Real-Time Trajectory Planning Design of Car-like Robots on Uneven Terrain with Dynamic Obstacles

Qingyi Ren

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Real-Time Trajectory Planning Design of Car-like Robots on Uneven Terrain with Dynamic Obstacles

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

Qingyi Ren

September 13, 2024

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

Real-Time Trajectory Planning Design of Car-like Robots on Uneven Terrain with Dynamic Obstacles

by

Qingyi Ren

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: September 13, 2024

Supervisor(s):

prof.dr.ir. T.K. Tamás Keviczky

ir. L.Z. Luyao Zhang

Reader(s):

Dr.ir. S.G. Sergio Grammatico

Abstract

This thesis introduces two improvements in trajectory planning on uneven terrain with dynamic obstacles: the Dynamic Traversability-based Hybrid A* (DT Hybrid A*) algorithm and the Safety Direction Velocity Obstacle (DSVO) replanning method. The DT Hybrid A* differs from traditional path-planning approaches by adopting a less conservative strategy, incorporating vehicle momentum into terrain traversability in order to more accurately reflect the vehicle's capability to navigate uneven terrain. Based on dynamic traversability, the DT Hybrid A* is developed by leveraging a kinematic bicycle model for steering, allowing variable travel lengths between adjacent nodes, and incorporating both distance and time heuristics. These enhancements significantly reduce CPU time costs during the planning process, resulting in shorter path lengths and faster planning times.

This thesis incorporates a collision detection mechanism within the global path planning framework, enabling the system to replan if a potential collision is predicted in the future. The replanning process initializes the global path to avoid all static obstacles. With the movement of dynamic obstacles, if a potential conflict is recognized, the vehicle discards the current global path and replans from its current location. To ensure that the replanned path is free of dynamic obstacles, the collision detection mechanism incorporates Safety and Direction Elements based on a Velocity Obstacle (VO) cost term into the global path algorithm (DSVO).

The DSVO algorithm enhances collision avoidance by integrating Safety and Direction Elements based on a VO cost term, surpassing traditional methods that use Euclidean distance penalties. DSVO avoids collisions by considering obstacle velocity and direction over time. It enhances the rate of successful replans for dynamic obstacle avoidance, producing the shortest collision-free paths with minimal computational overhead and completion time. The effectiveness of these approaches highlights the balance between computational efficiency and dynamic obstacle management, paving the way for more reliable path replanning solutions.

DSVO surpasses methods that use Euclidean distance as a penalty for dynamic obstacles because it considers the obstacle's velocity and direction over time. It also outperforms the Safety Element based on a VO cost term, which only ensures collision avoidance within a short time frame and provides the optimal velocity for that period but does not account for longer-term safety, potentially leading to collisions over extended durations. Additionally,

since these are cost terms within global path planning algorithms, there can be conflicts between obstacle avoidance and achieving the goal. The DSVO algorithm, which incorporates Safety and Direction Elements based on a VO cost term, mitigates conflicts between obstacle avoidance and goal achievement. Additionally, DSVO effectively adjusts the path to navigate around dynamic obstacles while maintaining a direct route toward the goal, balancing safety and efficiency.

Table of Contents

Preface

This document serves as my Master of Science thesis, centered on the emerging field of real-time trajectory planning for uneven terrain with dynamic obstacles. In this emerging field, mature and well-developed approaches are limited, and there lacks a structured framework suitable for navigating dynamic obstacles on uneven terrain. This technology is perceived as a frontier in autonomous driving, poised to extend the capabilities of autonomous vehicles to navigate through complex, unstructured, and uneven terrains, gradually advancing into maturity.

The inspiration for this thesis arose from the realization that trajectory planning on uneven terrain with dynamic obstacles represents one common case that has not received adequate analysis or resolution. This intriguing aspect deeply captivates my interest, and I am enthusiastic about contributing to the advancement of autonomous navigation in complex environments, starting with this graduation thesis.

Acknowledgements

I would like to express my deepest gratitude to my supervisor in Delft, Tamas Keviczky, for his unwavering support and professional guidance throughout this journey. His insightful advice, sense of responsibility, and dedication have helped me navigate every challenge. From offering precise steps to refine my work to encouraging critical thinking, his instruction has significantly contributed to the quality of this thesis. I am grateful for his patience, expertise, and encouragement at every stage of this research.

I would also like to extend my sincere appreciation to my daily supervisor, Luyao Zhang, for his constant support and the numerous insightful discussions we had on 2.5D trajectory planning methods, obstacle avoidance techniques, and open-source frameworks in ROS. His commitment to expanding my knowledge by sharing the latest research and developments in the field has been incredibly helpful in broadening my understanding. I am deeply grateful for his dedication and sense of responsibility toward my work, consistently providing timely and constructive feedback that has significantly contributed to the progress and depth of this thesis.

Delft, University of Technology
September 13, 2024

Qingyi Ren

“Courage doesn’t always roar. Sometimes courage is the quiet voice at the end of the day saying, ’I will try again tomorrow.”’

— *Mary Anne Radmacher*

Chapter 1

Introduction

In recent years, autonomous vehicle technology has witnessed significant advancements. Pioneering efforts from traditional car manufacturers like Google and Tesla have propelled the boundaries of self-driving technology. Google commenced its autonomous vehicle project in 2009, accumulating over 8 million physical kilometers and extensive simulated testing. In parallel, Tesla introduced the 'Autopilot' software function, allowing drivers to temporarily disengage from steering.

Today, the focus is on achieving full autonomy, enabling cars to navigate freely on uneven terrain. This evolution has extended into more complex environments, leading to the necessity for ground robots to navigate unstructured or uneven terrain autonomously. The challenge lies in the intricate physical characteristics of the terrain, including slope, roughness, and dynamic obstacles, directly influencing the behavior of car-like robots [28]. Real-time path planning on large-scale uneven terrain with dynamic barriers is essential for applications such as search and rescue missions.

While conventional 2D indoor navigation techniques are mature and widely used, their shortcomings become evident when applied to unpredictable outdoor terrains [47]. Designed for controlled environments, these techniques need more adaptability to the complexities of uneven terrain with dynamic obstacles. Effective real-time path planning is fundamental for autonomous navigation, requiring on-the-fly decisions considering the robot's state, the environment, and dynamic changes—critical for ensuring mission safety and success in challenging terrains.

Traditional approaches assume a flat surface or overlook the active height changes of ground robots, aiming to extend 2D navigation methodologies to uneven terrain [41]. While simplifying the terrain model aids in finding trajectories for car-like robots, it imposes limitations on motion capabilities. Such restrictions, especially in managing both passive and active height variations, significantly constrain the capabilities of ground robots, thereby limiting their potential applications.

The introduction of autonomous ground robot navigation on uneven terrain introduces a visionary concept poised to revolutionize the field of robotics. These advanced systems are anticipated to integrate cutting-edge technologies, including path-planning algorithms, dynamic

obstacle detection and avoidance strategies, and an MPC solver [24]. The process, rooted in terrain perception outcomes that provide knowledge and characterization of terrain, initiates path planning on uneven terrain. Emphasizing the vital role of dynamic obstacle detection and avoidance strategies, this approach aims to generate real-time collision-free trajectories. The MPC framework then formulates an optimization problem by incorporating the planned trajectory and predicting the robot's future states over a defined horizon. It computes optimal control inputs that guide the robot toward the goal pose while accurately following the planned path and adhering to physical constraints, ultimately completing the navigation task.

Traditional obstacle avoidance on 2D plane can be approached in two main ways: one method integrates obstacle constraints directly into the optimization problem, while the other conducts collision checks either after optimization or during the execution of a search-based algorithm. Both methods require defining conditions under which a collision occurs.

Currently, research on dynamic obstacle avoidance in uneven terrain remains limited, with few well-developed frameworks available. This challenge is largely due to the limitations of 2.5D motion planning. The common approach in 2.5D planning is to first create a 2D motion plan using a traversability map, then apply a perception model to map that path onto uneven terrain. This approach simplifies 2.5D motion planning by converting it into a 2D problem, allowing the use of existing algorithms. However, this method introduces issues for dynamic obstacle avoidance because a 2D collision check may not capture all potential collisions in 2.5D environments. Complex terrain features, such as caves or multiple layers at the same 2D spatial location, can lead to inaccurate collision detection. Additionally, collisions detected in 2D may not actually occur in 3D if the terrain's structure causes discrepancies between the 2D path and the real-world distance on uneven ground.

Another challenge is the lack of research on geometric modeling for collision detection based on the shape and size of dynamic obstacles. Without accurate geometric analysis, ensuring reliable obstacle avoidance in 3D terrain becomes difficult. In 2D environments, a common strategy is to approximate the obstacle region using a 2D convex shape, then formulate convex constraints based on the boundaries of this shape, which helps in solving non-convex optimization problems [18]. However, when dealing with uneven terrain, the situation is more complex. A three-dimensional convex shape is required to encapsulate the dynamic obstacle region, which significantly increases the difficulty and can lead to computational challenges that may render the problem unsolvable.

Chapter 2

Overview of Motion Planning Techniques

2-1 2D Motion Planning

2D trajectory planning involves charting a course for a car-like robot to travel from its starting position to a designated endpoint on a two-dimensional surface while adhering to specific constraints. Two common advanced techniques used in this process are Sampling-Based Motion Planning and Optimization-Based Motion Planning.

Optimization-Based Motion Planning [45] is proficient in producing optimal solutions by minimizing an objective function. However, it may face challenges when dealing with high-dimensional spaces, requiring substantial computational resources for real-time applications.

On the other hand, Sampling-Based Motion Planning [14] proves effective in navigating high-dimensional spaces and adeptly handling complex environments with obstacles. Yet, it may struggle to identify the absolute optimal path.

In many motion planning tasks, the car-like robot must navigate around both static and dynamic obstacles to ensure safe movement. By strategically combining these two approaches to leverage their strengths and incorporating obstacle avoidance, the comprehensive framework of Hybrid Motion Planning encompasses global planning, local path optimization, and obstacle avoidance.

