used in this thesis. After that, the state of the art motion planning methods are reviewed to provide a big picture of the different approaches and aspects in this research field. Finally, the contribution of this research work is highlighted.

1.1 Motivation

Modern robots can be equipped with advanced sensors, e.g., laser scanners, cameras, radars, ultrasonic sensors, inertial sensors, which provide rich information about the environment and the ego state. The rapid development of the computation hardware enables robots to perceive the information, build knowledge, and make decisions with sophisticated algorithms on board. Furthermore, as the things are getting connected to each other through the latest communication technologies, a robot can access to a huge amount of information as the “Big Data” and unlimited processing power from the “Cloud”. As a result, it is possible to build a robot system

that performs complicated tasks autonomously in dynamic environments.

In contrast to the conventional industrial robots in production automation domain, which repeat similar motions in a rather static environment, the mobile robots in service, household, and logistics domains highly rely on online motion planning, because the environment is dynamic and changing in real-time. For example, a service robot travels among crowds to provide guiding information or serve food and drinks; a vacuum-cleaner robot goes around furnished rooms with randomly deposed items to do cleaning jobs; multiple mobile platforms coordinately transport goods in a warehouse or a factory at the same time. An essential capability of these robots is to move to the right place based on the prior knowledge and posterior information about the environment without causing harm, e.g., colliding with other objects. Between knowing the world and performing the operation, motion planning is the central link of the "Perception-Planning-Action" paradigm to accomplish the mission.

In the wide spectrum of mobile robot applications, autonomous driving receives great attention in the recent years, as it proposes great improvements to safety, convenience, and efficiency of the mobility, and reveals a revolutionary future of automotive, transportation, and way of life [6]. Many research prototypes are built with the most cutting edge technologies. They have demonstrated autonomous driving functionality in field experiments and competitions [7, 8, 9], and created massive impacts in the society. At the same time, the automotive industry brings more and more advanced driver assistance systems (ADAS) to the market, from passive monitoring and warning functions to complete systems with active driving controls [10, 11]. Ambitious roadmaps and timetables are proposed to achieve autonomous driving in 2020s or even earlier. However, in order to reach the ultimate goal of fully automated driving, the following issues should be addressed:

- **Planning in dynamic environment**: An autonomous driving scenario often contains moving obstacles and live signals. In addition, the sensing ability of a vehicle is limited in both range and accuracy, so it only knows its surroundings to a certain space and time horizon with some degree of uncertainties. A planning agent should be able to perform kinodynamic planning with the perceived dynamic environment and online re-planning for the unexpected environment changes.

- **Handling traffic domain knowledge**: A motion planning problem is theoretically solved when a collision-free trajectory is found. However, this only satisfies a necessary condition of autonomous driving. As a part of the transportation system, an autonomous vehicle should follow traffic rules and behave rationally to the other traffic participants. On one hand, additional task constraints should be considered, which make a motion planning problem more difficult; on the other hand, the traffic domain knowledge can be exploited to restraint the scope of the problem and provide guidance to the planning.

- **Interacting with humans**: An autonomous vehicle interacts with human drivers and passengers not only through the human machine interfaces such as steering wheel, pedals, lights, and sounds, but also by the driving behavior itself. Therefore, the vehicle motion should comply with human feelings for the convenience of passengers and other human drivers in the traffic.

- **Software development and verification**: As most of the motion planning functionalities is implemented in software, the development of an autonomous driving system is a software intensive engineering process. A motion planning module must guarantee safety of the vehicle motion, even when no solution is found for the target driving maneuver. The system should be able to always maintain the vehicle in a safe state. The functions including the safety features should be tested and verified for the final product.

This thesis mainly focuses on the first two aspects by developing a novel motion planning framework as *Space Exploration Guided Heuristic Search* for kinodynamic planning in dynamic environment. A task planning approach is introduced to deal with domain knowledge and make decisions in a symbolic space. These two methods are integrated as a *Motion Planning Engine* providing a real-time motion planning service.

1.2 Motion Planning Problem

In general, the state of a robot can be generalized to a minimum set of variables $(q_0, q_1, \ldots, q_i)$ called *General Coordinates* or simply a *Configuration* of the robot, which is denoted by a vector $\vec{q}$. The state space of a robot is called *Configuration Space*, denoted by $\mathscr{C}$. The dimension of a configuration space is the degrees of freedom of the robot. The physical space, where a robot performs its tasks, is called *Workspace*, denoted by $\mathscr{W}$. A robot accepts a set of *Control Inputs* as a vector $\vec{u}$ that defines a transition from one configuration to another: $\vec{u} : \vec{q}_i \rightarrow \vec{q}_j$. A *Motion Primitive* m can be defined with an initial configuration $\vec{q}$ and control inputs $\vec{u}$ as $m = (\vec{q}, \vec{u})$. Let $\mathscr{U}$ be the *Control Space* of all the possible control inputs, a motion planning problem can be formulated as a state space problem $P = (\mathscr{C}, \mathscr{U}, \vec{q}_{\text{start}}, Q_{\text{goal}})$, in which $\vec{q}_{\text{start}} \in \mathscr{C}$ is a start state, and $Q_{\text{goal}} \subseteq \mathscr{C}$ is a set of goal states. The solution is a sequence of control inputs $\{u_1, \ldots, u_n\}$ or primitive motions $\{m_1, \ldots, m_n\}$ that moves the system from $\vec{q}_{\text{start}}$ to one of the target state in Q_{goal}.

The state space problem can be solved as a graph search problem in computer science. If a configuration space is modelled as a graph $G = (V, E)$ by associating each configuration $\vec{q}$ with a vertex v, and control inputs $\vec{u}$ with an edge e. A solution is a path with vertices $\{v_0, v_1, \ldots, v_n\}$ and edges $\{e_1, e_2, \ldots, e_n\}$ connecting the start vertex v_0 and a goal vertex v_n in the graph. Furthermore, edges can have weight as the cost of the motion. Thus, a graph search algorithm can return a path with the minimum total cost as the optimal solution.

Existing graph search techniques can easily solve the problem when a configurations space is discrete and the paths between the configurations are known, i.e., the search is well informed. However, in most mobile robot applications, the configuration space is continuous with infinite number of configurations. The set of valid configurations is usually implicitly defined by a function $f_{\text{validate}}(\vec{q})$, which decides whether a configuration is valid, e.g., no collision occurs. Furthermore, robot kinematics may introduce additional constraints, such as limited turning radius, that makes the interconnection in configuration space complicated. These constraints are classified as holonomic and nonholonomic in classical dynamics research. The former one only concerns the general coordinates as $f_{\text{holonomic}}(\vec{q}) = 0$. The latter one also involves the time derivative of the general coordinates as $f_{\text{nonholonomic}}(\vec{q}, \dot{\vec{q}}) = 0$. For example, a stationary collision-free constraint is holonomic, as it can be verified just with a configuration of the robot. The steering radius of a car-like robot is a nonholonomic constraint, as the robot changes its orientation proportional to its velocity. If a system only has holonomic constraints, it is called holonomic system. The metric of a holonomic configuration space is as simple as the Euclidean space, because the system can move in any direction except for invalid states. For nonholonomic constraints, the time derivative of the general coordinates, i.e., the moving direction, matters. In this case, it is non-trivial to find an optimal motion between two arbitrary configurations.

Therefore, motion planning with nonholonomic constraints is a great challenge, because the graph of the configuration space is unknown. A general approach is constructing a graph from the start state until it reaches the goal condition, as described in pseudo-codes in Algorithm 1. This graph search algorithm is an iterative process. It starts with a single vertex $\vec{q}_{\text{start}}$. In each iteration a vertex $\vec{q}_i$ and a set of control inputs U_i are selected to extend the graph. If the

Algorithm 1: Motion planning with graph search $(\mathscr{C}, \mathscr{U}, \vec{q}_{\text{start}}, Q_{\text{goal}})$

```
(V, E) ← ({q_start}, ∅);
while V ∩ Q_goal = ∅ do
    q_i ← SelectVertex(V);
    U_i ← SelectInputs(q_i, U);
    for each u_j ∈ U_i do
        q_{i,j} ← ExpandVertex(q_i, u_j);
        if IsValid(q_i, u_j) then
            (V, E) ← (V ∪ {q_{i,j}}, E ∪ {u_j});
```

motion $(\vec{q}_i, \vec{u}_j)$ is valid, a new vertex $\vec{q}_{i,j}$ is added to the graph with the edge $\vec{u}_j$ from the parent vertex $\vec{q}_i$. The loop continues until a vertex from the goal set Q_{goal} is reached. The following points are important to design an efficient graph search algorithm in this schema.

- **Vertex selection**: The function SelectVertex(V) selects a vertex from the existing graph as the parent state to generate new vertices in every iteration. It mainly determines the global construct direction of the graph. By selecting the most promising vertex, an algorithm can achieve better performance than the systematic depth-first or breadth-first search methods.
- **Control inputs selection**: The function SelectInputs$(\vec{q}_i, U)$ picks a set of control inputs to be applied to the selected state. As the control space is also continuous with infinite admissible control inputs, it is only possible to verify a certain subset of motion primitives in a search step. It produces the local formation of the graph. A wise choice of the atomic motions can produce more valid vertices that extend the graph.
- **State resolution**: The graph expansion with a new vertices as $V \cup \{\vec{q}_{i,j}\}$ prefers a compare to the existing states to reduce redundancy regarding a certain resolution. It not only reduces memory usage and computation time, but also enables a search algorithm to escape local minima, where many similar states could be created. A trade-off between completeness and efficiency should be made when deciding the state resolution. With a coarse resolution, an algorithm may return a result very fast, but miss some possible solutions. In contrast, a fine resolution may enable a planner to achieve a better result with high cost at a local minimum.

The motion planning algorithms developed in this thesis put effort mainly in these three aspects to improve the performance of the graph-search-based motion planning for nonholonomic vehicles.

1.3 State of the Art

The configuration space of a mobile robot is continuous, therefore the common discrete search methods can only be applied when it is discretized. The general combinatorial planning methods require space decomposition [1, 2, 12]. As a result, these approaches suffer from high preprocess overhead and states explosion in a large configuration space. Furthermore, the geodesic between the states should be further evaluated for a car-like robot due to the nonholonomic constraints. If a car can only move forwards with a limited turning radius, the shortest path between

two states without considering obstacles was defined by Dubins [13] as one of the 6 possible combinations of motion primitives. When reverse driving is also considered, the set of the possible shortest paths increases to 48 as discovered later by Reeds and Shepp [14]. These shortest paths can only be achieved by steering when stopped, or changing moving direction at the end of each primitive motion. When driving velocity, acceleration, and steering speed are involved, there is no close-form solution for the configuration space geodesic anymore, except for some special cases, e.g. the continuous curvature paths in [15].

In this case, various approaches are developed to address the motion planning problem with constraints in continuous space. They are introduced in the following subsections, and the techniques to enhance their performance are also discussed.

1.3.1 Potential Field Methods

Potential field methods do not provide an explicit path solution, but a motion strategy regarding obstacles and goals in the whole configuration space. The behavior of a robot is not deliberately planned but instantly determined based on a potential function. The potential field defines an artificial force that pushes the robot away from obstacles, while pulls it towards the goal. In this case, a potential field method can be regarded as a special version of search methods, which conducts the search procedure by always selecting the most promising vertex as the next one to the current robot state. As a result, the robot behaves as the basic collision avoidance or goal pursuit natural animal motions. Therefore, these methods are also called *Behavior-Based Methods*. For example, Khatib [16] developed an *Artificial Potential Field* to avoid collisions and joint singularities for robot arm motion planning. Arkin [17] introduced *Motor Schemas* for concurrent behaviors of a mobile robot. Brooks argued this approach as "intelligence without representation and reason" in [18, 19], that the complex behavior of a robot is only the reflection of the sophisticated environment.

The major advantage of a potential field method is that if the potential function is provided, it can produce instant robot motion for real-time applications. Therefore, it is widely used in motion planning for collision avoidance [20, 21, 22, 23] and simple automotive applications, e.g. lane following [24] and lane keeping [25].

The drawback of a behavior-based method is that building a potential field in a clustered environment with simple superposition may create many local minima, which hinder a robot to move further towards the goal. A thorough evaluation of the whole configuration space is necessary to design a potential function without any local minimum. For example, a global guidance function was constructed in [26] to connect the local minima. The panel method was introduced in [27] to create a harmonic potential function without local minima. With a space decomposition technique, it is possible to combine local potential functions in sub-regions for the global behavior as presented in [28, 29].

However, such preprocess is time consuming in a high dimensional configuration space. In case of dynamic environments, it is even more exhausting to keep updating the global potential field in real-time. Therefore, the potential field methods are usually applied to local problems, where the global navigation is tackled by other modules. For instance, an *Elastic Bands* [30] employs local potential fields to adapt the robot motion along a global path [31]. In this case, a global reference path is provided, and the robot motion is planned in a small neighborhood.

1.3.2 Random Sampling Methods

Random sampling methods explore the configuration space with random samples, and continuously build a graph regarding a distance metric of the sample points. The *Probabilistic*

Roadmaps (PRM) approach [32] creates a map with random samples in a learning phase. Then, a path can be found with a graph search algorithm in a query phase. The *Rapidly-Exploring Random Trees (RRT)* [33] algorithm grows a tree from an initial state following random samples until it reaches a goal state. According to the general graph search algorithm, these methods select vertices and control inputs in a random manner to construct the search graph.

These algorithms are generic and straightforward to implement, especially practical for problems in a high dimensional configuration space. PRM supports multiple queries after building the map, and the learning process can continue to improve the map resolution and coverage. RRT can deal with kinodynamic problems [34] as a tree is constructed in a forward propagation manner. Another nice property is that a random sampling method naturally provides a probabilistically complete guarantee, i.e., with enough time and memory, the probability of solving the problem converges to 1. However, this guarantee is so weak that if a problem is unsolvable, an random sampling algorithm will run forever. Furthermore, the planning time and result quality depend totally on the individual sampling procedure, which differs greatly among the planning attempts. In addition, a general uniform sampling schema for the whole configuration space is inefficient when planning with constraints such as narrow passages or nonholonomic kinematics.

To address the disadvantages mentioned above, various extensions of random sampling methods are developed, especially with the focus to improve the sampling efficiency. *RRT-Connect* [35] and *RRT-blossom* [36] enhance the configuration space exploration by creating multiple states each time for RRT. A *Path-Directed Subdivision Tree (PDST)* was developed in [37] for motion planning with drift, underactuation, and discrete system changes. Shkolnik et al. [38] evaluated the reachability of a nonholonomic system to direct the samples of RRT. The *LQR-RRT** from Perez et al. [39] applies results from a LQR-controller to guide the sampling with differential constraints. Plaku et al. introduced a *Discrete Search Leading Continuous Exploration* method [40], which exploits the qualitative knowledge obtained during the randomized exploration to selection the most promising branch of RRT to progress. *Kinodynamic Motion Planning by Interior-Exterior Cell Exploration (KPIECE)* [41, 42] is an approach that divides the explored region into interior and exterior cells to push the frontiers of the random exploration further. *EG-RRT* [43] handles the environment guidance knowledge for kinodynamic motion planning with the presence of narrow passages and control uncertainties. *RRT** [44] is an anytime RRT algorithm, which improves the completeness and optimality guarantees of the motion planning result. However, the trade-off, that real-time motion planning and optimal planning result cannot be achieved at the same time, is crucial when applying these methods in automotive applications.

1.3.3 Heuristic Search Methods

The heuristic search [45], or known as *A* Search*, is a best-first search algorithm, which benefits from the heuristics, i.e., the extra knowledge about the problem, to guide the search. In this case, the vertex selection in Algorithm 1 refers to a cost estimation function to choose the cheapest state to go. The total cost f of a vertex is the sum of the actual cost g from the start state to the current state and the estimated cost h from the current state to a goal state. If a heuristic estimation is optimistic, i.e., for any state the h value is no larger than the actual cost to reach the goal, the heuristic search algorithm is able to find the optimal solution with the minimum effort. This condition is called the admissible condition of the heuristic search.

The systematic search procedure and the optimum guarantee of the heuristic search are important for mobile robot applications to achieve a reproducible optimal result. For a holonomic mobile robot, an A* search can be easily performed after discretizing the continuous

configuration space to a finite number of states. Heuristics can be obtained by relaxing some of the constraints, e.g. the shortest path between two states without considering the obstacles or the robot kinematics. Several heuristics can be combined for a better cost estimation function. Furthermore, the incremental search technique can be applied to adapt robot motion in dynamic environments as *Dynamic A* (D*)* [46]. Similar strategies were developed to continuously improve the planning result as *Lifelong Planning A** [47] or *Anytime D** [48, 49].

However, decomposing the configuration space with a uniform grid is not optimal for a nonholonomic mobile robot. Due to the differential constraints, a robot may be not able to directly reach a neighboring grid cell from the current state. Furthermore, a path through the centers of the grid cells could have discretization artifacts regarding the grid resolution. In this case, the *Field D** algorithm [50] extends the D* search with an interpolation on the cell edges in order to produce a smoother path. The *Hybrid A** algorithm [51, 52] accepts hybrid states to apply continuous primitive motions in a discrete state grid for nonholonomic vehicles. However, the trade-off is still open to balance completeness and efficiency in a specific application. A multi-resolution lattice and anytime dynamic A* search algorithm was proposed by Likhachev et al. in [53] to achieve a compromise. The *FAHR* method [54] improves the heuristic function in order to deal with large discrepancies between a general heuristic estimation and the actual calculated cost.

In practical traffic scenarios, the road network already provides a natural heuristic for the global path. Many search methods are developed to generate trajectories regarding the reference path. Werling et al. [55, 56] combined lateral and longitudinal movements in a relative coordinates, Frenet coordinate system, and chose an optimal maneuver according to the cost function. In [57], a spatiotemporal lattice was built along a reference path to determine kinematic feasible motions for driving on the highway. A post optimization process was added to this motion planning framework to improve the quality of the discretized search result [58, 59]. The tenability of this approach was further discussed in [60]. Gu et al. [61] employed an elastic band method to plan the global path, followed by a speed planning that generates kinematic feasible trajectories. These methods work well for lane following, lane changing, and merging maneuvers. However, a reference path is mandatory, and the motion can only be adapted in a small neighborhood of the reference path.

1.3.4 Planning with Space Decomposition

Space decomposition is a powerful technique for robot motion planning. It can divide a large complex problem into simple sub-problems. Some combinatorial methods are directly based on configuration space decomposition, e.g. the visibility graph approach [62]. Potential field methods can be applied in each decomposed area to solve a problem in a "divide and conquer" manner. In [63], a vector field was designed for every grid cell, which drives a robot to an adjacent cell that leads to a goal configuration. However, the evaluation of the whole configuration space is time consuming, and frequent updates are necessary in dynamic scenarios.

Instead of examining the whole configuration space at a time, space decomposition can also be performed in an incremental exploration fashion. Then the space exploration starts from the initial state, and investigates the configuration space in a flood-like manner until it reaches a desired location. Thus, a planning agent collects useful information just in the region of interest, and obtains a path corridor from start to goal with moderate effort. The *Elastic Bands* [30] is a potential field method based on partial space decomposition, which only considers the neighborhood of a reference path. An expansive configuration space concept was introduced in [64], that samples relevant parts of the configuration space to improve the efficiency of PRM.

Motion planning for mobile robots usually deals with a large clustered environment and dif-

ferential constraints. In this case, exploration-based configuration space decomposition could still be too laborious for real-time motion planning. However, as the obstacles can be well presented in the workspace, it is possible to gather valuable knowledge with space decomposition in this low dimensional space. A Voronoi diagram based workspace decomposition was introduced in [65] to accelerate PRM. Brock et al. [66, 67] decomposed workspace with "bubbles" in order to apply potential fields that pull a robot through these "bubbles" to the goal. Rickert et al. proposed *EET* [68, 69], which uses the same technique to balance the exploration and exploitation effort of RRT especially for narrow passage problems. A survey of different types of workspace decomposition for sampling-based methods was presented in [70].

1.3.5 Planning with Dynamic Environments

Motion planning becomes more challenging with dynamic environments, which could be caused by insufficient prior knowledge, imperfect sensing, or moving obstacles. A conventional approach is taking a snapshot of the world, so that all the obstacles are static during planning. With an assumption that planning is much faster than environment changes, a robot can adapt its motion to dynamic environments with online re-planning. In this case, a robot motion can be planned in two steps: a path planning algorithm first finds a geometric path regarding the kinematic constraints and the static obstacles; then, a trajectory generation method calculates a trajectory concerning the system dynamics. A re-planning can be carried out in an incremental manner, e.g., *D** [46], *lifelong planning A** [47], *anytime D** [49]. However, as the planning is done in a static snapshot, the robot may run into an inevitable collision that cannot be escaped with a re-planning on the fly.

In contrast to the ambition of planning everything, it was argued by Brooks [18, 19] that with simple reactive mechanisms, a robot can perform complex behaviors in a sophisticated environments. The methods of behavior-based robotics are efficient to adapt the robot local motion to dynamic environments. "Elastic Roadmap" [30] is an example that reforms the reference path after environment changes. A global dynamic window approach was applied in [71] to follow a global grid-based path with real-time collision avoidance. Another similar approach for local motion adaptation is "Elastic Roadmap" [72, 73], which fulfills mobile manipulation constraints with a feedback to motion planning. The general idea of planning a global path in advance and then fine tuning local motions according to real-time perceptions provides a multi-level solution for motion planning in dynamic environments. For example, PRM and AD* are taken respectively for global and local planning, and combined in an anytime planning framework for dynamic environments [74]. However, these methods can only handle environment changes locally in a rather close time horizon. The global path should be updated when the local adaptation fails.

In order to directly deal with dynamic environments, the time domain should be included to the configuration space. Kinodynamic motion planning [5] considers the first and second order time derivatives of the state variables, i.e., speed and acceleration, to calculate a timestamp for every state. Therefore, it is possible to check collisions in a dynamic environment at a specific time. RRT [34] can solve a kinodynamic planning problem by sampling the control space for random motions to extend the tree. As a result, it can deal with moving obstacles [75]. In [76, 77], dynamic objects were treated as *Velocity Obstacles* in order to easily plan a collision-free motion. This approach was extended to arbitrary shapes for robots and obstacles in [78]. A car-like robot can apply the *Generalized Velocity Obstacles* method for collision avoidance in a dynamic environment [79]. Similar approaches are known as the *Dynamic Window* [80] or the *Forbidden Velocity Map* [81]. A time-bounded lattice was constructed in [82] to enable heuristic search in a dynamic environment. However, most of these approaches must represent

the whole dynamic environment as specific models in advance with extra overhead.

1.3.6 Task Planning

Besides space decomposition which segments a large motion planning problem regarding topology and geometry, task planning divides a complex driving maneuver into sub-tasks in a logical way. Then, these sub-tasks can be solved more efficiently with dedicated motion planning algorithms. Known as symbolic planning or classical planning, it is one of the earliest research themes of artificial intelligence [1]. Task planning performs logical reasoning in a symbolic space and looks for an action strategy or a sequence of tasks to achieve a goal. A formal problem domain definition for task planning can be found as STRIPS [83], PDDL [84], and their successors. According to the definition, a system state is represented with a number of predicates. Actions are defined with preconditions and effects. An action is applicable to a state if all its preconditions are satisfied. A state is changed by an action through its effects. A task planning algorithm can also be modelled as a graph search procedure as Algorithm 1 by associating system states with vertices and actions with edges. In this case, a task planning problem can be solved by forward chaining, backward chaining, or partial-order planning [1].

Symbolic planning can solve problems that are discrete, deterministic, and fully-observed. However, in robot motion planning domain, the robot configuration and the environment state are both continuous and may be partially observed. In addition, robot motion is also continuous and lasts for a certain time. Therefore, symbolic planning cannot be directly applied, but serves as a high-level task planner in a hierarchical planning architecture. A task planner works with a set of abstractions of the world and plans symbolic tasks for a robot based on the domain-specific semantic knowledge. The goal is to divide a large problem into small sub-problems and limit the context for an efficient motion planning [85]. Plaku and Hager [86] combined task planning and kinodynamic motion planning so that the tasks can be solved with a forward search of motions with differential constraints. A belief-space hierarchical planner was introduced by Kaelbling and Lozano-Pérez [87], which searches backward from the goal state in a probabilistic state space that models perception and manipulation uncertainties.

In autonomous driving scenarios, the environment contains a lot of traffic domain specific information such as traffic signals and rules. Therefore, it is possible to apply task planning to decompose a long route from the high-level navigation module into small driving tasks for the low-level motion planning. Especially for the traffic rules, which are usually abstract and difficult to model in the continuous configuration space for motion planning, while a task planner can handle them easily in a symbolic state space. Ontology-based approaches were developed to describe traffic situations, assess the driving behaviors, and perform high-level reasoning and planning [88, 89, 90]. The results can be used to manage low-level motion planning in an ADAS system, especially for intersections [91].

1.3.7 Advanced Driver Assistant Systems

The automotive industry approaches autonomous driving from another direction. Different from the complete top-down autonomous systems developed in scientific research, advanced driver assistant systems are built in a bottom-up manner to achieve partially or highly automated driving for commercial products. These functions mainly deal with specific motion planning problems such as collision avoidance, parking, and lane following. Therefore, the domain knowledge of the driving scenario can greatly simplify the motion planning strategy.

Take automated parking for example, an iterative method was proposed in [92] that steers a vehicle with sinusoidal functions into a parking lot. Collision avoidance was implemented as

a direct reaction to the the ultrasonic distance sensor output. The parking problem can also be solved in a two-step approach [93, 94] as mentioned above. First, a collision-free geometric path is planned without considering the vehicle kinematic, and then it is converted to a feasible trajectory. Predefined maneuvering schemas can be applied for certain type of parking problem as in [95, 96]. Most of these methods are specially designed for parking scenarios. As a result, it cannot be easily integrated into a general motion planning framework for autonomous driving, but triggered by a human driver.

The problem becomes more sophisticated when combining multiple ADAS functions as building blocks for a highly or fully automated driving system. A high-level administration module should decide which component to be activated and select the right configuration for the sub-module. A common approach is to integrate all the conditions, rules, and functions into a state machine based system architecture [97, 98]. In this case, the system complexity, in general, grows exponentially with respect to the number of states or conditions; test, verification, and modification of such systems are tedious and error prone. Kress-Gazit and Pappas [99] propose an approach to synthesize hybrid controllers from the high-level descriptions of urban driving behaviors, which guarantees correctness and facilitates the development of state machines. An alternative approach are behavior-based solutions, which select the current control mode by likeliness or as a fusion of a set of predefined actions [100, 101]. However, the result behavior may be a local optimum that does not comply with the goal in longer-term.

1.4 Contributions

This thesis proposes a hierarchical motion planning framework for autonomous driving with knowledge oriented methods. There are many different kinds of information involved in autonomous driving. Some is concrete and continuous, e.g., the geometry of a road, the motion of an object, the kinematics of the ego vehicle; some is abstract and symbolic, for example traffic rules and traffic signals. The decisions or results from the intelligent agents are also different. There are discrete decisions such as the selection of a route or a driving maneuver, as well as continuous ones such as a path or trajectory for a concrete motion. Some of them can serve as intermediate knowledge to facilitate solving a complex problem with an effective method. The *Space Exploration Guided Heuristic Search* (SEHS) approach developed in this thesis provides a generic framework that can exploit various types of knowledge in autonomous driving scenarios. First of all, SEHS is a general motion planning algorithm that plans motion for car-like robots regarding environment and kinematic constraints. At the same time, it can be tailored for specific driving scenarios regarding the high level semantic information and domain knowledge achieved by driving task planning. These two kinds of planning agents are integrated in a *Motion Planning Engine* to provide real-time motion planning service in an autonomous driving system. Furthermore, a C++ software library called *AutoDrive* is developed to implement the planning framework.

1.4.1 Space Exploration Guided Heuristic Search Algorithms

The *Space Exploration Guided Heuristic Search* method [102] plans motions for nonholonomic vehicles in two steps. First, it performs an exploration-based space decomposition to investigate the topology and dimension of the workspace. The workspace knowledge is used to construct a heuristic that balances path length and safety distance. Then, a heuristic search algorithm performs a graph search following the workspace corridor to find the solution. Thus, the environment constraints and kinematic constraints are evaluated in a progressive manner, which

simplifies the planning in each phase and achieve a great boost to the total planning performance. Furthermore, the space decomposition enables the search in continuous configuration spaces to be grid-free with adaptive state resolution and search step size. As a result, it obtains a good balance between efficiency and completeness.

A SEHS algorithm is a generic motion planning method since the two subroutines can be both generalized as the graph search in Algorithm 1, which provides the flexibility to customize SEHS for many practical problems in autonomous driving. For example, if moving obstacles are involved in a driving scenario, the vehicle speed and obstacle motion should be considered. In this case, the space exploration can be carried out in both space and time domains to enable kinodynamic planning as *Space Time Exploration Guided Heuristic Search* (STEHS) [103]. When complex maneuvering is required for driving through narrow passages or into small parking-lots, the vehicle orientation is evaluated in addition during the space exploration phase to exact the driving direction knowledge with *Orientation-Aware Space Exploration Guided Heuristic Search* (OSEHS) [104]. Furthermore, most incremental and anytime planning techniques for heuristic search can be applied to the exploration and search procedures to achieve rapid re-planning and continuous improvement of the solution quality.

1.4.2 Real-Time Planning Engine Combining Task Planning and Motion Planning

A motion planning algorithm alone is not sufficient for a autonomous driving system. The motion planning is a life-long process during the whole driving course, which requires the system to continuously plan, verify and re-plan the driving maneuver. Furthermore, a task planning method [105] is introduced to process the high-level information such as road network, traffic signals, and traffic rules with symbolic planning techniques. Thus, a specific driving maneuver is selected to generate a certain planning problem for the motion planner. A *Motion Planning Engine* (MPE) [106] is developed as the intelligent entity that performs these two planning procedures and provides interfaces to different types of perception inputs and domain knowledge. It verifies the planned motion online with the real-time environment, triggers a re-planning when necessary, and reserves a safe motion in case a goal is beyond reach.

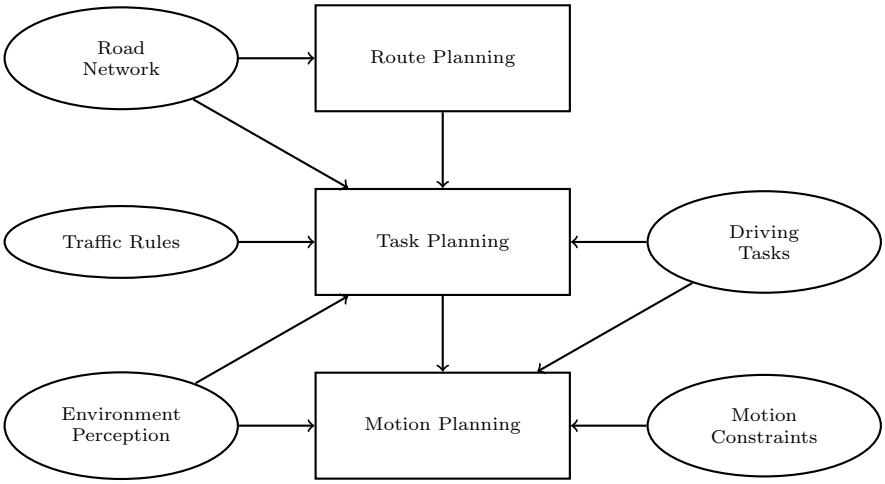


Figure 1.1: Motion Planning Engine system architecture

A general MPE system architecture is proposed in Fig. 1.1. Different types of information are

processed in different layers. The planning results are refined gradually through the layers until the final motion trajectory is generated. First, the route planning agent suggests a roadmap from the current location to the goal based on the road network. Then, a sequence of driving tasks is determined by the task planning agent to follow the route regarding the traffic signals and rules. Finally, each driving task is transformed to a motion planning problem for a dedicated planning algorithm, which produces a detailed trajectory for real-time execution. Most of the ADAS functions, such as automatic parking, traffic-jam assistant, and the more complex ones such as automatic valet parking or intersection assistant can be realized by the MPE. Therefore, it is possible to integrate these functions in a unified planning framework to build a fully autonomous driving system.

1.4.3 AutoDrive Library

In order to evaluate and demonstrate the algorithms in this thesis, a C++ library called *AutoDrive* for "autonomous driving", is developed. Comparing to other robotic motion planning libraries such as the Motion Planning Kit (MPK) [107], the Motion Strategy Library (MSL) [108], and the Open Motion Planning Library (OMPL) [109, 110], the *AutoDrive* library is especially designed for mobile robot motion planning. It contains the basic mathematical utilities, environment representations, vehicle kinematic and dynamic models, and various sampling-based and search-based planning methods. The motion planning progress and results can be visualized in a graphical simulation program with different scenarios configured in XML files. The *AutoDrive* library is developed with just a few external dependencies as only the STL [111, 112] and boost [113, 114] libraries for the core functionalities. Thus, it can be easily applied in many development environments for education, research, and autonomous driving prototyping. The *AutoDrive* library is compatible with both GCC in Linux and MSVC in Windows.

1.5 Outline

The remaining chapters of the thesis are organized as follows: Chapter 2 presents the general concept of the *Space Exploration Guided Heuristic Search* algorithm. Chapter 3 adds the orientation knowledge to the SEHS approach for parking and maneuvering. Chapter 4 introduces the space-time extension of the SEHS method for dynamic environments. Chapter 5 proposes the system architecture of *Motion Planning Engine* with task planning for high-level logical decisions and SEHS for low-level motion planning. Several example applications are also demonstrated in this chapter. Chapter 6 provides an overview of the *AutoDrive* library. The whole work is summarized in Chapter 7.

Chapter 2

Space Exploration Guided Heuristic Search

This chapter presents the general *Space Exploration Guided Heuristic Search* (SEHS) method. First, some basic ideas are discussed regarding the related work. Then, the SEHS framework is introduced in two sub-procedures: *Space Exploration* and *Heuristic Search*, both with problem statements and algorithm details. Finally, SEHS is compared to some other motion planning methods in several example scenarios to evaluate the performance of this approach.

2.1 General Idea

The motion planning for mobile robots concerns a relative low dimensional configuration space, in general, a three-dimensional space for position and orientation. In this case, a search-based planning method as Algorithm 1 is applicable. However, the motion of a car-like robot is subject to the nonholonomic kinematic constraints. The system is underactuated, i.e., it accepts fewer control inputs than the degrees of freedom. Therefore, the robot is only local controllable, but not small time controllable. The geodesic of configuration space is much more complex than that of a holonomic robot. As a result, it is difficult to find an edge between two arbitrary vertices while building a graph. The subsequent vertices are usually generated with numerical forward simulations of the vehicle dynamics. In this case, the vertices expansion is compute-intensive, not to mention the tedious collision checks. Since there are numberless states in the continuous configuration space, it is impossible to perform an exhaustive search.

The heuristic search provides the technique to improve the search efficiency by investing the effort in the most promising direction regarding the extra knowledge as heuristics. The two-dimensional workspace is a subspace of the configuration space, where a robot moves as a single entity. The environment constraints can be well represented in the workspace. In this case, an investigation of the workspace can bring valuable information to build effective search heuristics. As the workspace of a mobile robot is usually large or even unlimited, a thorough analysis of the whole workspace is rather time consuming and unnecessary for a single query problem. In this case, exploration-based space decomposition is a better choice as it only deals with the area of interest.

Following these ideas, a novel motion planning method is developed combining space exploration and heuristic search just as it is called: *Space Exploration Guided Heuristic Search*. In this approach, the space exploration first relaxes the nonholonomic constraints of the robot

kinematics in order to extract the free space knowledge as a path corridor in workspace; after that, the heuristic search looks for a sequence of control inputs or primitive motions that drive the robot following the workspace passage to the goal. The both subroutines can be abstracted as a general graph search algorithm as Algorithm 1, of which two kinds of heuristics are applied successively: a primal heuristic guiding the space exploration in the workspace, and an advanced heuristic leading the search-based motion planning in the configuration space. The second one is obtained from the result of the first planning procedure. Details are explained in the following sections.

2.2 Space Exploration

The space exploration is the first planning stage of the SEHS approach. The purpose of this procedure is to extract the dimension and topology information about the free space based on the prior knowledge and post perception of the environment.

As a mobile robot, especially an autonomous vehicle, usually moves on the ground surface, a two-dimensional workspace is sufficient in this case to model the environment constraints. The three-dimensional physical space is projected to a two-dimensional ground surface with obstacles as two-dimensional shapes. A space exploration algorithm studies this two-dimensional environment model for a path corridor, which can be decomposed in many geometric shapes.

There are many approaches to space decomposition, such as polygon triangulation [115], cylindrical decomposition [116], etc. Roadmaps can be created based on the space decomposition results for path planning, e.g. the visibility map for PRM [118, 119, 120]. Voronoi diagram [117] segments a space region regarding the distance to several reference objects, which is useful to build a roadmap as the street network [52]. Due to the enormous time and resource costs of the space decompose for the entire workspace of a mobile robot, it is usually performed offline with the prior environment knowledge. Thus, a roadmap is built for the multiple queries later. However, a single query is only interested in a certain region of the large workspace. In this case, an exploration-based space decomposition algorithm is more suitable. Instead of a systematic decomposition of the entire workspace, it only investigates the area connecting start and goal locations. As a result, the problem complexity is greatly reduced, so that it is possible to update the space knowledge online.

Furthermore, an autonomous driving application handles real-time sensor inputs in different forms with different update rates. Instead of combining them in a unified environment model, the space exploration procedure requires only several types of queries such as whether a position is collision-free, or what is the distance from a point to the nearest obstacle. These queries can be flexibly implemented based on different environment models depending on the perception inputs, e.g., an object list, an occupancy grid. Therefore, the space exploration is rather generic without strong requirements for environment representations.

2.2.1 Problem Definition

In order to formulate a problem definition for the space exploration, the workspace of a mobile robot is first defined in Definition 1.

Definition 1. *Workspace $\mathscr{W}$: The workspace of a mobile robot is a bounded two dimensional Cartesian space $\mathbb{R}^2$, where the robot motion is performed.*

An obstacle can be modelled as a subset of the workspace $O \subset \mathscr{W}$. A distance function $d_{\text{point}}(p, q)$ returns the distance between two points in the workspace (2.1). An obstacle distance function $d_{\text{obstacle}}(p, O)$ is defined as (2.2) for the distance to the nearest point of an

obstacle. A clearance distance $d_{\text{clearance}}(p)$ is the distance to the nearest obstacle (2.3). The clearance distance is 0, if the point collides with an obstacle.

$$d_{\text{point}}(p, q) = \|p - q\| \tag{2.1}$$
$$d_{\text{obstacle}}(p, O) = \min_{q \in O} d_{\text{point}}(p, q) \tag{2.2}$$
$$d_{\text{clearance}}(p) = \min_{O_i \subset \mathcal{W}} d_{\text{obstacle}}(p, O_i) \tag{2.3}$$

Regarding the clearance distance, a subset of the workspace is defined as free workspace in Definition 2. It is obvious that the free workspace is an upper bound of the projected valid configuration space, which is defined by collision checks and kinematic constraints. It is much convenient to inspect the free workspace than the valid configuration space, as only the point distance in a low dimensional subspace is relevant.

Definition 2. *Free Workspace $\mathcal{W}_{\text{free}}$: The free workspace is a subset of the workspace, which contains all the collision-free points.*

$$\mathcal{W}_{\text{free}} = \{p | p \in \mathcal{W}, d_{\text{clearance}}(p) > 0\} \tag{2.4}$$

The general space exploration problem in SEHS can be defined as Problem 1. A workspace $\mathcal{W}$ is given with a clearance distance function $d_{\text{clearance}}$, a start point p_{start}, and a goal point p_{goal}. The exploration result is a sequence of geometric shapes $\{S_0, S_1, \ldots, S_n\}$ in free workspace, which compose a path corridor from the start point to the goal point.

Problem 1. *Space Exploration*
Given: *$\mathcal{W}$, $d_{\text{clearance}}(p)$, $p_{\text{start}} \in \mathcal{W}_{\text{free}}$, and $p_{\text{goal}} \in \mathcal{W}_{\text{free}}$*
To Find: *$\{S_0, S_1, \ldots, S_n\}$ with $S_i \subset \mathcal{W}_{\text{free}}, i = 0, 1, \ldots, n$; $S_j \cap S_{j+1} \neq \emptyset, j = 0, 1, \ldots, n-1$; $p_{\text{start}} \in S_0$, and $p_{\text{goal}} \in S_n$*

2.2.2 Circle-Based Space Exploration Algorithm

The space exploration problem is solved by a circle-based algorithm in SEHS, which investigates the free workspace with circles for the topology and dimension knowledge. In a two-dimensional space a point with a certain clearance distance can be simply represented with a circle. In addition, it is convenient to acquire the geometric properties such as overlapping and distance with circles.

In general, a number of waypoints are generated as the vertices of a graph search algorithm in Algorithm 1. These points are treated as circle centers, while the circle radius is calculated by the clearance function. An edge starts from a waypoint with a distance regarding the circle size to the next one. Furthermore, the search algorithm follows a heuristic of the Euclidean distance to the goal point. Thus, a graph search algorithm constructs a tree of free space circles from the start position towards the goal in the workspace. The space topology information is contained in the circle constellation; while the space dimension knowledge is provided as the circle size. The result of the space exploration algorithm is a sequence of circles as a path corridor from the start point to the goal point in workspace. Each circle in the path overlaps with the adjacent ones, so that the free workspace is connected for a local controllable robot to travel. The space exploration algorithm is outlined with pseudo-code in Algorithm 2.

The graph consists of a set of vertices V and edges E. A vertex v is a tuple (p, r, g, h). p is a point in the workspace, i.e., the center point of a circle. r is the clearance distance, i.e., the

Algorithm 2: Space Exploration $(p_{\text{start}}, p_{\text{goal}})$

```
1  v_start ← (p_start, d_clearance(p_start), 0, d_heuristic(p_start, p_goal));
2  f_goal ← ∞;
3  V_closed ← ∅;
4  V_open ← {v_start};
5  E ← ∅;
6  while V_open ≠ ∅ do
7      Sort(V_open);
8      v_i ← PopTop(V_open);
9      if f_goal < f[v_i] then
10         break;
11     else if Exist(v_i, V_closed) then
12         continue;
13     else
14         (V_i, E_i) ← Expand(v_i);
15         V_open ← V_open ∪ V_i;
16         E ← E ∪ E_i;
17         V_closed ← V_closed ∪ {v_i};
18         if Overlap(v_i, p_goal) then
19             f_goal = min(f[v_i], f_goal);
20 if f_goal < ∞ then
21     return success;
22 else
23     return failure;
```

radius of the circle. g and h are the actual cost and heuristic cost respectively of a common heuristic search method. The sum of them, $f = g + h$, is the estimated total cost to reach the goal through the current vertex. The heuristic cost is obtained with a heuristic distance function $d_{\text{heuristic}}(p, p_{\text{goal}})$, which estimates the rest cost from the current vertex to the goal. The details of the heuristic function are explained in subsection 2.2.2.1. An edge e is a directed connection from a parent vertex to a child vertex. The weight of an edge is the distance between the two points. The accumulated weight from the start point is the actual cost to reach the current vertex.

The algorithm begins with a vertex at the start position. The graph is constructed by expanding vertices iteratively. The vertices are maintained in two sets: a closed set V_{closed} for the already evaluated vertices, an open set V_{open} for the fresh ones from expansion. The open set is initialized with the start vertex. The closed set is empty at the beginning. In each iteration, if the open set is not empty, the open vertices are sorted after the f-values by the function Sort(V_{open}). The function PopTop(V_{open}) picks the vertex with the minimum total cost from the open set as the parent for vertices expansion. First, the chosen vertex is compared with the closed vertices by the function Exist(v_i, V_{closed}) in order to reduce redundancy. Then, new vertices and edges are created from the parent vertex by the function Expand(v_i). The details of the circle expansion are explained in subsection 2.2.2.2. The child vertices are added to the open set, while the parent one is moved from the open set to the closed set.

A variable f_{goal} holds the best cost to reach the goal achieved so far. It is initialized with

an infinity value. In each iteration, if the parent circle overlaps with the goal point, f_{goal} is updated if a better path is found. If f_{goal} is smaller than the f-value of the parent vertex, an optimal solution has been found. Because according to the heuristic search principle, if the heuristic is admissible, all the other vertices in the open set will take larger cost than the existing solution to reach the goal. In this case, the space exploration is success and a path corridor is constructed with backtracking the edges. The space exploration fails when the goal is unreached and no more vertices can be expanded, i.e., the open set is empty.

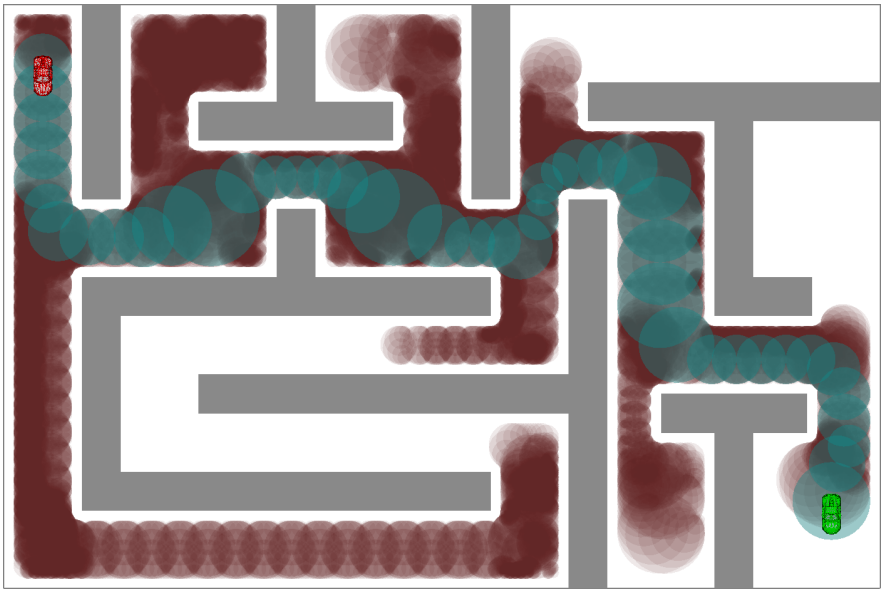


Figure 2.1: Space exploration in a labyrinth

Fig. 2.1 shows an example of the space exploration in a labyrinth environment. The red vehicle frame is the start position, and the green one is the goal position. The obstacles and the boundaries of the labyrinth are colored in grey. The exploration algorithm generates vertices as the brown circles. The final result is the path corridor of the cyan circles. In each iteration, 32 circles are created with a 1.0 m margin for the clearance distance. The circle radius are bounded between 0.5 m and 5.0 m.

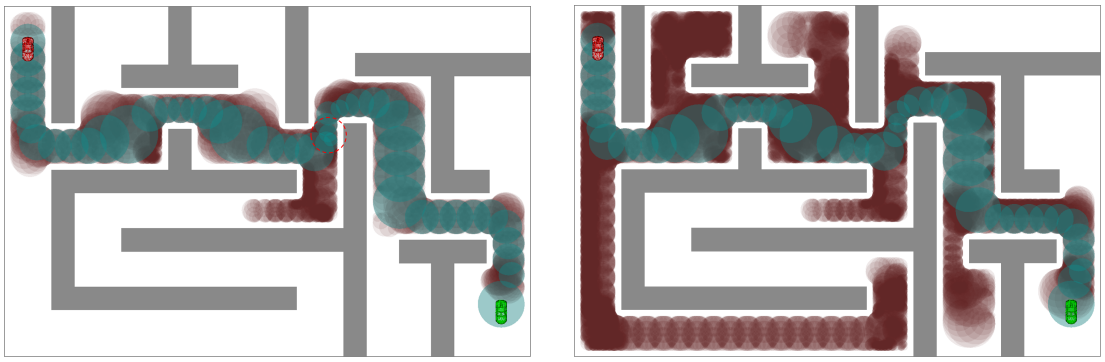
2.2.2.1 Euclidean Distance Heuristic Estimation

The heuristic function $d_{\text{heuristic}}(p, p_{\text{goal}})$ gives a cost estimation from a point to the goal. Together with the actual cost from the start vertex to the current vertex, a total cost is calculated in order to select the most promising vertex from the open set to extend the search graph. A heuristic estimation usually solves an easier problem than the original search problem by relaxing some of the constraints. In this case, if all the obstacles are ignored, the travel cost is simply the Euclidean distance between two points. This heuristic is admissible because it is the shortest path length between two points in the workspace. Therefore, the space exploration algorithm returns the optimal solution if it exists. Furthermore, it is obvious that this heuristic distance holds the triangle inequality (2.5) as the monotonic condition. As a result, the open set and close set technique can be applied to enhance the performance of the heuristic search

by reducing redundant vertices expansion.

$$d_{\text{heuristic}}(p_2, p_{\text{goal}}) \leq d_{\text{heuristic}}(p_1, p_{\text{goal}}) + d_{\text{point}}(p_1, p_2) \tag{2.5}$$

When two vertices have the same total cost value, a tiebreaker prefers the vertex with a larger actual cost. Because the heuristic cost is an optimistic estimation, the final cost is larger than or equal to this value. Therefore, the vertex with a larger actual cost is more likely to achieve a better final result. When the two cost components are also the same, the vertex with a larger clearance distance is chosen, as it explores a larger unknown area. In case all these parameters are identical, a vertex is selected in random.



(a) Space exploration with greedy search

(b) Space exploration with heuristic search

Figure 2.2: Comparing space explorations with greedy search and heuristic search.

The main advantage of heuristic space exploration is that it provides an optimal solution guarantee. Fig. 2.2 compares the results from greedy search and heuristic search. The greedy search algorithm simply expands the vertex that closest to the goal. It is obvious that the greedy search takes fewer vertices to find a solution. However, the result is not perfect at some places (red dashed circle), where several redundant circles are generated. As the greedy method does not care about the effort expended in the past, the closest circle is selected even if it makes little contribution to the exploration progress, especially around a local minimum. In contrast, the heuristic search follows the total cost estimation and evaluates all the possibilities for the optimal solution. In this example, it provides a path 1.03 m shorter than the greedy one.

2.2.2.2 Circle-Based Vertex Expansion

The Expand(v_i) subroutine expands new vertices from the selected parent vertex. Algorithm 3 shows the details of the circle-based vertex expansion. It takes a parent vertex v_{parent} and a bound of circle radius $[r_{\min}, r_{\max}]$ as inputs. First, a set of sample points P is generated relative to the parent circle as candidates for the child points. The radius of a child circle is calculated by the clearance distance function. The radius should not exceed the upper bound $r_{\max}$ to achieve certain resolution of the space decomposition. If the radius is less than the lower bound $r_{\min}$, the circle is neglected to avoid creating tiny circles. After a child circle is created, a g-value is calculated as the sum of the parent's g-value and the distance between the center points. The h-value is obtained by the heuristic estimation.

The bound $[r_{\min}, r_{\max}]$ restricts the circle radius in a range, so that neither too large nor too small circles are created. If a circle is too large, it may result in a suboptimal path corridor as the center point deviates far from the shortest path. If a circle is too small, it contributes

Algorithm 3: Circle Expansion $(v_{\text{parent}}, r_{\min}, r_{\max})$

```
V_children ← ∅;
E_children ← ∅;
P ← Sample(v_parent);
foreach p_i ∈ P do
    r_i = min(d_clearance(p_i), r_max);
    if r_i < r_min then
        continue;
    else
        v_i ← (p_i, r_i, g[v_parent] + ‖p_i − p[v_parent]‖, d_heuristic(p_i, p_goal));
        V_children ← V_children ∪ {v_i};
        E_children ← E_children ∪ {(v_parent, v_i)};
return (V_children, E_children);
```

little to the space exploration especially at a local minimum. These parameters are decided regarding the resolution of the space exploration, size of the mobile robot, safety margin of the motion, and further requirements of the motion planning problem.

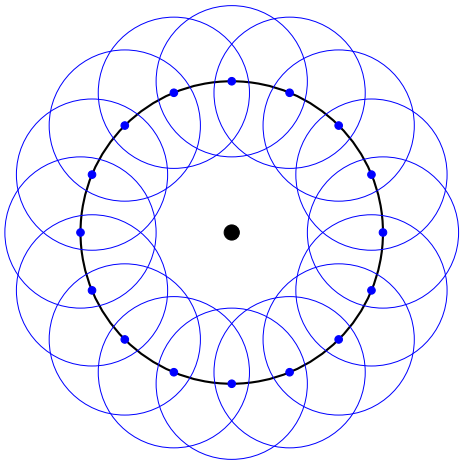
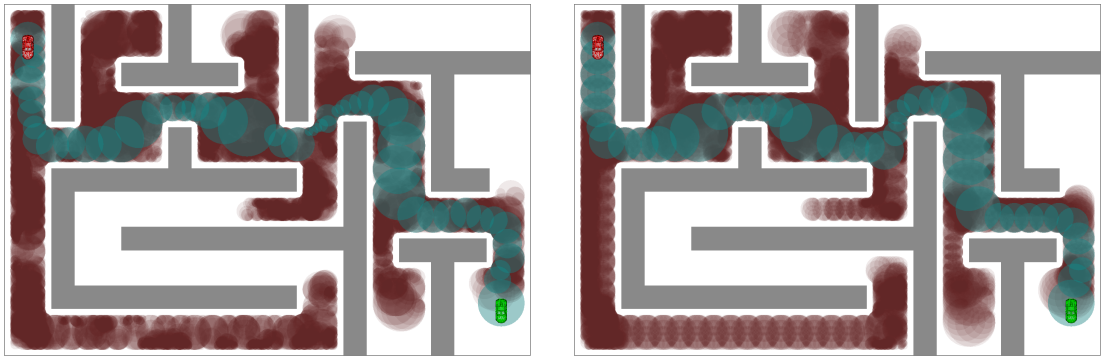


Figure 2.3: Circle expansion with uniform interpolation

A sampling function Sample(v_{parent}) decides where to create the child circles. Different strategies can be applied to create the sample points, e.g., random sampling the parent circle, or uniformly interpolating the circle. An interpolation method is adopted for more systematic and reproducible results. In addition, in order to maximize the exploration area and guarantee overlapping between the circles, the parent circle is equally interpolated for n points on the border. Fig. 2.3 shows the circle-based expansion with a parent circle in black and child circles in blue. The center points are the blue dots uniformly distributed on the parent circle.

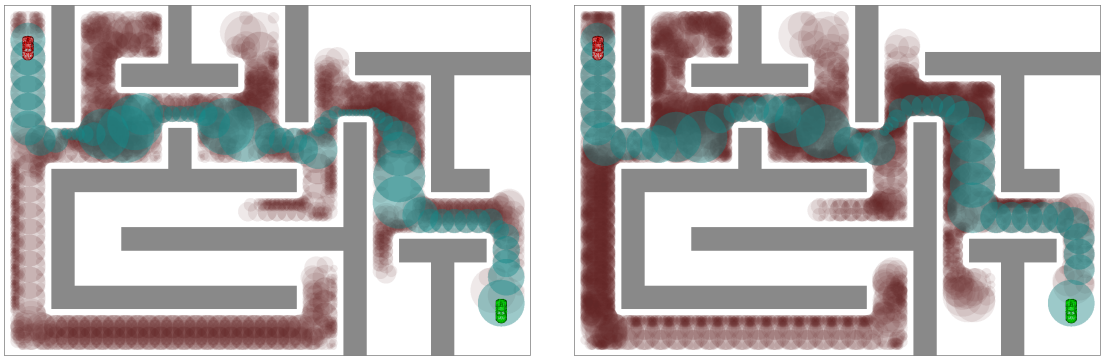
Fig. 2.4 compares the random expansion schema with the systematic interpolation one. The former one generates the same number of circles with center points randomly chosen on the parent circle. In general, the two methods explore similar areas in the workspace. However, in the local scale, the circles from the random approach diverse greatly in position and size, which



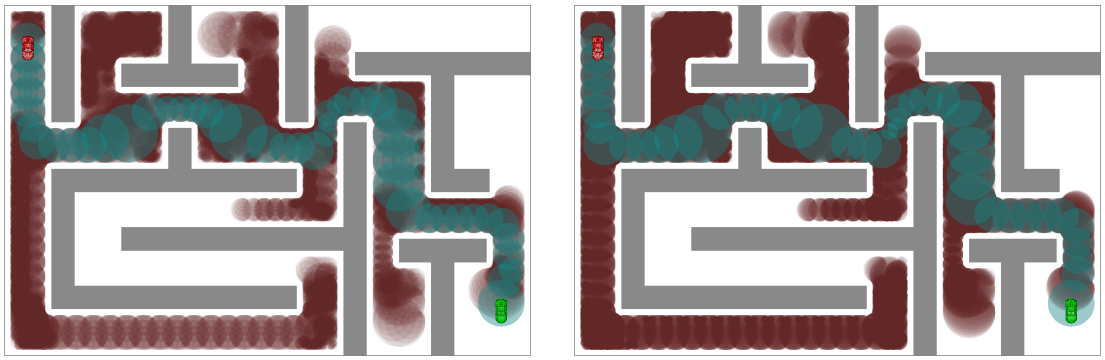
(a) Space exploration with random circle expansion (b) Space exploration with uniform circle expansion

Figure 2.4: Comparing random and uniform circle expansions.

produce different qualities of results in different trials. In contrast, the performance of uniform interpolation is stable.



(a) Circle expansion with 8 circles each step (b) Circle expansion with 16 circles each step



(c) Circle expansion with 32 circles each step (d) Circle expansion with 64 circles each step

Figure 2.5: Comparing number of vertices in each expansion step.

Fig. 2.5 compares the number of vertices created in each expansion step. The explored areas

are similar in all the four examples. It is clear that the more circles expanded in each step, the more vertices are generated in total to solve the problem. However, if the number is too small, the resolution is not fine enough to conduct explorations in all directions. A jitter effect can be observed in Fig.2.5(a), where new circles are only created in every 45° interval on the parent one. This effect is reduced by increasing the number of child circles. With 32 circles it is already hard to notice the artifact in the final result. A further increment to 64 circles does not bring obvious improvement, but large overhead in circle expansion. A rule of thumb is to determine the number based on the bound for the circle size as $\lceil \pi \times r_{\max}/r_{\min} \rceil$. Thus, if the parent circle has radius $r_{\max}$ and all the child circles have radius $r_{\min}$, the child circles can still cover the entire border of the parent circle. In our case, 32 is a good value for the radius bound $[0.5, 5.0]$.

Another important fact from the examples is that the global exploration behavior is decided by the heuristic estimation. The explored areas are similar with different sampling schemas and sampling numbers. As the heuristic search decides the progress direction according to the cost. The circle expansion only determines the local progress of the space exploration, such as detail positions and sizes of the circles. However, it plays an important roll especially around the local minimums.

2.2.2.3 Redundant Vertices with Circle Overlapping

By applying a closed set and an open set in the space exploration, it is possible to distinguish the interior and the frontier of the explored area. If a vertex is in the already explored area, it can be omitted to save the exploration effort. Thus, a redundant vertex is identified by evaluating the overlaps between circles. The function $\text{Exist}(v_i, V_{\text{closed}})$ in Algorithm 2 compares the circle of a vertex with the ones in the closed set to resolve this situation. If the center point is inside a circle from the closed set, except for its parent, the vertex is redundant, because the explored area by a redundant circle can be totally covered by the descendant circles from the existing one in the closed set.

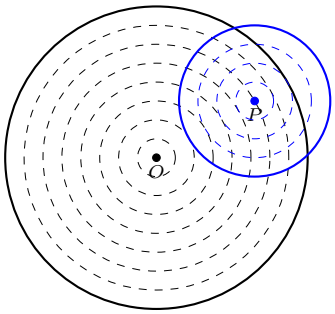


Figure 2.6: Circle overlapping

The redundancy condition can be proved with the wave theory. If the space exploration is modelled as a wave initialized at the center point and propagates in the free space. When the wavefront meets an obstacle, a free space circle is found. At the same time, each point on the wavefront is also a source for a new wave. Thus, the waves keep spreading in the free space. Fig. 2.6 shows a case that a point P is inside a free space circle centered at O. The dashed circles simulate the propagation of the wavefront. It is obvious that point P can be treated as a wave source when the wavefront from O comes. As a result, the free space circle at P is a subset of the explored area by the wave initiating from point O.

If a heuristic search problem has limited number of vertices, it can be solved without the open set and closed set mechanism. However, the space exploration problem is continuous with infinite number of vertices. Therefore, the redundancy check and the monotonic condition are important to enable the open set and closed set tactic to efficiently solve the problem.

2.2.3 Circle-Path Optimization

The result of the circle-based space exploration is a path corridor consists of overlapping circles, which is called *Circle-Path* in this thesis. It provides a compromise between travel distance and obstacle clearance. However, the original result from the vertices backtracking may still have improvement potentials due to the jitter effect of the circle expansion. Therefore, an iterative optimization procedure is designed to improve the quality of the circle-path.

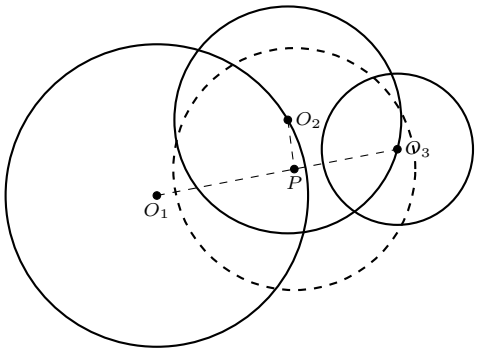


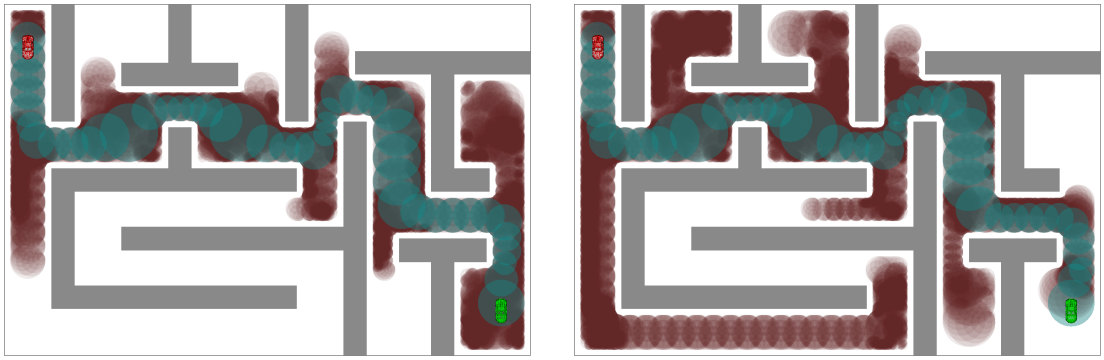
Figure 2.7: Circle-path optimization

Fig. 2.7 shows an example of the circle-path optimization. The circles centered at O_1, O_2 and O_3 are the initial circle-path with radius r_1, r_2 and r_3 respectively. The circle at O_2 can be optimized for a shorter path. In order to do so, a new center point P is calculated on the line segment O_1O_3 with a interpolation ratio $|O_1P| : |PO_3| = r_1 : r_2$. The dashed circle at point P is the optimized candidate for circle O_2. If the radius of the new circle P is no less than r_2, the dashed circle is accepted to replace circle O_2. Thus, a circle is optimized only if the new circle shortens the path without reducing the clearance. The interpolation ensures the overlapping between the new circle and its neighbors.

The optimization process is performed iteratively to each circle along the path. A flag indicates whether an optimization is possible for a circle. It is set true for every circle at the beginning. If no better solution is found, the according flag is cleared. If a circle is updated, the flags of the circle and its neighbors are set. The whole procedure repeats until no further optimization is possible.

2.2.4 Multi-Directional Exploration

The circle-based space exploration can be parallelized to boost the performance. A common technique is the bidirectional exploration, which constructs two graphs simultaneously from the start and goal positions towards each other. The exploration terminates when the two graphs meet. This approach can be extended to multi-source space exploration with several initial points other than the start and goal points. These points can be selected for specific scenarios. For example, points can be chosen inside narrow passages or areas of interests. In this case, the space exploration can be greatly accelerated in a large complex environment.



(a) Bidirectional space exploration (b) Single-directional space exploration

Figure 2.8: Comparing bidirectional and single directional space explorations.

Fig. 2.8 compares the results of bidirectional exploration and single directional exploration. In Fig. 2.8(a), two exploration processes are carried out by turns initiating from the start and goal positions respectively. A solution is found with smaller area explored and fewer vertices than the single thread exploration in Fig. 2.8(b).

A heuristic search method assures an optimal solution by examining all the vertices that may generate better result regarding the heuristic estimations. As an admissible heuristic is with optimistic assumptions, there is a discrepancy between the estimated cost and the actual cost. The larger the difference, the more vertices should be evaluated. The single directional space exploration expands far more circles, because of the great discrepancy between the Euclidean distance and the actual path length in the labyrinth. While each sub-procedure of the bidirectional exploration only needs to proceed to the middle of the labyrinth, the according estimation deviation is much smaller. As a result, it converges much faster.

2.3 Heuristic Search

Heuristic search is the second phase of the SEHS motion planning framework. The heuristic is created based on the space exploration result. The search algorithm forward propagates robot states with primitive motions concerning the nonholonomic constraints. The collisions are verified during the search procedure.

The heuristic search is performed in the configuration space of a car-like robot. The common grid-based search methods discretize the whole configuration space into a grid, where states are mapped to grid cells with transitions to its neighbors. However, due to the nonholonomic constraints, the motion of a car-like robot is anisotropic in the general coordinates. It is suboptimal to treat the position and orientation equally in a uniform grid world. Furthermore, some neighbors may only be reached with maneuvering through other grid cells. Therefore, in several search-based methods [53, 52], predefined primitive motions are applied to perform state transitions. In addition, a trade-off between efficiency and completeness should be accepted: a fine grid can achieve good completeness with the cost of a large number of states to evaluate; while a coarse resolution sacrifices completeness for a faster convergence. The SEHS method conducts the heuristic search in a more flexible way with the help of the workspace knowledge from the space exploration. The circle-path decomposes the path corridor into many circles, which enables a natural clustering of the states. A grid-free incremental search algorithm is

developed by adapting the state resolution and motion step size to the circle size and reduced it when no solution is found.

The design of a heuristic is the key of a heuristic search algorithm. A grid-based search method may apply a heuristic estimation based on the grid distance regarding the obstacles, which can be easily obtained with dynamic programming. The grid distance estimation is under the assumption that the robot can move independently in each coordinate of the grid. In addition, a Dubins or Reeds-Shepp curve [13, 14] based shortest path provides the minimum travel distance for a car-like robot without considering the obstacles. These two heuristics can be combined to serve as a better heuristic. However, such heuristics only concern about the travel distance, which may produce a shortcut close to the obstacles. In this case, a post-process is required to optimize the result regarding a safety margin. The circle-path from the space exploration naturally provides a path through the circle centers with the circle radius as a safe distance regarding obstacles in all directions. As a result, the distance estimation based on the circle-path provides a heuristic balancing the path length and the safety margin.

2.3.1 Problem Definition

The configuration space of a mobile robot is introduced in Section 1.2. A function $f_{\text{valid}}(\vec{q})$ validates a configuration $\vec{q}$ by checking collisions and the kinematic constraints. A motion $(\vec{q}, \vec{u})$, defined by the start configuration $\vec{q}$ and control inputs $\vec{u}$, is valid if all the internal states as well as the initial and final configurations are valid.

The heuristic search problem of SEHS can be defined as Problem 2. $\vec{q}_{\text{start}}$ is the start configuration. $\vec{q}_{\text{goal}}$ is the goal configuration. The valid configuration space is determined by a function $f_{\text{valid}}(\vec{q})$. The primitive motion are given as a set of control inputs U. The result from the space exploration is a sequence of shapes $\{S_0, S_1, \ldots, S_m\}$ that builds a path corridor in the free workspace. The desired solution is a sequence of control inputs $\{\vec{u}_0, \vec{u}_1, \ldots, \vec{u}_n\}$ that drives the robot from the start to the goal with valid motions.

Problem 2. *Heuristic Search*
Given: $\mathscr{C}$, $f_{\text{valid}}(\vec{q})$, $\vec{q}_{\text{start}} \in \mathscr{C}$, $\vec{q}_{\text{goal}} \in \mathscr{C}$, $U \subset \mathscr{U}$, *and* $\{S_0, S_1, \ldots, S_m\}$ *with* $S_i \subset \mathscr{W}_{\text{free}}, i = 0, 1, \ldots, m$
To Find: $\{\vec{u}_0, \vec{u}_1, \ldots, \vec{u}_n\}$ *with* $\vec{u}_i \in U, i = 0, 1, \ldots, n$ *and* $\{\vec{q}_0, \vec{q}_1, \ldots, \vec{q}_n\}$ *with* $\vec{q}_j \in \mathscr{C}, j = 1, 2, \ldots, n$ *that* $\vec{u}_0 : \vec{q}_{\text{start}} \rightarrow \vec{q}_0$, $\vec{u}_j : \vec{q}_{j-1} \rightarrow \vec{q}_j$, $\vec{q}_n \approx \vec{q}_{\text{goal}}$ *and every motion* $(\vec{q}_{i-1}, \vec{u}_i)$ *is valid.*

2.3.2 Vehicle Kinematic Models

The configuration space and control space are defined by the robot kinematics. The kinematics of a car-like robot can be modelled in different degrees of complexity for different kinds of motion planning scenarios. A wise decision about the kinematic model can save great effort of states evaluation in the search algorithm.

The state of a static car-like robot can be described with a vector (x, y, θ), of which x and y for the position and θ for the orientation in the two dimensional workspace. A holonomic robot can change each of the three coordinates independently, while a nonholonomic car-like robot is only able to change the position regarding a moving direction, as well as the orientation regarding a turning radius r. The curvature k of the motion is the inverse of the radius (2.6). If $k = 0$, the robot moves with an infinite radius, i.e., in a straight line. The minimum turning radius determines an upper bound of the curvature $k_{\max}$. When distinguishing the left turning and right turning with the sign, the motion curvature is bounded in $[k_{\min}, k_{\max}]$ with

$k_{\min} = -k_{\max}$.

$$k = \frac{1}{r} \tag{2.6}$$

For a moving robot, additional state variables are relevant such as velocity, acceleration, steering speed, etc. They are derivatives of the static states, which are related to the differential constraints. Four kinematic models for car-like robots are introduced in this section with different groups of the state variables for different planning purposes. Details about the kinematic models including equations of motion, configuration space geodesic, and primitive motions are presented in Appendix A.

2.3.2.1 Constant Curvature Model

The constant curvature model considers a constant turning radius during a primitive motion. In this case, the robot moves either in a circle or a straight line. The robot velocity is irrelevant to the path geometry. With a travel distance $s \in \mathbb{R}$ and a motion curvature $k \in [k_{\min}, k_{\max}]$, it is sufficient to determine the robot states of a primitive motion. The curvature is not smooth between motion primitives with different curvatures.

The configuration space of the constant curvature model is (x, y, θ), and the control space is (s, k). The dynamic properties such as velocity and acceleration are ignored, so as if the robot can change its moving direction and curvature immediately. A configuration space geodesic, i.e., the shortest path between two configurations, can be explicitly calculated with a certain curvature as in [13] for forward motion only and [14] for both forwards and backwards.

As the simplest kinematic model for a car-like robot, the constant curvature model is only suitable for motion planning scenarios with very low driving speed, e.g., parking or maneuvering. In these situations, a robot can stop and turn or change its moving direction at any time.

2.3.2.2 Continuous Curvature Model

In order to achieve smooth motion curvature, the continuous curvature model deals with a constant curvature change u_s proportional to travel distance s. In this case, the curvature becomes a state variable. The robot motion of a continuous curvature model can be defined with a travel distance $s \in \mathbb{R}$ and a steering parameter $u_s \in [u_{s,\min}, u_{s,\max}]$. The path of a primitive motion is an Euler spiral or clothoid. An Euler spiral can be calculated with Fresnel integral [121], which can be expanded to power series and solved with numerical methods. The Euler spirals provide smooth blending between straight lines and circle segments with different curvatures. Hence it is also widely used in road and railway designs.

The configuration space of the continuous curvature model is (x, y, θ, k), and the control space is (s, u_s). If $u_s = 0$, a continuous curvature model degenerates to a constant curvature model. An approximate configuration space geodesic can be calculated with the help of Dubins and Reeds-Shepp curves [15]. The result from Dubins or Reeds-Shepp is polished with Euler spirals at the ends to achieve smooth transitions between straight lines and circle segments. However, it only provides a solution for configurations with zero curvature, hence not complete for all the situations. A formal proof for optimality is also missing.

The continuous curvature model considers a constant change rate of the motion curvature for a smooth steering behavior. It is suitable for the driving scenarios, where a robot cannot dramatically change its velocity or steering.

2.3.2.3 Kinodynamic Curvature Model

The kinodynamic curvature model further takes the acceleration a into account. In this case, the velocity v becomes a state variable. Meanwhile, it is better to model the steering behavior with a time derivative of the curvature u_t. The traveling distance is determined by an initial velocity $v \in [v_{\min}, v_{\max}]$, an acceleration $a \in [a_{\min}, a_{\max}]$, and a time duration $t \in \mathbb{R}^+$. The curvature is calculated with an initial curvature k, a steering parameter $u_t \in [u_{t,\min}, u_{t,\max}]$, and time t.

The configuration space of the kinodynamic curvature model is (x, y, θ, k, v) with the control inputs (u_t, a, t). No closed form of configuration space geodesic is available for this kinematic model. Numerical methods should be applied to solve the motion equation by integrating the motion with a small time interval δt iteratively. When $a = 0$, a kinodynamic curvature model degenerates to a continuous curvature model. Otherwise, there is no explicit solution for the shortest path between two configurations, except for a few trivial cases.

A kinodynamic curvature state may drift with zero inputs, i.e., the robot continues moving with the current velocity and curvature. As a result, a robot state cannot be validated statically, because a collision-free configuration at one time may change to an invalid state at another time, even if the environment does not change. Therefore, the state validation should also consider the time effect.

The kinodynamic curvature model produces motions with continuous steering and velocity profiles. A trajectory can be directly obtained. Hence, a timestamp can be calculated for each state to verify collisions with moving obstacles. As a result, the kinodynamic curvature model is useful for motion planning in dynamic environments.

2.3.2.4 Single Track Model

The curvature variable in the previous three kinematic models only concerns about the geometric property of the path. In practice, it is determined by the steering angle of the wheels of a car-like robot, which can be calculated with a single track model. The motion curvature k can be determined with the wheel angle ϕ and the wheelbase l, e.g., for front steering as (2.7). A vehicle is controlled with an acceleration command $a \in [a_{\min}, a_{\max}]$ and a steering command $\omega \in [\omega_{\min}, \omega_{\max}]$, that change the speed and the angle of the wheels proportional to time.

$$k = \frac{1}{r} = \frac{\tan \phi}{l} \tag{2.7}$$

The configuration space of the single track model is (x, y, θ, v, ϕ), and the control space is (a, ω, t). Similar to the kinodynamic curvature model, there is no closed form solution for the motion. Only numerical integration can be applied with a small time step δt. A transform between a single track model and a curvature-based model is not trivial, as the relationship between the steering angle and the motion curvature is nonlinear. Therefore, this model does not degenerate to a continuous curvature model when $a = 0$.

The single track kinematic is close to the physical mechanism of a car-like robot. Hence, direct control inputs can be generated with this model for an autonomous vehicle. Similar to the kinodynamic curvature model, it produces smooth motions regarding velocity and curvature, and can be applied for motion planning in dynamic environments.

2.3.3 Primitive Motion Based Heuristic Search Algorithm

The heuristic search algorithm of SEHS can be generalized as a graph search algorithm in Algorithm 1. In each iteration, a vertex is selected regarding a circle-path-based heuristic

Algorithm 4: *HeuristicSearch*$(\vec{q}_{\text{start}}, Q_{\text{goal}}, U, \{S_0, S_1, \ldots, S_m\})$

```
1  v_start ← (q_start, S_0, 0, d_heuristic(q_start));
2  f_goal ← ∞;
3  k ← k_init;
4  V_closed ← ∅;
5  V_open ← {v_start};
6  E ← ∅;
7  while V_open ≠ ∅ do
8      Sort(V_open);
9      v_i ← PopTop(V_open);
10     if f_goal < f[v_i] then
11         break;
12     S_i ← MapCircle(q_i);
13     if Exist(q_i, S_i, k) then
14         continue;
15     else
16         (V_i, E_i) ← Expand(v_i, S_i, U, k);
17         if h[v_i] < r_goal then
18             (v̂_i, ê_i) ← GoalExpand(v_i, Q_goal, U);
19             if V̂_i ≠ ∅ then
20                 V_i ← V_i ∪ V̂_i;
21                 E_i ← E_i ∪ Ê_i;
22                 f_goal = min(f_min(V̂_i), f_goal);
23         V_open ← V_open ∪ V_i;
24         E ← E ∪ E_i;
25         V_closed ← V_closed ∪ {v_i};
26     if V_open = ∅ and f_goal = ∞ then
27         k = k × 0.5;
28         if k > k_min then
29             E_open ← E_closed;
30             E_closed ← ∅;
31 if f_goal < ∞ then
32     return success;
33 else
34     return failure;
```

which determines the global progress of the search. Control inputs are chosen regarding the primitive motions of a kinematic model to create new vertices from the selected one, so that the local structure of the graph is constructed. The details of the search procedure are outlined in Algorithm 4.

A vertex in the search graph consists of $(\vec{q}, S, g, h)$: a robot state $\vec{q}$, a reference circle S, and the g-value and h-value for the heuristic search. The mapping between a robot state and a circle is done by a function MapCircle$(\vec{q})$. The g-value is the sum of the motion costs from the start

state to the current state $\vec{q}$. The h-value is calculated with a heuristic function, which returns an estimation of the rest cost to reach the goal. An edge in the search graph is a primitive motion between two states.

The heuristic search is performed with an open set and a closed set to maintain the vertices. The open set is initialized with the start state $\vec{q}_{\text{start}}$. In each iteration, the vertex with the least f-value from the open set is selected to extend the graph. If the f-value of the selected vertex is greater than the goal cost f_{goal} already achieved, the planning is finished because an optimal solution has already be found. Otherwise, the selected state is mapped to a path circle for heuristic estimation and motion adaptation. By comparing with the states in the closed set, it is possible to skip redundant states to reduce the search effort. After that, the state is expanded with the primitive motions. In addition, if the state is inside a range r_{goal} of the goal region, a direct attempt is evaluated for the case that a goal can be reached with a simple motion primitive. The goal cost f_{goal} is updated when a better result is found. When the open set is empty, the algorithm terminates with a success if a solution is found or a failure when the goal is not reached.

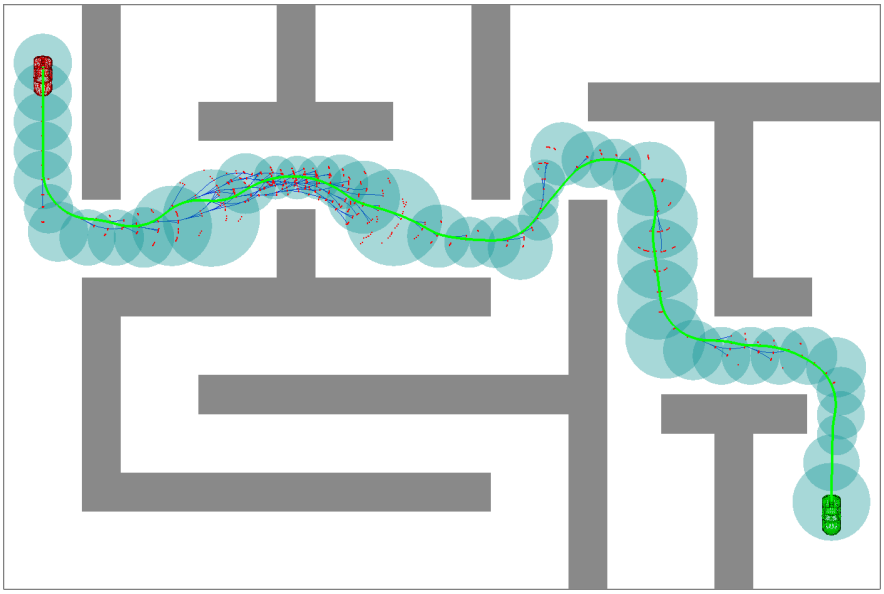


Figure 2.9: Heuristic search with the guidance of space exploration in a labyrinth

Fig. 2.9 shows the heuristic search progress and result with the guidance of space exploration heuristic in a labyrinth. The cyan circles are the path corridor suggested by the space exploration. The heuristic search expands states as the red dots with the blue segments as the primitive motions. The propagation of the states follows the path corridor and converges to a solution as the thick green line. A continuous curvature model is applied in this example for the primitive motions with a curvature $k \in [-0.2, 0.2]$ and a steering parameter $u_s \in [-0.2, 0.2]$. For every state expansion, five primitive motions are evaluated with different steering parameters of forward driving. The travel distance is with a ratio 0.5 to the according path circle radius. Collision checks are performed at internal states with a 0.5 cm interpolation step size.

The main advantage of this motion planning approach is contributed by the circle-path heuristic, which presents the workspace knowledge. In this case, the states can be clustered to the circles to resolve the similar ones; the heuristic cost can be estimated regarding the distance

along the circle-path; control inputs and the state resolution can be adapted according to the circle size. The details are explained in the following subsections.

2.3.3.1 Circle-Path States Clustering and Heuristic Estimation

A circle-based states clustering is an important preprocess for the most subroutines such as heuristic estimation and the nearest neighbor search in a search iteration. A robot configuration is projected to the explored sub-space, in this case the workspace, and mapped to the nearest circle regarding the Euclidean distance to the circle center.

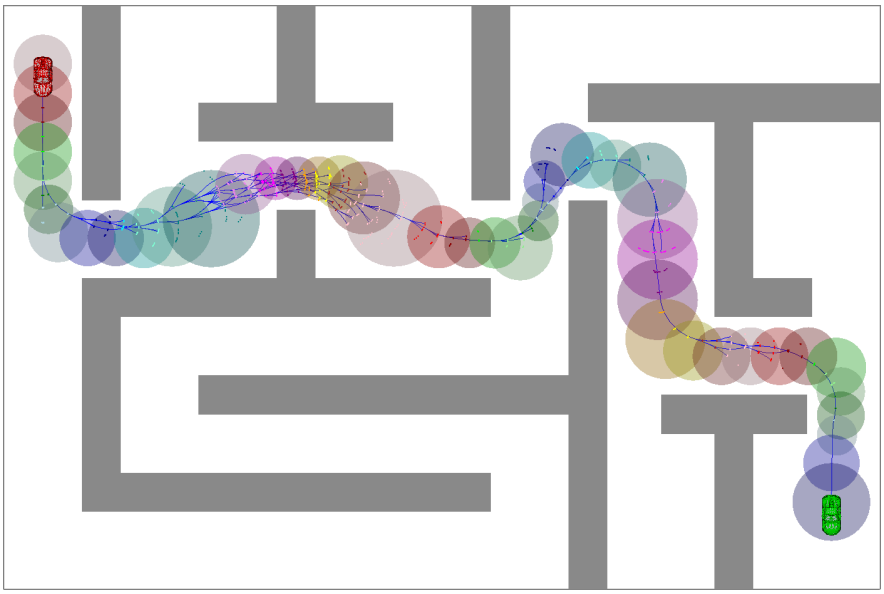


Figure 2.10: Circle based states clustering for heuristic search

Fig. 2.10 shows the clustering result in the labyrinth example. The circles are colored differently with the mapped state dots in the same color. Comparing to a grid-based search approach, the overhead of this type of clustering for each state is linear to the number of circles. However, no discretization is required, which provides great flexibility to adapt the resolution. The clustering can be further augmented with a distance margin so that the search algorithm may reject a state if it is far away from the path corridor. In this case, the vertices outside a certain range of the circle-path are pruned to speed up the search process. With the cost of the completeness of the heuristic search, the search effort is then concentrated in the circle-path region for a result compromising length and obstacle distance.

As the circle-path holds the topological information about the space from start to goal, the distance through the circle centers provides an estimation of the distance along the passage. After mapping a state to a circle, the estimation of the remaining distance to the goal is the sum of the distance to the next circle and the rest distance along the circle-path. As illustrated in Fig. 2.11, O_i, O_{i+1}, and O_{i+2} are path circles. The state at point P is mapped to circle O_i. The heuristic distance estimation consists of two parts: one is the distance between the point P and the next circle center O_{i+1}; the other is the sum of the distance from O_{i+1} to the goal circle as the arrows from O_{i+1} to O_{goal}.

This heuristic does not provide estimation for the shortest path distance, but the path

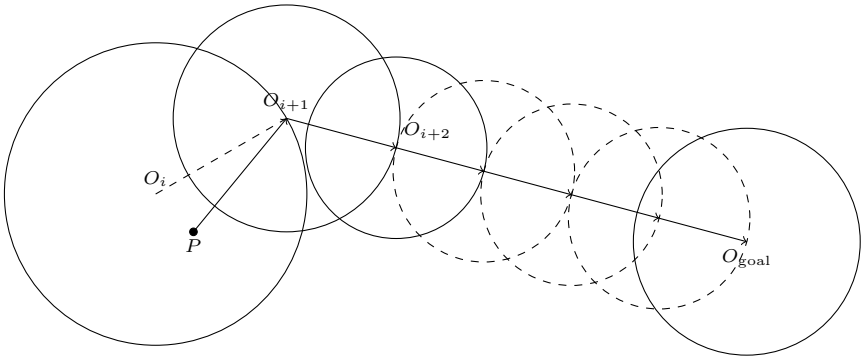


Figure 2.11: Circle-path heuristic distance estimation

distance along a centric line of a path corridor for a safer and more human like driving behavior. The space exploration determines the circle radius as the distance to the nearest obstacle with a safety margin. At the same time, it takes the Euclidean metric as heuristic to optimize the path length, which provides an admissible heuristic following the circle-path. Therefore, a trade-off between path length and safety distance is embodied in the circle-path. The heuristic search follows the passage as the global search direction with the local bias to the next circle center to avoid collisions.

If travel time is concerned in the motion planning, the estimated distance can be transformed to estimated time by dividing with a desired velocity. Thus, the planner can search for an optimal motion with the kinodynamic models.

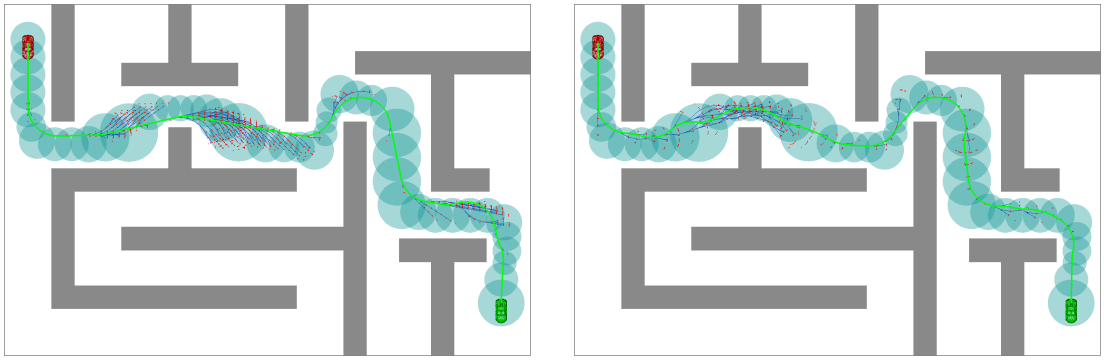
2.3.3.2 Search Step-Size and Resolution Adaptation

The size of a path circle holds the information about the local free space dimension, which is helpful to adapt the search step size and states resolution. The step size is updated proportional to the circle size (2.8). A step factor α decides the ratio between the step size and the according circle radius r_{c_i}. A minimum step $s_{\min}$ is defined as the lower bound of the step size. If a state is in the goal circle, the equation considers the Euclidean distance to the goal state instead of the circle radius. Thus, the heuristic search takes large steps for big circles, and proceeds carefully in small circles or close to the goal.

$$s = \begin{cases} \max(\alpha \cdot r_{c_i}, s_{\min}) & \vec{q} \in c_i \\ \max(\alpha \cdot d_{\text{point}}(\vec{q} - \vec{q}_{\text{goal}}), s_{\min}) & \vec{q} \in c_{\text{goal}} \end{cases} \tag{2.8}$$

Fig. 2.12 compares heuristic search with constant step size and adaptive step size. A constant step size 1.0 m is applied in Fig. 2.12(a). The same algorithm takes adaptive step size with a factor 0.5 in Fig. 2.12(b). The version with constant step size generates more vertices around the large circles, while the adaptive one takes large steps and quickly pass through the large free regions. In order to achieve good search completeness, a small step size should be chosen in the constant step size version, which is inefficient in large free space. However, the step size adaptation solves the problem with a good trade-off between completeness and efficiency. The step factor can be further reduced when no solution is found.