The specific implementation of Hybrid Motion Planning can vary depending on the approach to obstacle avoidance, such as conducting obstacle checks after optimization or integrating obstacle constraints into the optimization problem within Search-based Algorithms.

2-1-1 Global Planning

Global Planning adopts a sampling-based motion planning method to explore high-dimensional state space and generates the shortest piece-wise linear sampled path from the current position to the goal as the initial path. Sampling-based algorithms rely on a collision-checking

module to evaluate the viability of potential paths. This module helps generate a collection of points from the obstacle-free space, which are then connected to form a roadmap of feasible trajectories. This roadmap offers a flexible and adaptable approach for navigating complex environments. By bypassing the need to explicitly account for obstacles during construction, sampling-based methods can significantly reduce computational overhead [14].

In the realm of motion planning, two widely-used algorithms are Probabilistic Roadmaps (PRM), pioneered by Steven M. LaValle and James J. Kuffner Jr. in the late 1990s, and Rapidly-exploring Random Trees (RRT), introduced by Steven M. LaValle in 1998.

Probabilistic Roadmaps (PRM) employ a probabilistic sampling approach to construct a roadmap of the configuration space, enabling efficient path planning by connecting sampled configurations with feasible paths. This method is particularly effective for high-dimensional spaces and complex environments with obstacles.

Rapidly exploring Random Trees (RRT), on the other hand, aims to efficiently explore the configuration space by progressively growing a tree structure from the starting configuration towards the target. RRTs are well-suited for problems with complex, continuous spaces and dynamic environments, as they can quickly generate feasible paths.

2-1-2 Local Path optimization

Local Path Optimization involves taking the piece-wise linear sampled path and the initial state, creating an optimization problem to enhance the sampled path. This optimization task incorporates specific criteria, including but not limited to time efficiency, smoothness, and energy efficiency [46], and formulates the motion planning problem as a nonlinear optimization problem (OCP) [43]. The goal is to iteratively improve the path originally generated by the sampling-based method through this refined optimization process.

Nonlinear optimal control problems (OCP) have two main categories of optimization algorithms: numerical optimization techniques like Sequential Quadratic Programming (SQP) and Differential Dynamic Programming (DDP) inspired by numerical optimal control. Riccati Recursion and Condensing Solvers, including those in Model Predictive Control (MPC) frameworks like ACADOS [40], are vital for addressing KKT(Karush-Kuhn-Tucker) conditions and optimizing decision variables, reducing computational costs for large state dimensions.

DDP methods, exemplified by iterative Linear-Quadratic Regulator (iLQR) and its modifications (CROCODDYL [21] and ALTRO [11]), offer practicality for complex dynamic systems, showcasing proficiency in handling nonlinear constraints. In the realm of optimization solvers, ACADOS stands out with efficiency and compatibility, CROCODDYL utilizes Feasibility-driven Differential Dynamic Programming (FDDP) for computing state trajectories, ALTRO integrates iLQR for constrained optimization with rapid convergence, and FATROP [39] leverages nonlinear solvers for accelerated, robust trajectory optimization in complex robot scenarios. These solvers collectively contribute to advancing the field of trajectory optimization for car-like robot systems, offering diverse approaches and capabilities to meet various requirements and constraints.

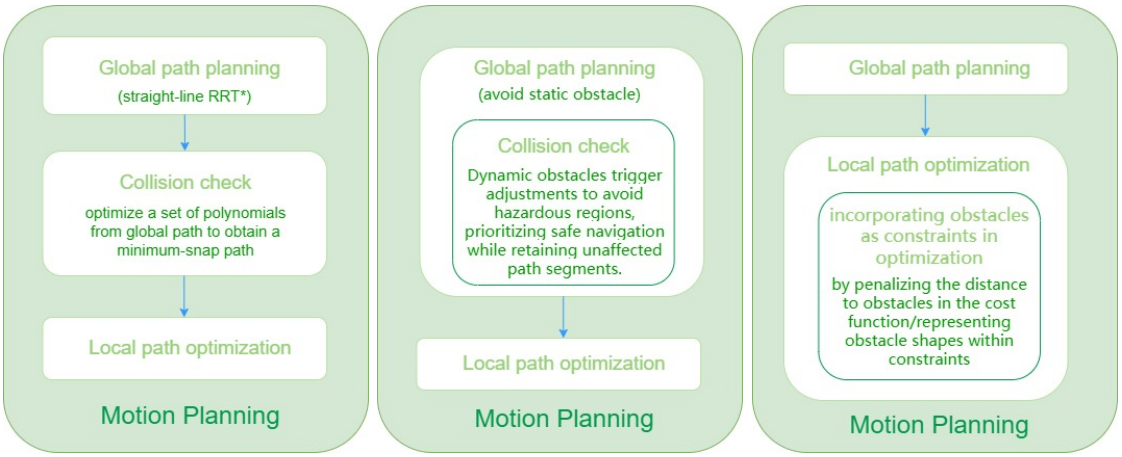

Figure 2-1: Three common implementations of Hybrid Motion Planning with different strategies of obstacle avoidance (a) Collision check post-optimization. (b) Collision check during a Search-based Algorithm. (c) Obstacle Avoidance by Constraints in the Optimization

2-1-3 Obstacle Avoidance

In motion planning, effectively navigating around obstacles is paramount. There are two primary methods for obstacle avoidance: the first involves incorporating obstacle constraints into the optimization problem, while the second entails performing collision checks either after the optimization process or during the execution of a search-based algorithm.

The first method involves directly incorporating obstacles as constraints within an optimization problem. This integration can be achieved within the cost function by penalizing proximity to obstacles or representing the shape of obstacles within the constraints through a process of successive convexification [8], [16], [37], [27], [26] and [18].

The second method involves collision checks during the execution of a search-based algorithm that employs a search-based algorithm for local path re-planning enable effective avoidance of dynamic obstacles. However, it is imperative to minimize the length of the re-planned path [17], [32], [15], [48], [34] and [25]. Conducting collision checks follows the optimization process, where stitched polynomial trajectories, guided by multiple waypoints obtained through search-based methods such as [6], [23], and [36], are utilized.

In Figure (2-1), three common implementations of Hybrid Motion Planning are shown with different approaches to avoid obstacles.

- To address collision checking and post-optimization, straight-line RRT* is employed to select waypoints from the optimal path. Nevertheless, this method overlooks dynamic constraints and may produce a path that does not align with the intended objective function. To achieve a minimum-snap trajectory, a set of polynomials is jointly optimized based on these waypoints.
- Collision check during a Search-based Algorithm creates the initial path using a search-based algorithm to navigate around static obstacles. If a dynamic obstacle emerges in proximity to the initial path, the dangerous region of the initial path is recognized by dangerous node detection. The adjustments to the initial path are made to circumvent

the dynamic obstacle effectively. The key strategy of this method involves retaining the unaffected segments of the path that avoid known static obstacles, with re-planning focused solely on the potentially hazardous region.

- Obstacle Avoidance by Constraints comprises two methods: by directly incorporating obstacles as constraints in optimization, often by penalizing the distance to obstacles in the cost function, or by representing obstacle shapes within constraints using techniques such as successive convexification or a convex decomposition of the environment.

2-2 2.5D Motion Planning

When guiding a car-like robot across rough terrain, the preferred approach is 2.5D motion planning. This method permits restricted variations in the third dimension while predominantly functioning within a two-dimensional plane.

2-2-1 Terrain Pose Mapping Based 2.5D Motion Planning

Uneven terrain has a changing altitude and curvature with a car-like robot's spatial location and heading direction, which directly influences the car-like robot's pose. Xu et al. proposed terrain pose mapping to evaluate how the uneven terrain impacts the robot [47]. In this method, three assumptions are made to simplify the problem

Assumption 1: The car-like robot maintains contact with the terrain throughout the entire trajectory.

Assumption 2: Wheel slip is not considered in the analysis.

Assumption 3: Obstacles and dynamic changes in terrain are not taken into consideration.

Based on these assumptions and the static property of uneven terrain, they preprocessed the point cloud data of the whole uneven terrain to obtain a one-to-one correspondence mapping from a 2D spatial position and heading direction to the car-like robot's 3D spatial location and altitude. Also, the 2D heading direction significantly enhances the uniqueness of the mapping results. This is because even within a small region of the same 3D spatial location, the uneven terrain's height and curvature can vary considerably based on different spatial orientations which can be projected onto unique 2D heading directions.

For Terrain Pose Mapping based Motion Planning shown in Figure (2-2), it makes full use of point cloud data to establish a direct correspondence from 2D spatial positions to 3D locations and orientations. By adopting this strategy, the optimization problem's variables can exist in a 2D space. Simultaneously, 3D objective functions that evaluate the quality of the generated 3D trajectory, such as jerk and surface orientation, can be defined using the 2D projected location. Furthermore, in the context of 2.5D trajectory planning, constraints on 3D factors, including speed, acceleration, and curvature, can be effectively imposed. These constraints are akin to those applicable in a 2D navigating scenario, but they are established solely through

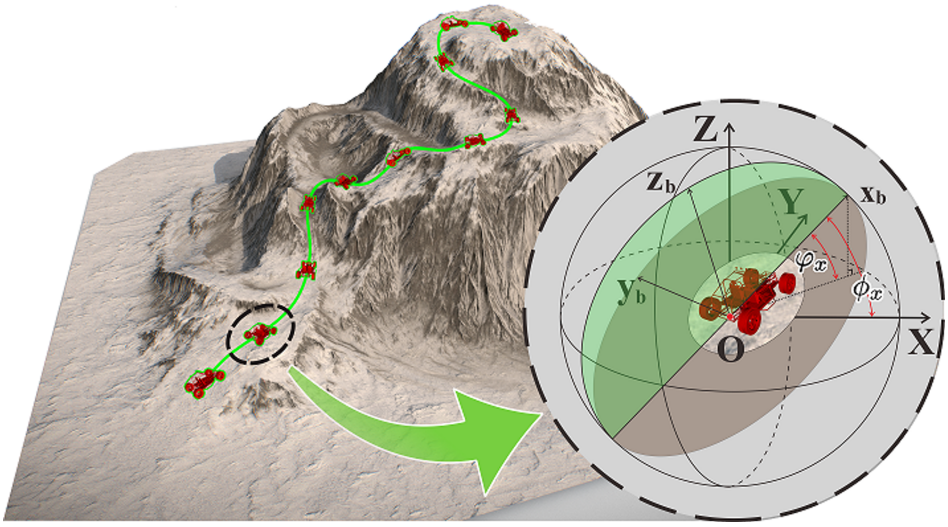

Figure 2-2: Visual representation of Terrain Pose Mapping based Motion Planning

the utilization of 2D projected locations. Acknowledging that physical variables might not consistently align with the 2D XY plane, yet recognizing that points on the 2D XY plane can uniquely define all corresponding 3D physical factors, this methodology guarantees a more comprehensive and effective trajectory planning process.