The state resolution relates closely to the motion step size, as it should be chosen accordingly to resolve the states produced by the motion primitives. As the step size is adapted to the circle radius, the resolution could also be adjust in the same way. A nearest neighbor search is performed to decide whether a state has already been evaluated regarding the resolution.



(a) Heuristic search with constant step size

(b) Heuristic search with adaptive step size

Figure 2.12: Comparing constant step size and adaptive step size in heuristic search

The circle-based state clustering provides the convenience that only a few states mapped to the same circle should be checked. Furthermore, as the step size is adapted for vertices expansion, the number of states in a circle would be relative small and hardly relevant to the circle size. As a result, the nearest neighbor search could even be conducted with a linear search method.

The distance metric for state resolution is not the actual motion distance between two states, which will be as complex as solving another motion planning problem, but an estimated distance as a lower bound of the actual distance. Taking the constant curvature model for example, an estimated distance between two states would be the maximum of the Euclidean distance and the distance required to compensate the orientation difference (2.9). $d_{\text{point}}(\vec{q_1}, \vec{q_2})$ is the Euclidean distance between $\vec{q_1}$ and $\vec{q_2}$. $d_{\text{angle}}(\theta_1, \theta_2)$ is the angle difference between the states in a range $(-\pi, \pi]$. The absolute value multiplying the minimum steering radius of the robot yields the minimum distance regarding the angle difference. Thus, a lower bound of the actual motion distance between two states is obtained. If this distance is less than the resolution of the circle, the state is redundant and skipped by the search algorithm.

$$d_{\text{resolution}}(\vec{q_1}, \vec{q_2}) = \max(d_{\text{point}}(\vec{q_1}, \vec{q_2}), |d_{\text{angle}}(\theta_1, \theta_2)| \times r_{\min}) \tag{2.9}$$

It is also possible to apply adaptive resolution with an octree data structure. However, it is suboptimal for the continuous nonholonomic motion and the circle-based space decomposition. A grid should be created in advance with the resolution information for the whole configuration space. In contrast, the SEHS algorithm makes use of the circle-based state clustering to achieve a more flexible resolution schema along the path corridor. The resolution can also be increased regarding the step size decrement when no solution is found. Thus, based on the circle-based segmentation, the configuration space is decomposed regarding the vertices, that each represents a hyper-polyhedron neighborhood.

2.3.3.3 Motion Cost Calculation

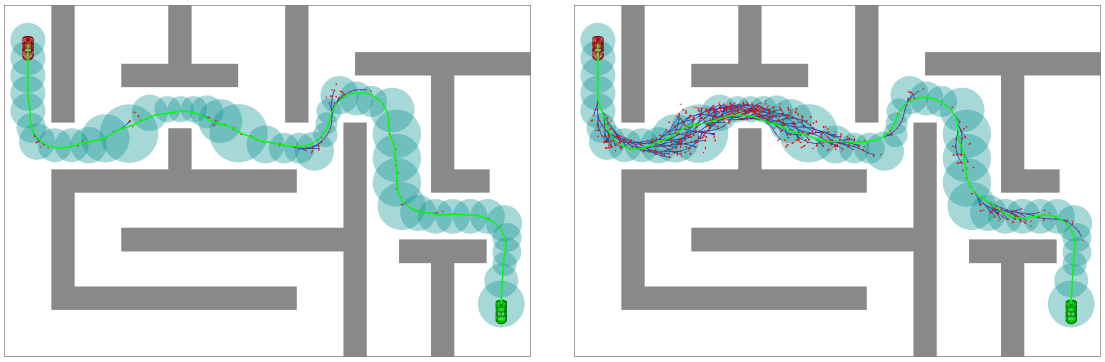
The actual motion cost is calculated based on the length and type of the motion. It not only counts the actual distance between two states, but also punishes the unwished motion behaviors with additional cost. An example of cost formulation is (2.10).

$$g_{\text{cost}} = s_{\text{length}} + \alpha \times s_{\text{reverse}} + \beta \times s_{\text{steering}} + \gamma \times n_{\text{cusp}} \tag{2.10}$$

The following types of costs are included:

- The basic cost is calculated regarding the absolute total length s_{length} of the motion. It can be the distance of the movement or the time duration of the motion. Usually it takes a factor of 1.0.
- The reverse driving cost is an additional cost for driving backwards. It is proportional to the reverse motion length s_{reverse} with a positive factor α.
- The steering cost is an additional cost for turning behavior. It is proportional to the steering motion length s_{steering} with a positive factor β.
- The direction switch cost is an additional cost for a motion cusp. A positive additive cost γ is counted for each direction change. n_{cusp} is the number of cusps.

As all the additional costs are positive, the final cost is larger than the actual length of the motion. Therefore, the admissible condition of the heuristic still holds.



(a) Heuristic search with a simple cost function

(b) Heuristic search with a complex cost function

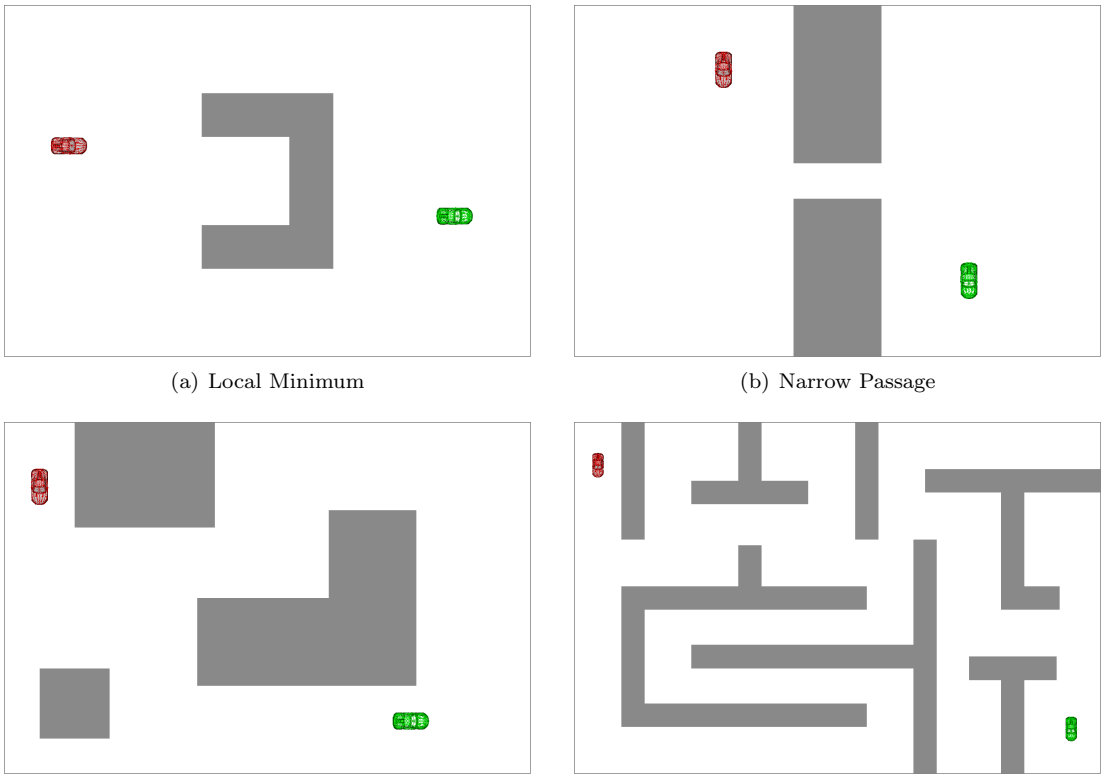
Figure 2.13: Comparing heuristic search with simple cost and complex cost functions

A complex cost structure produces an optimal result with the cost of extra heuristic search effort. Because the circle-path heuristic only provides the optimistic distance estimation, the discrepancy is larger with a complex actual cost. Therefore, more vertices should be evaluated to reach the termination condition. Fig. 2.13 compares the results from a simple cost function and a complex cost structure. In Fig. 2.13(a), the actual cost only accounts for the travel distance. The cost function in Fig. 2.13(b) consists of all the components in (2.10) with $\alpha = 0.5, \beta = 0.2, \gamma = 1.0$. Obviously, it takes more vertices to find the optimal motion in the second case, which has not much difference than the first one. Therefore, the cost function should be carefully selected for the specific scenarios. In this case, a post-optimization may be a better choice following a fast motion planning with a simple cost function. Further discussions are made in the Chapter 3 for parking and maneuvering.

2.4 Experiments

In order to evaluate the SEHS method, four experiments in Fig. 2.14 are designed and performed comparing with the state of the art approaches. A local minimum scenario in Fig. 2.14(a) and a narrow passage scenario in Fig. 2.14(b) assess the planning performance in the extreme situations. A simple navigation example in Fig. 2.14(c) evaluates the general motion planning

in a near range local scenario. In these three experiments, the results of SEHS are compared with RRT [34], EET [69], and Hybrid A* [51] methods, which are also implemented in *AutoDrive* library. Finally, a large labyrinth scenario in Fig. 2.14(d) is evaluated as a long range global planning problem. The extended RRT methods from OMPL [109] are also included as benchmarks in this scenario.



(a) Local Minimum

(b) Narrow Passage

(c) Local Navigation

(d) Large Labyrinth

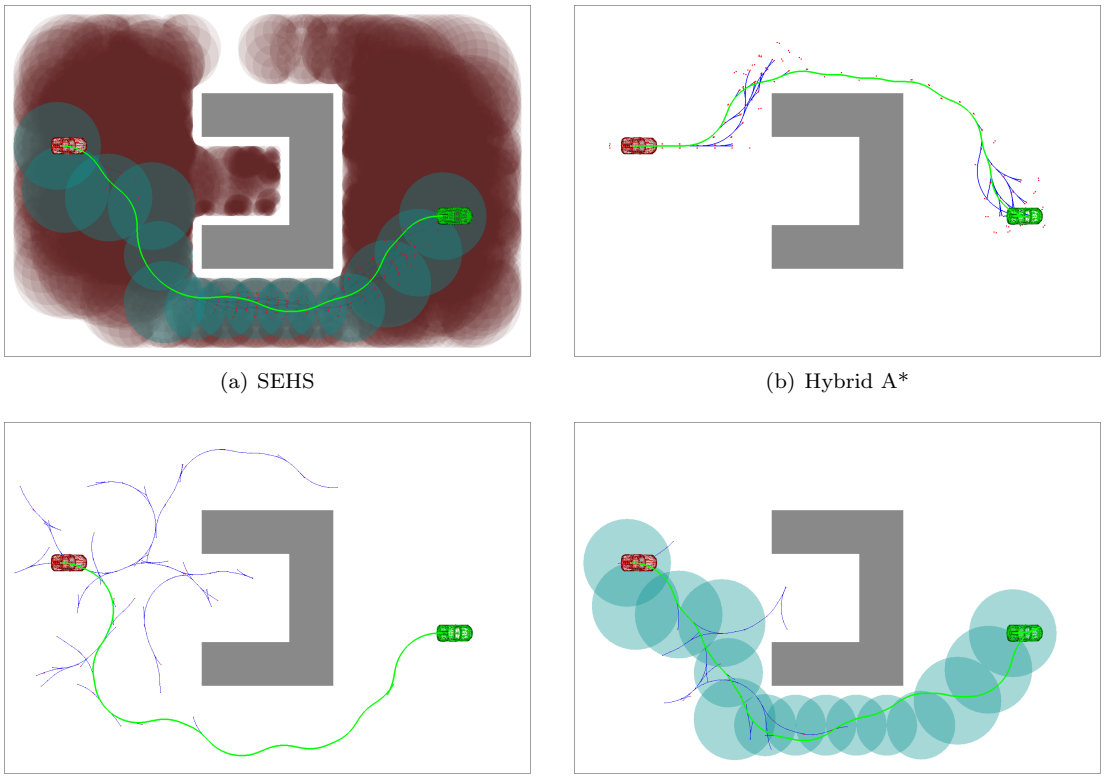
Figure 2.14: Experiment scenarios

The continuous curvature model in Section 2.3.2.2 with the maximum curvature $0.2\,\mathrm{m}^{-1}$ and steering $0.2\,\mathrm{m}^{-2}$ is applied for smooth motion results. The steering input range $[-0.2, 0.2]$ is interpolated for the primitive motions with different travel distances. The Hybrid A* and RRT methods take a constant step size for primitive motions. SEHS calculates the motion step size by multiplying the circle radius with a step factor 0.33. The state resolution is also adapted regarding a parameter 0.5. RRT method extends the random tree bias the goal direction in a period of 10 iterations.

For all the planning algorithms, the performance is measured with the results from 100 trials with different start and goal states randomly generated within ± 0.5 meter in x and y directions and ± 10 degree for orientation regarding the given configurations. All the algorithms are implemented in C/C++ with the same kinematic model, environment model, and collision check methods from the *AutoDrive* library. The experiments are performed on a desktop PC running Linux with an Intel Core i7 2.90 GHz CPU and 8 GB RAM.

2.4.1 Local Minimum Scenario

The local minimum scenario has a U-form obstacle in the middle. The start and goal positions are located on the different sides of the obstacle. A greedy search algorithm may have trouble in this local minimum setup. Example results of the four methods are illustrated in Fig. 2.15. The SEHS planner in Fig. 2.15(a) first investigates the workspace with space exploration (brown circles), which converges to a circle-path (cyan circles) going around the obstacle. Therefore, the heuristic search (red dots as nodes and blue lines as edges) reaches the goal configuration without bothering with the local minimum. The Hybrid A* in Fig. 2.15(b) handles the obstacle with a grid-distance heuristic, and goes around it from the other side. The search progress shows a jitter effect from the grid discretization. Furthermore, the path from Hybrid A* takes a smaller safety distance due to the grid resolution. The RRT method in Fig. 2.15(c) solves the problem with random samples in the whole configuration space. Therefore, the result consists of several arbitrary unnecessary motions. In contrast, the EET approach in Fig. 2.15(d) samples along the circle-path, which finds a result with fewer samples. But it still does not has the guarantee of no random moves.



(a) SEHS (b) Hybrid A*

(c) RRT (d) EET

Figure 2.15: Local minimum scenario

The statistics from the 100 trials are listed in Table 2.1 as the number of states, the number of verification or distance queries, the planning time, and the standard deviation of the planning time. There are two rows for the exploration and search subroutines of the SEHS method respectively. The circle number and distance queries are counted for the space exploration

procedure. SEHS has the best time performance in this example. Hybrid A* and EET are similar. RRT takes the longest time to solve the problem. The space exploration of SEHS only costs about 10% of the total planning time with the benefit of about 40% faster planning than Hybrid A* in this scenario. The circle-path heuristic of SEHS is better for diagonal motions, and the step size adaptation greatly reduces the number of states expended in a large free space. In contrast, Hybrid A* searches with a constant step size of 2 m and the according state resolution. Due to the nonholonomic constraints, the RRT method takes huge effort to find a valid vertex to extend the search tree. As a result, with three times as many verifications as Hybrid A*, RRT only produces a few more vertices. By applying the same circle-path from space exploration as a sampling guidance, the EET method achieves even slightly better time performance than Hybrid A* with an RRT planning schema.

The planning time deviation of the search-based methods is mainly caused by the final expansion, which has large influence to the total planning time. If the goal configuration is not easy to reach from the states systematically generated by a heuristic search, extra effort is required for the last mile. Hybrid A* suffers greater from this drawback, because its heuristic tends to take shortcuts, which left little space margin for the final maneuvering. In contrast, the SEHS method maintains good distance to the obstacle by following the circles centers that preserves enough space for the final motion. The sampling-based methods do not have this issue for the random sampling schema. As a result, Hybrid A* has the largest relative performance deviation in this scenario.

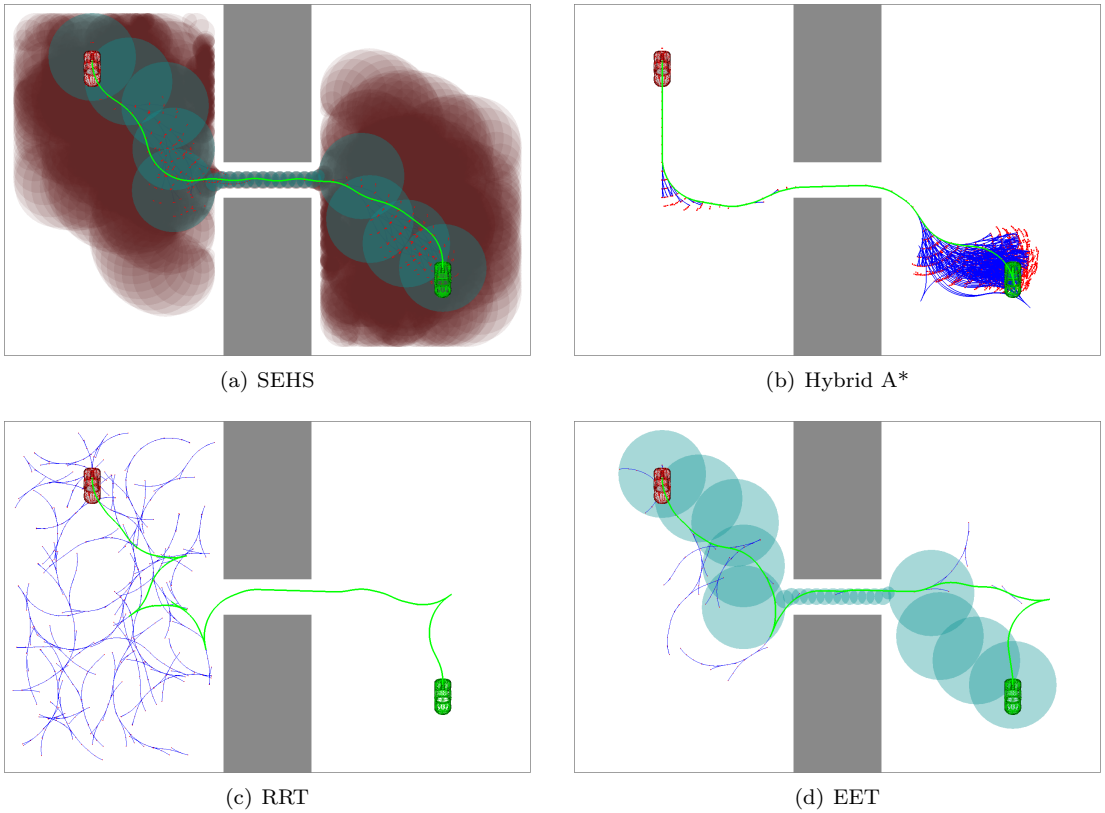
Table 2.1: Results of the local minimum scenario

Planner	States	Queries	Time (ms)	Time STD (ms)
$\text{SEHS}_{explore}$	1 671	2 084	3.73	1.47
SEHS_{search}	413	19 278	24.66	9.78
Hybrid A*	534	23 516	39.98	20.69
RRT	665	74 839	92.51	33.38
EET	245	30 511	38.97	11.72

2.4.2 Narrow Passage Scenario

In this scenario, the workspace is divided in two parts with a narrow passage connecting them. Such a problem is extremely difficult for random sampling methods to solve, as the probability of a sample in the narrow passage is very small. Example results of the four methods are illustrated in Fig. 2.16. The SEHS method explores the workspace from the both sides and finds the passage in the middle as in Fig. 2.16(a). The Hybrid A* can also extract this passage with grid-based heuristic. However, the final steps are rather time consuming as in Fig. 2.16(b). The RRT generates most of the vertices on the start side, and quickly reaches the goal when the tree goes through the passage as in Fig. 2.16(c). The EET takes less samples to find the passage, as it has already obtained the knowledge from the space exploration as in Fig. 2.16(d).

The statistical results are listed in Table 2.2. SEHS is the fastest among the four with dominant time performance. The random sampling methods have extra large deviations in planning time. Because of the complexity of nearest neighbor search, the planning time grows $O(\log n)$ regarding the number of vertices. Therefore, the moment when the random tree finds the passage makes large difference of the total planning duration. The EET can improve the time performance by one magnitude from RRT, as it already knows the path corridor. However, EET is mainly developed for holonomic motion planning, so it is still a great challenge



(a) SEHS (b) Hybrid A*

(c) RRT (d) EET

Figure 2.16: Narrow passage scenario

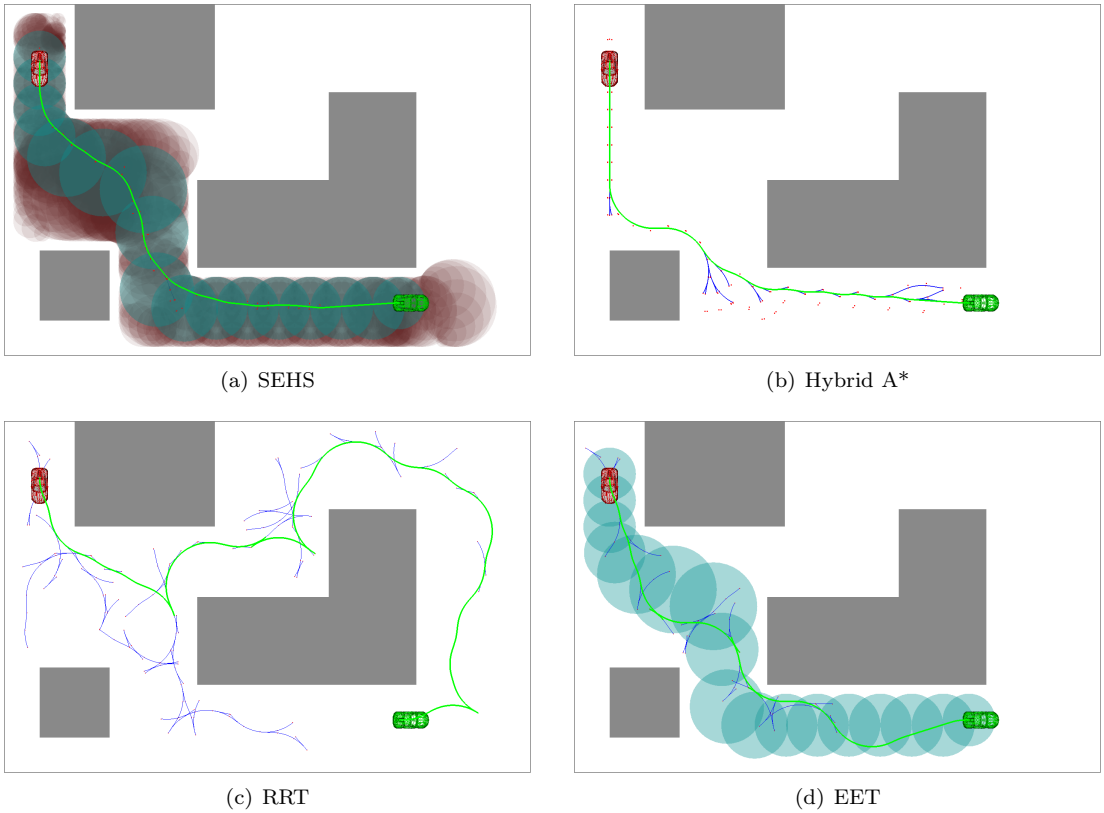
to progress with nonholonomic kinematics in a narrow passage. Between the two search-based methods, the step size and state resolution adaptation of SEHS brings a larger advantage against Hybrid A* in this scenario. Instead of a constant small step size and fine state resolution of Hybrid A* for the narrow passage, SEHS takes larger steps with adapted resolution elsewhere to make faster progress. The final extension is another setback of Hybrid A* in this scenario, which is enhanced by the small step size. As a result, Hybrid A* should pay a lot more state expansions for the shortcut it takes.

Table 2.2: Results of the narrow passage scenario

Planner	States	Queries	Time (ms)	Time STD (ms)
$\text{SEHS}_{explore}$	1 030	1 466	1.69	0.73
SEHS_{search}	2 043	49 974	66.85	106.12
Hybrid A*	16 135	347 491	816.01	308.95
RRT	10 543	1 109 109	9 762.84	23 279.65
EET	1 982	218 067	994.13	3 800.34

2.4.3 Simple Navigation Scenario

This scenario is a simple navigation example with a few obstacles. Example results of the four methods are illustrated in Fig. 2.17. The SEHS method plans a motion follows the center points of the circles, which stays away from the obstacles from all directions as in Fig. 2.17(a). The Hybrid A* method produces a path following the grid distance heuristic, which sometime goes closely along a side of an obstacle as in Fig. 2.17(b). The path of RRT is the longest one with many cusps as in Fig. 2.17(c). The result from EET follows the circle-path in a random manner as in Fig. 2.17(d).



(a) SEHS

(b) Hybrid A*

(c) RRT

(d) EET

Figure 2.17: Narrow navigation scenario

The statistical results are listed in Table 2.3. The SEHS method has the best time performance in this local navigation scenario. The Hybrid A* takes the second place with 50% longer time than SEHS. The final pose is aligned with the last expansions, therefore, the search-based methods do not have final expansion problem here. The RRT returns a result with the largest time cost and standard deviation. With the workspace knowledge, EET takes only about a half time than RRT in average.

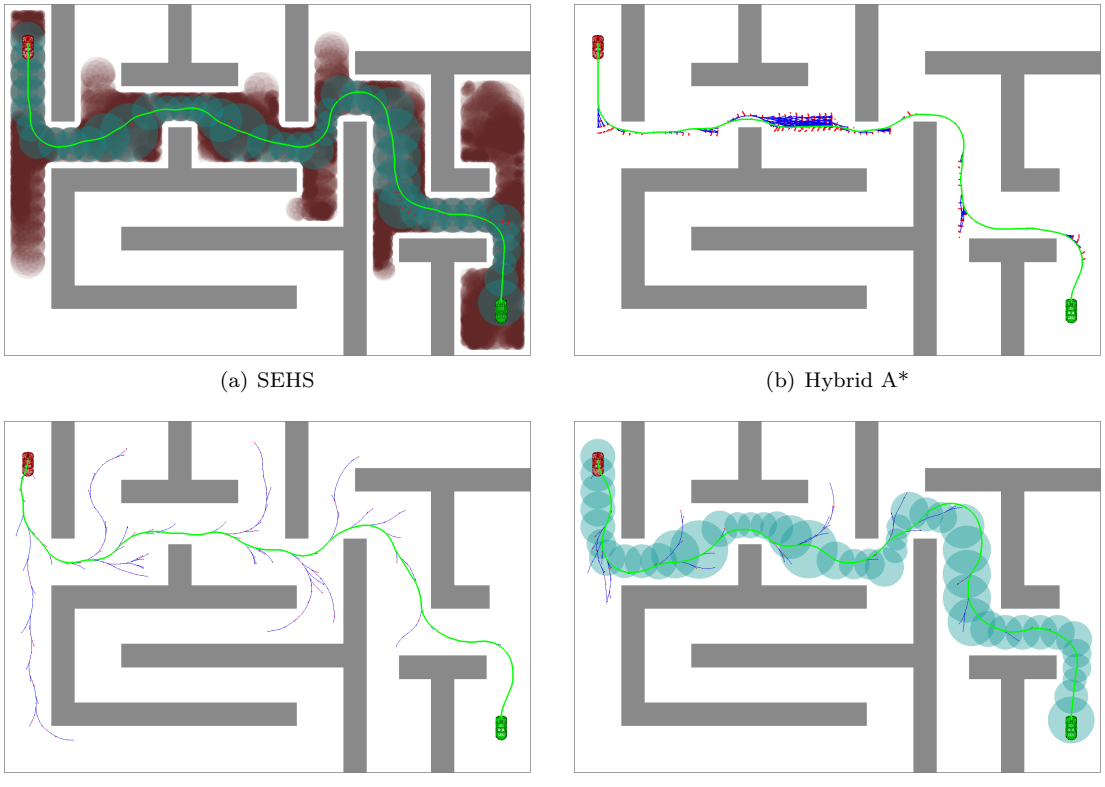
2.4.4 Large Labyrinth Scenario

This scenario is a large labyrinth of 90 m × 60 m. The robot should move diagonally across the labyrinth. The example results of the four planners from *AutoDrive* are illustrated in Fig. 2.18.

Table 2.3: Results of the simple navigation scenario

Planner	States	Queries	Time (ms)	Time STD (ms)
$\text{SEHS}_{explore}$	594	776	1.03	0.62
SEHS_{search}	338	9 253	15.45	3.25
Hybrid A*	289	12 888	23.53	5.10
RRT	293	61 479	70.55	38.74
EET	115	26 680	33.80	9.57

The SEHS method employs a bidirectional space exploration to find a path corridor avoiding detours and local minimums. The heuristic search then returns a smooth motion tracking the centers of the circles as in Fig. 2.18(a). The Hybrid A* with a constant gird resolution and step size produces a path close to the obstacles in Fig. 2.18(b). As discussed before, the RRT planner explores the configuration space with a random tree in all the directions in Fig. 2.18(c), while the EET generates samples bias the path corridor in Fig. 2.18(d).



(a) SEHS (b) Hybrid A*

(c) RRT (d) EET

Figure 2.18: Large labyrinth scenario

The quantitative results are listed in Table 2.4 including the three extended RRT methods from OMPL. In this scenario, the search-based methods and the sampling-based methods have large difference in time performance as in the narrow passage experiment. The space exploration

procedure of SEHS takes a larger proportion of 1/3 of the total planning time than in the previous small scenarios. However, the total average time cost of SEHS is still below 100 ms, which is only 1/4 of the Hybrid A* method. The Hybrid A* method requires an optimal configuration of the step size and grid resolution to get a good time performance without sacrificing the completeness, i.e., solving all the random problems. The RRT method has much longer planning time in seconds, as it explores randomly in the labyrinth. The EET planner is 3 times faster in average than RRT for the extra workspace knowledge.

Table 2.4: Results of the large labyrinth scenario

Planner	States	Queries	Time (ms)	Time STD (ms)
$\text{SEHS}_{explore}$	4 674	7 035	25.25	4.21
SEHS_{search}	1 171	25 520	43.08	14.18
Hybrid A*	5 598	124 909	260.91	150.24
RRT	4 707	739 640	4 193.88	9 911.00
EET	2 031	311 227	1 375.04	3 385.26
EST (OMPL)	31 524	855 955	1 603.56	284.27
PDST (OMPL)	38 952	483 176	1 424.43	821.86
KPIECE (OMPL)	18 770	1 213 745	2 179.06	5 963.98

Three advanced RRT algorithms, EST [75], PDST [37], and KPIECE [41] from the OMPL library, are also evaluated in this scenario. The library version is 0.13.0 from Ubuntu. The same kinematics and collision checking implementations from *AutoDrive* are applied. There are still several differences in the OMPL planning setup. An OMPL algorithm does not use a direct final expansion to exactly reach the goal configuration, but returns success when it arrives in a certain neighborhood of the goal configuration. The goal threshold is set to 0.5. In this case, the terminate condition is easier for the OMPL methods. The state expansion evaluates control variables for a time step between 1.0 and 5.0 in 0.1 steps, while the random sampling methods from *AutoDrive* take a constant step size. The collision checks are done in a resolution proportional to the scenario scene size. As the labyrinth is relatively large, the step factor is chosen as 0.0005. In contrast, the planners from *AutoDrive* check collisions for every 5 cm. All the three methods require a sub-space projection, which is the space of position (x, y) and orientation θ with a resolution of $(2.0, 2.0, 0.2)$. The states counter is also different in OMPL, which counts all the state evaluated, while the *AutoDrive* methods only count the vertices added to the search tree.

In Table 2.4, all the three OMPL methods have time performance in second range. The EST takes a range of 1.0 for the maximum length of a motion to be added in the tree. The planning time is larger than PDST and less than KPIECE. KPIECE produces the worst time result among the three. It is difficult to tune the cell score coefficients of KPIECE. Because the labyrinth environment is clustered with obstacles, a cell extension is much likely to fail. As a result, the cell scores quickly reach the limit of the float precision even when the both good and bad factor are set very close to 1.0. The result in the table is produced with a good factor of 0.8 and a bad factor of 0.4. Among the five sampling-based methods, EET provides the best time performance, while EST has the most stable results.

2.5 Summary

The SEHS method dominates the experiments in time performance. The exploration duration changes regarding the size and complexity of the scenario, while the search time does not change much. Due to the step size and state resolution adaptation, it has advantage over the grid-based Hybrid A* method. The random sampling methods have drawbacks for the inefficient random exploration schema, especially with narrow passages and in large complex scenarios.

Another advantage of the search based method is the deterministic and repeatable natures. In contrast, the paths from the sampling based methods are random in length and form, which require great optimization effort for practical autonomous driving applications. Furthermore, the SEHS method can produce better results concerning the safety margin, as it follows the centers of the path circles. The Hybrid A* applies the grid distance heuristic and the Reeds-Shepp shortest path heuristic, that may generate results close to the obstacles. Therefore, the *Space Exploration Guided Heuristic Search* approach is a good general motion planning method for nonholonomic car-like robots.

Chapter 3

Orientation-Aware Space Exploration Guided Heuristic Search

Parking or maneuvering a vehicle in a narrow clustered environment is among the most challenging tasks in daily driving. These problems are usually addressed by ADAS functions with specific planning algorithms, which provide highly customized solutions with strong assumptions about the scenario regarding extra domain knowledge. However, such an approach is only efficient when it is applied to the proper problem. In this case, a highly automated system should decide whether all the pre-requirements are satisfied to perform the dedicated driving function. The usability of these approaches is greatly limited, because there are often exceptions and corner cases that cannot be handled with the specific solution, for example a general maneuvering scenario with no clear rules or schema that can be applied. Therefore, a universal method for maneuvering tasks is essential for highly automated driving.

In this chapter, an extended version of SEHS, called *Orientation-Aware Space Exploration Guided Heuristic Search* (OSEHS), is introduced to solve the motion planning problem for maneuvering and parking. OSEHS considers the robot orientation and steering constraints to some extent during space exploration for the knowledge about driving directions and maneuvering regions. The extra information is exploited by the heuristic search to further adapt the cost estimation and primitive motions to improve the motion planning performance in difficult parking and maneuvering scenarios.

3.1 General Idea

The baseline SEHS method provides a two stage framework to solve a motion planning problem: space exploration in workspace investigates the free space topology and dimension information; heuristic search in configuration space follows the workspace guidance with adapted motion primitives and state resolution. As demonstrated in the previous chapter, this approach is efficient to solve many motion planning problems for nonholonomic car-like robots even in complex scenarios such as a labyrinth. However, when maneuvering is involved, the general circle-based space exploration cannot provide any further hints about the maneuvering motion. As the robot should move back and forth around a single position, this driving behavior can be neither captured in the space topology, nor the space dimension knowledge with a circle-path.

In this case, the circle-path heuristic is too optimistic when extra maneuvering is required. As a result, the heuristic search should evaluate a large number of states to compensate the discrepancy between the heuristic cost and the actual motion cost.

In order to create a more effective heuristic for maneuvering, the information about driving directions and maneuver locations is relevant. The first one indicates a preference of the moving directions along the path corridor. The second one provides hints about where to perform the maneuvering. Based on these heuristics, the search algorithm can adapt the primitive motions or cost functions at the according locations. Thus, OSEHS can follow the global path corridor with different local behaviors more efficiently than the general SEHS method.

The circles of SEHS only provides space topology and dimension information as position and radius. In order to evaluate the moving directions, the flow of the circles is studied in addition. In this case, each circle has an orientation which indicates the advance direction of the search graph, i.e., pointing from the parent circle to the child circle. By comparing the orientation of the neighboring circles, the moving direction can be determined. In addition, it is possible to estimate whether the robot can follow the path corridor considering the steering constraints. If the orientation change is too abrupt, extra maneuvering is required in the region. Thus, the space exploration is orientation-aware and provides the heuristic search with a directed circle-path suggesting moving directions and maneuvering regions.

3.2 Orientation-Aware Space Exploration

The basic circle-based space exploration relaxes the kinematic constraints of a car-like robot to a holonomic mobile robot without orientation. In this case, the robot can move directly from one point to any other point insight in the workspace. In a clustered environment, it may dramatically change the moving direction to achieve a circle-path with the shortest travel distance. Furthermore, the space exploration algorithm terminates when it reaches the goal position, regardless of a mismatch between the final traveling direction and the goal orientation. These facts increase the discrepancy between the circle-path heuristic distance and the actual motion cost, which requires large search effort to fill the gap.

In order to extract the direction information, the orientation-aware space exploration considers a holonomic robot with orientation that can turn freely during a motion. In this case, an orientation is defined for each circle. The change of orientation is evaluated with a cost function concerning the steering constraints of the nonholonomic robot. The result is a circle-path with suggested driving directions for each circle.

A vertex in the search graph for the orientation-aware space exploration is (p, θ, r, g, h) with θ for the orientation. The general procedure of the orientation-aware space exploration is similar as the basic space exploration algorithm in Algorithm 2. In addition, the circle orientation is calculated in the expansion subroutine. In this case, the exploration is not only in the two dimensional workspace, but also involves the orientation coordinate. Therefore, the cost function and the overlapping relation for should be extended to the additional dimension. Details are provided in the following subsections.

3.2.1 Cost Function for Orientation Change

Additional to the Euclidean distance heuristic for the basic circle-based exploration, the cost function (3.1) considers the start and end orientations and the steering constraint of the robot. $\|\vec{p}_i - \vec{p}_j\|$ is the Euclidean distance between the circle center points $\vec{p}_i$ and $\vec{p}_j$. $|\theta_i - \theta_j|$ is the absolute angle difference in $[0, \pi]$. If an orientation angle is represented with a point on a unit circle in Fig. 3.1. The difference between these two orientation angles θ_i and θ_j is the length

of the shorter arc between P_i and P_j, i.e., the angle $\angle P_iOP_j \in [0, \pi]$. If the robot motion is constrained with a maximum curvature k, the distance required to compensate this angle difference is greater or equal to $|\theta_i - \theta_j|/k$. This value serves as a distance metric for the angular difference. Thus, the distance between two directed circles is calculated as the maximum of the two kinds of distance.

$$d_{i,j} = \max(\|\vec{p}_i - \vec{p}_j\|, |\theta_i - \theta_j|/k) \tag{3.1}$$

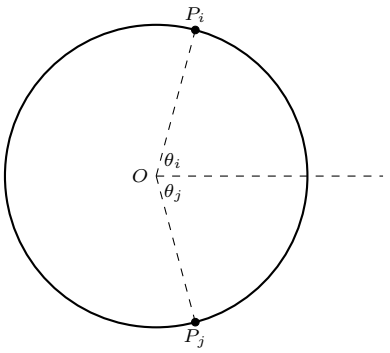


Figure 3.1: Circle angle difference

This cost calculation is still an admissible and monotone heuristic estimation. The admissible condition is trivial, because each of the two types of distance is admissible, which can be combined as an admissible estimation with a maximum operation. A proof for the monotone condition is provided as the following.

Proof. Let $\vec{p}_k$ be a point other than $\vec{p}_i$ and $\vec{p}_j$ with an orientation angle θ_k.

1. According to triangle inequality, it is trivial that $\|\vec{p}_i - \vec{p}_k\| + \|\vec{p}_j - \vec{p}_k\| \geq \|\vec{p}_i - \vec{p}_j\|$
2. Regarding Fig. 3.1, it is obvious that $|\theta_i - \theta_k| + |\theta_j - \theta_k| = |\theta_i - \theta_j|$ if the angle θ_k is on the shorter arc, and $|\theta_i - \theta_k| + |\theta_j - \theta_k| > |\theta_i - \theta_j|$ otherwise. Therefore, the inequality $|\theta_i - \theta_k| + |\theta_j - \theta_k| \geq |\theta_i - \theta_j|$ holds.
3. It is obvious that if $a_1 + b_1 \geq c_1$ and $a_2 + b_2 \geq c_2$, then $\max(a_1, a_2) + \max(b_1, b_2) \geq \max(c_1, c_2)$. So, the inequality $d_{i,k} + d_{j,k} \geq d_{i,j}$ holds.

□

3.2.2 Directed Circle Expansion

For directed circle expansion, the subroutine Expand(v_i) in Algorithm 2 calculates not only the center point and radius of a child circle, but also the orientation. As illustrated in Fig. 3.2, the black parent circle expands blue child circles with uniformly interpolated center points. The orientation of a child circle is defined by a vector aligning with the parent and child center points. Regarding the orientation of the parent circle as the thick arrow, the direction of a vector is selected for a smaller angle difference in $[0, \pi/2]$, which is indicated by the arrows. Two circles are created on both sides at $\pi/2$ with opposite directions.

The driving direction is determined by the orientation and relative position between the adjacent circles. If the vector from the parent center point to the child center point has the same

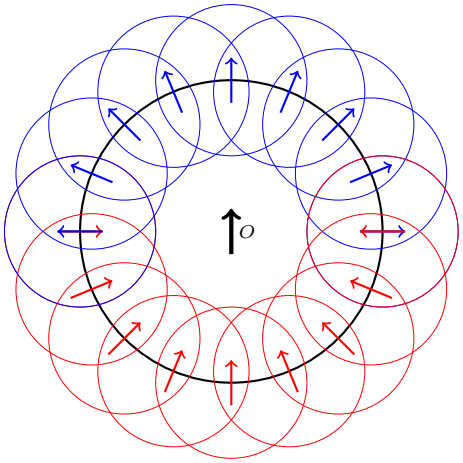


Figure 3.2: Orientation-aware circle expansion

orientation to the child circle, forward driving is preferred to reach the child circle. Otherwise, reverse driving is suggested. In Fig. 3.2, the forward driving circles are in blue, while the reverse driving ones are in red. Thus, the holonomic robot reaches the points on the front half circle with forward motion and the back half with reverse motion in order to reduce the steering cost.

Furthermore, when the orientation is available, different margins can be applied in the longitudinal and lateral directions to determine the circle radius. As the motion of a car-like robot is mostly performed along the longitudinal direction, it may take a larger distance margin than in the lateral direction. In this case, a directed circle can provide more precise information about free space dimension for the specific driving direction.

3.2.3 Directed Circle Overlapping

As the circles are generated with orientation, the geometric relationship should also be evaluated including the relative orientation, especially in the subroutine Exist(v_i, V_{closed}) of Algorithm 2. According to the distance function (3.1), all the equal distance points of a directed circle form the surface of a cylinder in a three-dimensional space of (x, y, θ). As illustrated in Fig. 3.3, circle O_0 has a radius r and orientation θ_0. If k is the motion curvature, i.e., the angle change proportional to travel distance, all the states closer than distance r lie in the dashed cylinder centered at O_0 with a radius r and a height $2rk$. Thus, each directed circle can be represented with a cylinder in the three-dimensional space.

The radius of the cylinder is the same as the circle, while the height is proportional to the circle radius regarding the motion curvature. A cylinder explores the free space by expanding from the center point until the projected circle reaches an obstacle. Similar as the circle-based exploration, each point inside a cylinder can be treated as a source for a child cylinder. In this case, if a selected cylinder is centered inside an evaluated cylinder from the closed set, it is redundant as it is inside the explored region of the existed ones. In Fig. 3.3, O_1 is redundant to O_0, as the point O_1 is inside the cylinder of O_0. In contrast, O_2 is not redundant, despite its center point is inside the circle O_0 when projected to the x-y plane. As a result, more circles are evaluated by the orientation-aware space exploration when the travel direction is considered.