2-2-2 Energy Efficiency Based 2.5D Motion Planning

The Energy Efficiency-Based 2.5D Motion Planning [7] involves applying sampling-based motion planning to navigate uneven terrain. Guiding a car-like robot across rough terrain presents a dilemma in optimizing between achieving the shortest path and ensuring energy efficiency. The challenge stems from the realization that the shortest path on uneven terrain may not be energy-efficient, given the presence of uphill and downhill sections incurring additional energy costs compared to moving on flat surfaces. While energy-optimal paths prioritize energy consumption, they often demand an extended traversal time. The novel search algorithm named NAMOA* builds a multi-objective path planning problem comprised of both distance and energy usage between two chosen waypoints on a grid-based elevation map. The grid-based elevation terrain map can be converted into a weighted digraph G made of 8-connected neighborhoods [7]. The nodes of the digraph represent the grid center connecting with 8 neighboring grids.

Based on distance and energy cost considerations, two cost functions are created. These functions are constrained by a maximum inclination angle, causing them to tend to infinity if this angle is exceeded. A path selection policy is then established by comparing the cost functions for different trajectories. Through this process, along with the implementation of monotone conditions for setting heuristic cost, a practical path selection algorithm is provided. This algorithm becomes particularly valuable when selecting the shortest or most energy-efficient path is impractical, all without the necessity for a complete re-planning process.

2-2-3 Safety Penalty Field Based 2.5D Motion Planning

Safety Penalty Field-based 2.5D Motion Planning [41] introduces additional safety considerations compared to 2D Motion Planning. On a 2D ground plane, employing collision-free constraints is typically sufficient to ensure safe navigation. However, when navigating uneven terrain, relying solely on collision-free constraints is inadequate for guaranteeing the safety of the entire trajectory. For instance, a trajectory adhering to collision-free constraints might lead a vehicle dangerously close to the edge of a cliff. Additionally, when selecting trajectory points, it is crucial to ensure that the neighboring areas are sufficiently flat to prevent collisions or toppling.

The identified hazards can be further categorized as follows:

- Collision between the car-like robot and an obstacle.
- The car-like robot standing at a very steep angle.
- The robot experiences significant positional shifts, such as falling off a step or undergoing dramatic changes in its posture.

This approach constructs safety penalty functions on a grid map. The Euclidean Signed Distance Functions (ESDF) [49] is employed to ensure collision-free navigation between the car-like robot and obstacles. Additionally, RANSAC [44] is utilized to prevent the car-like robot from assuming a dangerously tilted position and undergoing substantial positional changes. By incorporating the ESDF and RANSAC, this method extends beyond the limitations of relying solely on collision-free constraints on uneven terrain, effectively addressing a broader range of potentially unsafe scenarios. This enhancement establishes a robust strategy for guaranteeing the safety of the trajectory in diverse and challenging environments.

2-3 Conclusion

This Chapter begins by introducing traditional 2D motion planning methods, focusing on global planning, local optimization, and obstacle avoidance. These techniques serve as the foundation for understanding motion planning in simple, two-dimensional environments.

However, our work also considers the more complex scenario of dynamic obstacle avoidance on uneven terrain, which leads us to the introduction of 2.5D motion planning. In navigating uneven terrain, particularly for car-like robots, 2.5D motion planning is the preferred approach. Three existing studies that focus on terrain pose mapping, energy cost, and the safety of the entire trajectory are introduced to provide an overview of 2.5D motion planning.

In the following chapter, we adopt terrain pose mapping as the perception model. We will introduce all the essential components needed for dynamic obstacle avoidance on uneven terrain. This includes vehicle dynamics, terrain pose mapping, trajectory optimization, and model predictive control (MPC). Each concept will be explained in detail, with formulas provided to enhance understanding of the work.

Chapter 3

Background

In the preceding chapters, the scope, objectives, and focus of this thesis were outlined, along with an overview of motion planning. The terrain perception framework proposed by Xu et al. was adopted, and the details of terrain pose mapping were presented. The concepts of vehicle dynamics, terrain pose mapping, trajectory optimization, and MPC control were then introduced in sequence, offering a comprehensive foundation for understanding motion planning on uneven terrain.

3-1 Vehicle Dynamics

The bicycle kinematic model is a simplified representation of a vehicle's motion and is used to describe and predict the movement of wheeled vehicles. This model captures the essential geometry and constraints of steering and movement without the complexities of a full dynamic model.

The control input $\mathbf{u} = (a, \phi)$ is represented as a combination of acceleration a, which is longitudinal control input, and steering input ϕ as lateral control input. The vehicle state space x_v, including location x and y, orientation θ and velocity v, can be represented as

$$\mathbf{x}_v = [x, y, \theta, v]^\top$$

.

- When $\phi \neq 0$, the turning radius corresponding to each ϕ can be calculated as $\tan\phi/r$. The change in successive vehicle states when $\phi \neq 0$ can be represented as:

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \\ \Delta v \end{bmatrix} = \begin{bmatrix} r\cos\theta\sin\Delta\theta + r\sin\theta(1-\cos\Delta\theta) \\ r\sin\theta\sin\Delta\theta - r\cos\theta(1-\cos\Delta\theta) \\ d/r \\ a\Delta t \end{bmatrix} \tag{3-1}$$

  where d represents the travel length between successive vehicle states.

- When $\phi = 0$, the turning radius is infinite, which means the vehicle moves in a straight line. The difference between successive vehicle states can be represented as:

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \theta \\ \Delta v \end{bmatrix} = \begin{bmatrix} d\cos\theta \\ d\sin\theta \\ 0 \\ a\Delta t \end{bmatrix} \tag{3-2}$$

When a vehicle is navigating on uneven terrain, the state includes the spatial position denoted as $\mathbf{p} = \begin{bmatrix} x & y & z \end{bmatrix}^\top$ in the global frame and the robot's altitude described by $\mathbf{R} = \begin{bmatrix} \mathbf{x}_b & \mathbf{y}_b & \mathbf{z}_b \end{bmatrix} \in SO(3)$. Here, $\mathbf{x}_b$, $\mathbf{y}_b$, and $\mathbf{z}_b$ represent the car-like robot's body frame, a 3-dimensional vector with respect to the global frame's x, y, and z coordinates. It is worth mentioning that the body frame of a robot refers to a coordinate system fixed to the robot itself, and it is related to the terrain's orientation. The body frame could be used to describe the motion of the robot's position and orientation by the following equations:

$$\dot{\mathbf{p}} = \mathbf{x}_b \cdot v_x \tag{3-3}$$

$$\dot{\mathbf{R}} = \mathbf{R}\lfloor \mathbf{v}_w \rfloor \tag{3-4}$$

In these equations, v_x denotes the magnitude of the robot's forward velocity in the direction of $\mathbf{x}_b$ axis, which aligns with the x-direction of the body frame and the $\lfloor \mathbf{v}_w \rfloor$ converts angular velocity vector $\mathbf{v}_w$ in the body frame into a skew-symmetric matrix. (3-3) describes the rate of change of the robot's position, denoted as $\dot{\mathbf{p}}$, which represents the robot's forward velocity along the body frame $\mathbf{x}_b$. (3-4) signifies the alteration of the robot's altitude over time, driven by the angular velocity rate. By using the forward velocity along the body axis v_x, steering angle ϕ, and wheelbase length of the robot L_w, the angular velocity [35] could be further represented as

$$\mathbf{v}_w = \frac{v_x \tan\phi}{L_w} \cdot \mathbf{z}_b$$

The spatial location and orientation can be projected onto a 2D plane as the 2D state $\mathbf{s}_p = \begin{bmatrix} x & y & \theta \end{bmatrix}^\top \in SE(2)$, where x and y denote the spatial coordinates, and θ represents the projected heading direction on 2D plane.

3-2 Terrain Pose Mapping

As discussed earlier in Chapter 2-2-1, the impact of uneven terrain on the robot's pose has been a critical aspect of terrain navigation. Xu et al. introduced the concept of terrain pose mapping [47], which evaluates how variations in terrain height and curvature, in conjunction with the robot's spatial position and heading direction, affect the vehicle's pose.

To simplify the problem, three key assumptions were made:

Assumption 1: The car-like robot maintains contact with the terrain throughout the entire trajectory.

Assumption 2: Wheel slip is not considered in the analysis.

Assumption 3: Obstacles and changes in terrain dynamics are not considered. This means the terrain is obstacle-free and remains static, with no variation in its structure or characteristics.

These assumptions allow for a more tractable analysis of the terrain's influence on the robot, focusing on static terrain characteristics while excluding dynamic effects and wheel slip. This simplification is beneficial for initial analyses and models where the complexity of real-world interactions needs to be controlled.

Building on these assumptions and the static characteristics of uneven terrain, Xu et al. preprocessed the point cloud data to establish a one-to-one correspondence between the 2D spatial position and heading direction and the car-like robot's 3D spatial location and altitude. Notably, the 2D heading direction greatly guarantees the uniqueness of the mapping results. This is because, even within a small region of the exact 3D spatial location, the height and curvature of the uneven terrain can vary significantly depending on different spatial orientations, which can be projected onto distinct 2D heading directions.

The terrain pose mapping is then shown as follows:

$$\mathcal{F} : SE(2) \mapsto \mathbb{R} \times \mathbb{S}_+ \tag{3-5}$$

where elements in $SE(2)$ represent car-like 2D state, elements $\mathbb{R}$ and those in $\mathbb{S}_+$ represent car-like robot's height z and body axis $\mathbf{z}_b$. It is important to note that we restrict our use to the upper hemisphere $\mathbb{S}_+ \subset \mathbb{S} = \{\mathbf{x} \in \mathbb{R}^3 | \|\mathbf{x}\|_2 = 1\}$, rather than the entire 3D sphere $\mathbb{S}$. This restriction is due to the practical need to ensure that $\mathbf{z}_b$ remains in the upper half-plane. Otherwise, it would necessitate rapid speeds and introduce safety hazards in common tasks to keep $\mathbf{z}_b$ oriented toward the lower half-plane. For the sake of safety, the body axis $\mathbf{z}_b$ should be constrained as:

$$\mathbb{S}_+ = \{\mathbf{x} \in \mathbb{R}^3 | \|\mathbf{x}\|_2 = 1,\ \mathbf{x} \cdot \mathbf{b}_3 > 0\}$$

where $\mathbf{b}_3 = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^\top$.

The terrain pose mapping enables the determination of the car-like robot's unique height, denoted as z, and its body frame's vertical component, denoted as $\mathbf{z}_b$, based on the robot's 2D state represented by $\mathbf{s}_p = \begin{bmatrix} x & y & \theta \end{bmatrix}^\top$. The terrain pose mapping can be represented as:

$$z = f_1(\mathbf{s}_p) \tag{3-6}$$

$$\mathbf{z}_b = \mathbf{f}_2(\mathbf{s}_p) \tag{3-7}$$

Based on height z and body frame vertical component $\mathbf{z}_b$, the position $\mathbf{p}$ and altitude $\mathbf{R}$ of robot can be calculated as follows:

$$\mathbf{p} = [x, y, f_1(\mathbf{s}_p)]^\top \tag{3-8}$$

$$\mathbf{y}_b = \frac{\mathbf{z}_b \times \mathbf{x}_{yaw}}{\|\mathbf{z}_b \times \mathbf{x}_{yaw}\|} = \frac{\mathbf{f}_2(\mathbf{s}_p) \times \mathbf{x}_{yaw}}{\|\mathbf{f}_2(\mathbf{s}_p) \times \mathbf{x}_{yaw}\|} \tag{3-9}$$

$$\mathbf{x}_b = \mathbf{y}_b \times \mathbf{z}_b = \mathbf{y}_b \times \mathbf{f}_2(\mathbf{s}_p) \tag{3-10}$$

Algorithm 1: Get the result of $\mathscr{F}$ at a $SE(2)$ state

Input: state $\mathbf{s}_r \in SE(2)$, Iteration times N_{iter},
Ellipsoidal parameters (e_x, e_y, e_z)
Output: $\mathbf{z}_b$, z
begin
  $\mathbf{z}_b \leftarrow \mathbf{b}_3$;
  $z \leftarrow$ FindNearestXYPointZ($\mathbf{s}_r$);
  for ***each*** $i \in N_{iter}$ **do**
    $(p_i, R_i) \leftarrow$ CalculateSE3($\mathbf{s}_r, z, \mathbf{z}_b$);
    $G_i \leftarrow$ FindEllipsoidPoints(p_i, R_i, e_x, e_y, e_z);
    $p_{mean} \leftarrow$ GetMeanPosition(G_i);
    $Cov \leftarrow$ ZeroSquareMatrix3();
    for ***each*** $p_j \in G_i$ **do**
      $p_e \leftarrow p_j - p_{mean}$;
      $Cov \leftarrow Cov + p_e p_e^{\mathrm{T}}$;
    $p_{mean} \leftarrow p_{mean}/$NumOf($M_i$);
    $Cov \leftarrow Cov/$NumOf(M_i);
    $\mathbf{z}_b \leftarrow$ GetMinEigenVec(Cov);
    $z \leftarrow p_{mean}$.GetZ();
  return $\mathbf{z}_b$, z;

Figure 3-1: Algorithm of terrain pose mapping $\mathcal{F}$

where $\mathbf{x}_{yaw} = [\cos\theta, \sin\theta, 0]^\top$ represents the 2D heading direction vector in the global frame, the body axis vectors $\mathbf{x}_b$, $\mathbf{y}_b$ and $\mathbf{z}_b$ are unit vectors. In (3-1), the process of deriving the terrain pose mapping function $\mathcal{F}$ is depicted. This algorithm is developed from point cloud data using a repetitive plane-fitting approach, taking into account the robot's dimensions and orientation. The input to the algorithm consists of point cloud data representing the uneven terrain and the robot's 2D state $\mathbf{s}_p = \begin{bmatrix} x & y & \theta \end{bmatrix}^\top$. The output is the elevation height z and the body axis $\mathbf{z}_b$. The process begins by extracting the point cloud data around the 2D point (x, y), initializing z to the height at (x, y) and setting $\mathbf{z}_b$ to $\mathbf{b}_3$. Using (3-9) and (3-10), the SE(3) state is then computed. Next, a 3D ellipsoidal region is defined to include point cloud data points within the region. The size and altitude of this ellipsoidal region are determined by the robot's physical size, position, and orientation. Within the ellipsoidal region, the center point is initially calculated as the mean of all points inside the region. The covariance matrix is then computed based on the distribution of points relative to this mean. The difference from the mean point is calculated and used to update the covariance matrix for each point inside the ellipsoidal region. After this step, the mean of all covariance matrices is taken as the final covariance matrix. The smallest eigenvalue of the covariance matrix defines $\mathbf{z}_b$, while the mean point's elevation height determines z. This process is iteratively refined, leading to the final values for z and $\mathbf{z}_b$.

A noteworthy aspect is the ability to represent the 3D state trajectory and dynamic states using only the 2D state trajectory through terrain pose mapping. This mapping allows each point $\mathbf{s}_p = [x, y, \theta]^\top$ on the 2D trajectory to yield unique values for height z and body axis $\mathbf{z}_b$. Consequently, according to the work [47], the 3D dynamic states—essential components of the optimization problem's constraints—such as longitudinal velocity v_x, longitudinal acceleration

a_x, lateral acceleration a_y, angular velocity w_z, curvature κ, and steering δ, can be expressed as:

$$v_x = \frac{v}{\cos\phi_x} \tag{3-11}$$

$$a_x = \frac{a_t}{\cos\phi_x} + g\sin\varphi_x \tag{3-12}$$

$$a_y = \frac{a_t}{\cos\phi_y} + g\sin\varphi_y \tag{3-13}$$

$$w_z = \frac{w}{\cos\xi} \tag{3-14}$$

$$\kappa = \frac{v_x}{w_z} \tag{3-15}$$

$$\delta = \arctan(L_w \cdot \kappa) \tag{3-16}$$

where a_t and a_n are projected 2D tangential acceleration and centripetal acceleration in XY plane respectively; L_w denotes length of vehicle's wheelbase;

$$v = \sqrt{\dot{x}^2 + \dot{y}^2} \tag{3-17}$$

$$a_t = \ddot{x}\cos\theta + \ddot{y}\sin\theta \tag{3-18}$$

$$a_n = -\ddot{x}\sin\theta + \ddot{y}\cos\theta \tag{3-19}$$

$$w = \dot{\theta} \tag{3-20}$$

$$\cos\phi_x = \mathbf{x}_b^\top \mathbf{x}_{yaw} \tag{3-21}$$

$$\cos\phi_y = \mathbf{y}_b^\top \mathbf{y}_{yaw} \tag{3-22}$$

$$\sin\varphi_x = \mathbf{x}_b^\top \mathbf{b}_3 \tag{3-23}$$

$$\sin\varphi_y = \mathbf{y}_b^\top \mathbf{b}_3 \tag{3-24}$$

$$\cos\xi = \mathbf{z}_b^\top \mathbf{b}_3 \tag{3-25}$$

Then, the focus is put on calculating the gradients of terrain variable vector s_{var},

$$s_{\text{var}} = \begin{bmatrix}\cos\phi_x & \cos\phi_y & \sin\varphi_x & \sin\varphi_y & \cos\xi\end{bmatrix}^\top$$

explicitly by using the chain rule. Since these terrain variables play a crucial role in the objective function and are essential for transforming the current velocity v and pose of the vehicle into other physical quantities, such as v_x, a_x, a_y, w_z, κ, and δ, which are subject to the optimization constraints.