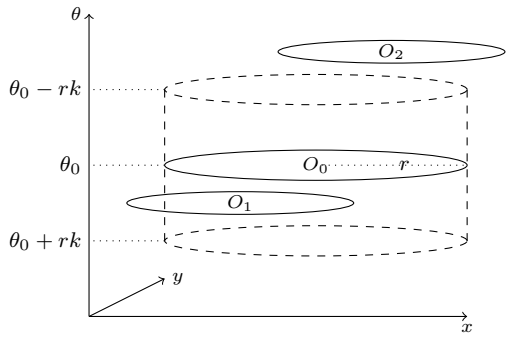


Figure 3.3: Directed circles overlapping

3.2.4 Example Scenario

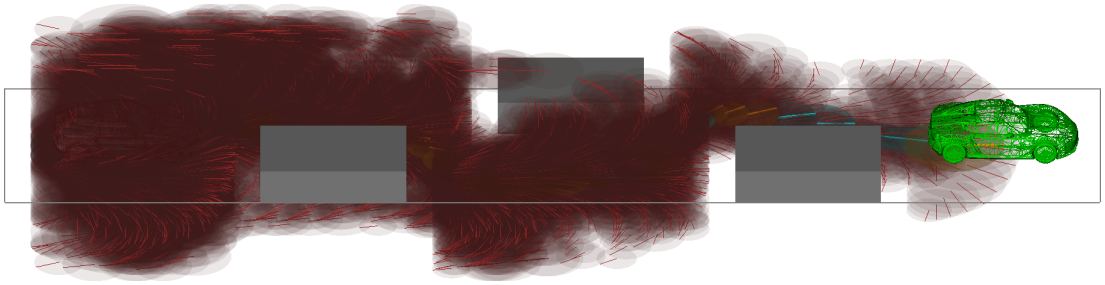
An example scenario of orientation-aware space exploration is presented in Fig. 3.4. A vehicle is required to maneuvering through three obstacles to reach the goal position. The progress of the orientation-aware space exploration is demonstrated in Fig. 3.4(a). The circles are drawn in a three-dimensional space with the third coordinate for orientation. The orientation of a directed circle is indicated with a line segment from the center point. The brown circles are the search vertices evaluated during the exploration. The flows of the circle directions can be observed in the side view. The exploration result is shown in Fig. 3.4(b).

The basic circle-based space exploration of SEHS is presented as a reference in Fig. 3.4(c). It is obvious that the baseline space exploration takes fewer circles to find a solution. However, the path corridor from SEHS has abrupt direction changes, while the one from the orientation-aware version is more convenient for steering behavior.

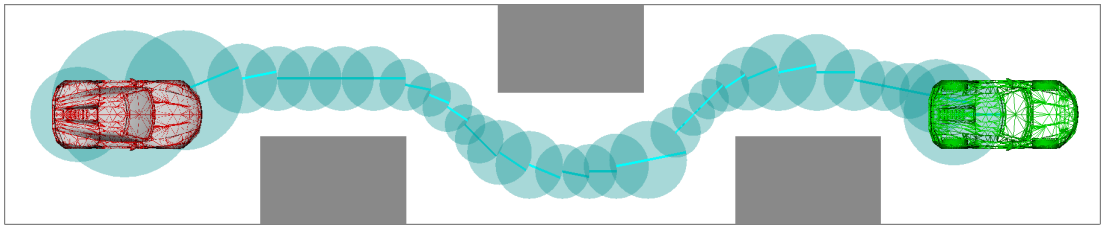
3.3 Orientation-Aware Heuristic Search

As discussed in the Section 2.3.3, SEHS conducts its global search progress following the space heuristic and adapts local state expansions with primitive motions. The orientation-aware heuristic search improves the algorithm both in the global heuristic guidance and the local motion selections for maneuvering problems.

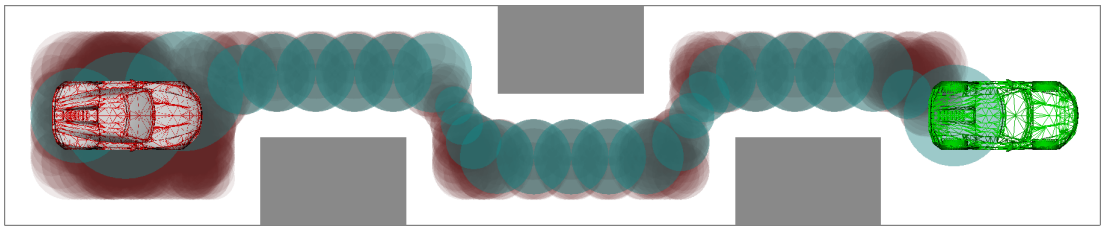
The result of the orientation-aware space exploration is a sequence of directed circles. As the space exploration applies a distance metric concerning orientation, the path circles are waypoints in a three-dimensional space for position and orientation providing the guidance about where to proceed and how to move with a recommended driving direction. A preprocess is performed to the circle-path in order to identify the possible driving directions and the maneuvering regions. Fig. 3.5 shows the different conditions of the driving directions. O_i and O_{i+1} are two adjacent circles. The arrow in the center indicates the circle direction. If the direction arrow of the previous circle points to the next one and the orientation difference is small, the circle (blue) is specified for forward driving as in Fig. 3.5(a). If the direction arrow points away from the next one with a small orientation difference, the circle (red) prefers reverse driving as in Fig. 3.5(b). In case of a large orientation difference, maneuvering is required in the circles (orange) in Fig. 3.5(c) and Fig. 3.5(d). The threshold for orientation difference is calculated as the product of the circle radius and the maximum motion curvature. It is assumed that a robot can change the orientation within the travel distance if the difference is smaller than the threshold, i.e., no maneuvering is required. Thus, the path circles are divided in three



(a) Orientation-aware space exploration progress (side view)



(b) Orientation-aware space exploration result (bird view)



(c) Space exploration of SEHS (bird view)

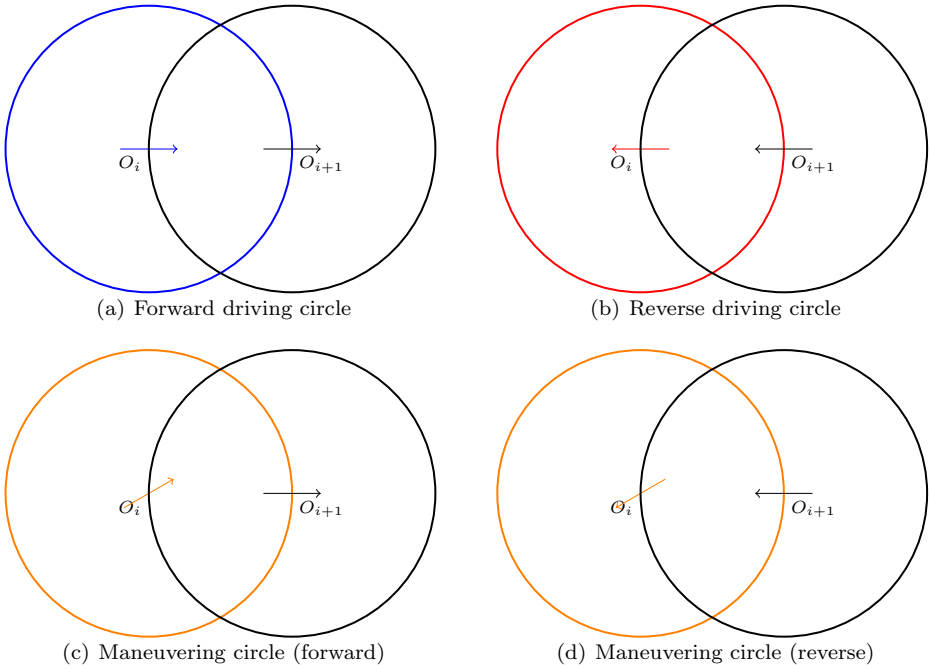
Figure 3.4: Space exploration of SEHS and OSEHS methods

types with driving directions: *Forward*, *Reverse* and *Bidirectional*.

The purpose of this pre-process is extracting the orientation knowledge for different primitive motions and cost functions in different situations. The general process of the orientation-aware heuristic search remains the same as Algorithm 4. States at the same position with different angles are treated as different vertices in order to find the one with the best orientation regarding the directed circle-path. By mapping a state to the closest directed circle, the primitive motions can be adapted to encourage the expansion in the most promising direction. Details are discussed in the following subsections.

3.3.1 Directed Circle-Path Heuristic

The heuristic estimation based on the directed circle-path concerns not only the position of the robot, but also the orientation regarding the circle directions. The distance metric 3.1 is applied to calculate the distance along the circle-path, as well as the distance between a state and a directed circle for state clustering. The heuristic cost consists of the distance to the next circle and the distance along the rest circle-path to the goal as introduced in Section 2.3.3.1. As proved in the previous section, this distance estimation is optimistic, i.e., it is a lower bound of the travel distance for a car-like robot to follow the target position and orientation. As a

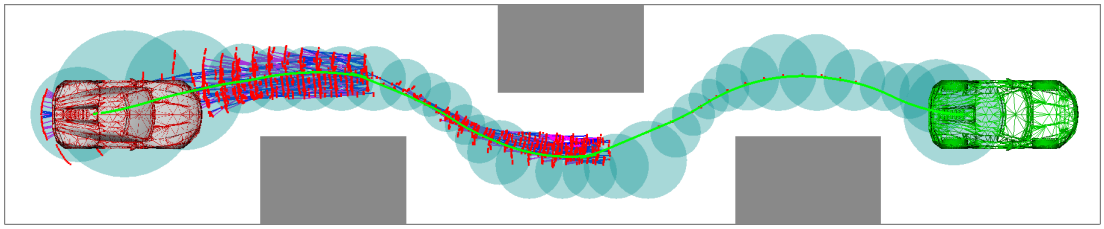


(a) Forward driving circle (b) Reverse driving circle

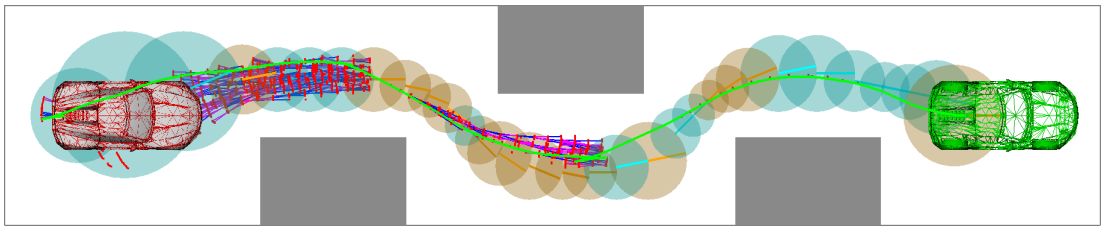
(c) Maneuvering circle (forward) (d) Maneuvering circle (reverse)

Figure 3.5: Circle driving directions

result, the heuristic search returns an optimal motion following the directed circle-path.



(a) Search with the circle-path heuristic of SEHS method (16885 vertices)



(b) Search with the directed circle-path heuristic of OSEHS method (10209 vertices)

Figure 3.6: Search with different heuristics of SEHS and OSEHS methods.

Fig. 3.6 compares the search results with the different heuristics based on the same circle-path from SEHS and OSEHS. The circle-path is generated by the same orientation-aware space

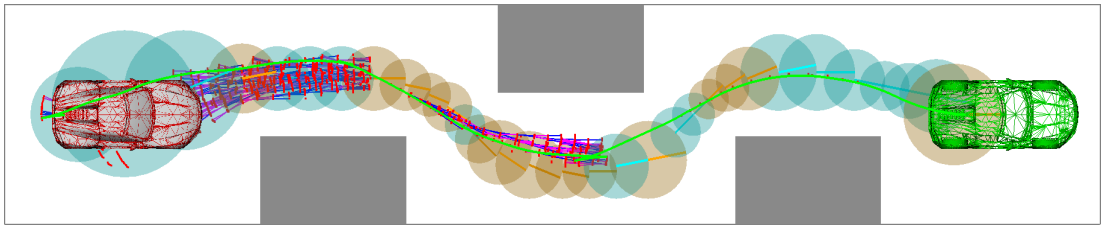
exploration procedure. The planner in Fig. 3.6(a) treats the directed circles as normal SEHS circles, so that only the Euclidean distance is calculated for the heuristic costs. The planner in Fig. 3.6(b) applies the orientation-aware heuristic estimation, in order to priories the states with better orientation during the search. The results from the two methods are similar with minor difference caused by the different distance metric. OSEHS solves the problem with 40% fewer vertices than SEHS.

3.3.2 Motion Cost Adaptation

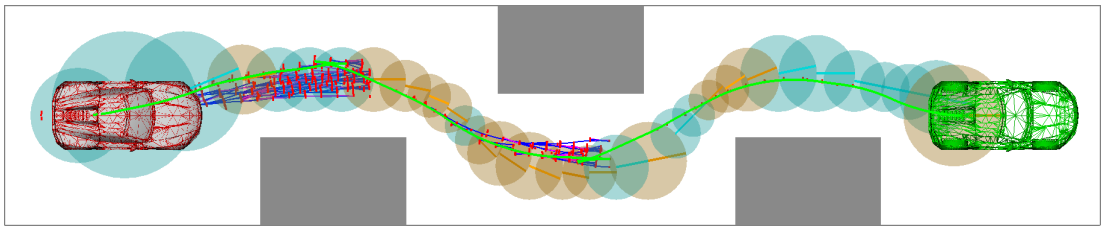
The directed circle-path not only provides a better distance heuristic, but also contains the information about driving directions, which can be exploited in vertex expansion. One option is to punish the motion in the other direction with a higher cost.

In this case, an extra cost value is added to the SEHS cost function (2.10) as (3.2). λ is a positive cost coefficient for moving in the opposite direction. s_{opposite} is the length of the motion in the other direction. Thus, in a reverse driving circle a reverse motion takes no additional cost, but a forward motion costs more. In a maneuvering circle, forward driving and reverse driving are treated equally with no additional cost. Furthermore, in a maneuvering circle the cups factor γ in (2.10) can be reduced to lift the maneuvering behavior. The completeness of the search algorithm is not harmed, because the planner can still traverse every possible vertex. The new cost function only sorts the vertices in a different order for the maneuvering scenario.

$$g^*_{\text{cost}} = g_{\text{cost}} + \lambda \times s_{\text{opposite}} \tag{3.2}$$



(a) Search with the normal cost function (10209 vertices)



(b) Search with the adapted cost function (6188 vertices)

Figure 3.7: Search with different cost functions.

Fig. 3.7 compares the search result with different cost functions. The planner in Fig. 3.7(a) applies the normal SEHS cost function (2.10) with a cusp factor 0.5. The cost function in Fig. 3.7(b) is (3.2) with an opposite coefficient 1.0 and a cusp factor 0.0 for states in maneuvering circles. As a result, the planner with the adapted cost function produces a similar solution with 40% fewer vertices.

3.3.3 Motion Direction Adaptation

A more direct way to apply the orientation knowledge is adapting the primitive motions according to the circle driving direction. In this case, if a state is mapped to a forward driving circle, only forward motions are evaluated to expand the child states. Similarly, a reverse driving circle only allows reverse motions. Moving in both directions is then accepted only in a maneuvering circle. This method may affect the completeness of the search algorithm as it deliberately ignores some branches of the search tree. However, in practical it is more efficient than the cost function adaptation, which introduces additional cost that increases the difference between the actual cost and the heuristic estimation. In contrast, the primitive motions adaptation prunes the search tree directly, so that fewer vertices are generated.

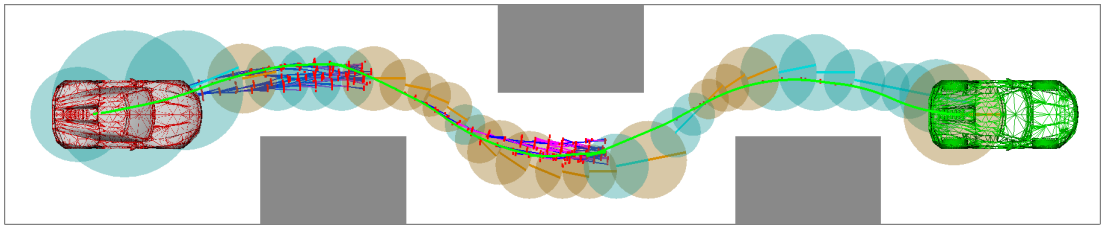


Figure 3.8: Search with adapted motion primitives (5329 vertices)

Fig. 3.8 shows the search result with the adapted motion primitives. The cost function remains unchanged as (2.10) from SEHS. In this scenario, it takes further 15% fewer vertices than the version with modified cost function.

3.4 Experiments

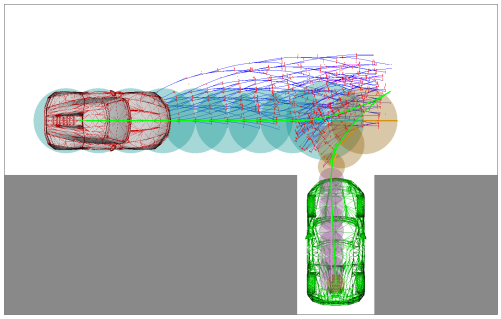
Three scenarios are evaluated in the experiment section including cross parking, parallel parking, and a general maneuvering problem. The OSEHS method is compared with three other algorithms: the sampling-based RRT method [34], the Hybrid A* search algorithm [51], and the general SEHS approach [102]. All these four planners are implemented in C++ and executed on a desktop machine running Linux with an Intel Core i7 2.90 GHz CPU and 8 GB RAM.

As parking or maneuvering is performed in low speed, motion dynamic is barely involved. A robot can even stop and then change the moving direction. In this case the kinematic model of constant curvature in Section 2.3.2.1 is applied by the motion planning methods. The maximum motion curvature is $0.2\,\mathrm{m}^{-1}$. The valid curvature range $[-0.2, 0.2]$ is interpolated for the primitive motions with different travel distances. The SEHS and OSEHS methods determine the motion step size by multiplying the circle radius with a step factor. They also adapt the state resolution with the a parameter. The Hybrid A* algorithm takes a set of primitive motions with a constant step size. Its grid resolution is also constant in (x, y, θ) coordinates. The same set of atomic control inputs is evaluated by the RRT method to extend the random tree, which bias the goal direction in a period of 10 iterations.

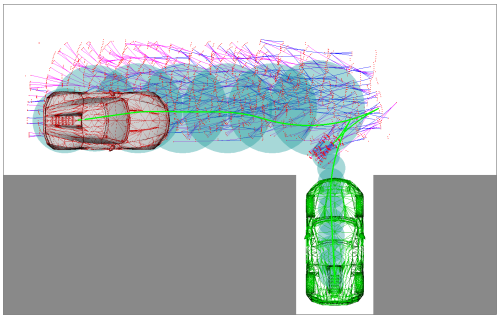
The performance of the methods is measured with the results from 100 trials. Each experiment is conducted with a random start pose sampled in a region of $(\pm 0.5\,\mathrm{m}, \pm 0.5\,\mathrm{m}, \pm 5^\circ)$ for (x, y, θ).

3.4.1 Cross Parking Scenario

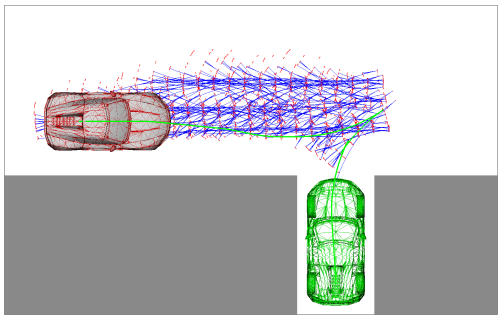
In the cross parking scenario, a vehicle is going to park into a perpendicular parking lot with 25 cm space margin to both left and right sides. According to the goal pose, the vehicle should maneuver backwards into the parking lot.



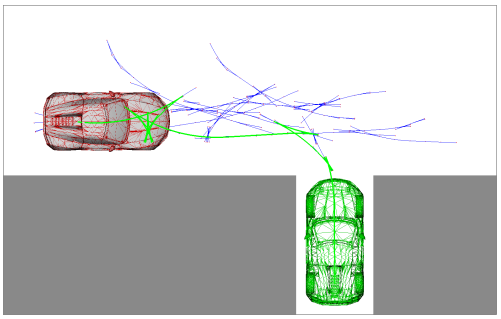
(a) OSEHS



(b) SEHS



(c) Hybrid A*



(d) RRT

Figure 3.9: Cross parking scenario.

In this experiment, the step and resolution factor of the OSEHS and SEHS methods are both 0.5. The Hybrid A* search employs a grid with 0.5 m resolution in (x, y) coordinates and 0.1 rad for orientation. The constant step size of the primitive motions is 0.5 m in Hybrid A* and RRT. An example result from each of the four methods is demonstrated in Fig. 3.9. The quantitative results from the four planning algorithms are listed in Table 3.1. Similar as the SEHS experiments, OSEHS has two rows for the exploration phase and search phase respectively. In addition, the successful rate is presented in case some algorithm cannot solve all the random problems.

The result motions from the three search-based methods are similar in geometric forms. With the guidance from the orientation-aware space exploration, the OSEHS algorithm decides to perform the parking maneuver in three segments: forwards driving (cyan circles), maneuvering (orange circles), and reverse driving (pink circles) in Fig 3.9(a). Thus, the heuristic search can adapt the cost and motion primitives accordingly to concentrate search effort in the maneuvering regions. In contrast, the other two search methods evaluate a lot more states exhaustively along the whole parking procedure as in Fig. 3.9(b) and Fig. 3.9(c). The solution from RRT in Fig. 3.9(d) is the longest with many cusp motions, which greatly differs from the human parking behavior.

The OSEHS algorithm has the best overall time performance with the fewest vertices. The

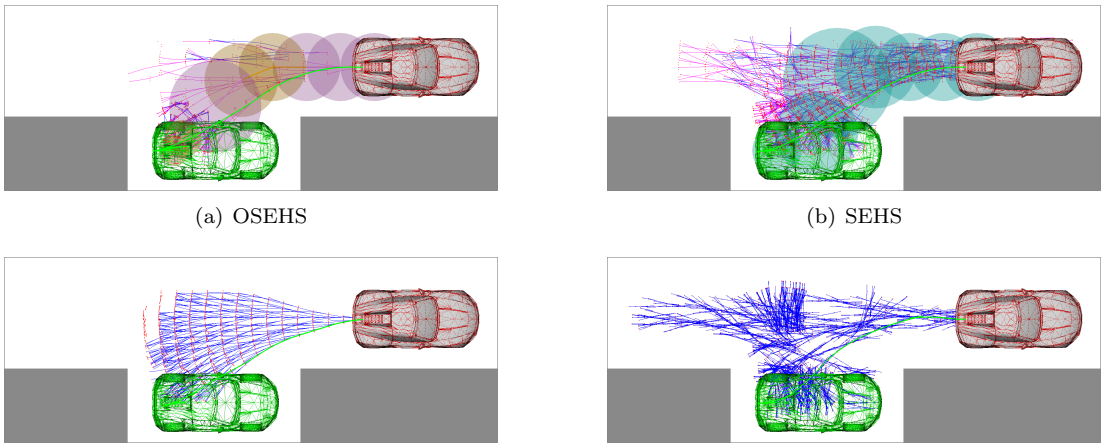
Table 3.1: Results of the cross parking scenario

Planner	Success (%)	States	Queries	Time (ms)	Time STD (ms)
$\text{OSEHS}_{explore}$	100	3 117	4 147	18.81	11.64
OSEHS_{search}	100	4 221	37 202	60.50	56.51
$\text{SEHS}_{explore}$	100	377	557	0.64	0.48
SEHS_{search}	100	6 627	60 489	91.79	44.80
Hybrid A*	100	11 729	78 127	221.46	29.60
RRT	100	1 668	69 816	162.71	228.03

orientation-aware space exploration has more overhead than the normal one in SEHS, but the orientation knowledge brings more benefit in the search phase. The SEHS method performs better than Hybrid A* search because the step size and resolution adaptation saves much effort around the large circles, while Hybrid A* should choose a small step size and resolution for the narrow parking lot. The RRT method is faster than Hybrid A* as there is large free space for the random samples to find a convenient internal pose to back into the parking space. However, the motions planned by RRT diverse greatly in the path length, the number of cusp motions, and planning time due to the random samples.

3.4.2 Parallel Parking Scenario

In the parallel parking scenario, a vehicle is going to park into a parallel parking lot with 70 cm margin to the front and the back.



(a) OSEHS (b) SEHS

(c) Hybrid A* (d) RRT

Figure 3.10: Parallel parking scenario.

In this experiment, the OSEHS and SEHS methods take the same parameters as in the previous one. The step size and resolution are automatically reduced when no solution is found. However, in order to achieve a better successful rate, the constant grid resolution of the Hybrid A* search is reduced to 0.2 m in (x, y) coordinates and the motion step size is reduced to 0.1 m. An example result from each of the four methods is demonstrated in Fig. 3.10. The statistic results from the four planning algorithms are compared in Table 3.2.

Similar solutions are observed from all the four planning methods. The orientation-aware space exploration of OSEHS suggests driving backwards with two maneuvering regions in Fig. 3.10(a). One is in the middle while backing into the parking space and the other is around the final position. The heuristic search finds a solution without direction change in the first maneuvering region, but multiple cusp motions in the second, where most search activities is performed. The other three methods evaluate a larger area around the whole parking procedure with much denser states in Fig. 3.10(b), Fig. 3.10(c), and Fig. 3.10(d).

Table 3.2: Results of the parallel parking scenario

Planner	Success (%)	States	Queries	Time (ms)	Time STD (ms)
$\text{OSEHS}_{explore}$	100	843	1 657	6.61	3.07
OSEHS_{search}	100	7 468	36 934	234.24	737.91
$\text{SEHS}_{explore}$	100	120	164	0.16	0.48
SEHS_{search}	100	13 911	93 685	661.68	2 128.93
Hybrid A*	92	44 909	109 773	645.08	656.39
RRT	64	51 415	1 561 558	13 811.86	105 383.96

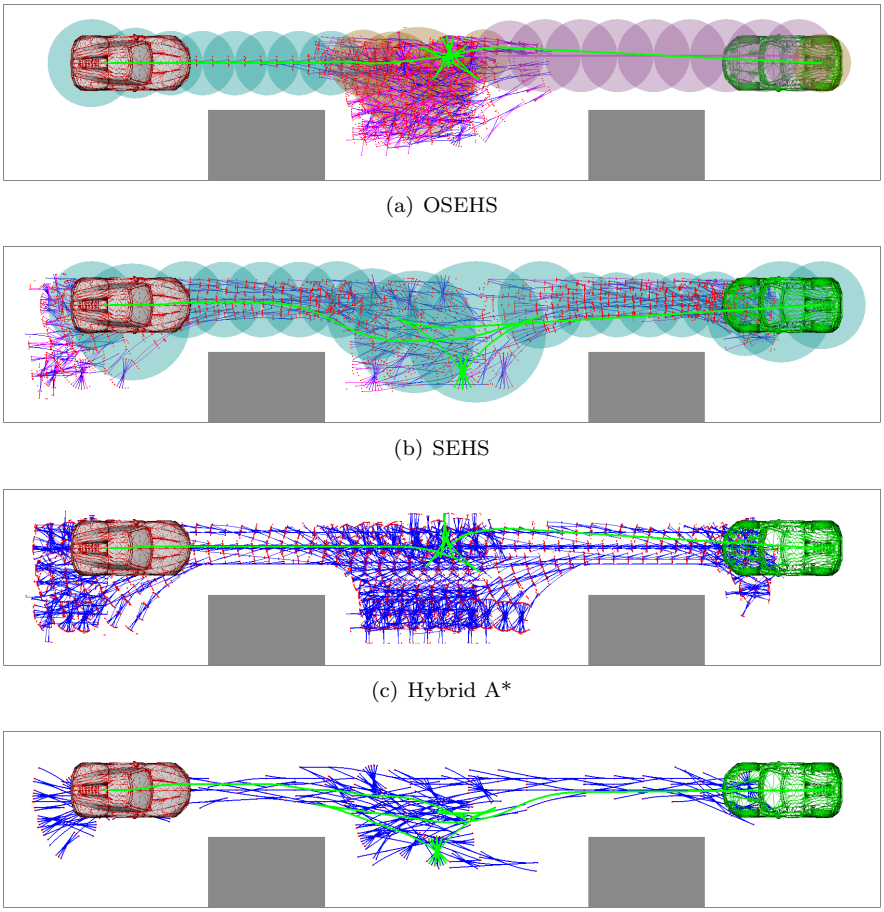
The OSEHS method still shows the best time performance in this scenario. The advantage of the SEHS algorithm against the Hybrid A* search is lost as the circle-path heuristic contributes less information than the Reeds-Shepp heuristic of Hybrid A* at the end. Unlike the first scenario that requires a direction change in the middle of the maneuver, this one only needs maneuvering around the final position. Therefore, Hybrid A* can follow the grid-based heuristic with reverse motions until it reaches the neighborhood of the final position, where the Reeds-Shepp metric provides a good estimation for the last a few steps of the motion. However, even with a reduced grid resolution and step size, the Hybrid A* search can only solve 92 of the total 100 random problems. The planning time varies greatly of the three search algorithms, because the complexity of the problem differs greatly for the different initial and final orientations generated with random. The success rate of RRT falls to 64% with a limitation of maximum 100000 vertices. As the final maneuvering is very close to the obstacles, it creates a narrow passage in the configuration space, which is extremely difficult for RRT to solve.

3.4.3 Maneuvering Through Narrow Space

The third scenario is maneuvering through a narrow passage. At the same time, the vehicle should turn-around for the goal pose with an opposite orientation to the start configuration. The required 180° turn is only possible in the middle of the maneuver path.

The same parameters as in the first experiment are applied for the four planning algorithms. An example result from each of the four methods is demonstrated in Fig. 3.11. The average results from the four planning algorithms are compared in Table 3.3.

In Fig. 3.11(a), the space exploration with directed circles returns a path corridor with a motion pattern that drives forwards at the beginning, moves backwards at the end, and makes the turn in the middle where the free-space is relatively large. Then, the OSEHS algorithm investigates most of the search effort in the middle maneuvering region to achieve the 180° turn. After that, the reverse driving motion is quite simple to plan. In contrast, the SEHS and Hybrid A* methods expand the search tree almost over the whole workspace as in the previous scenarios in Fig. 3.11(b) and Fig. 3.11(c). The results from the three search-based methods are slightly different. The Hybrid A* obtains the best solution among the three, as it performs a complete search based on the grid discretization. The OSEHS approach ignores



(a) OSEHS

(b) SEHS

(c) Hybrid A*

(d) RRT

Figure 3.11: Maneuvering with turn-around scenario.

some possibilities due to the primitive motion adaptation. The SEHS returns a longer motion, as it takes large steps for the large circles in the middle. The RRT is efficient in this scenario as there is naturally large free space in the middle for a random tuning motion as in Fig. 3.11(d). However, the path from RRT still contains more direction changes and unnecessary motions than the optimal solution.

Table 3.3: Results of the maneuvering with turn-around scenario

Planner	Success (%)	States	Queries	Time (ms)	Time STD (ms)
$\text{OSEHS}_{explore}$	100	2 419	4 814	26.11	6.33
OSEHS_{search}	100	8 011	79 902	148.55	207.14
$\text{SEHS}_{explore}$	100	228	403	0.58	0.57
SEHS_{search}	100	16 582	195 525	322.92	100.78
Hybrid A*	100	32 848	259 213	725.33	29.84
RRT	100	1 260	78 263	186.97	752.83

The OSEHS approach is the fastest one in this scenario. The RRT method takes the second place, as the maneuvering is performed in the largest free space, where the possibility is largest for random samples. However, the random sampling causes a great deviation in time performance among the different trials. The Hybrid A* search takes the longest planning time, because the grid-based heuristic does not considers the orientation difference between the start and goal states. As a result, it should evaluate a large number of states to compensate the discrepancy between the heuristic cost and the actual cost. However, its search schema provides a robust time performance for the different start and goal conditions.The SEHS method faces the same problem of a suboptimal heuristic for orientation, but it expands fewer vertices thanks to the step size and resolution adaptation.

3.5 Summary

The *Orientation-Aware Space Exploration Guided Heuristic Search* evaluates not only the free space geometry but also the driving directions during the space exploration phase to provide better heuristics for parking and maneuvering tasks. The heuristic search benefits from the orientation knowledge in the cost function design and the primitive motion adaptation. The results of the example scenarios show that it converges faster than the general SEHS method and other state of the art algorithms, especially when the other planners do not know where to perform the maneuvering.

The OSEHS method mainly deals with the low speed driving problems, where a robot can change its steering without considering the dynamic effects. The planning is carried out in a static environment without complex models for the moving obstacles, as the robot can stop almost immediately to avoid collision. In order to achieve stable performance in different maneuvering scenarios, the OSEHS algorithm can be integrated in an automated driving system as a general solution for parking and maneuvering. In this case, no extra module is required to decide the context for a dedicated path planning algorithm for a certain parking or maneuvering task.

Chapter 4

Space-Time Exploration Guided Heuristic Search

In Chapter 2, the *Space Exploration Guided Heuristic Search* approach performs a circle-based space exploration in the two-dimensional workspace with the assumption that the environment is static. However, in many driving scenarios, e.g. lane following, lane switching, overtaking, the movements of other objects plays an important role in the motion planning problem. The change of the environment could even be so rapid that it is almost impossible to adapt the motion with re-planning in an updated static world in real-time. In this chapter, the SEHS method is extended to *Space Time Exploration Guided Heuristic Search* (STEHS) that explores the three-dimensional space-time for motion planning in dynamic environments. The general idea and details of the STEHS method are presented in the following sections with evaluations in several scenarios.

4.1 General Idea

The space exploration procedure of SEHS relaxes the constraints of the car-like kinematics and the robot geometry to a holonomic point robot, so that it is convenient to explore the workspace with circles. In this case, the radius of a free space circle can be easily calculated as the point distance to the nearest obstacle in a static environment. If moving obstacles are involved in the problem, a common strategy is to plan in a snapshot of the dynamic environment by modeling a temporary obstacle position or the sweeping area of an obstacle in a certain time duration as a static object. The future environment changes are handled with re-planning, which could be carried out in an incremental manner to reduce the planning complexity. However, in some occasions, this approach may face inevitable collisions that cannot be detected when ignoring the obstacle motion. Furthermore, trade-off should be made when treating a moving object as its sweeping area, because it is an over approximation of the obstacle. If the selected time duration is too long, the artificial obstacle may be too large that little space remains for robot motion, while a short duration results in frequent re-planning and danger of inevitable collisions. Another solution is direct planning in dynamic environments. In this case, the movement of an obstacle should be known in advance, i.e. the obstacle is tracked by the sensors or it communicates its intentions to the planning agent. In this case, a dynamic environment model is considered in the first place in order to plan a valid motion to a time horizon.

In order to extract the space knowledge in a dynamic environment, the *Space Time Explo-*

ration Guided Heuristic Search method extends the workspace exploration in the time domain. It considers the speed of the point robot to explore the workspace with moving obstacles. The path corridor is constructed in the three-dimensional space and time to guide the heuristic search. The kinodynamic vehicle models, such as introduced in Section 2.3.2.3 or 2.3.2.4, are used in heuristic search, so that the velocity and acceleration are considered in the state expansion. Thus, each vertex obtains a timestamp, which enables collision checks with dynamic objects. The details about the STEHS algorithm are introduced in the following sections.

4.2 Cylinder-Based Space-Time Exploration

Unlike the explicit representation of a static obstacle in the two-dimensional workspace, a dynamic obstacle in the three-dimensional space including time is implicitly defined by its motion parameters. It can be modelled as a pillar leaning to the moving direction. Due to uncertainties, the outer contour may be inflated regarding time. As a result, it is sophisticated to model the whole dynamic environment in a way that a combinatorial geometric method is able to investigate the entire space and time.

The space exploration approach of SEHS provides the possibility to obtain the space knowledge incrementally in a complex environment. In this case, the exploration should be performed in the three-dimensional space. As a circle is a good geometry to investigate the free space in (x, y) dimensions, it is extended to the time domain t as a cylinder for space-time exploration. Thus, the local free space-time is captured with cylinders, while the global free space is evaluated with a tree of cylinders. The space-time exploration is also designed as the general graph search procedure with heuristics as Algorithm 2 in SEHS by replacing the circles with cylinders. The subroutines for vertex expansion and overlapping are adapted accordingly. The exploration result is a cylinder-path that does not only suggest where to go, but also when to move.

In the previous chapter, the cylinders are also used by OSEHS method to resolve the moving directions. However, the radius of the cylinder is first determined in a static environment in OSEHS, and the orientation is calculated later. In the space-time exploration, the time condition should be considered just when acquiring the free space dimension. As a result, the height and the radius of a cylinder should be calculated at the same time.

4.2.1 Heuristic Time Cost

The space exploration in SEHS takes the Euclidean distance as a heuristic in space exploration, so that an optimal path corridor regarding travel distance and safety margin could be found. In space-time exploration, the metric of time cost should be applied instead. The main reason is that as the search vertices are resolved in the three-dimensional space, a new vertex can be created at the same location at a different time. If the distance heuristic is applied, the heuristic search will not terminate because there could always be a vertex in the open-set with a better estimated total cost. This problem is solved when switching to the time cost. Thus, if the algorithm reaches the goal, an upper bound of the total travel duration is found. As each new vertex takes a nonzero additional time cost, the heuristic search procedure converges when the cost of every new vertex exceeds the already achieved best result. In this case, the space-time exploration returns a path corridor with the optimal travel time regarding the embodied safety distance guarantee of cylinders.

The time cost estimation is obtained by dividing the Euclidean distance $\|p_1 - p_2\|$ with a desired speed v as $\|p_1 - p_2\|/v$. Because the speed is positive, the monotonic condition still holds for the time heuristic. If a robot can move faster than the desired speed, the time estimation is not admissible anymore. However, if the maximum speed is used in heuristic estimation

and the robot can dramatically change the speed during the motion, the heuristic may be too optimistic. One extreme case is with an infinite speed, the heuristic cost approaches zero, so the heuristic search degenerates to exhaustive search. Regarding these concerns, the desired speed should be selected wisely so that a balance between result quality and exploration time is achieved. It is also possible to select different desired speed in different regions for a complex scenario. In this case, the heuristic estimation can be made by dynamic programming, which composes the total time cost with the estimations from the sub-regions.

4.2.2 Cylinder Expansion and Overlapping

A vertex in the space-time exploration algorithm consists of $(p, r, t, \Delta t, g, h)$. t is the start timestamp and Δt is the time duration. Thus, a circle in SEHS becomes a cylinder in STEHS in the time slot $[t, t + \Delta t]$.

Assuming a point robot moves with a constant velocity v, the reachable set in a time duration Δt is a cone with base radius $r = v\Delta t$ in the three-dimensional space of (x, y, t). As illustrated in Fig. 4.1, the reachable set from point O_0 is a cone starts from the point O_0 with a height of Δt and a base radius of r. Same as the circle-based exploration, every point inside a cone can be a starting point for a child cone. In this example, a new cone is created from the point O_1 on the top of the previous one. If the timestamp at O_0 is t_0, the time at O_1 is $t_0 + \Delta t$.

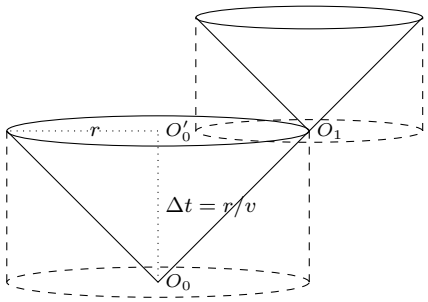


Figure 4.1: Cylinder-based space-time exploration

In order to explore the free space, a cone grows from a single point until it reaches an obstacle within a safety margin. However, it is sophisticated to determine the base radius with a time-dependent collision test or distance query. In order to simplify this procedure, a cone is approximated with a cylinder in the dashed lines in Fig. 4.1. A collision-free distance d_t is first acquired at the starting time t. Then, an initial time duration is calculated assuming the robot moves d_t with the desired velocity v as $\Delta t = d_t/v$. After that, another obstacle distance $d_{t,\Delta t}$ is calculated for the time-slot $[t, t + \Delta t]$. If the distance $d_{t,\Delta t}$ is no less than d_t, d_t is the valid base radius of the cylinder. Otherwise, $d_{t,\Delta t}$ is the cylinder radius and the time duration is adapted to $d_{t,\Delta t}/v$. Obviously, a cone is bounded by the according cylinder, i.e., a cylinder is an overestimation of the total reachable set. Therefore, this approximation does not harm the completeness of the exploration method.

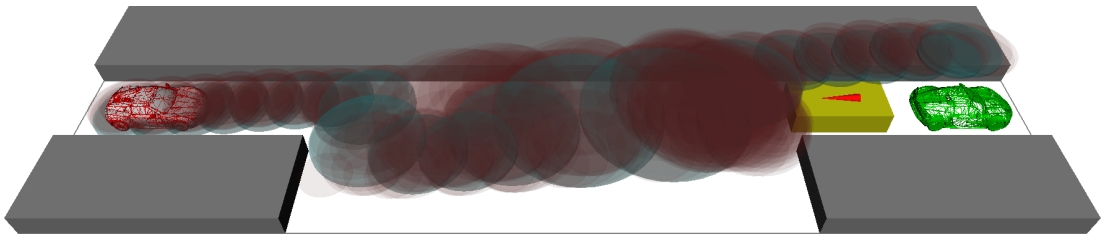
The vertex expansion in Algorithm 3 are performed with cylinders. Child cylinders are created on top of the parent cylinder. The starting points are obtained by interpolating the circle on the top. In addition, a cylinder is created at the circle center to extend the parent cylinder in the time domain. Thus, a robot can wait at the same location for a better occasion to move. The h-value for heuristic estimation is calculated regarding the distance between the starting point and the goal. The g-value for actual cost is obtained with the sum of the time

durations, i.e., the height, of the cylinders.

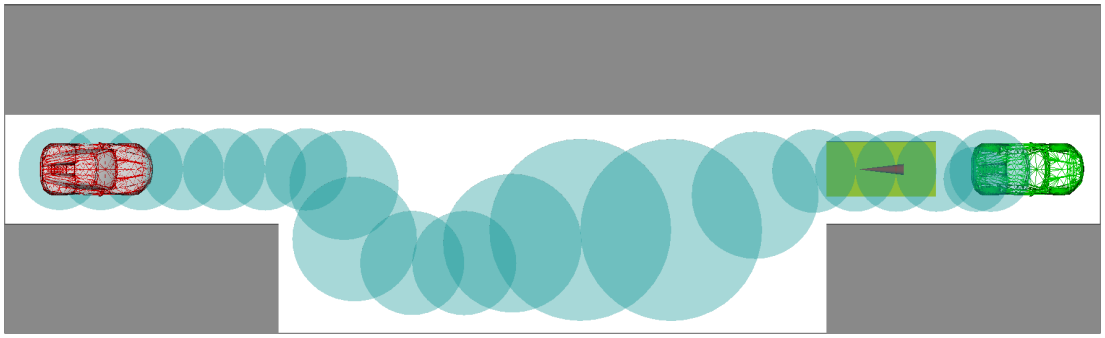
Overlapping is identified when two cylinders intersect both in space and time. As the propagation of cylinders is similar as the circle-based space exploration, each point in a cylinder can be a source for another cylinder. Therefore, if a new cylinder starts inside another cylinder from the closed-set other than its parent, it can be ignored as the reachable set is redundant.

The space-time exploration can also be done in an incremental manner based on the space exploration result from the static environment. The path circles are first converted to space-time cylinders with the time-dependent radius calculation. If a cylinder is smaller than a certain minimum size, it is removed from the path. The gap is then filled with the result from a local space-time exploration. In this case, the space-time exploration is only necessary when there is a moving obstacle blocking the path. Otherwise, the incremental space-time exploration has the same complexity as the circle-based space exploration. A drawback is that the incremental approach only fixes the path corridor locally, which may miss a global optimal solution.

4.2.3 Example Scenario



(a) Space-time exploration progress (side view)



(b) Space-time space exploration result (bird view)

Figure 4.2: Space-time exploration

Fig. 4.2 shows the space-time exploration in a dynamic scenario. A vehicle is driving in a narrow street, where another vehicle (the yellow rectangle with a red cone showing the driving direction) is coming from the opposite direction. There is extra space in the middle where the vehicle can go in order to let the other pass. With the red vehicle frame as the start point and the green one as the goal position, the space-time exploration expands the brown cylinders, which approximate the total reachable set in the three dimension space as in Fig. 4.2(a). The result is a cyan cylinder path in the reachable set. From the bird view in Fig. 4.2(b), it is clear that the vehicle should first move into the free space on the right and then steering back to reach the goal.

4.3 Cylinder Path Guided Heuristic Search

The result of the space-time exploration is a path corridor represented as a sequence of cylinders in the three-dimensional space including time. Then, a primitive motion based heuristic search is conducted to build a search graph following this path as Algorithm 4. Kinodynamic vehicle models are applied to expand states with timestamps for the collision verification in the dynamic environment. As a result, the graph vertex contains a time variable in addition as $(\vec{q}, t, S, g, h)$. The cost estimation and motion adaptation both concern a time metric. Details are discussed in the following subsections

4.3.1 Cylinder-Path Heuristic

The heuristic estimation of the search procedure in STEHS is based on the cylinder-path. The same as in the SEHS algorithm, a state is mapped to the nearest cylinder. The cost estimation consists of two parts: the cost to reach the nearest cylinder and the rest time cost along the cylinder-path to the goal.

In STEHS, the distance between a state and a cylinder should not only consider the space distance, but also the time distance. The space distance is calculated by projecting the state and the cylinder to the two-dimensional Cartesian space. It is the same as the distance between the state point and the circle center (4.1). The time distance is the difference between the state timestamp and the cylinder time frame. If the timestamp is inside the cylinder time frame, the time distance is zero. Otherwise, it is the time difference between them (4.2). It is also possible to treat the timestamp before and after the cylinder time slot differently, so that reaching a waypoint too early or too late have different effects. The maximum of the two kinds of distance is the combined distance (4.3) as the first part of the heuristic estimation. The space distance is divided by the same desired velocity v as in the space-time exploration, so that all the costs are measured with time.

The distance between the cylinders can be directly obtained as the difference between the timestamps. Thus, the heuristic provides an estimation of the time cost to reach the goal through the cylinder path corridor with the desired velocity. It is possible that the vehicle accelerates to a faster speed, so that it can reach the goal with an actual cost less than the estimation. In this case, the admissible condition of this heuristic is weak with additional velocity constraint.

$$d_{\text{space}}(\vec{q}, c) = \|(x_{\vec{q}}, y_{\vec{q}}) - (x_c, y_c)\| . \tag{4.1}$$

$$d_{\text{time}}(\vec{q}, c) = \begin{cases} t_c - t_{\vec{q}} & \text{if } t_{\vec{q}} < t_c \\ t_{\vec{q}} - (t_c + \Delta t_c) & \text{if } t_{\vec{q}} > (t_c + \Delta t_c) \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

$$d_{\text{combined}}(\vec{q}, c) = \max(\frac{d_{\text{space}}(\vec{q}, c)}{v}, d_{\text{time}}(\vec{q}, c)) \tag{4.3}$$

4.3.2 State Expansion and Resolution

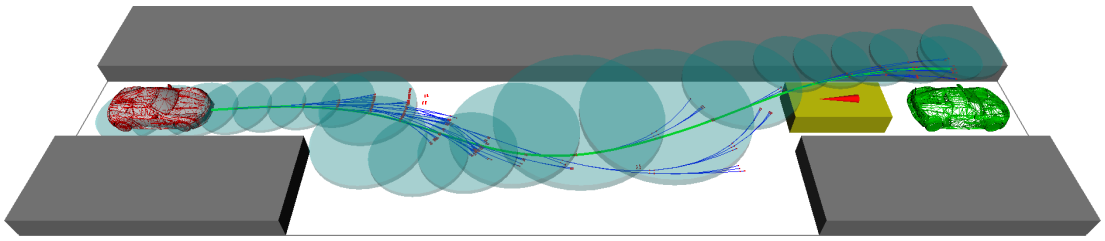
The states are created with primitive motions. Regarding the control inputs of a kinodynamic model, the length of a motion is measured with time. The timestamp of a child state is calculated by add the motion duration to the timestamp of the parent state. In this case, the

actual cost g can be obtained as the difference between the timestamps of the current state and the start state. When the starting time is 0, this g-value is identical with the state timestamp.

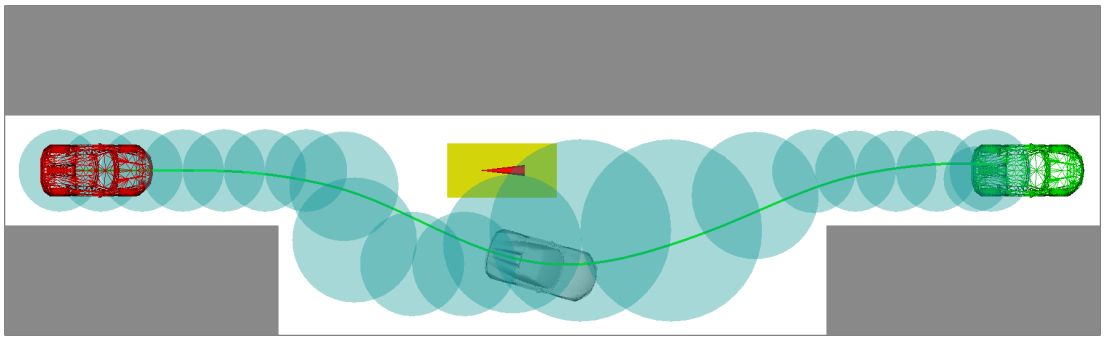
Regarding the state timestamp and motion duration, it is possible to perform collision checks in dynamic environments. Similar as in a static environment, a state expansion is verified at the interpolated points along the primitive motion. The collision check can also be done in an incremental way, that first examines the static obstacles. If no collision is found, the dynamic obstacles are verified with their sweeping areas. When the sweeping area collides, the detail motion is checked with the dynamic obstacle at the intermediate states. Thus, the average overhead of dynamic collision check is reduced.

The step size of a primitive motion is adapted to the cylinder time duration, which considers the cylinder base radius and the desired velocity. The state resolution is decided in the same way in time domain. In this case, besides the distance metric (2.9), the time difference $|t_i - t_j|$ is also relevant. If these two values are smaller than certain thresholds r_{distance} and r_{time} respectively, one state is redundant according to the resolution. r_{distance} takes a factor to the cylinder base radius, while r_{time} is proportional to the time duration.

4.3.3 Example Scenario



(a) Space-time exploration heuristic search progress (side view)



(b) Space-time exploration heuristic search result (bird view)

Figure 4.3: Space-time exploration heuristic search

Fig. 4.3 shows the heuristic search result based on the cylinder-path from the example in the previous section. A kinodynamic vehicle model is applied with a constant velocity identical to the desired velocity for space time exploration. Motion primitives are selected with different steering parameters and durations. The search graph grows following the path cylinders in the three dimensional space in Fig. 4.3(a). The result motion pursues the centers of the cylinders and forms a natural driving maneuver with large safety distance in the bird view Fig. 4.3(b).

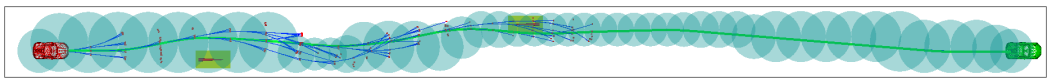
4.4 Experiments

In this section, the STEHS method is evaluated in two dynamic scenarios and compared with two search-based approaches, SEHS and Hybrid A*, as well as the random-sampling RRT method. All the planners apply the same implementation of a kinodynamic vehicle model and an environment model with collision check functions from the *AutoDrive* library. The SEHS algorithm performs the space exploration only in the static environment without the moving obstacles. The Hybrid A* calculates the grid-based heuristic also without considering the dynamic objects. In these two methods, the collision check with dynamic obstacles is done later during the states expansion. The performance is measured with 100 trials for each method with random start positions in a region of $1.0\,\mathrm{m} \times 1.0\,\mathrm{m}$. The experiments are performed on a PC with an Intel Core i7 2.90 GHz processor and 8 GB RAM running a Linux operating system.

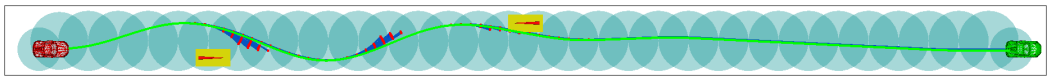
4.4.1 Overtaking Scenario

The first example is an overtaking scenario in Fig. 4.4. A vehicle is driving on a straight road. In front, a slow obstacle moves on the same side of the road in the same direction. The obstacle speed is $10\,\mathrm{m\,s^{-1}}$, while the ego vehicle is driving at $15\,\mathrm{m\,s^{-1}}$. In order to quickly reach the goal, the ego vehicle should perform an overtaking maneuver. At the same time, another obstacle is approaching from the opposite direction on the other side of the road. In this case, a planner should plan an overtaking motion without colliding with the two obstacles.

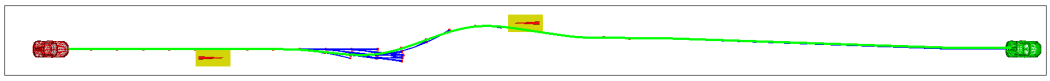
The kinodynamic model has a maximum speed of $20\,\mathrm{m\,s^{-1}}$ and can accelerate or decelerate with an maximum value of $1.0\,\mathrm{m\,s^{-2}}$. The maximum motion curvature is $0.2\,\mathrm{m^{-1}}$ and can be changed with a steering parameter of $0.2\,\mathrm{m^{-1}\,s}$. The integral step is 0.005 s for the forwards dynamic calculation. The set of motion primitives is constructed with the combinations of 3 different acceleration values and 3 different steering parameters. In Hybrid A*, a $0.5\,\mathrm{m} \times 0.5\,\mathrm{m}$ grid is applied for the heuristic estimation and states resolution. SEHS takes a step factor 0.33 and a resolution factor 0.06. STEHS takes a step factor 1.0 and a resolution factor 0.1, as it considers a time metric instead of a distance metric.



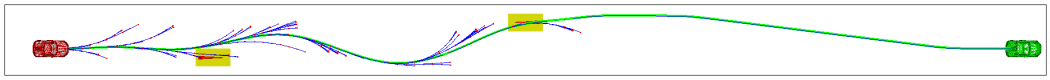
(a) STEHS result



(b) SEHS result



(c) Hybrid A* result



(d) RRT result

Figure 4.4: Overtaking scenario

Fig. 4.4 compares example results of the four methods. The cyan circles show the exploration results from STEHS and SEHS. The blue lines show the planning progress with red dots as the expanded states. The green line represents the result motion. From the qualitative aspect, the STEHS algorithm constructs a search graph following the cylinder-path in Fig. 4.4(a), which suggests that the ego vehicle first stays on the right, then goes to the left and performs the overtaking maneuver. The SEHS method does not extract the knowledge about the moving obstacle by the space exploration. As a result, the suboptimal heuristic causes the vehicle to perform an "S" form movement to avoid collisions in Fig. 4.4(b). The Hybrid-A* takes a grid-based heuristic, which result in a straight motion at the beginning, followed by an abrupt steering motion for the overtaking in Fig. 4.4(c). The result motion is dangerous with a little safety distance to the obstacle. The RRT algorithm in Fig. 4.4(d) generates a tree with random motions. Its result is also suboptimal.

Table 4.1: Results of the overtaking scenario

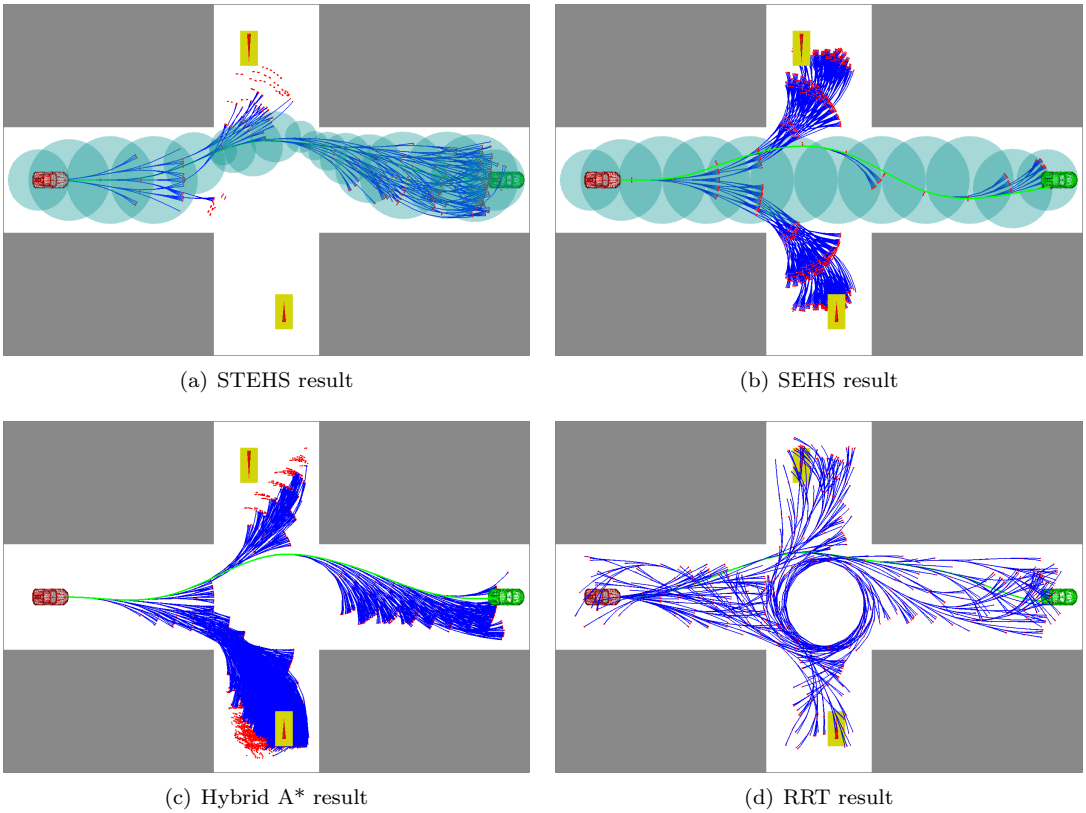
Planner	Success (%)	States	Queries	Time (ms)	Time STD (ms)
$\text{STEHS}_{explore}$	100	593	1 005	2.17	1.71
STEHS_{search}	100	694	3 710	15.37	6.18
$\text{SEHS}_{explore}$	100	628	1 156	1.28	0.60
SEHS_{search}	100	2 281	7 857	28.53	14.12
Hybrid A*	100	2 968	26 646	119.69	170.90
RRT	100	666	50 368	172.53	514.97

Table 4.1 lists the quantitative simulation results from the different algorithms with the successful rate, the number of states, the number of collision tests, the planning time, and its standard deviation. The STEHS and SEHS methods have two rows for the exploration phase and search phase respectively. The numbers in the exploration row are the counts of circles or cylinders and the distance queries instead.

In average, the STEHS method takes only half of the total planning time compared to the SEHS method, although the space-time exploration spends longer time than the space exploration of SEHS. As the space-time exploration provides better heuristics for the dynamic scenario, more time is saved in the search phase. The SEHS performs better than Hybrid A* in planning time, as the incremental adaptation of the search step and resolution requires less vertices to compensate the suboptimal heuristic estimation. In contrast, the Hybrid A* works with a constant step and constant grid resolution, which requires 0.5 m in x and y coordinates to solve all the random generated problems. It also suffers from a large deviation of the planning time, because the time window for overtaking is so small that occasionally hard to be hit with a static step size. The result from the random-sampling based RRT varies greatly in planning duration and path shape. The two moving obstacle create a difficult narrow passage problem for RRT to solve.

4.4.2 Intersection Scenario

The second example takes place in an intersection as in Fig. 4.5. A vehicle is going to move across the intersection to reach the goal position, while two obstacles are coming through the intersection at the same time on the crossing road. The speed of the obstacles is configured to cause a collision in the middle of the intersection if the ego vehicle does not change its moving direction. A planner should plan a motion that avoids the collision in the middle. The speed of one obstacle is $10\,\mathrm{m\,s^{-1}}$ and the other is $7\,\mathrm{m\,s^{-1}}$. The vehicle moves at a speed of $15\,\mathrm{m\,s^{-1}}$.



(a) STEHS result

(b) SEHS result

(c) Hybrid A* result

(d) RRT result

Figure 4.5: Intersection scenario

Fig. 4.5 compares qualitatively the example results from the four planners. The space-time exploration result of STEHS in Fig. 4.5(a) indicates a passage without colliding with the two obstacles. The ego vehicle should first steer to the left and then turn right to go through the intersection. The search algorithm follows this heuristic and solves the problem without wasting efforts in other possible directions. In contrast, the space exploration result of SEHS in Fig. 4.5(b) does not contain this knowledge, so the planner expands lots of states in the both left and right directions to find the result. The same phenomenon can be observed in Fig.4.5(c) for Hybrid A*. As it takes a constant step size and resolution, even more states are created one both sides. The RRT planner in Fig. 4.5(d) proceeds with random samples, which even creates circular motions in the middle of the intersection.

Table 4.2 lists the statistic numbers of 100 trials with the four algorithms. The ranking is similar as in the first scenario. The STEHS has the best time performance, followed by SEHS with about double as long in planning time. The time deviation of STEHS is larger in this scenario. As the space-time exploration does not consider the vehicle kinematics, a suboptimal path corridor may be found and the vehicle is unable to follow it due to the nonholonomic and differential constraints. The Hybrid A* and RRT cannot solve all the random problems. In order to achieve the successful rate, the Hybrid A* need to plan with a smaller step size and resolution as 0.3 m for x and y coordinates. The RRT approach requires longer time in average to expand a vertex for the overhead to find valid random samples.

Table 4.2: Results of the intersection scenario

Planner	Success (%)	States	Queries	Time (ms)	Time STD (ms)
$\text{STEHS}_{explore}$	100	833	1 280	2.83	1.94
STEHS_{search}	100	4 228	27 096	113.38	107.23
$\text{SEHS}_{explore}$	100	410	527	0.60	0.57
SEHS_{search}	100	4 460	39 264	225.35	391.47
Hybrid A*	96	25 855	209 909	1 089.57	890.84
RRT	99	4 676	179 460	1 807.40	3 152.33

4.5 Summary

The *Space Time Exploration Guided Heuristic Search* algorithm combines time-dependent space exploration in workspace with heuristic search in configuration space to solve the motion planning problem for car-like robots in dynamic environments. This method improves the SEHS approach with time modeling in space exploration. The space-time exploration creates a path corridor regarding the desired velocity of the robot and the time dynamics of the environment. The search procedure follows this time-dependent heuristic with kinodynamic motion primitives to generate the final solution.

The experiment results show that STEHS is more suitable than the basic SEHS in dynamic scenarios with high driving speed. It can be applied for the scenarios such as lane following, overtaking, driving through intersections, etc., where moving obstacles are involved.

Chapter 5

SEHS Motion Planning Engine with Task Planning

In the previous chapters, the baseline version and two extensions of *Space Exploration Guided Heuristic Search* method are introduced for motion planning with general purpose, parking and maneuvering, as well as dynamic scenarios. However, these algorithms alone are not sufficient to build a planning agent for daily autonomous driving. In practical autonomous driving scenarios, motion planning is not a single-shot fire-and-forget task but a continuous process throughout the whole driving period. In addition, online motion planning, trajectory verification regarding the real-time perception, and re-planning or motion adaptation should be addressed by the planning agent. Furthermore, the context of an autonomous driving task cannot be simply modelled with obstacles and free space as the examples in the previous sections. It contains other domain specific entities and semantic information, which requires reasoning and task planning with domain knowledge to extract the right motion planning problems and select the right planning methods. In this case, a real-time *Motion Planning Engine* is introduced in this chapter with a hybrid planning architecture, which integrates task planning and motion planning for highly automated driving applications. Several advanced automated driving scenarios are demonstrated with this integrated planning system.

5.1 System Architecture

Different types of information are relevant for a highly automated driving application. Some are obtained in prior, such as roadmap and kinematics of the vehicle; some are updated in real-time from sensing and communicating. Some are discrete, symbolic and semantic, such as traffic signals and traffic rules; some are continuous, e.g., obstacle movements and vehicle trajectories. The idea of a hierarchical planning architecture is to deal with different kinds of information in different system levels with suitable planning algorithms. Thus, the high-level planning can provide useful knowledge for the low-level planning, which reduces the planning complexity. At the same time, the low-level planning is able to give feedback to the high-level planning, so that the global planning strategy can be adapted.

A hybrid planning architecture is proposed in Fig. 5.1. The rectangle components in the middle are the intelligent agents planning routes, tasks, and motions in different layers. The ellipse units on the left are the perception modules providing real-time traffic information such as roadmaps, traffic signals, and objects lists. In addition, a set of driving tasks are defined

concerning traffic rules for the task planning. Motion planning problems can be derived from the specific driving tasks.

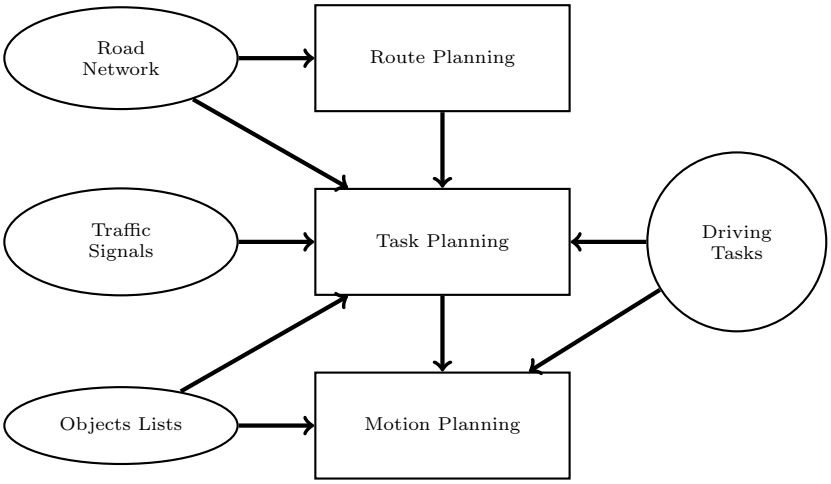


Figure 5.1: Hybrid planning system architecture

The route planning is a graph-based navigation agent, which plans an optimal route based on the prior-known road network with a graph search algorithm. The result serves as a global heuristic for the task planning in the next level. A route can be divided with several way-points as a list of targets for the task planning. If the task planner finds out that a target is unreachable, the route planner can suggest an alternative solution based on the feedback. Thus, the route planning layer deals mainly with the prior discrete topological information about the road network.

The task planning is a symbolic planner that considers the traffic rules and plans a sequence of driving tasks to reach the way-points along the route. The system state of task planning including the ego vehicle and the environment is transformed from the continuous configuration space to a discrete state space. A planning problem is solved by searching for an optimal task sequence to reach the goal conditions in the symbolic space. The task planning layer deals with both continuous states and discrete constraints, such as space topology, place occupation, and traffic rules. It bridges the gap between high-level navigation and low-level motion planning.

The motion planning solves the final planning problem for the specific driving tasks. According to the driving task provided by the previous layer, a dedicated planning method is selected with a suitable scope of the continuous problem space. For example, a lane following task should be carried out only inside the lane, which can be handled with a compact behavior-based approach. The motion planner can also give feedback to the task planner if a task cannot be executed. In this case, the driving maneuver is re-planned in the task level. The motion planning layers deals mainly with continuous planning problems regarding vehicle kinematics and obstacles.

This hybrid planning system integrates the high-level route planning and the low-level motion planning with domain specific task planning. Such a hierarchical decision making schema is analogous to the behavior of a human driver: while having a rough route direction in mind, a human driver usually makes discrete decisions, such as overtaking, lane switching, or turning based on the actual traffic conditions. Then, these driving tasks are performed with continuous steering, accelerating, or braking commands. If the situation is not clear, certain behaviors are

performed to acquire additional information or negotiate with the traffic to resolve the uncertainties. If a task cannot be accomplished as planned, the following actions or even the route can be adapted.

5.2 Driving Task Planning

The research work of task planning, reasoning, and problem solving can be traced back to the very beginning of artificial intelligence, and boomed in robotics domain. Fig. 5.2 shows the general concept of combining task planning in symbolic state space and motion planning in continuous configuration space. The plane below is the continuous configuration space for motion planning with robot configurations as red dots. The plane above is the discrete state space for symbolic planning with system states as green dots. The dashed blue lines show the mapping between the two spaces. First, the start q_0 and the goal q_4 are mapped to the symbolic states as s_0 and s_6, where a sequence of tasks is found connecting the states $\{s_0, s_1, s_3, s_4, s_6\}$. Then, the symbolic states are projected back to the configuration space as $\{q_0, q_1, q_2, q_3, q_4\}$, and concrete motions are planned. Thus, the complex problem about the motion from q_0 to q_4 is decomposed into sub-problems through the internal waypoints q_1, q_2, and q_3.

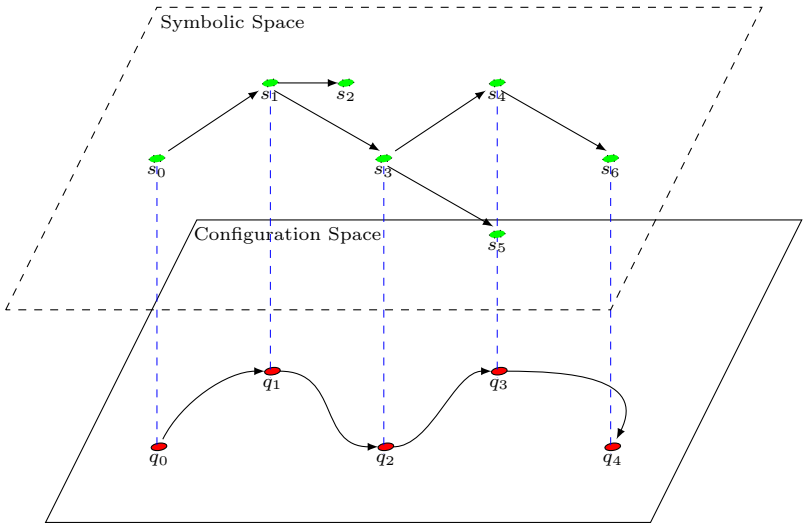


Figure 5.2: Combined task planning and motion planning.

In essence, the task planning approach for automotive domain is based on three necessary design principles that cannot be achieved by conventional state machine or signal-based architectures.

- **Symbolic-geometric hierarchy**: The whole problem is first processed in the symbolic space with limited discrete system states. Then, the tasks are passed to the motion planner as milestones, and a sequence of motions are planned to perform these tasks. Thus, a large complicated problem is divided into sub-problems, which can be solved more efficiently with specific methods in adequate scopes.

- **Easy to verify**: All symbolic domain description is organized as modules that can be verified individually and independently from the planning algorithm.

- **Generic AI planning**: Traffic rules and semantic information allow and require automatic planning and fully integrate into the task planning concept. Years of research in artificial intelligence have brought powerful task planning methods that are increasingly applied to robotics and can also be applied to automated driving.

In the following subsection, a compact domain definition for driving task planning is declared. Then, a general search-based task planning method is introduced, followed by several example scenarios to show the advantages of this approach.

5.2.1 Domain Definition

The problem domain for autonomous driving is defined with a tuple $\{\mathcal{W}, \mathcal{S}, \mathcal{P}, \mathcal{O}\}$. $\mathcal{W}$ is the continuous world model, which consists of the vehicle and environment states, such as vehicle speed, object position, place geometry, etc. $\mathcal{S}$ is the symbolic state space, which holds the discrete and semantic information, e.g., parking lot occupation, lane type, etc. A set of predicates $\mathcal{P}$ defines a projection from the continuous world $\mathcal{W}$ to the discrete world $\mathcal{S}$ by verifying certain propositions. $\mathcal{O}$ is a set of operations, which have effects on the world state.

A task can be defined as a function in Definition 3. The definition domain is determined by a set of predicates as the preconditions of the task. The value range is defined by the effects of the task, i.e., a set of operations.

Definition 3. *Task t: A task is a function from a set of inputs X that satisfies the predicates P_t as preconditions and produces the outputs Y by applying a set of operations O_t to the inputs.*

$$\begin{aligned} t :& X \to Y \\ & p(x) = \text{True}, x \in X, p \in P_t \\ & y = O_t(x), x \in X, y \in Y \end{aligned}$$

A simple version of domain definition is outlined in the following subsections as the world model, a predicates set, and a list of tasks.

5.2.1.1 World Model

A world model consists of *Places* and *Objects*. A place P is a particular area where a vehicle performs its motion, e.g., a lane, a parking lot. An object O is a traffic entity, e.g., a vehicle, a pedestrian, or a traffic signal. Fig. 5.3 is an example of a world model. Places l_0, l_1 and l_2 are three lanes with the driving direction from right to left. Places j_0 and j_1 are junctions at the both ends of the lanes. The axis below shows a lane coordinate starting from 0 m to 1 000 m. Object v is a vehicle at the location $(l_1, 240)$, i.e., on the lane l_1 at distance 240 m. Place p is a parking lot at the location $(l_0, 600)$. Object s is a traffic sign at the end of the three lanes $(l_0/l_1/l_2, 960)$. Thus, the topology between the places and objects is defined, so that a planner can reason about the relative positions.

A place in the world model has a unique *ID*, a *type* T, and attributes for other properties. It keeps a list of neighboring objects and a map of adjacent places. The place definition is mainly designed for the space topology information, while the detail geometry of a place is considered later in the motion planning problem. Three basic types of places are introduced as follows:

- **Lane (L)**: A lane can have a *subtype* as urban street, highway, motorway, etc. Further attributes are *speed limit* and *driving direction*. It has the access to the neighboring lanes

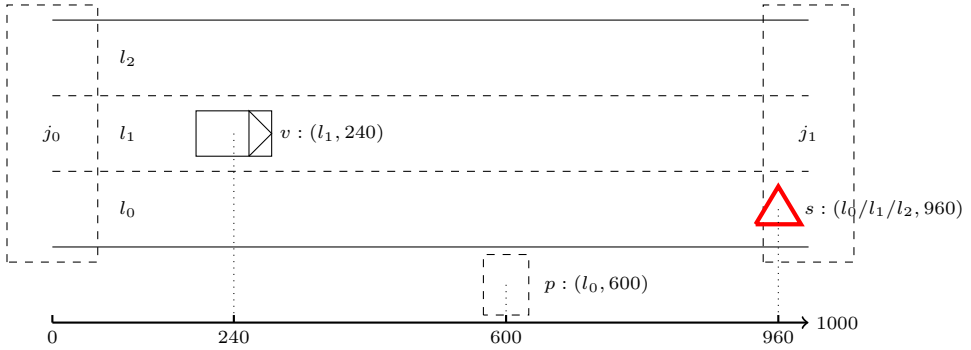


Figure 5.3: Example of world model

or the adjacent junctions. Vehicles, traffic signals, and parking lots are registered to a lane in different lists. They are located with a distance value d in the lane coordinate as (L, d). Finally, a lane has a certain geometric shape, defined as a reference path with width information.

- **Junction (J)**: A junction is a special type of lane, which holds the adjacency information of multiple inbound and outbound lanes as *connections*. The connections can have different *priorities* and *permissions* according to the traffic rules and signals. In this case, the planner can find out the region, where a vehicle is allowed to travel when driving through a junction from one lane to another.
- **Parking Lot (P)**: A parking lot is a hybrid of place and object, which can locate a vehicle and be located in a lane coordinate. A flag is set when it is occupied. Furthermore, it keeps a list of accessible lanes for parking and pull-out maneuver.

An object holds a unique *ID* and attributes such as *type*, *position*, and *speed*. One object can be mapped to multiple places, e.g., a traffic light can be referred by several lanes of the same road. The state of a dynamic object can be predicted according to its current state. Two types of objects are introduced as examples.

- **Vehicle (V)**: The state of a vehicle contains its location, speed, and *driving behaviors* such as light signals. A vehicle is a dynamic object.
- **Traffic Signal (S)**: Traffic signals are lane markers, traffic signs, and traffic lights. A traffic sign is mostly static, while a traffic light is usually dynamic. Regarding the current state and the phase period, it is possible to predict the light state in future.

5.2.1.2 Predicates

A predicate is a proposition about the world state. It can take arguments as predefined constants or variables from the run-time. The value of a predicate is a trilean: *True*, *False*, or *Unknown*. If the information is not sufficient to verify a proposition, the value of the predicate is *Unknown*. In this case, the decision about a predicate can be "soft" as a possibility value between 0.0 and 1.0. A predicate can be a combination of several sub-predicates. The following are some examples.

1. $At(O, T)$ takes the value *True* if object O is at a place of a specific type T.

2. $At(O, P)$ takes the value *True* if object O is located at place P.

3. $Before(O, L, d)$ takes the value *True* if object O is on lane L and before the location (L, d). It has a sub-predicate $At(O, P)$.

4. $In(O, L, d_0, d_1)$ takes the value *True* if object O is inside the coordinate range $[d_0, d_1]$ of lane L. It is the combination of $Before(O, L, d_1)$ and $\neg Before(O, L, d_0)$.

5. $Beside(L_1, L_2)$ takes the value *True* if lane L_1 is beside lane L_2.

6. $Left(L_1, L_2)$ takes the value *True* if lane L_1 is left to lane L_2. It has a sub-predicate $Beside(L_1, L_2)$.

7. $Free(L, d_0, d_1, t_0, t_1)$ takes the value *True* when an object can travel from location (L, d_0) to location (L, d_1) in the time duration $[t_0, t_1]$ without collisions. In this case, all the objects in lane L are treated as point objects in order to perform a simple collision test.

8. $Connect(L_1, L_2, J)$ takes the value *True* if lane L_1 and lane L_2 are connected in junction J. L_1 is the inbound lane, while L_2 is the outbound lane.

9. $Clear(L_1, L_2, J)$ takes the value *True* if the connection from lane L_1 to lane L_2 is clear to pass in junction J. This predicate considers both the space clearance and the permission regarding traffic rules. $Connect(L_1, L_2, J)$ is a sub-predicate of it.

5.2.1.3 Tasks

Tasks are defined as primitive actions in the task planning, which are composed of *Preconditions* and *Effects*. The *Preconditions* are a set of predicates which decide whether a task is applicable. The effects of a task are a set of operations that mainly change the state of the ego vehicle. A task can take parameters, which serve as arguments for preconditions and effects. In addition, a task may have a cost value, so that the planner can search for the optimal solution regarding the cost metric. If one of the predicates is *Unknown*, the task is uncertain. The effect of a task can also be nondeterministic. In this case, the planner can deal with the uncertainties and select a most promising solution. In this thesis, only the deterministic task effects are considered to elaborate the task planning framework. The following tasks are defined for the example scenarios.

1. $Park(V, P)$: Vehicle V parks into the parking lot P.
   Preconditions: Vehicle V is on a lane L, which has access to parking lot P. The vehicle coordinate is near the parking lot location d_P within a certain range $[d_P - \Delta d, d_P + \Delta d]$. Parking lot P is free during the parking duration t_{park}. If t_0 is the start time, the final time is $t_p = t_0 + t_{\text{park}}$. The distance range Δd and the time duration t_{park} are parameters of this task.
   $$At(P, L) \wedge In(V, L, d_P - \Delta d, d_P + \Delta d) \wedge Free(P, t_0, t_p)$$
   Effects: Vehicle V parks in parking lot P.
   Cost: The time cost t_{park} to park the vehicle.

2. $PullOut(V, L)$: Vehicle V pulls out from a parking lot to lane L.
   Preconditions: Vehicle V is inside a parking lot P, which is located at lane L. Lane L is free around the parking lot coordinate $[d_P - \Delta d, d_P + \Delta d]$ during the pull out duration

t_{pullout}. If t_0 is the start time, the final time is $t_p = t_0 + t_{\text{pullout}}$. The distance range Δd and the time duration t_{pullout} are parameters of this task.

$$At(V, P) \wedge At(P, L) \wedge Free(L, d_P - \Delta d, d_P + \Delta d, t_0, t_p)$$

Effects: Vehicle V is on lane L at the parking lot coordinate d_P. Parking lot P is free.
Cost: The time cost t_{pullout} to pull out the vehicle.

3. $FollowLane(V, L, d, v)$: Vehicle V follows lane L to coordinate d with speed v.
Preconditions: Vehicle V is on lane L and before coordinate d. Lane L is free at the relevant location during the driving task. The start location and time (L, d_0, t_0) can be obtained directly from the vehicle state. $t_{\text{follow}} = \frac{(d-d_0)}{v}$ is the estimated time to drive from d_0 to d with a constant speed v. The time is $t_d = t_0 + t_{\text{follow}}$ at the final location (L, d). The speed v and the duration t_{follow} are the parameters fo this task.

$$Before(V, L, d) \wedge Free(L, d_0, d, t_0, t_d)$$

Effects: Vehicle V is at the desired coordinate d with the desired speed v at time t_d.
Cost: The time cost t_{follow} of the task.

4. $SwitchLane(V, L_1, L_2)$: vehicle V switches from lane L_1 to lane L_2.
Preconditions: Vehicle V is on lane L_1, which is beside lane L_2. The both lanes are free at the relevant location during the driving task. (L_1, d_0, t_0) is the start location and time. (L_2, d_s, t_s) with $t_s = t_0 + t_{\text{switch}}$ and $d_s = d_0 + v \times t_{\text{switch}}$ is the final location and time. t_{switch} is a constant time cost for lane switching with a constant speed v. The speed v and the duration t_{switch} are the parameters fo this task.

$$\begin{aligned} At(V, L_1) \wedge Beside(L_1, L_2) \\ \wedge Free(L_1, d_0, d_s, t_0, t_s) \\ \wedge Free(L_2, d_0, d_s, t_0, t_s) \end{aligned}$$

Effects: Vehicle V is on lane L_2 at coordinate d_s and time t_s.
Cost: The time cost t_{switch} of the lane switching.

5. $ChangeLane(V, L_1, L_2, J)$: Vehicle V changes from lane L_1 to lane L_2 at junction J.
Preconditions: Vehicle V is close to the end of lane L_1 within a range d_{change}. Lane L_1 and L_2 are connected in junction J and is clear to drive. The last part of lane L_1 and the first part of lane L_2 are free during the lane changing duration t_{change}. l_1 is the length of lane L_1. If t_0 is the start time, the final time is $t_c = t_0 + t_{\text{change}}$. The distance margin d_{change} and the duration t_{change} are the parameters of this task.

$$\begin{aligned} In(V, L_1, l_1 - d_{\text{change}}, l_1) \wedge Clear(L_1, L_2, J) \\ \wedge Free(L_1, l_1 - d_{\text{change}}, l_1, t_0, t_c) \\ \wedge Free(L_2, 0, d_{\text{change}}, t_0, t_c) \end{aligned}$$

Effects: Vehicle V is on lane L_2 at coordinate d_{change} at time t_c.
Cost: The time cost t_{change} to change the lane.

These tasks are the most basic driving maneuvers concerning only the topology and occupation of lanes and parking lots. Further traffic rules can be applied to these tasks by extending the preconditions and providing extra condition parameters.

5.2.2 Task Planning

According to the domain definition, a task planning problem for automated driving can be defined with a triple $\{s_0, G, \mathcal{T}\}$, of which s_0 is an initial state, G is a set of goal conditions, and $\mathcal{T}$ is a group of relevant tasks. This problem can also be solved with a graph-search approach as Algorithm 1. A vertex of the search graph is (s, g, h) including a world state s, the actual cost g, and the heuristic cost h. An edge is (t, c) with a task t and its cost c, which reaches the current state from a parent vertex. The heuristic cost is estimated based on a route from the navigation layer. Such a heuristic may not always consistent, as it could be faster to travel from A to B via C than a direct connection between A and B due to speed limits, or traffic conditions. As a result, the open set and close set strategy is not sufficient to resolve the redundant states. In this case, additional cost evaluation is required to decide whether a closed state should be revisit with a better g-value. The modified heuristic search algorithm for task planning is formulated as Algorithm 5.

Algorithm 5: TaskPlanning$(s_0, G, \mathcal{T})$

$v_{\text{start}} \leftarrow (s_0, 0, d_{\text{heuristic}}(s_0, G))$;
$f_{\text{goal}} \leftarrow \infty$;
$V_{\text{closed}} \leftarrow \emptyset$;
$V_{\text{open}} \leftarrow \{v_{\text{start}}\}$;
$E \leftarrow \emptyset$;
while $V_{\text{open}} \neq \emptyset$ **do**
    Sort(V_{open});
    $v_i \leftarrow$ PopTop(V_{open});
    if $f_{\text{goal}} < f[v_i]$ **then**
        return success;
    else
        if Exist(v_i, V_{closed}) & $f[v_i] \geq f[v_{\text{closed}}]$ **then**
            continue;
        else
            $\mathcal{T}_i \leftarrow$ ChooseTasks$(\mathcal{T}, s_i)$;
            for each $t_j \in \mathcal{T}_i$ **do**
                if Verify$(s_i,$ PreConditions$(t_j))$ **then**
                    $s_{i,j} \leftarrow$ ExecuteTask(s_i, t_j);
                    $V_{\text{open}} \leftarrow V_{\text{open}} \cup \{(s_{i,j}, g_i + c_j, d_{\text{heuristic}}(s_{i,j}, G))\}$;
                    $E \leftarrow E \cup \{(t_j, c_j)\}$;
            $V_{\text{closed}} \leftarrow V_{\text{closed}} \cup \{v_i\}$;
            if Verify(s_i, G) **then**
                $f_{\text{goal}} = \min(f[v_i], f_{\text{goal}})$;
if $f_{\text{goal}} < \infty$ **then**
    return success;
else
    return failure;

The heuristic search procedure is similar as the ones in the previous chapters. An open set

V_{open} holds the vertices to be evaluated, while a closed set V_{closed} is for the visited vertices. In each iteration, the function $\text{Sort}(V_{\text{open}})$ and $\text{PopTop}(V_{\text{open}})$ select the vertex with the minimum total cost from the open set. If the total cost $f[v_i]$ is larger than the already achieved goal cost $f[v_{\text{closed}}]$, the planning is successful, i.e., an optimal sequence of tasks is found. Otherwise, the vertex v_i is compared with the vertices in the closed set. Only when a duplicated state is found with total cost less than or equal to $f[v_i]$, the vertex v_i can be skipped.

In order to expand states, the function $\text{ChooseTasks}(\mathcal{T}, s)$ chooses a subset of tasks $\mathcal{T}_i$ from $\mathcal{T}$ regarding the world state s. Tasks are mapped to specific kinds of states in order to reduce the effort of checking all the tasks. For instance, a lane-switching task is only possible when a vehicle is in a lane. The task parameters are decided according to the state, e.g., the distance and speed of a lane following task are decided based on the speed limit, traffic condition, and sensing range of the vehicle. If the preconditions of a task are satisfied, the effects of the task are applied to the current state with the function $\text{ExecuteTask}(s, t)$. The actual cost of the new vertex is obtained by accumulating the cost of the tasks. Afterwards, the parent vertex goes to the closed set, while the child vertices are added to the open set. Finally, the goal cost is updated if the state satisfies the goal condition with less total cost.

Due to the sensing and communication limits, the available information may be insufficient to determine the whole task sequence to reach the goal. In this case, an uncertain task is still evaluated by the planner. All the following vertices are labelled as uncertain. Thus, the planner returns a possible solution with several confident tasks at the beginning, with which the motion for the next moment can be generated. In case of uncertain tasks, the planner can continue to create a complete solution assuming each uncertain task can fail. Thus, the autonomous vehicle can surely reach the goal or remain in a safe state in any circumstances.

5.2.3 Example Scenarios

In this section, two task planning scenarios are evaluated: an overtaking scenario on the motorway and a rerouting scenario in a round-about. The former one focuses on the traffic rules, and the latter one deals with uncertainties and re-planning.

5.2.3.1 Overtaking on the Motorway

The overtaking scenario is illustrated in Fig. 5.4. A motorway has three lanes in one direction with a speed limit of $50\,\text{m}\,\text{s}^{-1}$. The three lanes have the same length, and their coordinates are aligned. The ego vehicle (cyan) starts on the right lane l_0 at distance 0 m with a speed of $35\,\text{m}\,\text{s}^{-1}$. There is another vehicle (red) on the middle lane l_1 80 m ahead travelling at $30\,\text{m}\,\text{s}^{-1}$. The goal is to reach the 1 000 m mark on any of the three lanes. According to German traffic rules [122], overtaking from the right is forbidden on a motorway. In addition, a vehicle should drive on the most possible right lane on the motorway. Therefore, the ego vehicle should either stay on the right lane reducing its speed as the states $\{s_0, s_1\}$, or switch to the left lane l_2, overtake the slow vehicle, and switch back to the right lane via the states $\{s_0, s_2, s_3, s_4, s_5\}$. Following the slow vehicle on the middle lane is not allowed, because the right lane is not occupied.

Assuming the sensing range of the vehicle is 100 m on the motorway. Therefore, the distance parameter for a lane following task is 100 m or the distance to the goal location if it is in sight. The lane following speed is the minimum of the speed limit, the speed of the vehicles in front or on the left lane within a certain distance. Thus, a vehicle will not overtake other vehicles from the right or threaten the vehicle in front. An additional precondition for lane following task is that the lane on the right should not be free in a certain look-ahead distance. Otherwise the vehicle must switch to the right lane. Assuming the time cost for a lane-switching task is 5 s.