According to (3-9) and (3-10), the $\cos\phi_x$ can be further written as

$$\begin{aligned}\cos\phi_x &= \mathbf{x}_b^\top \mathbf{x}_{yaw} = (\mathbf{y}_b \times \mathbf{z}_b)^\top \cdot \mathbf{x}_{yaw} \\ &= (\mathbf{y}_b \times \mathbf{z}_b)^\top \cdot \mathbf{x}_{yaw} \\ &= \frac{(\mathbf{z}_b \times \mathbf{x}_{yaw} \times \mathbf{z}_b)^\top \mathbf{x}_{yaw}}{\|\mathbf{z}_b \times \mathbf{x}_{yaw}\|}\end{aligned}$$

Given that $\|\mathbf{z}_b\| = \|\mathbf{x}_{yaw}\| = 1$, we assume $\mathbf{z}_b = [a, b, c]^\top$, Since $\mathbf{z}_b$ is unit vector, it follows that c can express as $c = \sqrt{1 - a^2 - b^2}$. By following the vector triple product identity, the cross product $\mathbf{z}_b \times \mathbf{x}_{yaw} \times \mathbf{z}_b$ can be written as:

$$\begin{aligned}
\mathbf{z}_b \times \mathbf{x}_{yaw} \times \mathbf{z}_b &= (\mathbf{z}_b \cdot \mathbf{z}_b)\mathbf{x}_{yaw} - (\mathbf{z}_b \cdot \mathbf{x}_{yaw})\mathbf{z}_b = \|\mathbf{z}_b\|^2 \mathbf{x}_{yaw} - (\mathbf{z}_b \cdot \mathbf{x}_{yaw})\mathbf{z}_b \\
&= \mathbf{x}_{yaw} - (\mathbf{z}_b \cdot \mathbf{x}_{yaw})\mathbf{z}_b \\
&= \begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \end{bmatrix} - \left(\begin{bmatrix} a \\ b \\ c \end{bmatrix} \cdot \begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \end{bmatrix} \right) \cdot \begin{bmatrix} a \\ b \\ c \end{bmatrix} \\
&= \begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \end{bmatrix} - (a\cos\theta + b\sin\theta) \cdot \begin{bmatrix} a \\ b \\ c \end{bmatrix} \\
&= \begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \end{bmatrix} - \begin{bmatrix} a(a\cos\theta + b\sin\theta) \\ b(a\cos\theta + b\sin\theta) \\ c(a\cos\theta + b\sin\theta) \end{bmatrix} = \begin{bmatrix} \cos\theta - a(a\cos\theta + b\sin\theta) \\ \sin\theta - b(a\cos\theta + b\sin\theta) \\ -c(a\cos\theta + b\sin\theta) \end{bmatrix}
\end{aligned}$$

As we have $\mathbf{z}_b \times \mathbf{x}_{yaw} = [-c\sin\theta, c\cos\theta, a\sin\theta - b\cos\theta]^\top$ and $c = \sqrt{1 - a^2 - b^2}$, the norm of the cross product can be written as:

$$\begin{aligned}
&\|\mathbf{z}_b \times \mathbf{x}_{yaw}\| = \sqrt{(-c\sin\theta)^2 + (c\cos\theta)^2 + (a\sin\theta - b\cos\theta)^2} \\
&= \sqrt{c^2 + a^2\sin^2\theta + b^2\cos^2\theta - 2ab\sin\theta\cos\theta} \\
&= \sqrt{1 - a^2 - b^2 + a^2\sin^2\theta + b^2\cos^2\theta - 2ab\sin\theta\cos\theta} \\
&= \sqrt{1 - a^2(1 - \sin^2\theta) - b^2(1 - \cos^2\theta) - 2ab\sin\theta\cos\theta} \\
&= \sqrt{1 - a^2\cos^2\theta - b^2\sin^2\theta - 2ab\sin\theta\cos\theta} = \sqrt{1 - (a\cos\theta + b\sin\theta)^2}
\end{aligned}$$

Then $\cos\phi_x$ can be further written as:

$$\begin{aligned}
\cos\phi_x &= \frac{(\mathbf{z}_b \times \mathbf{x}_{yaw} \times \mathbf{z}_b)^\top \mathbf{x}_{yaw}}{\|\mathbf{z}_b \times \mathbf{x}_{yaw}\|} \\
&= \begin{bmatrix} \cos\theta - a(a\cos\theta + b\sin\theta) \\ \sin\theta - b(a\cos\theta + b\sin\theta) \\ -c(a\cos\theta + b\sin\theta) \end{bmatrix}^\top \cdot \begin{bmatrix} \cos\theta \\ \sin\theta \\ 0 \end{bmatrix} \cdot \frac{1}{\|\mathbf{z}_b \times \mathbf{x}_{yaw}\|} \\
&= \frac{\cos\theta(\cos\theta - a(a\cos\theta + b\sin\theta)) + \sin\theta(\sin\theta - b(a\cos\theta + b\sin\theta))}{\|\mathbf{z}_b \times \mathbf{x}_{yaw}\|} \\
&= \frac{\cos^2\theta - a^2\cos^2\theta - ab\sin\theta\cos\theta + \sin^2\theta - ab\sin\theta\cos\theta - b^2\sin^2\theta}{\sqrt{(-c\sin\theta)^2 + (c\cos\theta)^2 + (a\sin\theta - b\cos\theta)^2}} \\
&= \frac{1 - (a\cos\theta + b\sin\theta)^2}{\sqrt{1 - (a\cos\theta + b\sin\theta)^2}} = \sqrt{1 - (a\cos\theta + b\sin\theta)^2} = \sqrt{1 - (\mathbf{z}_b \cdot \mathbf{x}_{yaw})^2}
\end{aligned}$$

Assuming $r = a\cos\theta + b\sin\theta = \mathbf{z}_b \cdot \mathbf{x}_{yaw}$, $\cos\phi_x = \sqrt{1 - r^2}$.

For $\sin\varphi_x$, it can be written as:

$$\sin\varphi_x = \mathbf{x}_b^\top \mathbf{b}_3 = \frac{(\mathbf{z}_b \times \mathbf{x}_{yaw} \times \mathbf{z}_b)^\top \mathbf{b}_3}{\|\mathbf{z}_b \times \mathbf{x}_{yaw}\|} \tag{3-26}$$

$$= \begin{bmatrix} \cos\theta - a(a\cos\theta + b\sin\theta) \\ \sin\theta - b(a\cos\theta + b\sin\theta) \\ -c(a\cos\theta + b\sin\theta) \end{bmatrix}^\top \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \cdot \frac{1}{\|\mathbf{z}_b \times \mathbf{x}_{yaw}\|} \tag{3-27}$$

$$= \frac{-c(a\cos\theta + b\sin\theta)}{\sqrt{1-(a\cos\theta + b\sin\theta)^2}} = \frac{-cr}{\sqrt{1-r^2}} \tag{3-28}$$

Similarly, $\cos\phi_y$ can be written as:

$$\cos\phi_y = \mathbf{y}_b^\top \mathbf{y}_{yaw} = \frac{(\mathbf{z}_b \times \mathbf{x}_{yaw})^\top \mathbf{y}_{yaw}}{\|\mathbf{z}_b \times \mathbf{x}_{yaw}\|}$$

$$= \begin{bmatrix} -c\sin\theta \\ c\cos\theta \\ a\sin\theta - b\cos\theta \end{bmatrix}^\top \cdot \begin{bmatrix} -\sin\theta \\ \cos\theta \\ 0 \end{bmatrix} \cdot \frac{1}{\|\mathbf{z}_b \times \mathbf{x}_{yaw}\|}$$

$$= \frac{c\sin^2\theta + c\cos^2\theta}{\sqrt{1-r^2}} = \frac{c}{\sqrt{1-r^2}}$$

Then $\sin\varphi_y$ can be denoted as

$$\sin\varphi_y = \mathbf{y}_b^\top \mathbf{b}_3 = \frac{(\mathbf{z}_b \times \mathbf{x}_{yaw})^\top \mathbf{b}_3}{\|\mathbf{z}_b \times \mathbf{x}_{yaw}\|}$$

$$= \begin{bmatrix} -c\sin\theta \\ c\cos\theta \\ a\sin\theta - b\cos\theta \end{bmatrix}^\top \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \cdot \frac{1}{\|\mathbf{z}_b \times \mathbf{x}_{yaw}\|}$$

$$= \frac{a\sin\theta - b\cos\theta}{\sqrt{1-r^2}} = \frac{s}{\sqrt{1-r^2}}$$

where $s = a\sin\theta - b\cos\theta$. Finally, the $\cos\xi$ can be simplified as:

$$\cos\xi = \mathbf{z}_b^\top \mathbf{b}_3 = c \tag{3-29}$$

By doing the above simplification, the components of terrain variable vector s_{var} are converted in the form of using the combination of c, $r = a\cos\theta + b\sin\theta$, and $s = a\sin\theta - b\cos\theta$. Then the terrain gradients with respect to time can be denoted as $\nabla_{\mathbf{x}}\cos\phi_x$, $\nabla_{\mathbf{x}}\cos\phi_y$, $\nabla_{\mathbf{x}}\sin\varphi_x$, $\nabla_{\mathbf{x}}\sin\varphi_y$ and $\nabla_{\mathbf{x}}\cos\xi$ respectively:

$$\nabla_{\mathbf{x}}\cos\phi_x = \nabla_{\mathbf{x}}\sqrt{1-r^2} = -\frac{r}{\sqrt{1-r^2}}\nabla_{\mathbf{x}} r$$

$$\nabla_{\mathbf{x}}\cos\phi_y = \nabla_{\mathbf{x}}\frac{c}{\sqrt{1-r^2}} = (1-r^2)^{-\frac{1}{2}}\nabla_{\mathbf{x}} c + r(1-r^2)^{-\frac{3}{2}} c\nabla_{\mathbf{x}} r$$

$$\nabla_{\mathbf{x}}\sin\varphi_x = \nabla_{\mathbf{x}}\frac{-cr}{\sqrt{1-r^2}} = -\frac{r}{\sqrt{1-r^2}}\nabla_{\mathbf{x}} c - (1-r^2)^{-\frac{3}{2}} c\nabla_{\mathbf{x}} r$$

$$\nabla_{\mathbf{x}}\sin\varphi_y = \nabla_{\mathbf{x}}\frac{s}{\sqrt{1-r^2}} = (1-r^2)^{-\frac{1}{2}}\nabla_{\mathbf{x}} s + r(1-r^2)^{-\frac{3}{2}} s\nabla_{\mathbf{x}} r$$

$$\nabla_{\mathbf{x}}\cos\xi = \nabla_{\mathbf{x}} c$$

In the terrain pose mapping algorithm, the derivatives of a and b, which are components of vector $\mathbf{z}_b$, with respect to the pose $\mathbf{p} = [x, y, \theta]^\top$, can be calculated as $\nabla_{\mathbf{x}} a$ and $\nabla_{\mathbf{x}} b$ by trilinear interpolation method [42].

- Adjust pose- Adjust the current pose $\mathbf{p}$ by half of the resolution.

$$\mathbf{p}_m = \mathbf{p} - [\Delta x/2, \Delta y/2, \Delta\theta/2]^\top$$

  where Δx, Δy and $\Delta\theta$ are grid map resolution in x, y and θ respectively.

- Convert pose to index and compute differences- For pose $\mathbf{p}_m$, we first determine its corresponding index i in the grid map. This index i represents the discrete grid cell that contains the pose $\mathbf{p}_m$. Next, we use this index i to retrieve the actual pose $\mathbf{p}_{idx}$ corresponding to the grid cell's center indexed by i. ($\mathbf{p}_m$ adjusted from $\mathbf{p}$ represents a pose within the discrete grid cell i, while the corresponding $\mathbf{p}_{idx}$ denotes the center pose of the same grid cell.). Then, the difference between the original pose $\mathbf{p}$ and the indexed pose $\mathbf{p}_{idx}$ is scaled by the inverse of the grid resolution.

$$\begin{aligned}
\mathrm{diff_v} &= \mathbf{p} - \mathbf{p}_{\mathrm{idx}} \\
\mathrm{diff_x} &= \frac{\mathrm{diff_v}(0)}{\Delta \mathrm{x}} \\
\mathrm{diff_y} &= \frac{\mathrm{diff_v}(1)}{\Delta \mathrm{y}} \\
\mathrm{diff}_\theta &= \frac{1}{\Delta\theta} \arctan\left(\frac{\sin(\mathbf{p}(2) - \mathbf{p}_{\mathrm{idx}}(2))}{\cos(\mathbf{p}(2) - \mathbf{p}_{\mathrm{idx}}(2))}\right)
\end{aligned}$$

- Interpolate along the x-axis- For pose $\mathbf{p}$, considering its surrounding 8 grid points, P_{000}, P_{001}, P_{010}, P_{011}, P_{100}, P_{101}, P_{110} and P_{111}, the interpolated values along x-axis can be obtained as:

$$\begin{aligned}
v_{00} &= \mathrm{P}_{000}(1 - \mathrm{diff_x}) + \mathrm{P}_{100}\mathrm{diff_x} \\
v_{01} &= \mathrm{P}_{001}(1 - \mathrm{diff_x}) + \mathrm{P}_{101}\mathrm{diff_x} \\
v_{10} &= \mathrm{P}_{010}(1 - \mathrm{diff_x}) + \mathrm{P}_{110}\mathrm{diff_x} \\
v_{11} &= \mathrm{P}_{011}(1 - \mathrm{diff_x}) + \mathrm{P}_{111}\mathrm{diff_x}
\end{aligned}$$

- Interpolate along the y-axis

$$\begin{aligned}
v_0 &= v_{00}(1 - \mathrm{diff_y}) + v_{01}\mathrm{diff_y} \\
v_1 &= v_{01}(1 - \mathrm{diff_y}) + v_{11}\mathrm{diff_y}
\end{aligned}$$

- Interpolate along the yaw-axis

$$v = v_0(1 - \mathrm{diff}_\theta) + v_1\mathrm{diff}_\theta$$

- Calculate gradients

$$\begin{aligned}
\mathbf{g}_\theta =& \frac{v_1 - v_0}{\Delta\theta} \\
\mathbf{g}_y =& \frac{(v_{10} - v_{00})(1 - \text{diff}_\theta) + (v_{11} - v_{01})\text{diff}_\theta}{\Delta x} \\
\mathbf{g}_x =& ((1 - \text{diff}_\theta)(1 - \text{diff}_\text{y})(\text{P}_{100} - \text{P}_{000}) + (1 - \text{diff}_\theta)\text{diff}_\text{y}(\text{P}_{110} - \text{P}_{010}) + \\
& + \text{diff}_\theta(1 - \text{diff}_\text{y})(\text{P}_{101} - \text{P}_{001}) + \text{diff}_\theta\text{diff}_\text{y}(\text{P}_{111} - \text{P}_{011}))/\Delta\text{y}
\end{aligned}$$

  Then the terrain gradients can be written as

$$\begin{bmatrix} \mathbf{g}_x & \mathbf{g}_y & \mathbf{g}_\theta \end{bmatrix} = \begin{bmatrix} \nabla_x a & \nabla_y a & \nabla_\theta a \\ \nabla_x b & \nabla_y b & \nabla_\theta b \\ \nabla_x c & \nabla_y c & \nabla_\theta c \end{bmatrix} \tag{3-30}$$

  where $\nabla_\mathbf{x} a = \begin{bmatrix} \nabla_x a & \nabla_y a & \nabla_\theta a \end{bmatrix}$, $\nabla_\mathbf{x} b = \begin{bmatrix} \nabla_x b & \nabla_y b & \nabla_\theta b \end{bmatrix}$ and $\nabla_\mathbf{x} c = \begin{bmatrix} \nabla_x c & \nabla_y c & \nabla_\theta c \end{bmatrix}$.

The derivative of c with respect to the state $\mathbf{x}$ can be represented as:

$$\nabla_\mathbf{x} c = \nabla_\mathbf{x} \sqrt{1 - a^2 - b^2} = \frac{-2a\nabla_\mathbf{x} a - 2b\nabla_\mathbf{x} b}{2\sqrt{1 - a^2 - b^2}} = -(a\nabla_\mathbf{x} a + b\nabla_\mathbf{x} b)/c \tag{3-31}$$

The derivative of r with respect to the state $\mathbf{x}$ can be denoted as:

$$\nabla_\mathbf{x} r = \nabla_\mathbf{x}(a\cos\theta + b\sin\theta) = \cos\theta\nabla_\mathbf{x} a + \sin\theta\nabla_\mathbf{x} b \tag{3-32}$$

The derivative of s with respect to the state $\mathbf{x}$ can be denoted as:

$$\nabla_\mathbf{x} s = \nabla_\mathbf{x}(a\sin\theta - b\cos\theta) = \sin\theta\nabla_\mathbf{x} a - \cos\theta\nabla_\mathbf{x} b \tag{3-33}$$

3-3 Trajectory Optimization

Quintic piecewise polynomials are employed to describe trajectories in the SE(2) space. Each trajectory segment can be expressed as:

$$\begin{aligned}
x_i(t) &= c_{x_i}^\top \gamma(t), \quad t \in [0, T_i] \\
y_j(t) &= c_{y_j}^\top \gamma(t), \quad t \in [0, T_j] \\
\theta_k(t) &= c_{\theta_k}^\top \gamma(t), \quad t \in [0, T_k]
\end{aligned} \tag{3-34}$$

where N_i, N_j, N_k denote the number of trajectory segments, and T_i, T_j and T_k represent the time durations for each segment in dimensions x, y and θ, in that order; For simplicity the N_i and N_j are set to be the same; the natural basis for the polynomial is given by $\gamma(t) = [1, t, t^2, t^3, t^4, t^5]^\top$; $c_* \in \mathbb{R}^{6\times 1}$ with $* \in i, j, k$ is the coefficient vector of polynomial.

The remarkable aspect lies in the ability to encapsulate the 3D state trajectory and dynamic states using only the 2D state trajectory through terrain pose mapping. Through this mapping, each point $\mathbf{s}_p = [x, y, \theta]^\top$ on the 2D trajectory yields distinct values for height z and

body axis $\mathbf{z}_b$. The optimization problem can be formulated as:

$$\min_{\mathbf{c}_{xy},\mathbf{c}_\theta,\mathbf{T}_{xy},\mathbf{T}_\theta} \int_0^{T_s} (\mathbf{j}(t)^\top \mathbf{j}(t) + \rho_{ter}\sigma(\mathbf{x}(t)))dt + \rho_T T_s \tag{3-35}$$

$$s.t. \quad \dot{x}\sin\theta - \dot{y}\cos\theta = 0 \tag{3-36}$$

$$\mathbf{G}_{xy}\mathbf{c}_{xy} = \mathbf{b}_{xy}, \quad \mathbf{G}_\theta \mathbf{c}_\theta = \mathbf{b}_\theta \tag{3-37}$$

$$\mathbf{T}_{xy} \succeq 0, \quad \mathbf{T}_\theta \succeq 0 \tag{3-38}$$

$$v_x^2 - v_{max}^2 \leq 0 \tag{3-39}$$

$$a_x^2 - a_{mlon}^2 \leq 0 \tag{3-40}$$

$$a_y^2 - a_{mlat}^2 \leq 0 \tag{3-41}$$

$$\frac{w_z^2}{v_x^2 + \delta_+} - \frac{\tan^2\delta_{max}}{L_w^2} \leq 0 \tag{3-42}$$

$$c_{min} - \cos\xi \leq 0 \tag{3-43}$$

$$\sigma(\mathbf{x}) - \sigma_{max} \leq 0 \tag{3-44}$$

where the smoothness of trajectory can be represented by the square term $\mathbf{j}(t)^\top \mathbf{j}(t)$, with trajectory's jerk $\mathbf{j}(t) = \mathbf{x}^{(3)}(t)$; $\sigma(\mathbf{x}(t))$ is the surface variation which can be computed in the process of generating terrain pose mapping $\mathcal{F}$ based on the current point on the trajectory $\mathbf{x}(t)$, ρ_{ter} is a constant; $\mathbf{c}_{xy}$, $\mathbf{c}_\theta$ are the coefficient matrix of trajectory segments,

$$\mathbf{c}_{xy} = \begin{bmatrix} \mathbf{c}_{1xy} \\ \mathbf{c}_{2xy} \\ \vdots \\ \mathbf{c}_{Mxy} \end{bmatrix} = \begin{bmatrix} c_{x_1} & c_{y_1} \\ c_{x_2} & c_{y_2} \\ \vdots & \vdots \\ c_{x_M} & c_{y_M} \end{bmatrix} \in \mathbb{R}^{6M\times 2}, \quad \mathbf{c}_\theta = \begin{bmatrix} c_{\theta_1} & c_{\theta_2} & \dots & c_{\theta_\Omega} \end{bmatrix}^\top \in \mathbb{R}^{6\Omega\times 1}$$

$\mathbf{T}_{xy} \in \mathbb{R}^M$ and $\mathbf{T}_\theta = \in \mathbb{R}^\Omega$ represent the time vectors, with each component of these vectors being non-negative and $\|\mathbf{T}_{xy}\|_1 = \|\mathbf{T}_\theta\|_1 = T_s$; $\mathbf{G}_{xy}$ and $\mathbf{G}_\theta$ are constraint matrices while $\mathbf{b}_{xy}$ and $\mathbf{b}_\theta$ are corresponding constraint vectors; Dynamic inequalities conditions include limits on v_x, a_x, a_y, κ, attitude $\cos\xi$ and terrain curvature $\sigma(\mathbf{x})$. The constants involved in these inequalities are v_{max}, a_{mlon}, a_{mlat}, δ_{max}, c_{min}, σ_{max}.