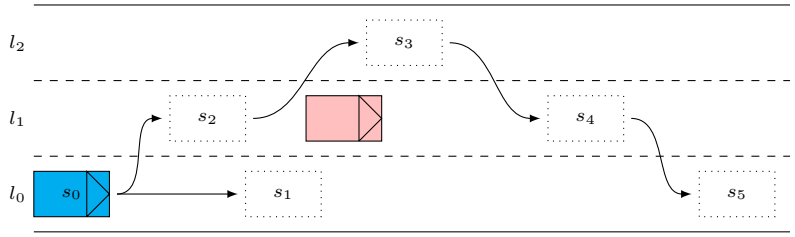


Figure 5.4: Overtaking scenario

The planning result is presented in Fig. 5.5 with the task sequence result in blue. The solution is a complex maneuver that first switches twice to the left lane, overtakes the slow vehicle, then switches back to the right lane, and follows it to the target location.

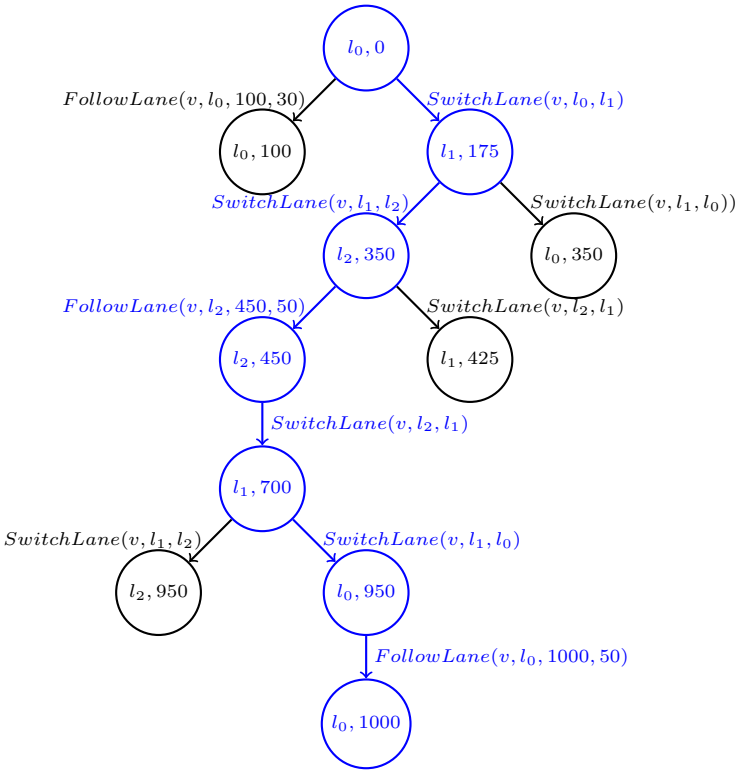


Figure 5.5: Task planning for the overtaking scenario

If the overtaking problem is solved with a state machine approach, each possible transition between the states should be defined regarding the traffic rules and conditions. This state machine can only be applied for the specific kind of scenarios. Each tiny change of the traffic rules or scenario setup may result in large modification of the state machine, e.g., overtaking from the right is allowed in the urban street. In contrast, the task planning approach employs a generic planner to solve the problem. The rules and conditions are defined in the task domain. Therefore, it is more flexible to select the right subset for the specific scenario.

5.2.4 Round-About with Blockage

In the second scenario, the ego vehicle is going to drive through a round-about, where the initially planned route is blocked. The scenario is illustrated in Fig. 5.6. For simplicity, the round-about has only one entrance and three exits. l_0, l_1, l_2 and l_3 are lanes connecting to the round-about with the arrows for the directions. The four lanes in the round-about, l_4, l_5, l_6 and l_7, are each 20 m long and counterclockwise directed. The four junctions, j_0, j_1, j_2 and j_3, connect the adjacent lanes. The speed limit is $10\,\mathrm{m\,s^{-1}}$ in all the lanes. The ego vehicle (blue) is coming from lane l_0 and can choose one of the three exits with different points as the rest time to go. The red traffic sign shows a temporary blockage at l_2. The time cost of a lane-changing task in a junction is 5 s. In this case, the difference of the rest time costs at the three exits are larger than driving around the whole round-about once again.

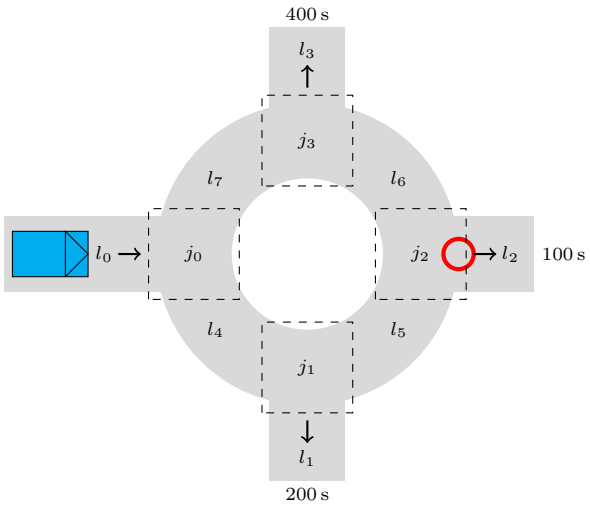


Figure 5.6: Round-about scenario

The sensors of the vehicle can only reach the next junction in this clustered environment, so the blockage can only be detected after the vehicle entering lane l_5. According to the planning result in Fig. 5.7, the exit at lane l_2 is chosen initially for the minimum time cost. After detecting the blockage, the vehicle continues driving in the round-about to the exit at lane l_1 for the second best solution, rather than taking the next exit at lane l_3. The dashed tasks are planned after detecting the task $ChangeLane(v, l_5, l_2, j_2)$ is not executable (red). The final solution is colored in blue.

Alternatively, the planner can assume that each uncertain task may fail at the first place. When the vehicle plans its tasks at the beginning, it has only enough information for the first lane-changing task. If a lane following task fails later, the goal is unreachable. In this case, stopping the vehicle is an acceptable safe task. If an exit lane-changing task fails, the vehicle should continue driving in the round-about to another exit. Thus, the entire tree in Fig. 5.7 is generated as a complete solution. At junction j_1, the vehicle is uncertain about the situations at j_2 and j_3, so it continues driving to junction j_2 for the possible best choice. Then, at junction j_3, the vehicle has all the information to choose from the rest two options.

The advantage of task planning in this scenario is that the heuristic search can incrementally re-plan the solution regarding new events. Multiple possibilities can be evaluated for a complete solution. The search algorithm can easily check the duplicated states to avoid deadlocks, e.g.,

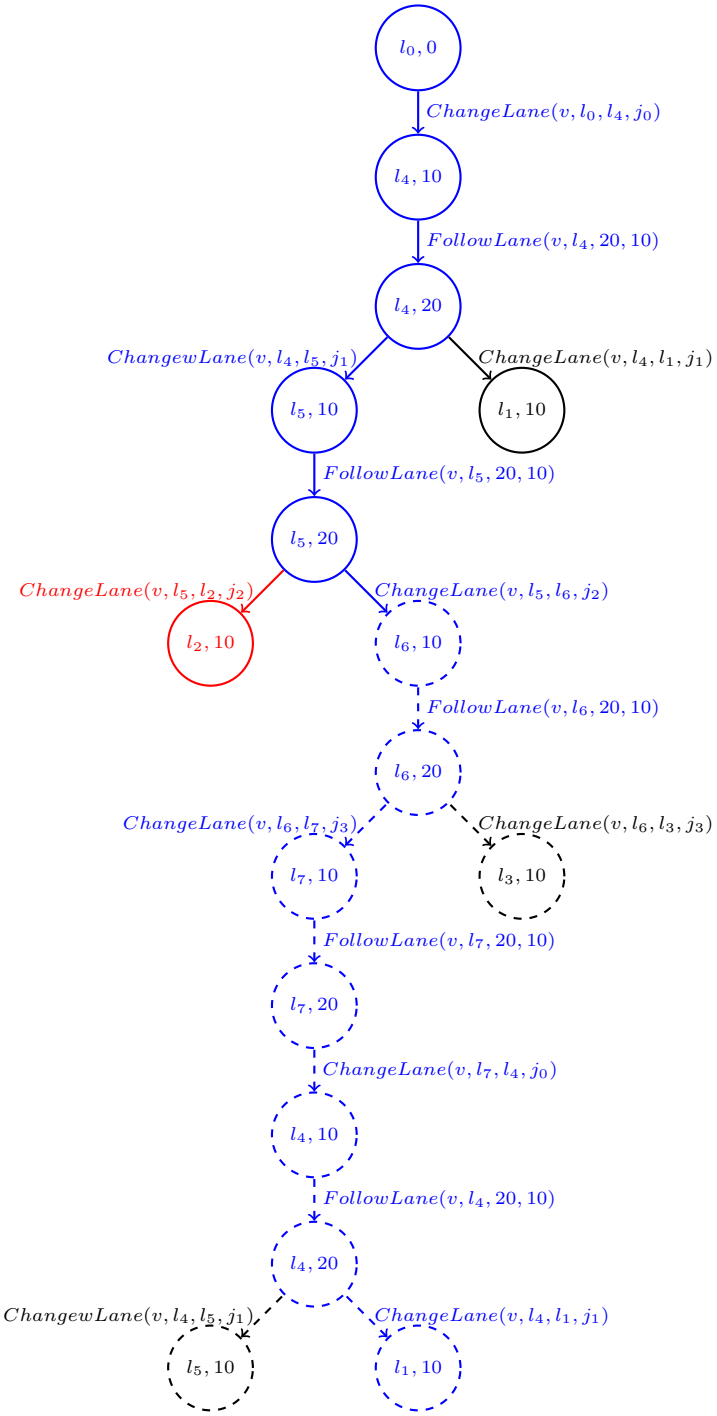


Figure 5.7: Task planning for round-about scenario.

keeping driving in the round-about when all the exits are blocked.

5.3 SEHS Motion Planning Engine

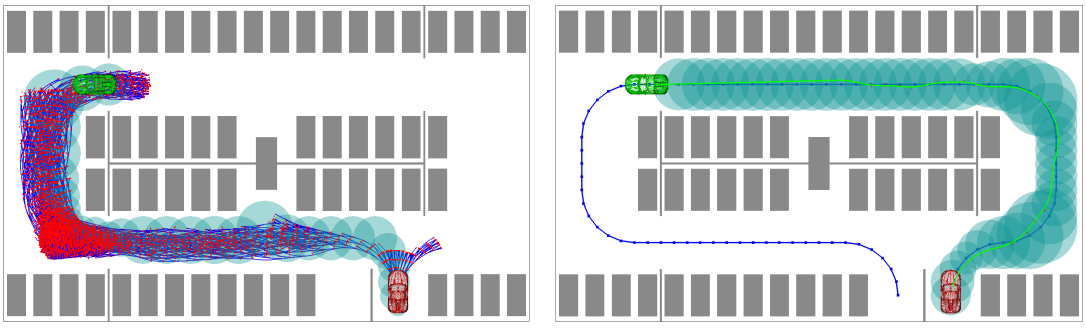
The task planning decomposed a complex driving scenario into a sequence of simple driving tasks. A real-time *Motion Planning Engine* plans the detail trajectory for each single driving task with the *Space Exploration Guided Heuristic Search* approach. In this case, the domain definition of the task planning provides the motion planner with various useful traffic knowledge, e.g., desired driving behavior of the task, predefined roadmap, prior static environment. The SEHS motion planning framework is able to take advantage of these different kinds of traffic knowledge in its two generic planning phases to solve practical automated driving problems. Meanwhile, an online verification and re-planning mechanism is designed to handle unexpected situations, and guarantee safety of the motion.

5.3.1 Apply Traffic Knowledge in SEHS

Traffic knowledge is a general concept for all kinds of information related to a practical driving task, road networks, traffic signals and rules, common driving behaviors, etc. Two of them, roadmap and desired steering behavior, are taken as examples to show the flexibility and extensibility of the SEHS method with benefit from traffic knowledge. As the space exploration of SEHS evaluates the geometric properties of the free space, and a roadmap already contains certain topological information, it can speed up the exploration procedure. The heuristic search in SEHS employs primitive motions to propagate the robot states. A good choice of the primitive motions regarding a steering guidance will dramatically improve the search performance and produce more desirable results.

5.3.1.1 Roadmap Knowledge

In practice, not all the physically free space is allowed for a vehicle to drive. A common constraint is that a vehicle should stay in its lane, and a lane shift is performed only when necessary. However, it is not easy to obtain such behaviors with a generic motion planning algorithm. In this case, if the space exploration of SEHS is initialized with a roadmap containing the lane information, the planner can produce free space circles initially along the lane. Only when there is an obstacle blocking the road, it starts to explore the rest free space to avoid the collision. As a result, it is more likely to achieve a roadmap oriented motion result.



(a) Motion planning in a car park without roadmap (b) Motion planning in a car park with roadmap

Figure 5.8: Roadmap based motion planning in a car park scenario

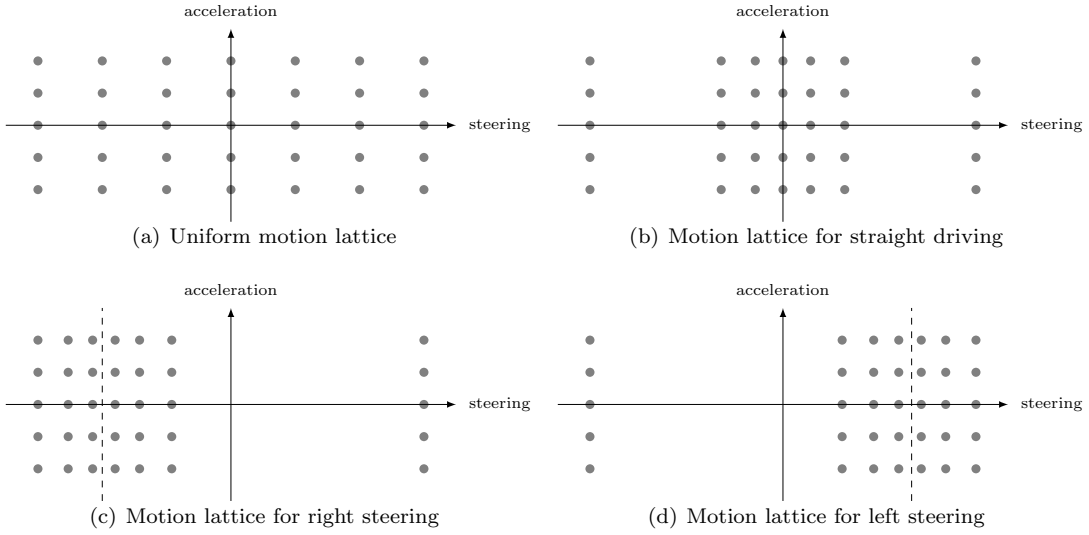
Fig. 5.8 compares motion planning with and without roadmap knowledge in a car park scenario. The red vehicle frame is the start pose and the green one is the goal pose. Fig. 5.8(a)

shows the result without roadmap knowledge. The space exploration suggests a path corridor in the clockwise direction, in which the vehicle travels with an opposite orientation at the end. Therefore, the heuristic search expands lots of states exhaustively to find a solution. In contrast, a roadmap is provided in Fig. 5.8(b) with many way-points (blue vertices) and the path segments (blue edges) between them. It provides a counterclockwise driving direction aligning to the goal orientation. The space exploration easily constructs a circle-path based on the roadmap. Then, the heuristic search follows the circles with a few vertices and finds a solution.

5.3.1.2 Driving Behavior Knowledge

As introduced in Section 2.3.2, a primitive motion is a combination of control inputs. The SEHS approach employs a set of primitive motions to extend the robot states. In SEHS, the length of a motion primitive is adapted to the circle size. However, other parameters such as acceleration and steering are uniformly interpolated in the valid range, which is invariant throughout the planning. With additional traffic knowledge, this simple control parameter lattice can be modified for different driving behaviors to improve the search efficiency.

The information about driving behaviors can be provided by the task planning or roadmap. For example, it is possible to extract the driving directions such as left turn, right turn or straight forward from the roadmap in Fig. 5.8(b). A task planning can divide an overtaking maneuver into lane switching and lane following tasks with different steering intentions. In this case, different sets of primitive motions can be defined with modified control parameter lattices.



(a) Uniform motion lattice (b) Motion lattice for straight driving

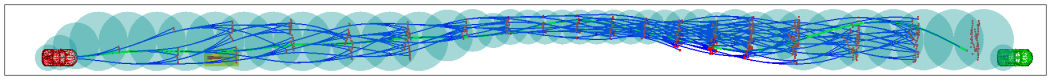
(c) Motion lattice for right steering (d) Motion lattice for left steering

Figure 5.9: Behavior-based steering motion primitives

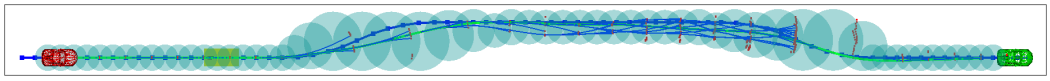
Fig. 5.9 shows different types of primitive motion lattices. X-axis is for the steering behavior and Y-axis is for the acceleration input. The grey dots are the sample points for motion primitives. Fig. 5.9(a) is a uniform motion lattice, the samples are distributed at equal distance. This motion primitive set is applied in the general SEHS motion planning method. Fig. 5.9(b) is for straight driving, as the samples are denser around the zero steering line. It produces mild steering motions for straight driving. Fig. 5.9(c) and Fig. 5.9(d) are for right turn or left turn respectively. The samples are biased to a desired steering input at the dashed line. The steering

input at the right or left extreme position is still provided in the three adapted motion sets, so that the state expansion can still cover the whole reachable set. Applying the specific motion lattice, the planner obtains finer resolution in the required moving direction than the standard uniform lattice with a same step size. As a result, the heuristic search can take larger steps to find a solution with fewer vertices, or return a smoother trajectory with the same planning effort. The acceleration parameter can also be handled in the same way for speed up, slow down, or constant cruising driving behaviors.

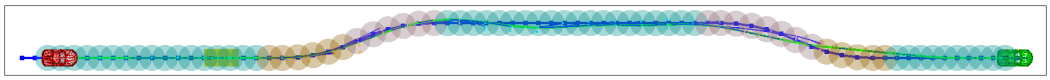
The behavior-based motion lattice in Fig. 5.9 is created with a normal distribution. The mean value is chosen according to a central steering parameter and the variance is decided regarding the valid range of the steering control parameters. Then, the value of a cumulative distribution function (cdf) is uniformly interpolated for a series of input parameters $\{x_1, x_2, \ldots, x_n\}$ that $\mathrm{cdf}(x_k) - \mathrm{cdf}(x_{k-1}) = \mathrm{const.}$. More natural primitive motions can be built with a close study of the human driving behaviors.



(a) Overtaking without roadmap knowledge



(b) Overtaking with roadmap knowledge



(c) Overtaking with roadmap and driving behavior knowledge

Figure 5.10: Overtaking with roadmap and driving behavior knowledge

Fig. 5.10 compares the effect of applying the different driving knowledge in an overtaking scenario. The start state is the red vehicle frame and the green one shows the goal pose. A slower moving obstacle (yellow rectangle) is in front of the ego vehicle. Fig. 5.10(a) is the direct result from a STEHS planner without considering any further information. The space exploration suggests taking the entire width of the road for driving, which should be divided into two lanes. The heuristic search proceeds at the beginning by moving the vehicle to the middle of the road, which is optimal for the safety distance but violates the traffic rules. After the overtaking, it takes huge effort to reach the goal position as the path corridor is not optimal for lane switching. Provided with a predefined roadmap, the planner in Fig. 5.10(b) first builds a path corridor based on the right lane, then fixes it with space explorations connecting the path corridor of the left lane. The heuristic search finds a more human like lane-switching motion. As the circles are smaller regarding the lane width, the planner expands fewer states to follow the passage. In addition, it is easier to switch back to the right lane to hit the goal. In Fig. 5.10(c), the planner is further aided by the behavior-based steering motion primitives. The circles in orange are labeled for left steering, while the pink circles for right steering. The cyan ones are the straight driving circles. By mapping the states to the circles, the planner evaluates different group of primitive motions to expand the states. The color of the state points is matched to the color of circles. It is more convenient to produce a left steering behavior at the beginning of the first lane switching, and then right steering to stay in the left lane. The second lane switching is just an easy opposite procedure. In this scenario, it generates the best

overall performance in both planning time and path quality.

5.3.2 Incremental Re-planning with Safety Motion

The *Motion Planning Engine* encapsulates the SEHS framework and provides interfaces to the traffic knowledge as in Fig. 5.1. It arranges the planning, validation, and re-planning functions in such a way that the system can react to the environment changes in time and ensure the safety of the motion. Different operation modes are defined with dynamic time schedules to produce a valid motion in time, as well as a fallback safe motion solution. The validation and re-planning are carried out in an incremental manner in order to reuse the knowledge gained during the previous planning and reduce the reaction time. Details are explained in the following subsections.

5.3.2.1 Operation Modes and Planning Period

The *Motion Planning Engine* has mainly three operation modes: *planning*, *validation* and *re-planning*. The planning procedure is the initial step of an autonomous driving process, which constructs an exploration tree and a search tree while producing the first motion result. After that, the vehicle can start to execute the motion as long as it is valid. A validation thread is started immediately, in order to make sure the motion is safe until the goal is reached. Otherwise, a re-planning procedure is triggered to adapt the motion. The states flow diagram is illustrated in Figure 5.11.

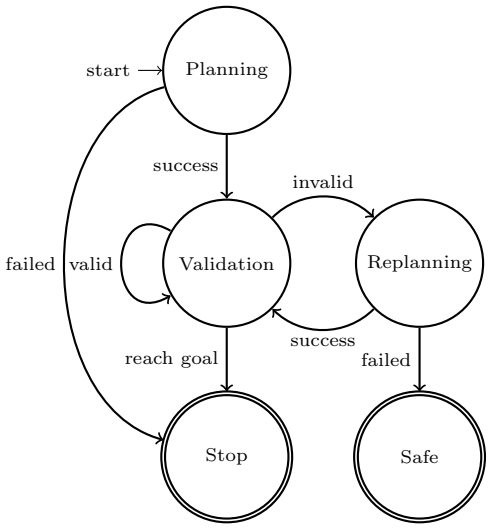


Figure 5.11: SEHS motion planning engine states diagram

The validation thread runs synchronously regarding the update rate of the environment perception, while the planning and re-planning are asynchronous in order to invest as much time as possible to achieve the best planning result. Through validation of the motion trajectory, the incoming invalid state or collision is known to the *Motion Planning Engine*. Any state point before the critical state can be selected as the starting point for the re-planning, so that the adapted motion can be attached to the existing one seamlessly. In this case, the whole time duration from the current state to the start state is available for the re-planning. Thus, a trade-off should be made between the time and space for re-planning. Because if the starting

point moves further from the current state, the planning time is longer, but the critical point is closer to the starting state leaving less space for the re-planning. When the starting point is chosen far from the invalid state, the situation is vice versa. In general, the time to the start point should be larger than the specified planning time, and the distance to the invalid point should be enough for the safety motion.

A predefined safety motion, e.g., a pull-over or emergency brake maneuver, is temporarily appended to replace the motion after the selected starting point. In this case, if the re-planning fails, the vehicle will naturally perform the safety motion instead.

5.3.2.2 Validation and Re-planning

In principle, a re-planning procedure should be triggered when the current motion is invalid or a better result exists. The first situation is straight forward to verify with collision checks, traffic rules, etc. The second one is almost as complex as the motion planning problem itself. However, regarding the additional traffic knowledge, it is possible to identify an optimal motion in certain circumstances. For example, if the motion is planned with a collision-free roadmap, the result motion is optimal if the path corridor remains unchanged. In this case, a re-planning is only necessary if the passage is blocked by obstacles. The other way around, if a previous invalid waypoint is now collision-free, a better solution may exist. Thus, human-like driving behaviors can be produced, which follows the road, avoids collisions, and steers back to the initial path.

The re-planning with SEHS is incremental that reuses the result from the last planning event as much as possible. It is performed in the following steps:

1. Redo the exploration, as the circle-based or cylinder-based exploration is extra fast and consumes only a small portion of the total planning time.
2. Prune the search graph by removing the vertices and edges other than the current state and its descendants.
3. Remap the vertices to the circles and update the heuristic costs.
4. Add all the vertices back into the open-set and start the heuristic search.

As a result, the states already evaluated by the previous search procedure remain for the re-planning, which saves the effort of states propagation with forward dynamics. The collision checks are performed in a lazy manner, which verifies the path vertices only when a new solution is found. In this case, if an internal state is invalid, the solution is rejected. The planning continues after removing the invalid vertex and its descendants.

Furthermore, a third set of vertices, called *Reserved-Set*, can be introduced beside the open-set and closed-set of the heuristic search. All the vertices that are rejected due to collision with dynamic obstacles go to this set. Then, these vertices can be reused during re-planning, as the environment may change during the time, so that some vertices could become valid to produce a better result. The vertices rejected regarding the state resolution can also be revisited in the same way when the resolution is refined.

5.4 Applications

Three example applications are presented in this section with the focus at practical autonomous driving scenarios. In these examples, the *Motion Planning Engine* needs to handle additional domain knowledge and adapt motion to the real-time sensing inputs.

5.4.1 Precise Parking onto an Inductive Charging Station

For electric vehicles, inductive charging is a convenient way to refresh the battery without plugging a cable from the charging station. Regarding the physical laws, the charging efficiency depends greatly on the relative position of the charging devices from the vehicle side and the station side. Therefore, good parking precision is the key point of this application, which is not easy for an untrained human driver to achieve. In this case, autonomous parking can provide a convenient solution to assure the efficiency of inductive charging. The *Motion Planning Engine* can adapt the vehicle movement to the real-time localization of the charging spot in order to reach the desired charging position.

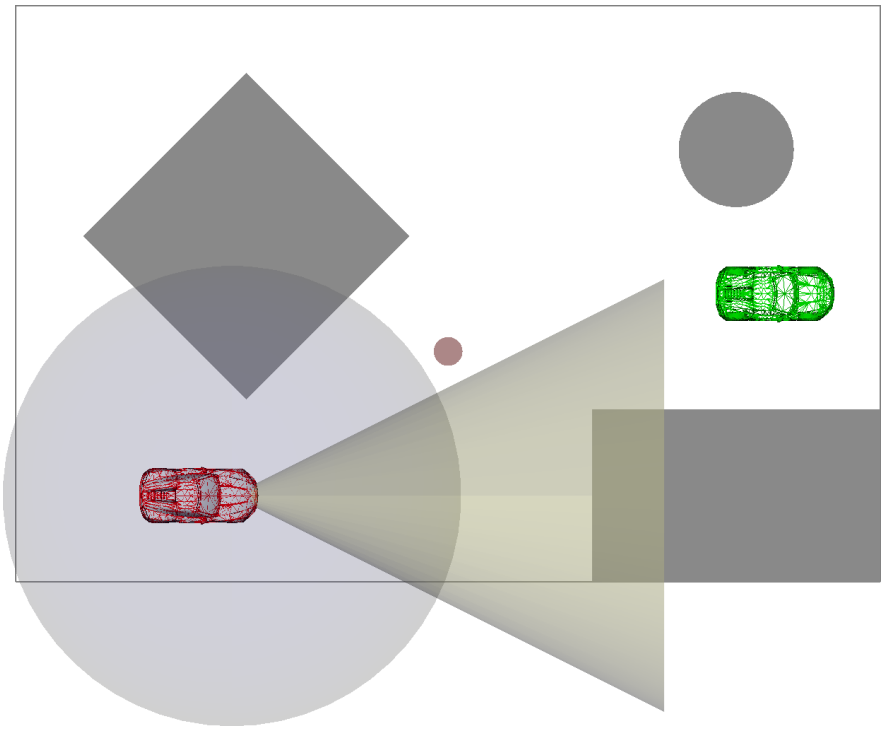
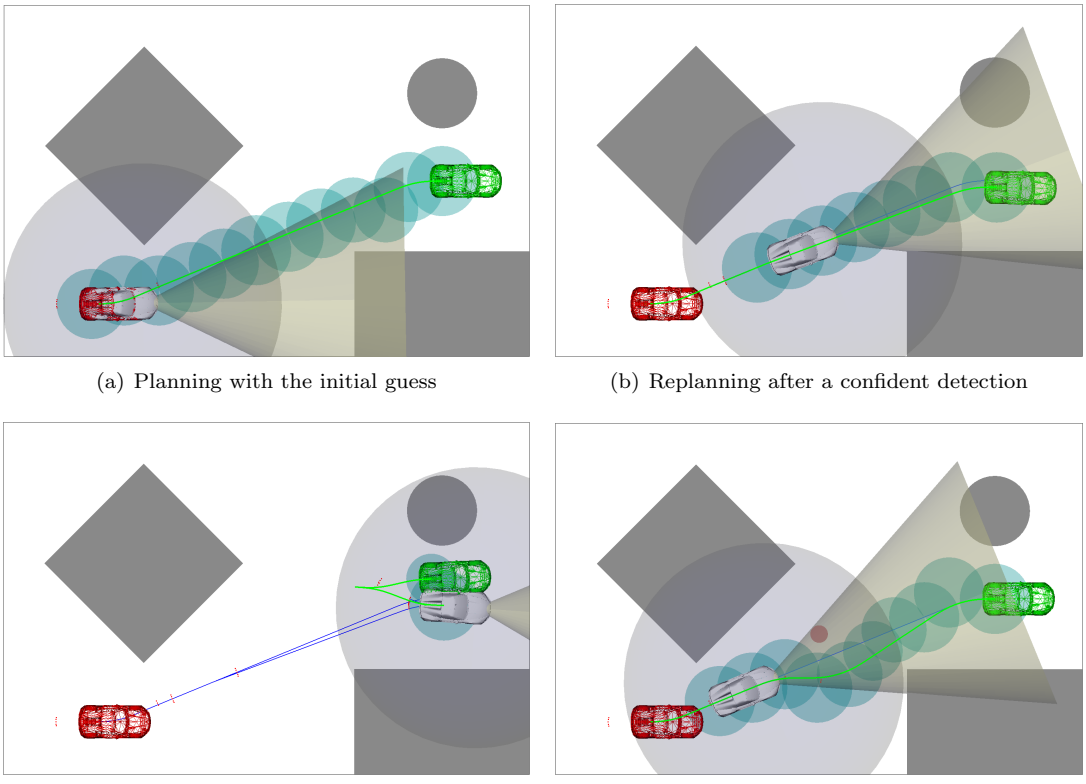


Figure 5.12: Autonomous charging scenario

Fig. 5.12 shows a setup of autonomous charging scenario. A vehicle starts in a close neighborhood (about 20 m away) from a charging station. The start pose is displayed as a red vehicle frame. An initial guess of the charging pose is made regarding the prior knowledge or communicated information about the charging station location. A green vehicle frame represents the goal pose. Some prior knowledge about the environment is also available as the grey obstacles. Unknown obstacles are colored in pink, which can be detected by the vehicle when it is in its sensing range. The vehicle has certain near field sensing ability in a range of 8 m as the light blue circle. In addition, the vehicle is equipped with a specific sensor, which can detect the charging station in a certain range (15 m) and field of view (60°) as the light yellow cone. The uncertainty of the charging station localization is simulated with random sampling near the initial estimated position. The translation is generated with a two dimensional normal distribution with a 10 cm standard deviation. The angle difference is normally distributed with a 10° standard deviation.

The general procedure of the autonomous charging can be divided into three steps. The first step is driving the vehicle towards the assumed charging location, so that it can detect the target with the onboard sensor. Then, after detecting the goal location with a certain confidence, a re-planning is triggered to adjust the vehicle motion to the actual charging pose. Due to the sensing uncertainty or accumulated error from the execution, the final state of the vehicle may not satisfy the precision requirement. In this case, an extra maneuver is planned to reduce the error. This action can repeat until a desired precision is achieved. All the motion planning is performed in a local coordinate system, which localizes the objects relative to the vehicle. In this case, a new target detection updates the goal pose instead of the vehicle pose, which is convenient for the trajectory controller to continue driving the vehicle after the existing trajectory without abrupt state jumps. The same as the re-planning procedure in the previous section, a future state along the trajectory is selected as the start pose for re-planning, which guarantees that the adapted trajectory can be smoothly appended to the current one.



(a) Planning with the initial guess

(b) Replanning after a confident detection

(c) Final correction

(d) Collision avoidance

Figure 5.13: Precise parking onto an inductive charging station

The results are illustrated in Fig. 5.13. As the vehicle speed is small (about $1\,\mathrm{m\,s^{-1}}$) in this charging scenario, the kinematic model with continuous curvature in Section 2.3.2.2 is applied in a baseline SEHS algorithm. The first three figures demonstrate a general three-step workflow of the autonomous charging process. Fig. 5.13(a) shows the initial planning result with the first guess of the charging pose. While the vehicle driving towards the rough charging position, the sensor cone of vision is pointed to its direction. In Fig. 5.13(b), an actual charging pose is detected, which is slightly different from the initial one. In this case, the trajectory is updated

from a future position to the new goal pose. Fig. 5.13(c) shows the final correction maneuver. The *Motion Planning Engine* also verifies the trajectory continuously with the live sensing inputs and adapts the motion when necessary. Fig. 5.13(d) shows the situation that when an unknown object is detected by the vehicle, the *Motion Planning Engine* suggests a collision avoidance motion to reach the final position. The re-planning is carried out in an incremental manner, as the search tree with red vertices and blue edges is constructed incrementally.

A further improvement is to consider the sensing and control uncertainty in the motion planning. In practical, a sensor, e.g., a camera for QR-code detection, returns good results when the target is close in distance and in the middle of the field of view. The actuators can produce precise motions when following a straight trajectory. However, the default SEHS planning tends to return a solution with steering motions at the beginning and end for the shortest travel distance, which is not optimal for sensing and control. In this case, a better motion strategy is to firstly reach a good internal position, where the vehicle can move straight forward to the charging pose. Thus, it has a better chance to detect the target pose and achieve good final precision.

5.4.2 Transit and Parking in Semi-Structured Environments

Parking in a semi-structured area is time consuming, not only because of looking for an empty parking-lot, but also for the time to transit between the parking-lot and the entrance or the exit. In this case, an automated valet parking system is helpful to save time. The driver only needs to drop the vehicle at the entrance of the parking area, and then it will drive automatically to a free parking-lot. To pick-up a vehicle is just as simple. The driver waits at the exit of the parking area, and the vehicle comes from the parking position to the waiting area by itself. In this scenario, it is required that the autonomous driving system not only plans the parking and transit maneuvers, but also follows simple traffic rules or roadmaps in the parking area, and negotiates with the prior known and post detected objects. These functions can be easily realized with the *Motion Planning Engine*, which handles the roadmap knowledge a SEHS planner, and automatically re-plans the motion regarding the detected obstacles.

In an example scenario in Fig. 5.14, a 50 m $\times$ 30 m parking area is constructed with many parking-lots and separated entrance and exit. The static environment, such as walls and parking-lots, is known, as well as a roadmap (the blue dots and lines) around the parking area from the entrance to the exit. The prior information is provided to the planner at the beginning of the valet parking task. There could be some unexpected obstacles in the parking area, e.g., pedestrians or other vehicles. The small dark yellow circle is a dynamic obstacle with the moving direction indicated by the red triangle. As in the previous scenario, the vehicle can detect objects within a sensing range (10 m) as the light blue circle. In a drop-off scenario, a free parking-lot is assigned to the vehicle with a pre-defined internal position (the blue vehicle frame) on the roadmap as the goal position for the transit maneuver and start pose for the parking procedure. For pick-up, the vehicle remembers the parking position with an internal pose on the roadmap as the goal position to pull-out. Then the vehicle can continue to drive along the roadmap to an other location. Combining a pick-up maneuver and a drop-off maneuver, a vehicle can change its parking position in case some special services, e.g., refueling, charging, washing, can only be performed in specific positions.

Fig. 5.15 shows the different scenarios in a semi-structured parking area combining autonomous parking and transit. The drop-off procedure is demonstrated in Fig. 5.15(a) and Fig. 5.15(b). The vehicle first performs a transit maneuver following the roadmap to the internal position. As it reaching the final destination, a second motion planning is triggered for the parking maneuver. The pick-up procedure is just the opposite. At first, a pull-out maneuver

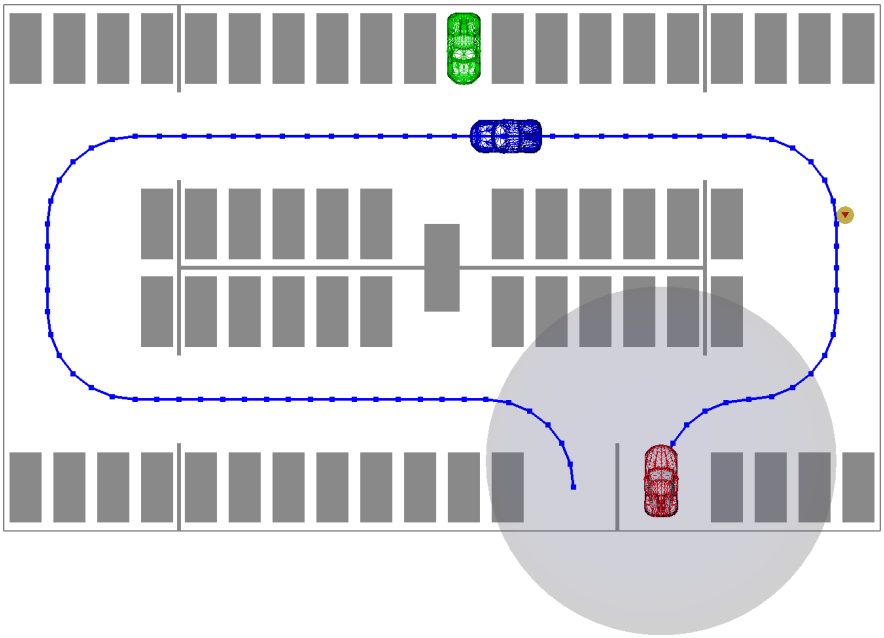


Figure 5.14: Automated parking scenario

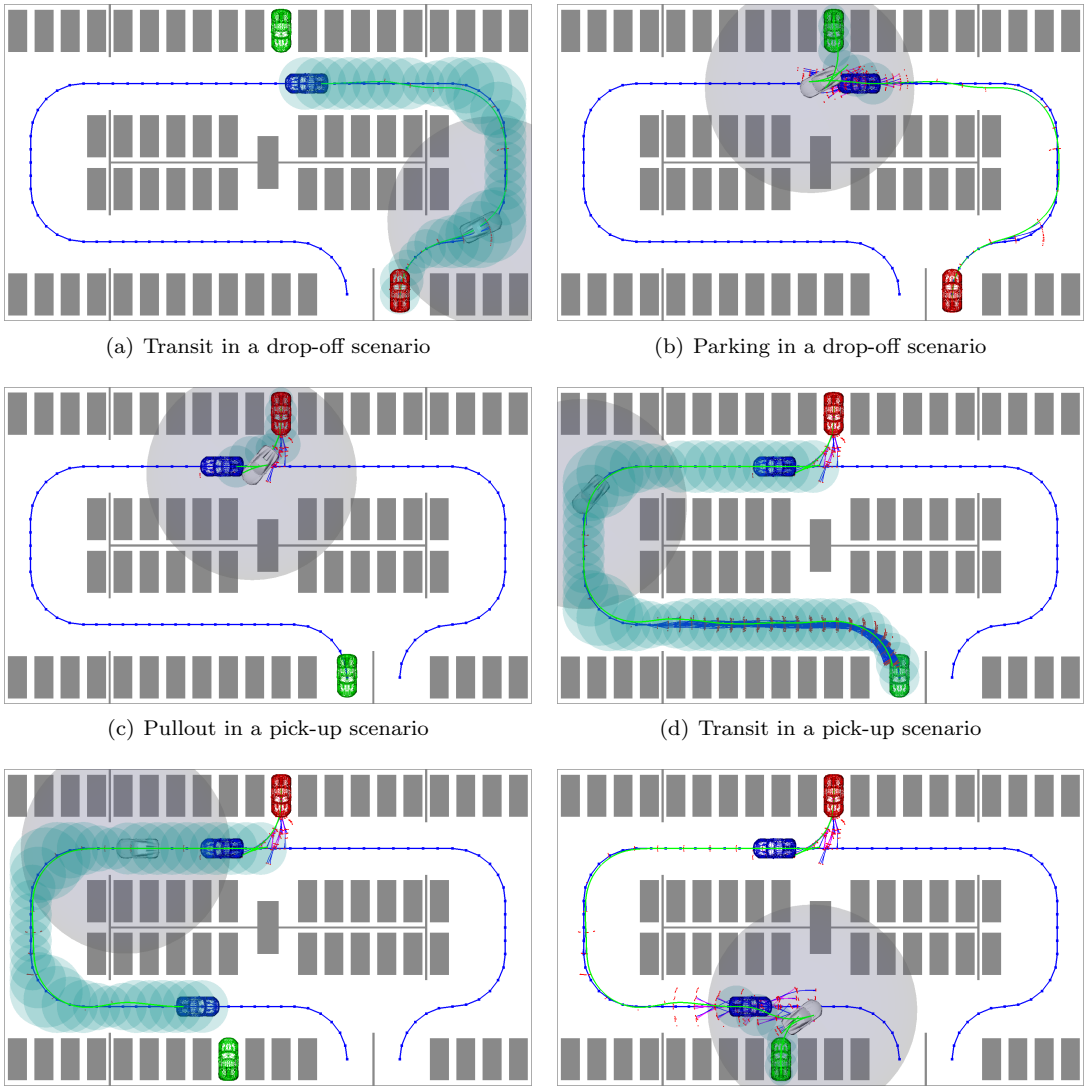
is planned and executed in Fig. 5.15(c). Then the transit motion can be generated when the pull-out goal pose is in sight in Fig. 5.15(d). In a changing parking position scenario, the vehicle first perform a pick-up maneuver to the internal pose for the new parking-lot in Fig. 5.15(e). Then it parks into the new position as in Fig. 5.15(f). For the pull-out and parking maneuvers, OSEHS can be applied to improve the planning performance.

With the help of the *Motion Planning Engine*, the vehicle can adapt its motion to the real-time perception during the transit maneuver. As demonstrated in Fig. 5.16, a dynamic object is unknown in the first place. Therefore, the initial trajectory in Fig. 5.16(a) may cause a collision. After the vehicle detects the object, the trajectory verify process of the *Motion Planning Engine* discovers that a collision may occur in about 4.35 s. The time is enough to re-plan the trajectory in the next 500 ms. In this case, a 500 ms later state is selected as the start position, and a new motion is planned as in Fig. 5.16(b) to avoid the collision and reach the goal position. STEHS is applied here to plan motion in the dynamic environment.

If the infrastructure is not so intelligent to maintain the parking-lots distribution, the vehicle may still drive autonomously around the parking area to look for an empty place, and perform the parking and pull-out maneuvers just with the common parking assistant functions. With a central planning agent for the whole parking area, a more complex and efficient parking system can be built, which manages the parking-lots and roadmaps, and optimizes the parking procedures of multiple vehicles at the same time.

5.4.3 Automatic Driving through Intersections

Intersections and junctions are the most vulnerable locations to traffic accidents due to the complex traffic conditions. According to the data from the German Federal Statistical Office [123], about 47.5% of the traffic accidents in 2013 happened at intersections and junctions in Germany. The statistics from the United States Department of Transportation, National Highway



(a) Transit in a drop-off scenario

(b) Parking in a drop-off scenario

(c) Pullout in a pick-up scenario

(d) Transit in a pick-up scenario

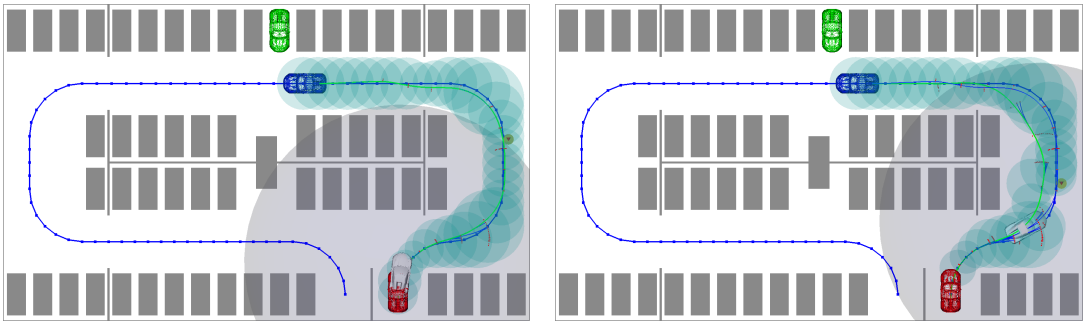
(e) Transit in a switch position scenario

(f) Parking in a switch position scenario

Figure 5.15: Transit and parking in a semi-structured parking area

Traffic Safety Administration [124] shows that 40% of the traffic accidents in the United States in 2008 occurred at intersections. Among them, 84.9% accidents are caused by incorrect driver perception or decision. Therefore, a driving assistant system that helps drivers to process traffic information, make decisions, or even autonomously drives the vehicle through intersections can greatly improve the traffic safety in these situations.