3-3-1 Constraints in the Optimization Problem

(3-36) ensures that the vehicle follows the non-holonomic constraint. This constraint arises from the kinematic model of the car, which limits the motion to a direction aligned with the wheels' orientation. According to **Assumption 2**, wheel slip is not considered. Then the component of the velocity perpendicular to the heading direction must be zero

$$v_x\sin\theta = v_y\cos\theta \rightarrow \quad \dot{x}\sin\theta - \dot{y}\cos\theta = 0 \tag{3-45}$$

To ensure a smooth trajectory, the condition of being continuously differentiable four times at the segmented points, along with boundary conditions for the trajectory, is incorporated

into the constrain (3-37). For example, the segment i, the x and y can be represented as

$$\begin{aligned}
\mathbf{p}(t) &= \mathbf{c}_{ixy}\gamma(t) = \mathbf{c}_{i0} + \mathbf{c}_{i1}t + \mathbf{c}_{i2}t^2 + \mathbf{c}_{i3}t^3 + \mathbf{c}_{i4}t^4 + \mathbf{c}_{i5}t^5 \\
\frac{\mathrm{d}\mathbf{p}(t)}{\mathrm{d}t} &= \mathbf{c}_{i1} + 2\mathbf{c}_{i2}t + 3\mathbf{c}_{i3}t^2 + 4\mathbf{c}_{i4}t^3 + 5\mathbf{c}_{i5}t^4 \\
\frac{\mathrm{d}^2\mathbf{p}(t)}{\mathrm{d}t^2} &= 2\mathbf{c}_{i2} + 6\mathbf{c}_{i3}t + 12\mathbf{c}_{i4}t^2 + 20\mathbf{c}_{i5}t^3 \\
\frac{\mathrm{d}^3\mathbf{p}(t)}{\mathrm{d}t^3} &= 6\mathbf{c}_{i3} + 24\mathbf{c}_{i4}t + 60\mathbf{c}_{i5}t^2 \\
\frac{\mathrm{d}^4\mathbf{p}(t)}{\mathrm{d}t^4} &= 24\mathbf{c}_{i4} + 120\mathbf{c}_{i5}t
\end{aligned} \tag{3-46}$$

In order to ensure that the trajectory is four times continuously differentiable at the segment boundaries, the constraints should be built as follows:

$$\begin{aligned}
p_i(t)\mid_{t=T_i} &= p_{i+1}(t)\mid_{t=0} \\
\frac{\mathrm{d}\mathbf{p}_i(t)}{\mathrm{d}t}\Big|_{t=T_i} &= \frac{\mathrm{d}\mathbf{p}_{i+1}(t)}{\mathrm{d}t}\Big|_{t=0} \\
\frac{\mathrm{d}^2\mathbf{p}_i(t)}{\mathrm{d}t^2}\Big|_{t=T_i} &= \frac{\mathrm{d}^2\mathbf{p}_{i+1}(t)}{\mathrm{d}t^2}\Big|_{t=0} \\
\frac{\mathrm{d}^3\mathbf{p}_i(t)}{\mathrm{d}t^3}\Big|_{t=T_i} &= \frac{\mathrm{d}^3\mathbf{p}_{i+1}(t)}{\mathrm{d}t^3}\Big|_{t=0} \\
\frac{\mathrm{d}^4\mathbf{p}_i(t)}{\mathrm{d}t^4}\Big|_{t=T_i} &= \frac{\mathrm{d}^4\mathbf{p}_{i+1}(t)}{\mathrm{d}t^4}\Big|_{t=0}
\end{aligned} \tag{3-47}$$

Also, the start and the endpoints of the trajectory should align with the ego vehicle's initial pose and the set goal pose,

$$\begin{aligned}
\begin{bmatrix}\mathbf{p}_1(0) & \dot{\mathbf{p}}_1(0) & \ddot{\mathbf{p}}_1(0)\end{bmatrix} &= \begin{bmatrix}\mathbf{p}_{\text{init}} & \dot{\mathbf{p}}_{\text{init}} & \ddot{\mathbf{p}}_{\text{init}}\end{bmatrix} \\
\begin{bmatrix}\mathbf{p}_M(T_M) & \dot{\mathbf{p}}_M(T_M) & \ddot{\mathbf{p}}_M(T_M)\end{bmatrix} &= \begin{bmatrix}\mathbf{p}_{\text{final}} & \dot{\mathbf{p}}_{\text{final}} & \ddot{\mathbf{p}}_{\text{final}}\end{bmatrix}
\end{aligned} \tag{3-48}$$

where $\mathbf{p}_{\text{init}}$ and $\mathbf{p}_{\text{final}}$, $\dot{\mathbf{p}}_{\text{init}}$ and $\dot{\mathbf{p}}_{\text{final}}$, and $\ddot{\mathbf{p}}_{\text{init}}$ and $\ddot{\mathbf{p}}_{\text{final}}$ are the initial and goal spatial locations, velocity and acceleration respectively. By combining (3-47) and (3-48), the equality $\mathbf{G}_{xy}\mathbf{c}_{xy} = \mathbf{b}_{xy}$ can be further represented as:

$$\mathbf{G}_{xy} :=$$

$$\begin{bmatrix}
1 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
0 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
0 & 0 & 2 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\
0 & 0 & 0 & 6 & 24t & 60t^2 & 0 & 0 & 0 & -6 & 0 & 0 & \cdots & \cdots & \cdots \\
0 & 0 & 0 & 0 & 24 & 120t & 0 & 0 & 0 & 0 & -24 & 0 & \cdots & \cdots & \cdots \\
1 & t & t^2 & t^3 & t^4 & t^5 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & \cdots & \cdots \\
1 & t & t^2 & t^3 & t^4 & t^5 & -1 & 0 & 0 & 0 & 0 & 0 & \cdots & \cdots & \cdots \\
0 & 1 & 2t & 3t^2 & 4t^3 & 5t^4 & 0 & -1 & 0 & 0 & 0 & 0 & \cdots & \cdots & \cdots \\
0 & 0 & 2 & 6t & 12t^2 & 20t^3 & 0 & 0 & -2 & 0 & 0 & 0 & \cdots & \cdots & \cdots \\
 & & & \mathbf{0}_{6\times6} & & & & & \mathbf{A} & & & & \cdots & \cdots \\
 & & & \mathbf{0}_{6\times6} & & & & & \mathbf{0}_{6\times6} & & & \mathbf{A} & \cdots & \cdots \\
 & & & \mathbf{0}_{6\times6} & & & & & \mathbf{0}_{6\times6} & & & \mathbf{0}_{6\times12} & \mathbf{A} & \cdots \\
 & & & \vdots & & & & & \vdots & & & \vdots & \vdots & \vdots \\
 & & & \cdots & & & & & \cdots & & & \cdots & \cdots & \mathbf{B}
\end{bmatrix}$$

where

$$\mathbf{A} = \begin{bmatrix}
0 & 0 & 0 & 6 & 24t & 60t^2 & 0 & 0 & 0 & -6 & 0 & 0 \\
0 & 0 & 0 & 0 & 24 & 120t & 0 & 0 & 0 & 0 & -24 & 0 \\
1 & t & t^2 & t^3 & t^4 & t^5 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & t & t^2 & t^3 & t^4 & t^5 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 2t & 3t^2 & 4t^3 & 5t^4 & 0 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 2 & 6t & 12t^2 & 20t^3 & 0 & 0 & -2 & 0 & 0 & 0
\end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix}
1 & t & t^2 & t^3 & t^4 & t^5 \\
0 & 1 & 2t & 3t^2 & 4t^3 & 5t^4 \\
0 & 0 & 2 & 6t & 12t^2 & 20t^3
\end{bmatrix}$$

$\mathbf{0}_{6\times6}$ represents zero matrix with the size 6×6. Then $\mathbf{b}_{xy}$ can be represented as

$$\mathbf{b}_{xy} = \begin{bmatrix} \mathbf{p}_{\text{init}}^\top & \dot{\mathbf{p}}_{\text{init}}^\top & \ddot{\mathbf{p}}_{\text{init}}^\top & \mathbf{E}^\top & \mathbf{p}_{\text{final}}^\top & \dot{\mathbf{p}}_{\text{final}}^\top & \ddot{\mathbf{p}}_{\text{final}}^\top \end{bmatrix}^\top$$

$$\mathbf{E}^\top = \begin{bmatrix} \mathbf{E}_{\text{init}} & \mathbf{E}_{\text{mid}} & \mathbf{E}_{\text{mid}} & \cdots & \mathbf{E}_{\text{final}} \end{bmatrix}$$

where $\mathbf{E}_{\text{init}} = \begin{bmatrix} \mathbf{p}_{\text{init}}^\top & \dot{\mathbf{p}}_{\text{init}}^\top & \ddot{\mathbf{p}}_{\text{init}}^\top \end{bmatrix}$, $\mathbf{E}_{\text{mid}} = \begin{bmatrix} \mathbf{0}_{2\times1} & \mathbf{0}_{2\times1} & \mathbf{I}_g^\top & \mathbf{0}_{2\times1} & \mathbf{0}_{2\times1} & \mathbf{0}_{2\times1} \end{bmatrix}$ and $\mathbf{E}_{\text{final}} = \begin{bmatrix} \mathbf{p}_{\text{final}}^\top & \dot{\mathbf{p}}_{\text{final}}^\top & \ddot{\mathbf{p}}_{\text{final}}^\top \end{bmatrix}$, $\mathbf{l}_g$ is the expected pose at the end of segment i, which can be obtained from the initial path. The first three rows $\mathbf{G}_{xy}\mathbf{c}_{xy} = \mathbf{b}_{xy}$ constrain the initial conditions shown in (3-48).