Thanks to the modern sensing and communication technologies, a vehicle can access to various traffic information online. For example, the ego position and motion are available from GPS, odometry or inertial sensor. The lane markers, traffic signs, and traffic lights are detected by cameras. Other vehicles, pedestrian, and obstacles are recognized by radar, laser scanners, or cameras. Furthermore, a vehicle can communicate with other vehicles or the infrastructure



(a) Initial motion planning (b) Replanning after obstacle detection

Figure 5.16: Collision avoidance during transit

to get an overview of the traffic situation. Altogether, these modules provide an intersection assistant system enough information to make decisions in difficult situations.

These different types of information can be processed by the *Motion Planning Engine* in the three planning levels in Fig. 5.1. The route planner suggests a road to take at an intersection based on the street map and vehicle position. The task planner evaluates the symbolic state of the traffic situation at the intersection for a sequence of driving tasks that follow the navigation and compliant to the traffic rules. The motion planner finally produces detail trajectories for the tasks regarding the vehicle kinematics and obstacles position. These three procedures can be executed with different frequency in parallel, planning and verifying the driving behavior with the real-time situation at an intersection, which is an excellent example to show that the *Motion Planning Engine* can manage the most sophisticated traffic scenarios in autonomous driving.

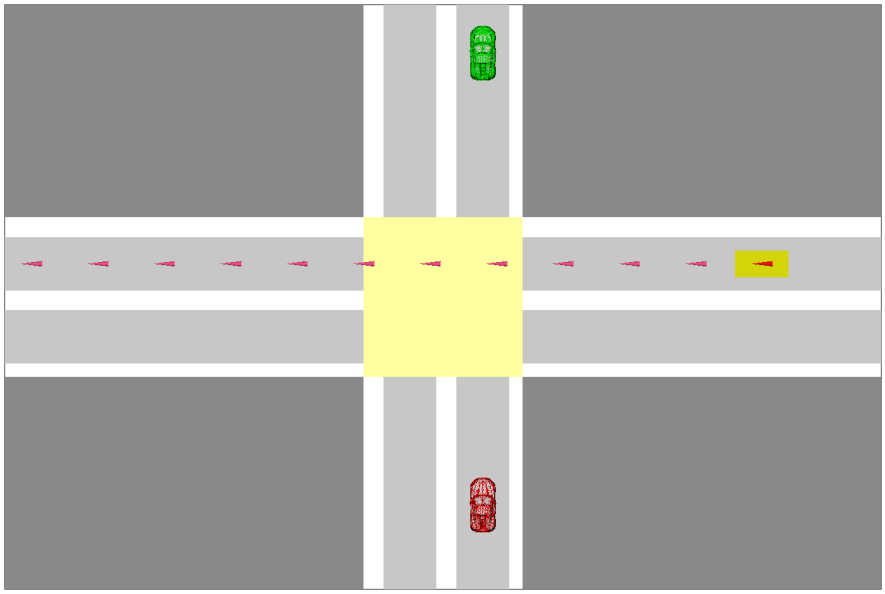
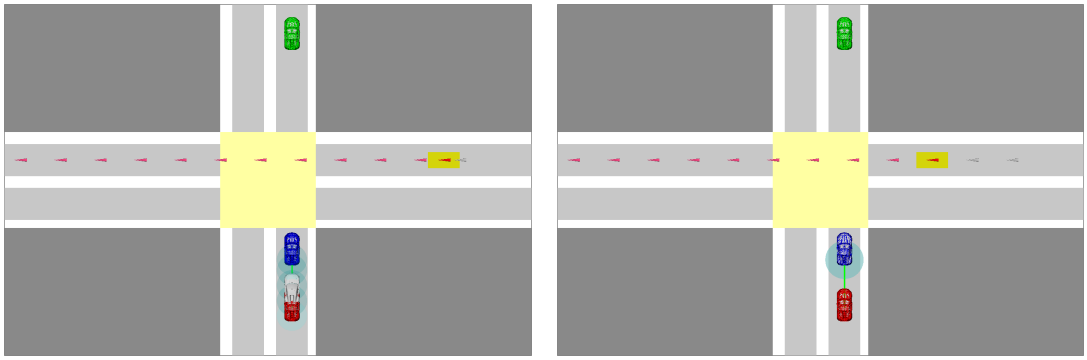


Figure 5.17: Simple intersection yield scenario

Fig. 5.17 shows a simple example of driving through an intersection without any traffic lights, signs, or crosswalk for the pedestrian. The yellow area is an intersection with two lanes in light grey in four directions. The dark grey area is not allowed to traverse. The ego vehicle is going to cross the intersection straight from the red position to the green spot, while another vehicle (the yellow rectangle with a red triangle for the moving direction) is coming from another street. Instead of planning a collision avoidance maneuver as in Fig. 4.5, the ego vehicle should behave correctly regarding the traffic rules. According to the German traffic laws [122], the traffic from the right has higher priority in this situation. Therefore, the ego vehicle should give way at the intersection, and cross it after the other vehicle passed. In this case, three kinds of tasks are relevant to plan the driving-through-intersection maneuver: *FollowLane* and *ChangeLane* as introduced in Section 5.2.1.3, and a *Wait* task, which just stop the vehicle before the intersection for a certain time duration.



(a) Approach the intersection

(b) Wait at the intersection

(c) Across the intersection

(d) Left the intersection

Figure 5.18: Yield and driving through an intersection

If the vehicle knows the coming traffic, four tasks are planned and executed sequentially as in Fig. 5.18. First, the vehicle approach the junction in its lane to a position before the stop line with a *FollowLane* task in Fig. 5.18(a). Then, it stops and waits the other vehicle to pass through the junction with a *Wait* task in Fig. 5.18(b). After that, the junction is clear to drive, and a motion is planned within the junction area to reach a position at the other side as a *ChangeLane* task in Fig. 5.18(c). Finally, the vehicle proceeds in the target lane with a *FollowLane* task in Fig. 5.18(d). The internal goals of the tasks are illustrated as the blue vehicle frames. The whole process is rather straightforward and trivial for the task planning to

solve.

If the vehicle is not aware of the situation around the intersection at the beginning, but perceives the environment in its sensing range, the case is a little more interesting. If the task planner assumes no traffic at the intersection, there may be not enough time to stop the vehicle if the vehicle speed is too high. Therefore, a better strategy is to prepare for coming traffic from outside the sensing range in anytime. Thus, driving through the intersection with a high speed is not 100 % safe, a *FollowLane* task with a moderate speed is suggested. The task planner initially plans three driving tasks without waiting at the intersection. Then, as the ego vehicle approaching the intersection and the other vehicle is detected, the task planner recognizes the precondition of the *ChangeLane* task is violated, and inserts a *Wait* task before it. The speed of the first *FollowLane* task can be chosen depending on the sensing ability and information availability of the ego vehicle in the specific scenario. Thus, an autonomous vehicle behaves similar as a human driver, who proceeds carefully when the traffic situation at an intersection is not clear.

For a simple intersection scenario, the choices of different behaviors can also be easily modeled with a state machine. However, for the various intersection types and traffic rules, the state machine should be extended to handle all the cases. Furthermore, if a decision should be made not only based on the current system state, but also the possible evolution of the environment, the state transition becomes sophisticated. The advantage of applying task planning is that the logic conditions are modeled in the domain definition as preconditions and effects of tasks. With a simple search method, the *Motion Planning Engine* can evaluate all the task combinations for an optimal solution. Moreover, it is easy to extend and modify a subset of tasks for new rules and scenarios without changing the planner. The motion planning is performed in the context of a task, e.g., the lane following motion is planned in a lane, while the lane change motion is planned in the designated area of the intersection. Therefore, the problem is solved in a hierarchical way as moderate sub-problems.

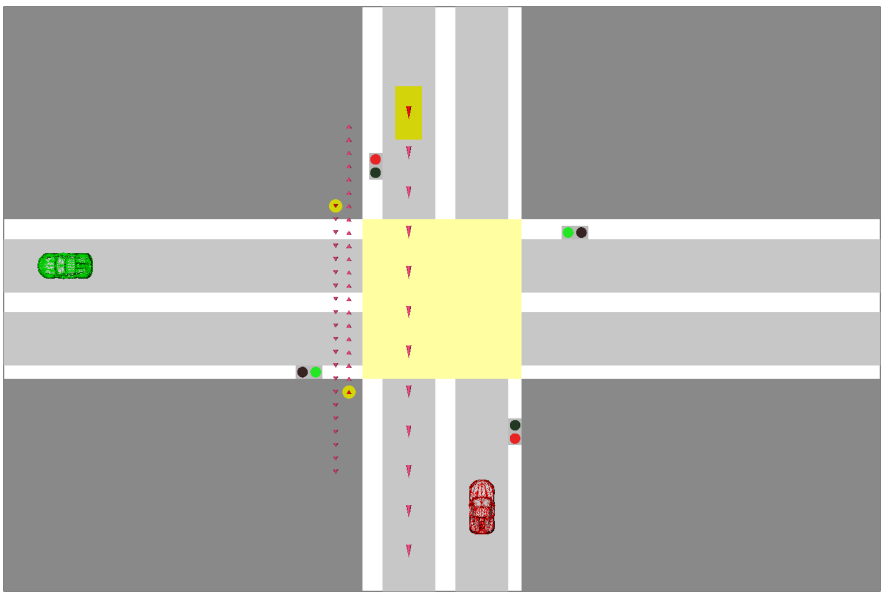


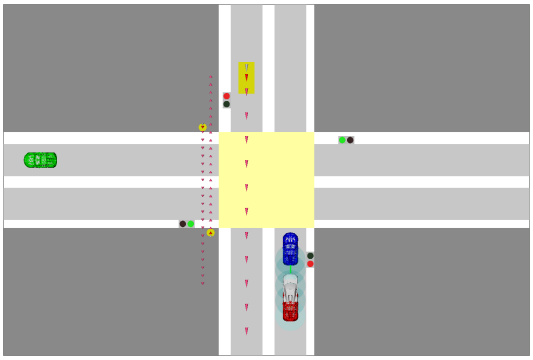
Figure 5.19: Complex left turn scenario

A more complicated scenario is illustrated in Fig. 5.19. A vehicle is going to turn left at an

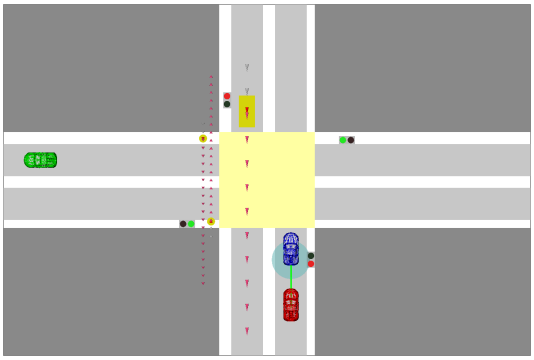
intersection from the red vehicle frame as the start position to the green vehicle frame as the goal pose. There are traffic lights control at the intersection. The state of the traffic light is displayed as the red and green dots on the right of the road. At the same time, another vehicle (the yellow rectangle with a red triangle for the moving direction) is coming from the opposite direction across the junction, which has a higher priority according to the traffic rules. In addition, two pedestrians (the yellow circles with red triangles for the moving direction) are crossing the road, where the vehicle is going to enter. In order to plan a human-like maneuver in this scenario, two new tasks are introduced beside the simple *LaneChange* task, a *LaneChangePrepare* task and a *LaneChangeProceed* task. The former one moves the vehicle to a pre-defined internal position in the junction, so that it can wait for the straight driving traffic to pass. The precondition is weaker than the *LaneChange* task, which does not require the complete connection and the destination lane to be free. The latter one finish a lane changing maneuver continuing from the internal position, which requires the rest of the connection and the final lane position to be free. Furthermore, a precondition of a green light is added to these lane changing tasks.

The result is demonstrated in Fig. 5.20. The ego vehicle first approaches and stops at the junction for the red light in Fig. 5.20(a) and Fig. 5.20(b). When the light turns green, the vehicle cannot directly turn left, because the connection is occupied by the other vehicle with higher priority. It can either wait at its place or move to the middle position to prepare the turning later. In Fig. 5.20(c), it chooses the second one because it saves time by performing a half of the turning. After the other vehicle passes by, the ego vehicle still waits in the middle for the two pedestrians to cross in Fig. 5.20(d) until the target lane is free. Finally, the vehicle finishes the turning maneuver and goes on driving in Fig. 5.20(e) and Fig. 5.20(f).

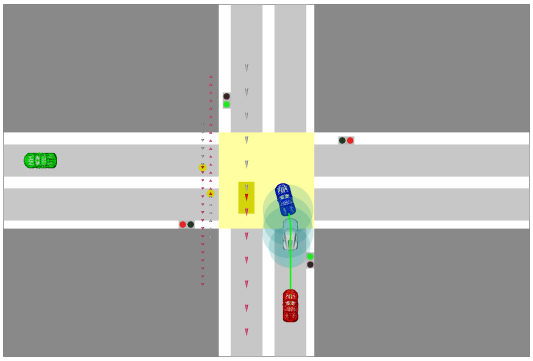
In this example, the task planner remains unchanged. The domain definition is extended with two tasks and the additional preconditions for traffic lights and pedestrians. These conditions for the new scenario is not explicitly coded in a state machine, but defined as task definitions. The modified domain knowledge is backwards compatible with the previous simple example. Thus, a knowledge base for autonomous driving can be built up in an incremental way.



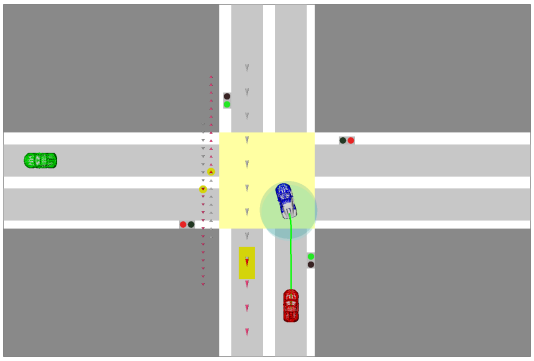
(a) Approach the intersection



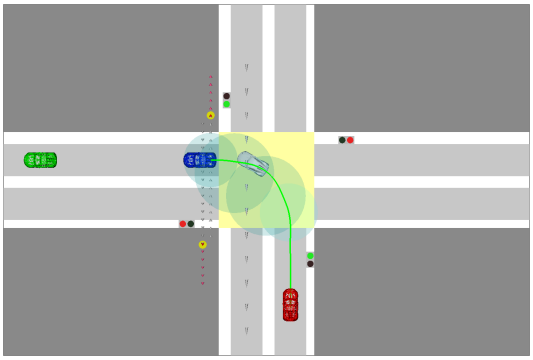
(b) Stop before the intersection for the traffic light



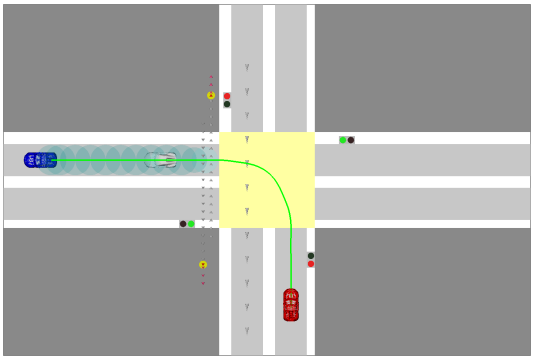
(c) Moving to a middle position



(d) Wait for the vehicle and pedestrians to pass



(e) Proceed to turn left



(f) Continue driving

Figure 5.20: Turning at an intersection with traffic light, incoming traffic, and pedestrians

5.5 Summary

As demonstrated in the three example scenarios above, the *Motion Planning Engine* can provide real-time planning and verification services for different automotive applications from a simple precise parking function to a complicated intersection assistance system. The task planning enhance the ability of the planning agent with access to domain knowledge and high level navigation results to produce more human-like driving behaviors in compliance with traffic rules. It is easy to scale and adapt the hierarchical architecture of *Motion Planning Engine* for complex scenarios and rules, so that this concept can be applied to develop advanced driver assistant systems towards fully autonomous driving.

Chapter 6

AutoDrive Library

In order to implement, demonstrate, verify, and compare the motion planning algorithms, a software library, named *AutoDrive* as abbreviation for "autonomous driving", is developed. This library contains components from the basic mathematical functions to graphical simulations for motion planning methods, which provides great assistance in algorithm development and application prototyping for autonomous driving.

6.1 Structure Overview

The structure of the AutoDrive library is visualized in Fig. 6.1. The library architecture is constructed according to the data and algorithms required by the car-like robot motion planning algorithms. The different models of vehicle kinematics are defined in the *Vehicle* sub-library. The environment information is represented with a grid map or an object list from the *Environment* part. The *Plan* stack contains the planning algorithms including motion planning and task planning. All these components are supported by the *Math* modules for real number arithmetic, geometric operations, random distributions, etc. Unit tests are available in the *Tests* section for the four core-components mentioned above. Several examples are implemented in *Demos* to demonstrate the motion planning methods in a 3D scene, including a XML-based scenario configuration and a control GUI.

Comparing to other path or motion planning libraries, such as MSL [108] or OMPL [110], the *AutoDrive* library is specific designed for car-like robot motion planning, which provides kinematic models of car-like robot and environment models for sensing and perception. The planners are tailored for car-like robot motion planning problems, especially the planners of the SEHS family. The task planning is also developed for driving maneuvers with a compact domain definition.

Another major purpose of the *AutoDrive* library is providing a common basis to test and benchmark different planning algorithms. A generic planner is defined with general motion validation and visualization interfaces. It is convenient to implement and test other planning methods with the same system model and in the same simulation environment.

Furthermore, the library only depends on the C++ STLs, and a few parts (shared pointer, random, and graph) of boost library. Therefore, it has good compatibility, and can be easily applied in different development and run-time environments.

Figure 6.1: AutoDrive Library

6.2 Mathematical Tools

The general robot motion planning involves mathematical problems in high dimensional configuration space, for examples, Jacobian matrix, reverse kinematics, nearest neighbor search, etc. Therefore, it is necessary to link external libraries, which are dedicated for these tasks. However, the car-like robot motion planning handles a kinematic model in a low dimensional but heterogeneous configuration space. In this case, the *AutoDrive* library develops a stand-alone *Math* section for the basic mathematical types and operations.

6.2.1 Real Number Arithmetic

First of all, a data type *Real* is defined for the real number physical values. Thus it is convenient to switch the precision of the whole program. Several frequently used constants are also provided

such as $\sqrt{2}$. An ϵ is defined as an infinite small value for the real number comparison. Some functions are designed to verify the properties or compare the real numbers in Table 6.1.

Table 6.1: Arithmetic Functions

isfinite(a)	true if $\|a\| < \infty$.
isnormal(a)	true if $0 < \|a\| < \infty$.
ispositive(a)	true if $0 < a < +\infty$.
isnegative(a)	true if $0 > a > -\infty$.
notnegative(a)	true if $0 \leq a < +\infty$.
notpositive(a)	true if $0 \geq a > -\infty$.
sign(a)	-1 if $a < 0$, 0 if $a = 0$, 1 if $a > 0$.
isepsilon(a)	true if $\|a\| \leq \epsilon$.
equal(a, b)	true if $\|a - b\| \leq \epsilon$.
greater(a, b)	true if $a > b + \epsilon$.
less(a, b)	true if $a < b - \epsilon$.
greaterequal(a, b)	true if $a \geq b - \epsilon$.
lessequal(a, b)	true if $a \leq b + \epsilon$.
compare(a, b)	-1 if $a < b - \epsilon$, 0 if $\|a - b\| \leq \epsilon$, 1 if $a > b + \epsilon$.
isbounded(a, inf, sup)	true if $inf \leq a \leq sup$.
truncate(a, inf, sup)	truncate $a \in [inf, sup]$.
round(a)	round a to an integer.
fraction(a, b)	return fraction of a/b

Fresnel integrations are implemented with power series expansions 6.1 to calculate the Euler spiral. The terms are summed up until a desired precision is reached. The factorial values are pre-defined from 1! to 20!, because the terms converge rapidly to zero.

$$\begin{aligned} S(x) &= \int_0^x \sin t^2 dt = \sum_{n=0}^{\infty} (-1)^n \frac{x^{4n+3}}{(2n+1)!(4n+3)} \\ C(x) &= \int_0^x \cos t^2 dt = \sum_{n=0}^{\infty} (-1)^n \frac{x^{4n+1}}{(2n)!(4n+1)} \end{aligned} \tag{6.1}$$

6.2.2 Geometric Models and Operations

Angles are represented in a range of $(-\pi, \pi]$ or $[0, 2\pi)$. Modulo functions are provided to convert an angle into these ranges. Some constants are defined, such as π, 2π, and the ratio for rad and degree conversions. The difference between two angles are calculated with a function considering the 2π period and returns a value in $(-\pi, \pi]$.

The spacial and geometric property of the robot and obstacles are important for collision checks and distance queries. An abstract class *Shape* holds the general geometric operation interfaces declared as in Table 6.2. All the operations are performed in the two-dimensional space.

Table 6.2: Geometric Operations

translate(x, y)	translate the shape with a displacement (x, y).
rotate(α)	rotate the shape with an angle α.
transform(x, y, α)	transform the shape with a transformation (x, y, α).
mapping(x, y, α)	map the shape to a frame at (x, y, α).
boundingBox()	return the bounding box of the shape.
inside(S)	true if the shape is inside of another shape S.
collide(S)	true if the shape collides with another shape S.
distance(S)	return the distance to the shape S.

Several types, such as *Point*, *Pose*, *Line*, *Circle*, and *Rectangle*, are derived from the *Shape* class for the concrete geometry. The geometry of a robot and obstacles can be represented with a shape or a group of shapes. The motion of an object is calculated as transformation of the shapes.

- *Point*: a point at position (x, y) in two-dimensional space.
- *Pose*: a vector at position (x, y) with an orientation α in two-dimensional space, inherited from *Point*.
- *Line*: a line segment in two-dimensional space with two points for the termini.
- *Circle*: a circle at position (x, y) in two-dimensional space with a radius r, inherited from *Point*.
- *Rectangle*: a rectangle at position (x, y) with an orientation α, whose size is defined with width w and length l, inherited from *Pose*.

6.2.3 Random Distributions and Generators

Random number generators are implemented to obtain random configurations for the sampling-based methods or random benchmarking scenarios. The random distributions and random number generators encapsulate the according modules of the boost library. Two types of random number generators are implemented:

- *Uniform random generator* generates uniform distributed numbers in a certain range.
- *Normal random generator* generates normal distributed numbers regarding given mean and σ values.

In addition, the both distributions provide a probability density function, a cumulative distribution function, and a quantile function for advanced usage.

6.2.4 Matrix and Grid

A two-dimensional grid is developed for a grid map environment model, which discretizes the space with constant resolution in x and y dimensions. The core data type is a two-dimensional matrix with real values that hold the confidence values for an occupancy grid. The grid is also used in A* search algorithms to discretize the configuration space.

6.3 Vehicle Models

As introduced in the section 2.3.2, the car-like robot kinematics and dynamics can be modelled in different degrees. These models are implemented in the *Vehicle* section of the library. The *State* class and the classes derived from it hold the robot state variables. The motions are defined with control inputs of *Segment* class that build up a *Path*. The execution of a motion, i.e., the robot state transformation regarding its kinematics, is implemented in the *Vehicle* class and its descendants. The shortest path between the configurations are provided by the *Metric* class.

6.3.1 States and Trajectory

The robot state variables such as position, orientation, velocity, etc. are contained in *State* class. For the according kinematic models, *KinState*, *KCState*, *KDState*, and *STState* are defined as follows.

- *KinState*: position (x, y) and orientation α for the constant curvature model in 2.3.2.1, inherited from *State*.
- *KCState*: position (x, y), orientation α, and curvature k for the continuous curvature model in 2.3.2.2, inherited from *KinState*.
- *KDState*: position (x, y), orientation α, curvature k, and speed v for the kinodynamic curvature model in 2.3.2.3, inherited from *KCState*.
- *STState*: position (x, y), orientation α, front wheel steering angle β, and speed v for the single track model in 2.3.2.4, inherited from *KinState*.

Furthermore, a class *TrajState* is defined for trajectory points derived from *KDState* class with a timestamp t.

6.3.2 Segments and Path

The primitive motions, i.e., the fundamental motion vector of a kinematic model, is abstracted as the *Segment* class. There are four types of path segments for the corresponding kinematic models.

- *KinSegment*: travel distance s and curvature k for the constant curvature model in 2.3.2.1, inherited from *Segment*.
- *KCSegment*: travel distance s and steering u regarding distance, for the continuous curvature model in 2.3.2.2, inherited from *Segment*.
- *KDSegment*: moving duration t, acceleration a, and steering u regarding time for the kinodynamic curvature model in 2.3.2.3, inherited from *Segment*.
- *STSegment*: moving duration t, acceleration a, and the angular steering velocity of the front wheel ω for the single track model in 2.3.2.4, inherited from *Segment*.

The steering input in *KCSegment* is the curvature change proportional to distance, while in *KDSegment* is the curvature change per unit time. A *Segment* is a single motion vector. A list of segments consists a *Path*, which can represent complex motions.

6.3.3 Vehicle Motions

A *Vehicle* class holds the according *State* and can perform the motion of a *Segment* or a *Path* that changes the state. A *Vehicle* takes constant parameters in addition for the according kinematic model, which define the symmetric bounds of the states and motion inputs as follows.

- *KinVehicle*: the maximum curvature $k_{\max}$ for the constant curvature model in 2.3.2.1, inherited from *Vehicle*.
- *KCVehicle*: the maximum curvature $k_{\max}$ and the maximum steering speed $u_{\max}$ for the continuous curvature model in 2.3.2.2, inherited from *Vehicle*.
- *KDVehicle*: the maximum speed $v_{\max}$, the maximum curvature $k_{\max}$, the maximum acceleration $a_{\max}$, the maximum steering $u_{\max}$, and the integration time-step Δt for the kinodynamic curvature model in 2.3.2.3, inherited from *Vehicle*.
- *STVehicle*: the vehicle wheel base l, the maximum speed $v_{\max}$, the maximum steering angle $\beta_{\max}$, the maximum acceleration $a_{\max}$, the maximum steering speed $w_{\max}$, and the integration time-step Δt for the single track model in 2.3.2.4, inherited from *Vehicle*.

A *Vehicle* can validate whether a state is in the valid range. A segment can be adapted by a vehicle so that it generates a valid state, i.e., the motion is valid.

6.3.4 Configuration Metrics

For the kinematic models with explicit configuration space metric, such as *KinVehicle* and *KCVehicle*, it is possible to explicitly calculate the shortest path between two states. In addition, a pseudo distance function is defined for a rough distance estimation, e.g., Euclidean distance considering only the state positions. These distance metrics are useful for the heuristic estimations and direct state expansion towards the goal of the planning algorithms.

- *KinMetric*: the metric of constant curvature model in 2.3.2.1 is the Reeds-Shepp [14] shortest path between two configurations, inherited from *Metric*..
- *KCMetric*: the metric of continuous curvature model in 2.3.2.2 is the modified Reeds-Shepp path with clothoid blending in [15], inherited from *Metric*.
- *KDMetric*: the kinodynamic curvature model in 2.3.2.3 does not have an explicit metric. However, if the motion velocity is constant, it degenerates to a continuous curvature model, which can be calculated using *KCMetric*. It is derived from *Metric*.
- *STMetric*: the single track model in 2.3.2.4 is similar to *KDMetric*, which can be obtained only in certain conditions. It is also a descendant from *Metric*.

A flag defines whether a metric considers only forward driving or motion in both directions. If the robot is only allowed to move forward, Dubins curves [13] are applied instead of the Reeds-Shepp metric for the shortest path. Another important function of the metric is to calculate the weight of a path with three types of costs: a multiplicative cost for backwards driving, a multiplicative cost for steering, and an additive cost for cusp motions.

6.4 Environment Models

The motion planner investigates the environment by querying whether a position is occupied by an object, whether the robot collides with an obstacle, or how large is the distance to the nearest obstacle. A general environment model is defined to provide the basic interface for these kinds of queries as in Table 6.3. The concrete implementation and data format can be different regarding the information sources, e.g., an *Object List* for a radar sensor, or a *Grid Map* for a laser scanner.

Table 6.3: Environment Queries

collide(O)	collision check for object O in a static environment.
collide(O, t)	collision check for object O in a dynamic environment at time t.
collide($O, t, \Delta t$)	collision check for object O in a dynamic environment in time duration $[t, t + \Delta t]$.
distance(O)	distance query for object O in a static environment.
distance(O, t)	distance query for object O in a dynamic environment at time t.
distance($O, t, \Delta t$)	distance query for object O in a dynamic environment in time duration $[t, t + \Delta t]$.

6.4.1 Object List

An *Object List* models the environment as a number of objects. The rest of the space is considered as collision-free. An *Object* has a *Shape* holding its position and geometry. The object motion is represented with a speed vector. Thus, it is straightforward to identify a static object, and estimate the position of a dynamic object in the future. Collision checks and distance queries are implemented based on the geometry operations. For functions with time duration, the sweeping area of a dynamic object is considered.

The memory requirement of an object list is linear to the number of objects, and so is the computational complexity of collision checks and distance queries. Furthermore, it is convenient to model moving obstacles in an object list.

6.4.2 Grid Map

A *Grid Map* represents the whole environment as an occupancy-grid. Obstacles are mapped to a grid cells with a certain resolution. Every grid cell has a flag or confidence value to indicate whether the location is taken or not. In this case, it is not easy to model a moving object in a grid. However, the grid cells can hold other information, such as distance to the nearest obstacle, the estimated distance to the goal, etc. In this case, the planner can exploit the additional knowledge for motion planning, such as the grid-based distance heuristic for the Hybrid A* algorithm.

A grid map is efficient for collision checks, as only the grid cells overlapping with the object shape need to be checked. A distance query could also be simple as directly returning the pre-calculated clearance value of a grid cell. However, to build a grid map with distance knowledge takes the overhead of dynamic programming. In addition, comparing to an object list, a grid map usually consumes a constant amount of memory to store the grid data. The memory

requirement is proportional to the grid resolution and the environment size. In this case, a grid map should balance the complexity and completeness by choosing a suitable resolution.

6.5 Motion Planning

The motion planning methods are developed in the *Plan* packet, including different space exploration and heuristic search algorithms for the SEHS approach, and several search-based or sampling-based planning algorithms as references for benchmarking. The planners are developed with generic interfaces to vehicle models and environment models. Therefore, the planners can be easily applied for all kinds of vehicle models and environment models in the *AutoDrive* library.

6.5.1 Planner Abstraction

An abstract planner class is defined as basis for the advanced planning algorithms, in order to simplify the implementation, demonstration, and benchmarking. In addition, a problem definition is provided for the problem specific data, such as the vehicle model, the start and goal states, the vehicle geometry, etc. The planner specific configurations are contained in a planning parameter structure, e.g., the maximum number of iterations, the cost factors. A planner is initialized with a problem definition and a set of parameters.

A planning procedure starts with an *initial* method, which performs tasks such as validating the problem definition, initializing the planning sequence, etc. Then, the planning algorithm is called by an *iterate* method to carry out the planning steps until a final condition is reached, which is verified with a *terminate* function. The motion planning result is a *Path*, while the number of iterations and expanded nodes are recorded for benchmarking purpose.

Furthermore, a generic heuristic search object is developed as a basis for the algorithms in SEHS approaches. The generic heuristic search is a graph search algorithm with nodes and edges. A node with a heuristic cost h, an actual cost g, and a pointer to its parent node is defined. An open-set and a closed-set are defined as queues of nodes to hold the graph. The nodes are sorted regarding to the total cost with a *priority_queue* in the open-set. The general heuristic search procedure is modeled with several pure virtual methods for nodes expansion, heuristic estimation, directly goal expansion, and examining whether a state is already evaluated.

The collision checks and distance quires are encapsulated in an abstract class named *Validation*. It provides interfaces for the planners to access the space knowledge. By inheriting from this class, different environment models can be applied to representing the prior knowledge and post perception information.

Interfaces for visualization are provided with an abstract class *Vision*. A planner can draw the search graph and the search result with it. By inheriting from the *Vision* class, different types of visualizations can be implemented for the demonstrations.

6.5.2 Heuristics

The heuristic cost estimation is the key component for space exploration and heuristic search in SEHS methods. Different types of heuristics and their combinations can be used in a heuristic search algorithm. Therefore, an abstraction of heuristic is defined to providing the generic interfaces to the planners. The basic function of a heuristic is estimating the cost from a start state to the goal state. The following heuristics are developed in the library:

- *EuclideanHeuristic*: The heuristic cost is the Euclidean distance between the state position and the goal position.

- *GridHeuristic*: If the environment model is a grid map, grid-based distance to a goal cell can be calculated with a dynamic programming method considering the occupied cells. This grid-based distance can serve as an optimistic cost estimation regardless of vehicle kinematics.
- *MetricHeuristic*: For a kinematic model with explicit metric in configuration space, a cost estimation without considering the obstacles can be directly obtained. In addition, different cost coefficients can be applied to weight the metric motions, e.g., extra cost for reverse motion or switching the motion direction. Thus, the nodes with desired motions are prioritized in the search.
- *BubbleHeuristic*: Bubble heuristic is the heuristic cost defined in Section 2.3.3.1 is implemented, which is calculated based on the circle-based space exploration result.
- *DBHeuristic*: Directed bubble heuristic is the heuristic cost defined in Section 3.3.1 regarding the orientation-aware space exploration result.
- *TBHeuristic*: Time bubble heuristic is the heuristic cost defined in Section 4.3.1 based on the space-time exploration result.
- *KBHeuristic*: Knowledge bubble heuristic is the heuristic cost for SEHS methods with additional traffic knowledge, such as steering behaviors for tuning and lane switching.

Furthermore, the heuristics for SEHS methods are able to map a state to a space geometric entity, e.g., a circle or a cylinder, and provide information about free space dimension such as circle radius. Thus, the SEHS approach can adapt the search step size and state resolution.

6.5.3 Space Explorers

The space exploration in the SEHS approach is a special version of heuristic search. Instead of producing a sequence of motions, it returns a sequence of geometric shapes. Therefore, the nodes involved in the heuristic search are geometric shapes instead of robot states. Different types of nodes are defined as follows.

- *Bubble*: The basic class as a circle in two-dimensional space.
- *DirectedBubble*: It descents from *Bubble* with an orientation angle for the orientation-aware space exploration in Section 3.2.
- *TimeBubble*: It descents from *Bubble* with a timestamp and a time duration values for the space-time exploration in Section 4.2.
- *KnowledgeBubble*: It extends *TimeBubble* with additional flags for different driving behaviors in Section 5.3.1.2.

According to the exploration algorithms introduced in the previous sections, the space explorers are implemented with different node expansion procedures according to the bubble type. In addition, it can take a list of bubbles as the initial condition, and adapts the path corridor incrementally.

- *BubbleExplorer*: The basic version of space exploration explores from the start position to the goal position with *Bubble* in Section 2.2.

- *BDBExplorer*: A bidirectional explorer combines two *BubbleExplorer*, which explores simultaneously from the start or goal position respectively. The iteration method calls the planning functions of the both explores in turn. In every step, the selected bubble of one exploration is checked with the closed-set of the other for overlapping. If so, a path corridor is constructed with the results from the both explorers.
- *DBExplorer*: An orientation-aware explorer in Section 3.2 works with *DirectedBubble*.
- *TBExplorer*: A space-time explorer in Section 4.2 proceeds with *TimeBubble*.

6.5.4 Heuristic Search Planners

Several heuristic search planners are implemented to plan the robot motion with a kinematic model and a heuristic estimation. The planners apply different node definitions, node expansion and resolution methods. The following heuristic search planners are provided by the library.

- *HSPlanner*: A general heuristic search planner takes a generic heuristic and a set of motion primitives with constant step size for states expansion. The states are resolved with a constant metric resolution.
- *AstarPlanner*: The Hybrid A* algorithm [52] resolves the states with a discretized grid in the configuration space. The states are expanded with a set of motion primitives with constant step size.
- *BubblePlanner*: The basic heuristic search method of the SEHS approach in Section 2.3.3. The primitive motions adapt their step size according to the bubble size. The resolution is also dynamic and grid-free.
- *DBPlanner*: An extended version of *BubblePlanner* for *DBHeuristic*, which adapts the primitive motions to the orientation of the bubbles as in Section 3.3.
- *TBPlanner*: An extended version of *BubblePlanner*, which applies the *TBHeuristic* as in Section 4.3.
- *KBPlanner*: The knowledge-based version of *TBPlanner*, which adapts the primitive motions to the knowledge from *KBHeuristic* as in Section 5.3.1.2.

6.5.5 Sampling Based Planners

The sampling based methods are implemented for comparing and benchmarking with the SEHS methods in the same conditions of robot kinematics and environment models including collision checks. Two types of sampling based methods, PRM and RRT, are included in this library.

The PRM planner [32] requires a kinematic model providing explicit metric motions in the configuration space, as the planner needs to connect random sampled states. Therefore, it is only applicable for the *KinVehicle* and *KCVehicle* kinematics. The planner tries to connect a random generated state with the existing map vertices in a certain range. The roadmap is maintained with an *adjacency_list* from the *boost* library. After building the map, a Dijkstra shortest path query is employed to find the shortest path for the result. The PRM planner also accepts a predefined roadmap to acceleration the exploration phase.

The RRT planner [33] constructs a random tree during the planning. Instead of explicit metric motions, a group of primitive motions are applied to propagate the states either towards a random sample or the goal state. A parameter controls the proportion between the two types

of vertex expansions. When the goal state is reached, the planner traces back to the start state to create the result. The EET method [69] is also implemented, which takes the result from *BubbleExplorer* to improve the effectiveness of the random sampling. If the kinematic model has explicit metric motions, RRT can be extended to RRT* [44], which optimizes the result incrementally within a dynamic range.

6.5.6 Trajectory Generator

The result of motion planning in *AutoDrive* library is a *Path*, which is a list of *Segment*s with control inputs. Generally, the robot motion is controlled by a trajectory controller to follow the path. A trajectory controller usually requires a sequence of reference points to calculate the control commands. A trajectory is a list of robot states with timestamps. *AutoDrive* library provides a simple version of trajectory generator, which converts a path to a trajectory.

For the kinodynamic models, *KDVehcile* and *STVehcile*, it is easy to generate robot states with equal time interval along the path. The *KinVehicle* and *KCVehicle* models do not consider velocity and time. Therefore, it requires a boundary condition and a velocity profile to generate a trajectory. A simple velocity profile contains the maximum acceleration, deceleration, velocity, and yaw-rate. Regarding a start velocity, a robot accelerates to the maximum velocity if possible, and then decelerates to the velocity of the goal state. Thus, a simple trajectory is generated with bang-bang controls. The velocity profile can be extended for more convenient motions.

6.5.7 Task Planner

The task planning is a symbolic planning algorithm. However, it shares the basic heuristic search principle with the motion planners. A simple domain definition is constructed according to the discussion in Section 5.2.1. Abstract classes are defined for *Object*, *Place*, and *Task*. The predicates are implemented as methods of the domain entities. A world model consists of objects and places. The ego vehicle state is contained in a vehicle object in the world model. A task planner is implemented based on the generic heuristic search with system states for the search nodes as Algorithm 5.

6.6 Demo in 3D Scene

The *AutoDrive* library provides the core functions of motion planning for car-like robots. In order to demonstrate and verify the planning algorithms in different scenarios, a set of demo components are developed with the focus of handling the scenario and planning configurations, visualizing the scenarios and planning progress, performing the planning and the result motion, and providing a control GUI to the user.

- *DemoSetup*: A planning scenario is described with a configuration file in XML format. A world model, a kinematic model, the start and goal configurations, the exploration and planning parameters, and a trajectory profile are included in a configuration file for a general planning scenario. A *DemoSetup* entity loads and verifies these configurations. The parameters are prepared as public members for easy access. The *DemoSetup* can be extended for additional parameters of motion planning engine, or task planners.

- *DemoScene*: A planning scenario and the planning procedure can be visualized with a *DemoScene* component. As the motion planning could be performed in space-time domain, or including the orientation during the exploration, a 3D scene is implemented based on

the *Coin3D* and *SoQt* libraries. It implements all the virtual methods of the abstract *Vision* class as Qt slots. Objects such as obstacles, the start and goal configurations, the search tree, and the result trajectory are processed in different groups. Thus, it is convenient to select the visual combinations, or update the pose of an object. The whole scene can be captured in a picture file.

- *DemoPlanner*: The planning is carried out in an independent thread. It initializes the planner, and then calls the iterate method in a loop. The planning can also run in a slow motion by waiting a moment between the iterations. In addition, *DemoPlanner* also inherits from *Vision* and *Validation* to provide interfaces to the visualization and an environment model. The *Validation* methods, e.g., collision checks and distance queries are implemented by accessing a pointer to an environment model. The *Vision* functions emit signals, which can be connected to the slots of *DemoScene* to visualize the planning elements. The planning results are also communicated through signals and slots.

- *DemoVehicle*: A simple vehicle thread is implemented, which sends signals to update the vehicle pose regarding a given trajectory. With the planner thread and the vehicle thread, the planning and execution run parallel. It is important for the *Motion Planning Engine* demos, as the vehicle keeps driving while the planner is verifying or re-planning the motion.

- *DemoWidget*: All the components mentioned above are integrated in a control widget. It connects the signals and slots between them, and provides a Qt GUI for the demonstration, e.g. load configuration, generate random start and goal configurations, start planning, start driving, capture image. The planning performance is evaluated with average planning time, number of iterations, number of search nodes, number of collision checks, etc.

All the basic demo components can be extended for advanced demonstrations, for example, a *Motion Planning Engine* demo, a task planning demo, or the application scenarios in Section 5.4. All the experiments result and the figures in this thesis are produced with this framework.

6.7 Summary

The *AutoDrive* library focuses on the specific motion planning for car-like robots. It provides useful tools and a generic codebase to develop and evaluate different motion planning algorithms. In addition, the scenarios can be easily configured and visualized with the demonstration framework. The goal of this library is contribution to motion planning and autonomous driving software development in education, research, and automotive industry.

Chapter 7

Conclusion

Motion planning for autonomous driving is an fascinating and challenging research topic. The system model is subject to the nonholonomic constraints of vehicle kinematics or dynamics. The environment is partially observed with dynamic obstacles and sensing uncertainties. Many traffic laws and common rules should be complied. Furthermore, the convenience of the human passengers and the interaction with other human drivers should be considered. On one hand, these constraints and rules make it difficult to directly search the configuration space for an optimal motion solution. On the other hand, the various kinds of knowledge can aid a planning agent from different aspects, so that a motion planning problem can be gradually approached in an effective manner.

Space Exploration Guided Heuristic Search provides a general framework to take advantage of the knowledge about subspace for motion planning. First of all, the workspace is investigated with a simple geometry based exploration procedure. Waypoints are generated in the two-dimensional workspace with collision-free distance to form a path corridor, which extracts the knowledge of workspace dimension and topology. Then, a heuristic is constructed based on the geometric path providing the cost estimation for the heuristic search algorithm, which propagates the vehicle states under kinematic or dynamic constraints following the guidance. Furthermore, the step size and states resolution are adapted to the free space dimension for a good trade-off between completeness and efficiency.

The SEHS framework can be extended to exploit many kinds of knowledge. In a low speed maneuvering scenario, the driving direction can be evaluated in the exploration phase with orientation-aware waypoints. In this case, the search procedure can improve the states propagation by adjusting the cost function or selecting the motion primitives regarding the preferred diving directions. In a high speed driving scenario, the moving objects in the environment can be evaluated in a space-time exploration procedure with time labeled waypoints. Thus, a path corridor is obtained in the three-dimensional space and time for a time-dependent motion strategy. Furthermore, the space exploration can be conducted with a prior known roadmap; and the primitive motions can be tailored for specific driving behaviors. All these kinds of domain knowledge about driving and traffic can contribute in the SEHS motion planning framework to improve the result planning time and motion quality.

In principle, the SEHS method is a heuristic search approach in a discretized state space. It does not apply all the information directly as heuristics for the search, but gradually builds the advanced knowledge based on the primitive ones. For example, the space exploration processes the environment information with the help of an Euclidean distance heuristic. The result serves as a path corridor heuristic for the following search procedure. The distance to obstacle

information is used to create an adaptive state space discretization, which can be further refined to improve the search completeness and the result quality. In this case, it takes benefit from the incremental search techniques to provide an anytime motion planning solution.

Although the SEHS algorithm can efficiently solve a large scale motion planning problem, due to sensing limitations and imperfect execution, motion planning for autonomous driving is usually performed within a space and time horizon in practice. In this case, a symbolic planning method is introduced to handle the long term decisions of the entire route and generate the context for the motion planning problems of the coming driving tasks. The domain knowledge of driving in the traffic is evaluated in the high-level task planning procedure, especially the traffic rules, in order to create a sequence of driving tasks for the low-level motion planning. The motion planning layer obtains a problem in an adequate scale with task information to select the right method to solve it. Thus, a hierarchical planning system is designed for the autonomous driving maneuvers.

In real-time autonomous driving applications, the motion planning is a continuous process. The vehicle motion should be maintained in the whole life time of a driving maneuver. The concept of *Motion Planning Engine* is introduced to integrate the planning agents in different levels and provide the planning as a service in an autonomous driving system. It not only plans the tasks and motions, but also continuously verifies them regarding the real-time information, and adjusts the plan when necessary to guarantee safety in any circumstances.

In order to implement and test the methods mentioned above, a software library called *AutoDrive* is developed especially for car-like robot planning applications. It provides software stacks for mathematical tools, kinematic models, environment models, and different planning algorithms including heuristic search based methods and random sampling approaches. *AutoDrive* is a self contained library, whose core functionality only depends on the C++ STLs and a few parts of the boost library. Thus, it can be easily applied in different development environments. Furthermore, the planning process can be visualized in a 3D demo program with different scenario configurations in XML format. With the generic class definitions in this library, it is convenient to develop further autonomous driving algorithms and modules for education, research, and practical applications.

This dissertation provides some general ideas, concepts, and methods for the upcoming autonomous driving applications. There is still much work to do for this promising future of artificial intelligence. A base of domain knowledge for autonomous driving should be built up including a complete set of traffic rules and driving tasks. Different planning methods, e.g., extensions of SEHS, should be developed to cover the requirements of the various driving maneuvers. The uncertainties in perception and execution should be addressed in planning, as well as the human experience and behaviors. When multiple autonomous vehicles participate in a same scenario, the planning should be optimized regarding multiple agents. It is a way of evolution, that the autonomous driving technology will gradually improve the quality of the modern transportation, change the industry landscape, and inspire new life style.

Appendix A

Vehicle Kinematic Models

The vehicle kinematic model defines the configuration space and the control space for the motion planning. Four vehicle kinematic models are introduced with increasing complexities for different motion planning problems.

A.1 Constant Curvature Model

The constant curvature model considers a constant turning radius during an atomic motion. The vehicle state can be described with a vector (x, y, θ), in which x and y for the position and θ for the orientation. Due to the minimum turning radius, a vehicle can only change the position and orientation regarding a motion curvature, which is defined in (2.6). As the radius can be calculated with an arc length s and angle α, a curvature can also be defined as (A.1). Hence, the curvature determines how a vehicle changes its orientation, while it is travelling.

$$k = \frac{1}{r} = \frac{1}{s/\alpha} = \frac{\alpha}{s} \tag{A.1}$$

If the curvature is constant during a motion, the equation of motion is (A.2) with v for the velocity.

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \cos\theta \\ \sin\theta \\ k \end{pmatrix} v \tag{A.2}$$

This equation can be integrated regarding a start configuration (x_0, y_0, θ_0). The result is (A.3) with s for the travel distance.

$$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} = \begin{pmatrix} x_0 + \frac{2}{k}\cos\frac{2\theta_0 + ks}{2}\sin\frac{ks}{2} \\ y_0 + \frac{2}{k}\sin\frac{2\theta_0 + ks}{2}\sin\frac{ks}{2} \\ \theta_0 + ks \end{pmatrix} \text{ with } s = vt \tag{A.3}$$

Without loss of generality, assuming that the motion starts from $(0, -1/k, 0)$, the states in (A.3) holds the condition $x^2 + y^2 = 1/k^2$, which means that the vehicle moves in a circle

with a radius $1/k$. When $k = 0$, the equation (A.3) degenerates to equation (A.4), i.e., the robot motion is a straight line with an angle θ_0.

$$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \\ \theta_0 \end{pmatrix} + \begin{pmatrix} \cos\theta \\ \sin\theta \\ 0 \end{pmatrix} s \text{ with } s = vt \tag{A.4}$$

As a result, a primitive motion with constant curvature can be determined with a travel distance $s \in \mathbb{R}$ and a curvature $k \in [k_{\min}, k_{\max}]$. The configuration space of the constant curvature kinematics model is (x, y, θ), and the control input is (s, k). The dynamic properties such as velocity and acceleration are ignored, so that the robot can change its moving direction and curvature immediately.

The control space of the constant curvature model is illustrated in Fig. A.1. The primitive motions can be divided into six groups: stop, driving forwards, driving backwards, driving forwards with turning left, driving backwards with turning left, driving forwards with turning right, and driving backwards with turning right. Each subset is a continuous subspace with infinite possible control inputs. Therefore, the control space is usually interpolated or sampled for a limited number of primitive motions.

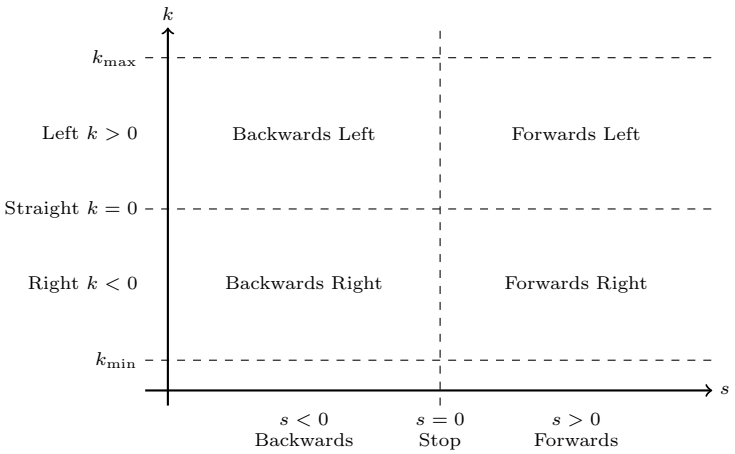


Figure A.1: Control space of a constant curvature model

A.2 Continuous Curvature Model

The continuous curvature model considers a constant curvature change variable in addition to the previous one. In this case, the curvature becomes a state variable. The configuration space is (x, y, θ, k).

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{k} \end{pmatrix} = \begin{pmatrix} \cos\theta \\ \sin\theta \\ k \\ 0 \end{pmatrix} v + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} u_t \tag{A.5}$$

The equation of motion is (A.5), when a time derivative of the curvature u_t is considered. There is no closed form to integrate the motion equation for an explicit solution of the vehicle

state. However, the ratio of a constant steering rate and a constant velocity yields the curvature change regarding the travel distance as u_s in (A.6). In this case, the curvature change is linear with the curve length, i.e., the vehicle moves in an Euler spiral. Many existing methods [121] are available to efficiently calculate the points on an Euler spiral.

$$u_s = \frac{u_t}{v} = \frac{dk/dt}{ds/dt} = \begin{cases} \dfrac{dk}{ds} & v \neq 0 \\ 0 & v = 0 \end{cases} \tag{A.6}$$

As an Euler spiral is defined with a curvature derivative with respect to the curve length, the robot motion of a continuous curvature model can be determined with a travel distance $s \in \mathbb{R}$ and a steering parameter $u_s \in [u_{s,\min}, u_{s,\max}]$. As a result, the control space of the continuous curvature model is (s, u_s).

The primitive motions are combinations of travel distance and steering, which can be divided into several groups as in Fig. A.2. The steering parameter u_s only decides the change rate of the curvature. In this case, a vehicle keeps its motion curvature when $u_s = 0$, i.e., it moves either in a straight line or a circle. The sign of the steering parameter indicates the steering direction. The sign of the curvature is related to the turning direction and the driving direction. Hence, a positive steering parameter increases the curvature (steering left) while driving forwards, and decreases (steering right) it when backwards. A negative steering parameter is vice versa. Furthermore, as the curvature k is bounded in $[k_{\min}, k_{\max}]$, a vehicle is not able to perform a steering motion when the curvature is already at its limit.

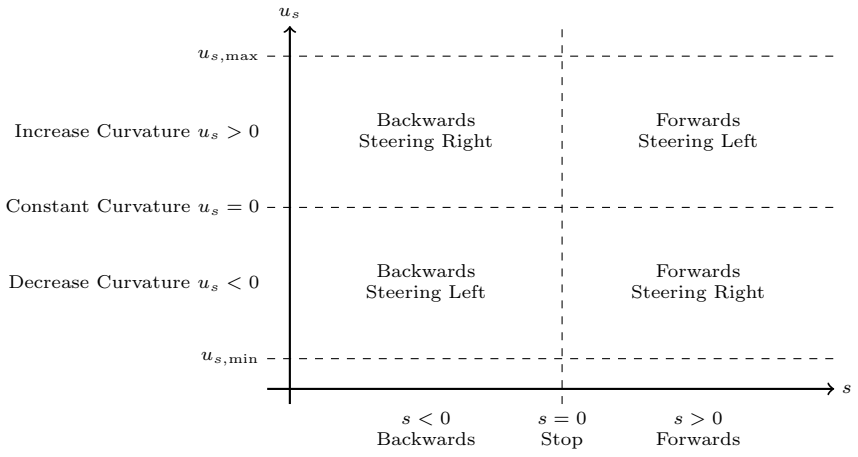


Figure A.2: Control space of a continuous curvature model

If $u_s = 0$, then u_t is also 0 regarding (A.6). The motion equation (A.5) degenerates to (A.2). Thus, the motion can be solved with a constant curvature model.

A.3 Kinodynamic Curvature Model

The kinodynamic curvature model considers a constant acceleration a in addition to the previous model. The velocity v becomes a state variable. The configuration space is (x, y, θ, k, v).

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{k} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} \cos\theta \\ \sin\theta \\ k \\ 0 \\ 0 \end{pmatrix} v + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} u_t + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} a \tag{A.7}$$

The equation of motion is (A.7). In this case, the time factor cannot be eliminated as in (A.6), because the velocity is a variable. As the first term on the right of (A.7) only consists of the state variables, a state may change with zero inputs, which is called the drift effect. No closed form is available to integrate (A.7). Numerical techniques should be applied to solve the motion equation by integrating a vehicle motion iteratively with a small time interval δt (A.8).

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \\ \theta_{i+1} \\ k_{i+1} \\ v_{i+1} \end{pmatrix} = \begin{pmatrix} x_i \\ y_i \\ \theta_i \\ k_i \\ v_i \end{pmatrix} + \begin{pmatrix} v\cos\theta \\ v\sin\theta \\ vk \\ u_t \\ a \end{pmatrix} \delta t \tag{A.8}$$

The control space is (u_t, a, t). u_t and a are bounded in $[u_{t,\min}, u_{t,\max}]$ and $[a_{\min}, a_{\max}]$ respectively. $t \in \mathbb{R}^+$ is the time duration of the motion. The primitive motions are defined in a three dimensional space. As the time t can only take positive values, the type of a primitive motion is determined by the other two parameters as illustrated in Fig. A.3. The input a only decides the change of the velocity, but not the driving direction. The sign of u_t determines the steering direction alone (left when positive and right when negative), because the time input t is always positive. For the constraints of the maximum motion curvature and velocity, whether a primitive motion is executable should be decided regarding the specific state.

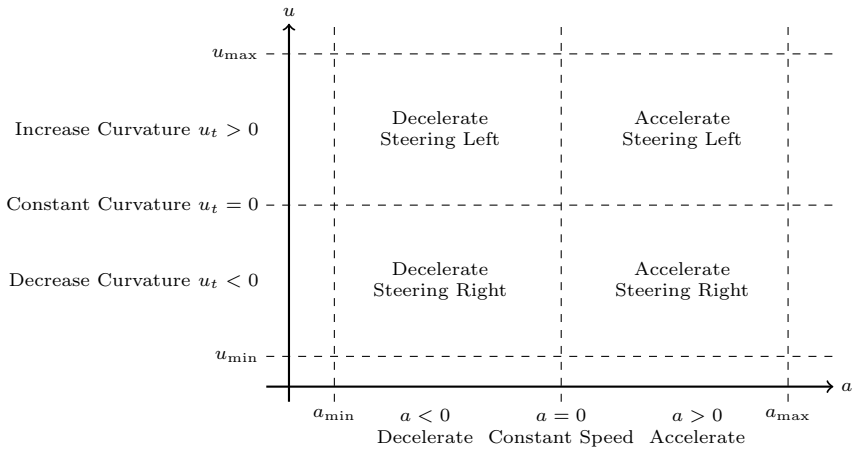


Figure A.3: Control space of a kinodynamic curvature model

When $a = 0$, the velocity remains unchanged. Thus, the kinodynamic curvature model degenerates to the continuous curvature model.

A.4 Single Track Model

The kinematic models with curvature mainly describe the geometric property of the vehicle motion. In practice, the physical steering mechanism determines the motion curvature by setting the angle of the wheels. The simplest one is the single track model, or known as the bicycle model.

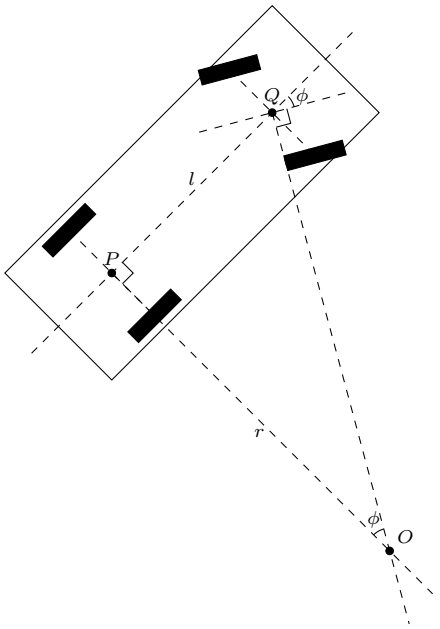


Figure A.4: Single track kinematics model

Fig. A.4 shows an example of a single track model with front-wheel steering. The reference point P is in the middle of the rear axis, where the position (x, y) of the vehicle is defined. The steering is performed by the front wheels. In fact, the two front wheels are set with slightly different angles regarding the Ackermann steering geometry. In the single track model, it is simplified to one steering angle ϕ at the center point Q of the front axis. Assuming there is no sliding between the wheels and the ground surface during the motion, the vehicle will travel in a circle, which is centered at the intersection point O of the perpendicular lines through P and Q regarding the directions of the wheels. As P is the reference point of the system, the turning radius r is the length of segment PO. In this case, the curvature k of the motion can be calculated based on the steering angle ϕ and the wheelbase l in (A.9).

$$k = \frac{1}{r} = \frac{\tan\phi}{l} \tag{A.9}$$

If the rear wheels are the driving wheels with a linear velocity of v, the equation of motion is (A.10). a is the linear acceleration, and ω is the steering speed of the front wheels. If the vehicle is front wheel driving, an equivalent velocity v is proportional to the linear speed of the front wheels v_{front} regarding the steering angle as $v = v_{\text{front}} \cos\phi$. In this case, only the cosine part of v_{front} contributes to the position change, while the sine part is responsible for the

orientation change.

$$\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\phi} \end{pmatrix} = \begin{pmatrix} \cos\theta \\ \sin\theta \\ \tan\phi/l \\ 0 \\ 0 \end{pmatrix} v + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} a + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \omega \tag{A.10}$$

There is no closed form solution for the equation of motion in (A.10). Only numerical integration can be applied with a small time step δt similar to the kinodynamic curvature model. A transform between a single track model and a curvature-based model is not trivial, as the relationship between the steering angle and the motion curvature is nonlinear. Therefore, the model does not degenerate to a continuous curvature model when $a = 0$. But when $\omega = 0$, the curvature is invariant as a constant curvature model.

The configuration space of the single track model is (x, y, θ, v, ϕ). The control space is (a, ω, t). The primitive motions are similar to the kinodynamic curvature model as in Fig. A.5.

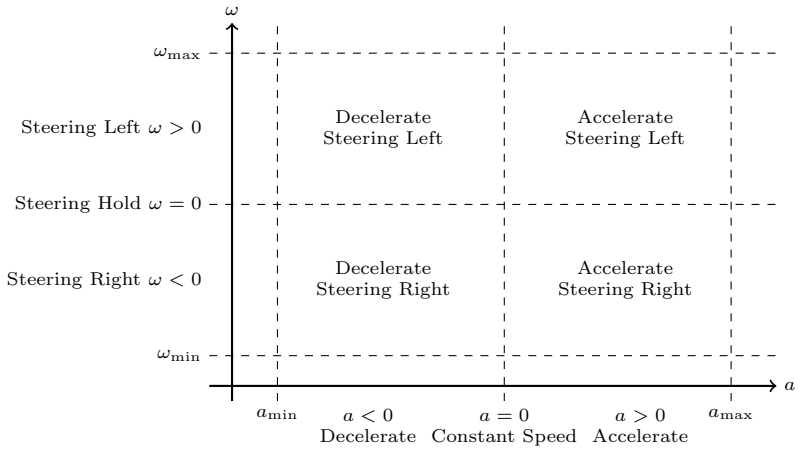


Figure A.5: Control space of a single track kinodynamic model.

References

[1] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2009. 1, 4, 9

[2] Steven LaValle. *Planning Algorithms*. Cambridge University Press, 2006. 1, 4

[3] Bruno Siciliano and Oussama Khatib, editors. *Springer Handbook of Robotics*. Springer, 2008. 1

[4] Jean-Paul Laumond, editor. *Robot Motion Planning and Control*. Springer, 1998. 1

[5] Bruce Donald, Patrick Xavier, John Canny, and John Reif. **Kinodynamic Motion Planning**. *Journal of the ACM*, **40**(5):1048–1066, 1993. 1, 8

[6] Markus Maurer, Christian Gerdes, Barbara Lenz, and Hermann Winner, editors. *Autonomes Fahren*. Springer, 2015. 2

[7] **The Home Page of the 2004 DARPA Grand Challenge**, 2004. `http://archive.darpa.mil/grandchallenge04/`. 2

[8] **The Home Page of the 2005 DARPA Grand Challenge**, 2005. `http://archive.darpa.mil/grandchallenge05/`. 2

[9] **The Home Page of the 2007 DARPA Urban Challenge**, 2007. `http://archive.darpa.mil/grandchallenge/index.html`. 2

[10] Azim Eskandarian, editor. *Handbook of Intelligent Vehicles*. Springer, 2012. 2

[11] Hermann Winner, Stephan Hakuli, Felix Lotz, and Christina Singer. *Handbook of Driver Assistance Systems: Basic Information, Components and Systems for Active Safety and Comfort*. Springer International Publishing, 2015. 2

[12] Howie Choset, Kevin Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia Kavraki, and Sebastian Thrun. *Principles of Robot Motion : Theory, Algorithms, and Implementation*. The MIT Press, 2005. 4

[13] L. Dubins. **On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents**. *American Journal of Mathematics*, **79**(3):497–516, 1957. 5, 24, 25, 98

[14] J. Reeds and L. Shepp. **Optimal Paths For a Car that Goes both Forwards and Backwards**. *Pacific Journal of Mathematics*, **145**(2):367–393, 1990. 5, 24, 25, 98

[15] Thierry Fraichard and Alexis Scheuer. **From Reeds and Shepp's to Continuous-Curvature Paths**. *IEEE Transactions on Robotics*, **20**:1025–1035, December 2004. 5, 25, 98

[16] Oussama Khatib. **Real-Time Obstacle Avoidance for Manipulators and Mobile Robots**. *The International Journal of Robotics Research*, **5**(1):90–98, 1986. 5

[17] Ronald Arkin. **Motor Schema - Based Mobile Robot Navigation**. *The International Journal of Robotics Research*, **8**(4):92–112, 1989. 5

[18] Rodney Brooks. **Intelligence without Representation**. *Artificial Intelligence*, **47**:139–159, 1991. 5, 8

[19] Rodney Brooks. **Intelligence without Reason**. In *International Joint Conference on Artificial Intelligence*, pages 569–595, August 1991. 5, 8

[20] Johann Borenstein and Yoram Koren. **Real-Time Obstacle Avoidance for Fast Mobile Robots**. *IEEE Transactions on Systems, Man and Cybernetics*, **19**(5):1179–1187, 1989. 5

[21] Johann Borenstein and Yoram Koren. **The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots**. *IEEE Transactions on Robotics and Automation*, **7**(3):278–288, 1991. 5

[22] Yong Hwang and Narendra Ahuja. **A Potential Field Approach to Path Planning**. *IEEE Transactions on Robotics and Automation*, **8**(1):23–32, 1992. 5

[23] Elon Rimon and Daniel Koditschek. **Exact Robot Navigation Using Artificial Potential Functions**. *IEEE Transactions on Robotics and Automation*, **8**(5):501–518, 1992. 5

[24] Stefan Gehrig and Fridtjof Stein. **Collision Avoidance for Vehicle-Following Systems**. *IEEE Transactions on Intelligent Transportation Systems*, **8**(2):233–244, 2007. 5

[25] Eric Rossetter. *A Potential Field Framework For Active Vehicle Lanekeeping Assistance*. PhD thesis, Stanford University, 2003. 5

[26] Jérôme Barraquand and Jean-Claude Latombe. **Robot Motion Planning: A Distributed Representation Approach**. *The International Journal of Robotics Research*, **10**(6):628–648, 1991. 5

[27] Jin-Oh Kim and Pradeep Khosla. **Real-Time Obstacle Avoidance Using Harmonic Potential Functions**. *IEEE Transactions on Robotics and Automation*, **8**(3):338–349, 1992. 5

[28] David Conner, Alfred Rizzi, and Howie Choset. **Composition of Local Potential Functions for Global Robot Control and Navigation**. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3546–3551, October 2003. 5

[29] David Conner, Howie Choset, and Alfred Rizzi. **Integrated Planning and Control for Convex-bodied Nonholonomic systems using Local Feedback Control Policies**. In *Proc. Robotics: Science and Systems*, pages 57–64, August 2006. 5

[30] Sean Quinlan and Oussama Khatib. **Elastic Bands: Connecting Path Planning and Control**. In *Proc. IEEE International Conference on Robotics and Automation*, pages 802–807, May 1993. 5, 7, 8

[31] Oussama Khatib, H. Jaouni, Raja Chatila, and Jean-Paul Laumond. **Dynamic Path Modification for Car-Like Nonholonomic Mobile Robots**. In *Proc. IEEE lntemational Conference on Robotics and Automation*, pages 2920–2925, April 1997. 5

[32] Lydia Kavraki, Petr Švestka, Jean-Claude Latombe, and Mark Overmars. **Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces**. *IEEE Transactions on Robotics and Automation*, **12**(4):566–580, 1996. 6, 102

[33] Steven LaValle. **Rapidly-Exploring Random Trees: A New Tool for Path Planning**. Technical report, Department of Computer Science, Iowa State University, October 1998. 6, 102

[34] Steven LaValle and James Kuffner Jr. **Randomized Kinodynamic Planning**. *The International Journal of Robotics Research*, **20**(5):378–400, 2001. 6, 8, 33, 49

[35] James Kuffner Jr. and Steven LaValle. **RRT-Connect: An Efficient Approach to Single-Query Path Planning**. In *Proc. IEEE International Conference on Robotics and Automation*, pages 995–1001, April 2000. 6

[36] Maciej Kalisiak and Michiel van de Panne. **RRT-blossom: RRT with a local flood-fill behavior**. In *Proc. IEEE International Conference on Robotics and Automation*, pages 1237–1242, May 2006. 6

[37] Andrew Ladd and Lydia Kavraki. **Motion planning in the Presence of Drift, Underactuation and Discrete System Changes**. In *Proc. Robotics: Science and Systems*, pages 233–241, June 2005. 6, 39

[38] Alexander Shkolnik, Matthew Walter, and Russ Tedrake. **Reachability-Guided Sampling for Planning Under Differential Constraints**. In *Proc. IEEE International Conference on Robotics and Automation*, pages 2859–2865, May 2009. 6

[39] Alejandro Perez, Robert Platt Jr., George Konidaris, Leslie Kaelbling, and Tomas Lozano-Perez. **LQR-RRT* : Optimal Sampling-Based Motion Planning with Automatically Derived Extension Heuristics**. In *Proc. IEEE International Conference on Robotics and Automation*, pages 2537–2542, May 2012. 6

[40] Erion Plaku, Lydia Kavraki, and Moshe Vardi. **Discrete Search Leading Continuous Exploration for Kinodynamic Motion Planning**. In *Proc. of Robotics: Science and Systems*, June 2007. 6

[41] Ioan Şucan and Lydia Kavraki. **Kinodynamic Motion Planning by Interior-exterior Cell Exploration**. In *Workshop on the Algorithmic Foundations of Robotics*, December 2008. 6, 39

[42] Ioan Şucan and Lydia Kavraki. **A Sampling-Based Tree Planner for Systems with Complex Dynamics**. *IEEE Transactions on Robotics*, **28**(1):116–131, 2012. 6

[43] Leonard Jaillet, Judy Hoffman, Jur van den Berg, Pieter Abbeel, Josep Porta, and Ken Goldberg. **EG-RRT: Environment-Guided Random Trees for Kinodynamic Motion Planning with Uncertainty and Obstacles**. In *Proc. of*

IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 2646–2652, September 2011. 6

[44] Sertac Karaman and Emilio Frazzoli. **Sampling-based Algorithms for Optimal Motion Planning**. *The International Journal of Robotics Research*, **30**(7):846–894, 2011. 6, 103

[45] Peter Hart, Nils Nilsson, and Bertram Raphael. **A Formal Basis for the Heuristic Determination of Minimum Cost Paths**. *IEEE Transactions on Systems Science and Cybernetics*, **4**(2):100–107, 1968. 6

[46] Anthony Stentz. **Optimal and Efficient Path Planning for Partially-Known Environments**. In *Proc. IEEE International Conference on Robotics and Automation*, pages 3310–3317, May 1994. 7, 8

[47] Sven Koenig, Maxim Likhachev, and David Furcy. **Lifelong Planning A***. *Artificial Intelligence Journal*, **155**(1-2):93–146, 2004. 7, 8

[48] Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun. **Anytime Dynamic A*: An Anytime, Replanning Algorithm**. In *International Conference on Automated Planning and Scheduling*, pages 262–271, June 2005. 7

[49] Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun. **Anytime Search in Dynamic Graphs**. *Artificial Intelligence Journal*, **172**(14):1613–1643, 2008. 7, 8

[50] Dave Ferguson and Anthony Stentz. **Field D*: An Interpolation-Based Path Planner and Replanner**. In *Proc. International Symposium on Robotics Research*, pages 1926–1931, October 2004. 7

[51] Dmitri Dolgov and Sebastian Thrun. **Autonomous Driving in Semi-Structured Environments: Mapping and Planning**. In *Proc. IEEE International Conference on Robotics and Automation*, pages 3407–3414, May 2009. 7, 33, 49

[52] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. **Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments**. *The International Journal of Robotics Research*, **29**(5):485–501, 2010. 7, 14, 23, 102

[53] Maxim Likhachev and Dave Ferguson. **Planning Long Dynamically Feasible Maneuvers for Autonomous Vehicles**. *The International Journal of Robotics Research*, **28**(8):933–945, 2009. 7, 23

[54] Matthew McNaughton and Chris Urmson. **FAHR: Focused A* Heuristic Recomputation**. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4893–4898, October 2009. 7

[55] Moritz Werling. *Ein neues Konzept fr die Trajektoriengenerierung und -stabilisierung in zeitkritischen Verkehrsszenarien*. PhD thesis, Karlsruher Institut fr Technologie, 2010. 7

[56] Moritz Werling, Sören Kammel, Julius Ziegler, and Lutz Gröll. **Optimal Trajectories for Time-critical Street Scenarios Using Discretized Terminal Manifolds**. *The International Journal of Robotics Research*, **31**(3):346–359, 2012. 7

[57] Matthew McNaughton, Chris Urmson, John Dolan, and Jin-Woo Lee. **Motion Planning for Autonomous Driving with a Conformal Spatiotemporal Lattice**. In *Proc. IEEE International Conference on Robotics and Automation*, pages 4889–4895, May 2011. 7

[58] Wenda Xu, Junqing Wei, John Dolan, Huijing Zhao, and Hongbin Zha. **A Real-Time Motion Planner with Trajectory Optimization for Autonomous Vehicles**. In *Proc. IEEE International Conference on Robotics and Automation*, pages 2061–2067, May 2012. 7

[59] Tianyu Gu, Jarrod Snider, John Dolan, and Jin-Woo Lee. **Focused Trajectory Planning for Autonomous On-Road Driving**. In *Proc. IEEE Intelligent Vehicles Symposium*, pages 547–552, June 2013. 7

[60] Tianyu Gu, John Dolan, and Jin-Woo Lee. **On-Road Trajectory Planning for General Autonomous Driving with Enhanced Tunability**. In *Proc. International Conference on Intelligent Autonomous Systems*, pages 247–261, July 2014. 7

[61] Tianyu Gu, Jason Atwood, Chiyu Dong, John Dolan, and Jin-Woo Lee. **Tunable and Stable Real-Time Trajectory Planning for Urban Autonomous Driving**. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 250–256, October 2015. 7

[62] Tomás Lozano-Pérez and Michael Wesley. **An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles**. *Communications of the ACM*, **22**(10):560–570, 1979. 7

[63] Liangjun Zhang, Steven LaValle, and Dinesh Manocha. **Global Vector Field Computation for Feedback Motion Planning**. In *Proc. IEEE International Conference on Robotics and Automation*, pages 477–482, May 2009. 7

[64] David Hsu, Jean-Claude Latombe, and Rajeev Motwani. **Path Planning in Expansive Configuration Spaces**. *International Journal of Computational Geometry and Applications*, **9**(4-5):495–512, 1999. 7

[65] Mark Foskey, Maxim Garber, Ming Lin, and Dinesh Manocha. **A Voronoi-Based Hybrid Motion Planner**. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 55–60, October 2001. 8

[66] Oliver Brock and Lydia Kavraki. **Decomposition-based Motion Planning: A Framework for Real-time Motion Planning in High-dimensional Configuration Spaces**. In *Proc. IEEE International Conference on Robotics and Automation*, pages 1469–1474, May 2001. 8

[67] Yuandong Yang and Oliver Brock. **Efficient Motion Planning Based on Disassembly**. In *Proc. Robotics: Science and Systems*, pages 97–104, June 2005. 8

[68] Markus Rickert, Oliver Brock, and Alois Knoll. **Balancing Exploration and Exploitation in Motion Planning**. In *Proc. IEEE International Conference on Robotics and Automation*, pages 2812–2817, May 2008. 8

[69] Markus Rickert, Arne Sieverling, and Oliver Brock. **Balancing Exploration and Exploitation in Sampling-Based Motion Planning**. *IEEE Transactions on Robotics*, **30**(6):1305–1317, 2014. 8, 33, 103

[70] Erion Plaku, Lydia Kavraki, and Moshe Vardi. **Impact of Workspace Decompositions on Discrete Search Leading Continuous Exploration (DSLX) Motion Planning**. In *Proc. IEEE International Conference on Robotics and Automation*, pages 3751–3756, May 2008. 8

[71] Oliver Brock and Oussama Khatib. **High-speed Navigation Using the Global Dynamic Window Approach**. In *Proc. IEEE International Conference on Robotics and Automation*, pages 341–346, May 1999. 8

[72] Yuandong Yang and Oliver Brock. **Elastic Roadmaps: Globally Task-Consistent Motion for Autonomous Mobile Manipulation in Dynamic Environments**. In *Proc. Robotics: Science and Systems*, August 2006. 8

[73] Yuandong Yang and Oliver Brock. **Elastic Roadmaps - Motion Generation for Autonomous Mobile Manipulation**. *Autonomous Robots*, **28**(1):113–130, 2010. 8

[74] Jur van den Berg, Dave Ferguson, and James Kuffner. **Anytime Path Planning and Replanning in Dynamic Environments**. In *Proc. IEEE International Conference on Robotics and Automation*, pages 2366–2371, May 2006. 8

[75] David Hsu, Robert Kindel, Jean-Claude Latombe, and Stephen Rock. **Randomized Kinodynamic Motion Planning with Moving Obstacles**. *The International Journal of Robotics Research*, **21**(3):233–255, 2002. 8, 39

[76] Lou Tychonievich, David Zaret, John Mantegna, Robert Evans, Eric Muehle, and Scott Martin. **A Maneuvering-Board Approach to Path Planning with Moving Obstacles**. In *Proc. International Joint Conference on Artificial Intelligence*, pages 1017–1021, August 1989. 8

[77] Paolo Fiorini and Zvi Shiller. **Motion Planning in Dynamic Environments Using Velocity Obstacles**. *The International Journal of Robotics Research*, **17**(7):760–772, 1998. 8

[78] Animesh Chakravarthy and Debasish Ghose. **Obstacle Avoidance in a Dynamic Environment: A Collision Cone Approach**. *IEEE Transactions on Systems Man and Cybernetics*, **28**(5):562–574, 1998. 8

[79] David Wilkie, Jur van den Berg, and Dinesh Manocha. **Generalized Velocity Obstacles**. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5573–5578, October 2009. 8

[80] Dieter Fox, Sebastian Thrun, and Wolfram Burgard. **The Dynamic Window Approach to Collision Avoidance**. *IEEE Robotics & Automation Magazine*, **4**(1):23–33, 1997. 8

[81] Bruno Damas and José Santos-Victor. **Avoiding Moving Obstacles: the Forbidden Velocity Map**. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4393–4398, October 2009. 8

[82] Aleksandr Kushleyev and Maxim Likhachev. **Time-bounded Lattice for Efficient Planning in Dynamic Environments**. In *Proc. IEEE International Conference on Robotics and Automation*, pages 1662–1668, May 2009. 8

[83] Richard Fikes and Nils Nilsson. **STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving**. *Artificial Intelligence*, **2**:189–208, 1971. 9

[84] Malik Ghallab, Adele Howe, Craig Knoblock, Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. **PDDL - The Planning Domain Definition Language**. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998. 9

[85] Stéphane Cambon, Rachid Alami, and Fabien Gravot. **A Hybrid Approach to Intricate Motion, Manipulation and Task Planning**. *The International Journal of Robotics Research*, **28**(1):104–126, 2009. 9

[86] Erion Plaku and Gregory Hager. **Sampling-Based Motion and Symbolic Action Planning with Geometric and Differential Constraints**. In *Proc. IEEE International Conference on Robotics and Automation*, pages 5002–5008, May 2010. 9

[87] Leslie Kaelbling and Tomás Lozano-Pérez. **Integrated Task and Motion Planning in Belief Space**. *The International Journal of Robotics Research*, **32**(9-10):1194–1227, 2013. 9

[88] Michael Hülsen, J. Marius Zöllner, and Javier Ibañez-Guzman. **Ontology-Based Context Awareness for Driving Assistance Systems**. In *Proc. IEEE Intelligent Vehicles Symposium*, pages 227–233, June 2014. 9

[89] Alexandre Armand, David Filliat, and Christian Weiss. **Traffic Intersection Situation Description Ontology for Advanced Driver Assistance**. In *Proc. IEEE Intelligent Vehicles Symposium*, pages 993–999, June 2011. 9

[90] Ralf Kohlhaas, Thomas Bittner, Thomas Schamm, and J. Marius Zöllner. **Semantic State Space for High-level Maneuver Planning in Structured Traffic Scenes**. In *Proc. IEEE International Conference on Intelligent Transportation Systems*, pages 1060–1065, October 2014. 9

[91] Lihua Zhao, Ryutaro Ichise, Tatsuya Yoshikawa, Takeshi Naito, Toshiaki Kakinami, and Yutaka Sasaki. **Ontology-based Decision Making on Uncontrolled Intersections and Narrow Roads**. In *Proc. IEEE Intelligent Vehicles Symposium*, pages 83–88, June 2015. 9

[92] Igor Paromtchik and Christian Laugier. **Motion Generation and Control for Parking an Autonomous Vehicle**. In *Proc. IEEE International Conference on Robotics and Automation*, pages 3117–3122, April 1996. 9

[93] Bernhard Müller, Joachim Deutscher, and Stefan Grodde. **Continuous Curvature Trajectory Design and Feedforward Control for Parking a Car**. *IEEE Transactions on Robotics and Automation*, **15**(3):541–553, 2007. 10

[94] Hélène Vorobieva, Sebastien Glaser, Nicoleta Minoiu-Enache, and Saïd Mammar. **Automatic Parallel Parking with Geometric Continuous-Curvature Path Planning**. In *Proc. IEEE Intelligent Vehicles Symposium*, pages 465–471, June 2014. 10

[95] Prasanth Jeevan, Frank Harchut, Bernhard Mueller-Bessler, and Burkhard Huhnke. **Realizing Autonomous Valet Parking with Automotive Grade Sensors**. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3824–3829, October 2010. 10

[96] Christian Löper, Claas Brunken, George Thomaidis, Stephan Lapoehn, Paulin Fouopi, Henning Mosebach, and Frank Köster. **Automated Valet Parking as Part of an Integrated Travel Assistance**. In *Proc. IEEE Conference on Intelligent Transportation Systems*, pages 2341–2348, October 2013. 10

[97] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, M. Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, Michele Gittleman, Sam Harbaugh, Martial Hebert, Thomas Howard, Sascha Kolski, Alonzo Kelly, Maxim Likhachev, Matt McNaughton, Nick Miller, Kevin Peterson, Brian Pilnick, Raj Rajkumar, Paul Rybski, Bryan Salesky, Young-Woo Seo, Sanjiv Singh, Jarrod Snider, Anthony Stentz, William Whittaker, Ziv Wolkowicki, Jason Ziglar, Hong Bae, Thomas Brown, Daniel Demitrish, Bakhtiar Litkouhi, Jim Nickolaou, Varsha Sadekar, Wende Zhang, Joshua Struble, Michael Taylor, Michael Darms, and Dave Ferguson. **Autonomous Driving in Urban Environments: Boss and the Urban Challenge**. *Journal of Field Robotics*, **25**(8):425–466, 2008. 10

[98] Michael Montemerlo, Jan Becker, Suhrid Bhat, Hendrik Dahlkamp, Dmitri Dolgov, Scott Ettinger, Dirk Haehnel, Tim Hilden, Gabe Hoffmann, Burkhard Huhnke, Doug Johnston, Stefan Klumpp, Dirk Langer, Anthony Levandowski, Jesse Levinson, Julien Marcil, David Orenstein, Johannes Paefgen, Isaac Penny, Anna Petrovskaya, Mike Pflueger, Ganymed Stanek, David Stavens, Antone Vogt, and Sebastian Thrun. **Junior: The Stanford Entry in the Urban Challenge**. *Journal of Field Robotics*, **25**(9):569–597, 2008. 10

[99] Hadas Kress-Gazit and George Pappas. **Automatically Synthesizing a Planning and Control Subsystem for the DARPA Urban Challenge**. In *Proc. IEEE Conference on Automation Science and Engineering*, pages 766–771, August 2008. 10

[100] Julio Rosenblatt. **DAMN: A Distributed Architecture For Mobile Navigation**. *Journal of Experimental and Theoretical Artificial Intelligence*, **9**(2/3):339–360, 1997. 10

[101] Andrew Bacha, Cheryl Bauman, Ruel Faruque, Michael Fleming, Chris Terwelp, Charles Reinholtz, Dennis Hong, Al Wicks, Thomas Alberi, David Anderson, Stephen Cacciola, Patrick Currier, Aaron Dalton, Jesse Farmer, Jesse Hurdus, Shawn Kimmel, Peter King, Andrew Taylor, David Van Covern, , and Mike Webster. **Odin: Team VictorTango's Entry in the DARPA Urban Challenge**. *Journal of Field Robotics*, **25**(8):467–492, 2008. 10

[102] Chao Chen, Markus Rickert, and Alois Knoll. **Combining Space Exploration and Heuristic Search in Online Motion Planning for Nonholonomic Vehicles**. In *Proc. IEEE Intelligent Vehicles Symposium*, pages 1307–1312, June 2013. 10, 49

[103] Chao Chen, Markus Rickert, and Alois Knoll. **Kinodynamic Motion Planning with Space-Time Exploration Guided Heuristic Search for Car-Like Robots in Dynamic Environments**. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2666–2671, October 2015. 11

[104] Chao Chen, Markus Rickert, and Alois Knoll. **Path Planning with Orientation-Aware Space Exploration Guided Heuristic Search for Autonomous Parking and Maneuvering**. In *Proc. IEEE Intelligent Vehicles Symposium*, pages 1148–1153, June 2015. 11

[105] Chao Chen, Andre Gaschler, Markus Rickert, and Alois Knoll. **Task Planning for Highly Automated Driving**. In *Proc. IEEE Intelligent Vehicles Symposium*, pages 940–945, June 2015. 11

[106] Chao Chen, Markus Rickert, and Alois Knoll. **A Traffic Knowledge Aided Vehicle Motion Planning Engine Based on Space Exploration Guided Heuristic Search**. In *Proc. IEEE Intelligent Vehicles Symposium*, pages 535–540, June 2014. 11

[107] **Motion Planning Kit**. `http://ai.stanford.edu/~mitul/mpk/`. 12

[108] **Motion Strategy Library**. `http://msl.cs.uiuc.edu/msl/`. 12, 93

[109] Ioan Şucan, Mark Moll, and Lydia Kavraki. **The Open Motion Planning Library**. *IEEE Robotics & Automation Magazine*, **19**(4):72–82, 2012. 12, 33

[110] **The Open Motion Planning Library**. `http://ompl.kavrakilab.org`. 12, 93

[111] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *The C++ Standard Library: A Tutorial and Reference*. Addison Wesley, 2nd edition, March 2012. 12

[112] **Containers - C++ Reference**. `http://www.cplusplus.com/reference/stl/`. 12

[113] Björn Karlsson. *Beyond the C++ Standard Library: An Introduction to Boost*. Pearson Education, 2015. 12

[114] **Boost C++ Libraries**. `http://www.boost.org/`. 12

[115] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry*. Springer, 3rd edition, 2008. 14

[116] Bob Caviness and Jeremy Johnson, editors. *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Springer, 1st edition, 1998. 14

[117] Franz Aurenhammer. **Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure**. *ACM Computing Surveys*, **23**(3):345–405, 1991. 14

[118] Jason Janét, Ren Luo, and Michael Kay. **The Essential Visibility Graph: An Approach to Global Motion Planning for Autonomous Mobile Robots**. In *Proc. IEEE International Conference on Robotics and Automation*, pages 1958–1963, May 1995. 14

[119] Carole Nissoux, Thierry Siméon, and Jean-Paul Laumond. **Visibility Based Probabilistic Roadmaps**. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1316–1321, October 1999. 14

[120] Thierry Siméon, Jean-Paul Laumond, and Carole Nissoux. **Visibility-Based Probabilistic Roadmaps for Motion Planning**. *Advanced Robotics*, **14**(6):477–493, April 2000. 14

[121] Raph Levien. **The Euler Spiral: A Mathematical History**. Technical report, Electrical Engineering and Computer Sciences, University of California at Berkeley, September 2008. 25, 109

[122] **Straßenverkehrs-Ordnung**, 2013. vom 6. März 2013 (BGBl. I S. 367), die durch Artikel 1 der Verordnung vom 22. Oktober 2014 (BGBl. I S. 1635) geändert worden ist. 73, 88

[123] Statistisches Bundesamt. **Verkehrsunfälle - Fachserie 8 Reihe 7**, 2013. 85

[124] U.S. Department of Transportation, National Highway Traffic Safety Administration. **Crash Factors in Intersection-Related Crashes: An On-Scene Perspective**, 2010. 86