$$\begin{aligned}
&\mathbf{p}_1(0) = \mathbf{c}_{1xy}\gamma(t)\Big|_{t=0} = (\mathbf{c}_{10} + \mathbf{c}_{11}t + \mathbf{c}_{12}t^2 + \mathbf{c}_{13}t^3 + \mathbf{c}_{14}t^4 + \mathbf{c}_{15}t^5)\Big|_{t=0} = \mathbf{c}_{10} = \mathbf{p}_{\text{init}} \\
&\dot{\mathbf{p}}_1(0) = (\mathbf{c}_{11} + 2\mathbf{c}_{12}t + 3\mathbf{c}_{13}t^2 + 4\mathbf{c}_{14}t^3 + 5\mathbf{c}_{15}t^4)\Big|_{t=0} = \mathbf{c}_{11} = \dot{\mathbf{p}}_{\text{init}} \\
&\ddot{\mathbf{p}}_1(0) = (2\mathbf{c}_{12} + 6\mathbf{c}_{13}t + 12\mathbf{c}_{14}t^2 + 20\mathbf{c}_{15}t^3)\Big|_{t=0} = 2\mathbf{c}_{12} = \ddot{\mathbf{p}}_{\text{init}} \\
&\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \end{bmatrix} \mathbf{c}_{1xy} = \begin{bmatrix} \mathbf{p}_{\text{init}} \\ \dot{\mathbf{p}}_{\text{init}} \\ \ddot{\mathbf{p}}_{\text{init}} \end{bmatrix}
\end{aligned} \tag{3-49}$$

Similarly, the last three rows constrain the goal conditions,

$$\begin{aligned}
&\mathbf{p}_M(T_M) = (\mathbf{c}_{M0} + \mathbf{c}_{M1}t + \mathbf{c}_{M2}t^2 + \mathbf{c}_{M3}t^3 + \mathbf{c}_{M4}t^4 + \mathbf{c}_{M5}t^5)\Big|_{t=T_M} = \mathbf{p}_{\text{final}} \\
&\dot{\mathbf{p}}_M(T_M) = (\mathbf{c}_{M1} + 2\mathbf{c}_{M2}t + 3\mathbf{c}_{M3}t^2 + 4\mathbf{c}_{M4}t^3 + 5\mathbf{c}_{M5}t^4)\Big|_{t=0} = \dot{\mathbf{p}}_{\text{final}} \\
&\ddot{\mathbf{p}}_M(T_M) = (2\mathbf{c}_{M2} + 6\mathbf{c}_{M3}t + 12\mathbf{c}_{M4}t^2 + 20\mathbf{c}_{M5}t^3)\Big|_{t=T_M} = \ddot{\mathbf{p}}_{\text{final}} \\
&\begin{bmatrix} 1 & t & t^2 & t^3 & t^4 & t^5 \\ 0 & 1 & 2t & 3t^2 & 4t^3 & 5t^4 \\ 0 & 0 & 2 & 6t & 12t^2 & 20t^3 \end{bmatrix} \mathbf{c}_{Mxy} = \mathbf{B}\mathbf{c}_{Mxy} = \begin{bmatrix} \mathbf{p}_{\text{final}} \\ \dot{\mathbf{p}}_{\text{final}} \\ \ddot{\mathbf{p}}_{\text{final}} \end{bmatrix}
\end{aligned} \tag{3-50}$$

The middle rows correspond to the trajectory having continuous fourth-order differentiability at the segment junctions, taking segment i as an example:

- First row in Matrix $\mathbf{A}$:

$$\begin{aligned}
&\frac{\mathrm{d}^3\mathbf{p}_i(t)}{\mathrm{d}t^3}\Big|_{t=T_i} = \frac{\mathrm{d}^3\mathbf{p}_{i+1}(t)}{\mathrm{d}t^3}\Big|_{t=0} \\
&\rightarrow (6\mathbf{c}_{i3} + 24\mathbf{c}_{i4}t + 60\mathbf{c}_{i5}t^2)\Big|_{t=T_i} = (6\mathbf{c}_{(i+1)3} + 24\mathbf{c}_{(i+1)4}t + 60\mathbf{c}_{(i+1)5}t^2)\Big|_{t=0} \\
&\rightarrow 6\mathbf{c}_{i3} + 24\mathbf{c}_{i4}T_i + 60\mathbf{c}_{i5}T_i^2 = 6\mathbf{c}_{(i+1)3} \\
&\rightarrow \begin{bmatrix} 0 & 0 & 0 & 6 & 24t & 60t^2 & 0 & 0 & 0 & -6 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{c}_{ixy} \\ \mathbf{c}_{(i+1)xy} \end{bmatrix} = 0
\end{aligned}$$

- Second row in Matrix $\mathbf{A}$:

$$\begin{aligned}
&\frac{\mathrm{d}^4\mathbf{p}_i(t)}{\mathrm{d}t^4}\Big|_{t=T_i} = \frac{\mathrm{d}^4\mathbf{p}_{i+1}(t)}{\mathrm{d}t^4}\Big|_{t=0} \\
&\rightarrow (24\mathbf{c}_{i4} + 120\mathbf{c}_{i5}t)\Big|_{t=T_i} = (24\mathbf{c}_{(i+1)4} + 120\mathbf{c}_{(i+1)5}t)\Big|_{t=0} \\
&\rightarrow 24\mathbf{c}_{i4} + 120\mathbf{c}_{i5}T_i = 24\mathbf{c}_{(i+1)4} \\
&\rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 & 24 & 120t & 0 & 0 & 0 & 0 & -24 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{c}_{ixy} \\ \mathbf{c}_{(i+1)xy} \end{bmatrix} = 0
\end{aligned}$$

- Third row in Matrix $\mathbf{A}$:

$$\begin{aligned}
&p_i(t)\Big|_{t=T_i} = \mathbf{l}_g \\
&\rightarrow (\mathbf{c}_{i0} + \mathbf{c}_{i1}t + \mathbf{c}_{i2}t^2 + \mathbf{c}_{i3}t^3 + \mathbf{c}_{i4}t^4 + \mathbf{c}_{i5}t^5)\Big|_{t=T_i} = \mathbf{l}_g \\
&\rightarrow \mathbf{c}_{i0} + \mathbf{c}_{i1}T_i + \mathbf{c}_{i2}T_i^2 + \mathbf{c}_{i3}T_i^3 + \mathbf{c}_{i4}T_i^4 + \mathbf{c}_{i5}T_i^5 = \mathbf{l}_g \\
&\rightarrow \begin{bmatrix} 1 & t & t^2 & t^3 & t^4 & t^5 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{c}_{ixy} \\ \mathbf{c}_{(i+1)xy} \end{bmatrix} = \mathbf{l}_g
\end{aligned}$$

- Fourth row in Matrix $\mathbf{A}$:

$$
\begin{aligned}
& p_i(t)\Big|_{t=T_i} = p_{i+1}(t)\Big|_{t=0} \\
& \rightarrow (\mathbf{c}_{i0} + \mathbf{c}_{i1}t + \mathbf{c}_{i2}t^2 + \mathbf{c}_{i3}t^3 + \mathbf{c}_{i4}t^4 + \mathbf{c}_{i5}t^5)\Big|_{t=T_i} \\
& \quad = (\mathbf{c}_{(i+1)0} + \mathbf{c}_{(i+1)1}t + \mathbf{c}_{(i+1)2}t^2 + \mathbf{c}_{(i+1)3}t^3 + \mathbf{c}_{(i+1)4}t^4 + \mathbf{c}_{(i+1)5}t^5)\Big|_{t=0} \\
& \rightarrow \mathbf{c}_{i0} + \mathbf{c}_{i1}T_i + \mathbf{c}_{i2}T_i^2 + \mathbf{c}_{i3}T_i^3 + \mathbf{c}_{i4}T_i^4 + \mathbf{c}_{i5}T_i^5 = \mathbf{c}_{(i+1)0} \\
& \rightarrow \begin{bmatrix} 1 & t & t^2 & t^3 & t^4 & t^5 & -1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{c}_{ixy} \\ \mathbf{c}_{(i+1)xy} \end{bmatrix} = 0
\end{aligned}
$$

- Fifth row in Matrix $\mathbf{A}$:

$$
\begin{aligned}
& \frac{\mathrm{d}\mathbf{p}_i(t)}{\mathrm{d}t}\Big|_{t=T_i} = \frac{\mathrm{d}\mathbf{p}_{i+1}(t)}{\mathrm{d}t}\Big|_{t=0} \\
& \rightarrow (\mathbf{c}_{i1} + 2\mathbf{c}_{i2}t + 3\mathbf{c}_{i3}t^2 + 4\mathbf{c}_{i4}t^3 + 5\mathbf{c}_{i5}t^4)\Big|_{t=T_i} \\
& \quad = (\mathbf{c}_{(i+1)1} + 2\mathbf{c}_{(i+1)2}t + 3\mathbf{c}_{(i+1)3}t^2 + 4\mathbf{c}_{(i+1)4}t^3 + 5\mathbf{c}_{(i+1)5}t^4)\Big|_{t=0} \\
& \rightarrow \mathbf{c}_{i1} + 2\mathbf{c}_{i2}T_i + 3\mathbf{c}_{i3}T_i^2 + 4\mathbf{c}_{i4}T_i^3 + 5\mathbf{c}_{i5}T_i^4 = \mathbf{c}_{(i+1)1} \\
& \rightarrow \begin{bmatrix} 0 & 1 & 2t & 3t^2 & 4t^3 & 5t^4 & 0 & -1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{c}_{ixy} \\ \mathbf{c}_{(i+1)xy} \end{bmatrix} = 0
\end{aligned}
$$

- Sixth row in Matrix $\mathbf{A}$:

$$
\begin{aligned}
& \frac{\mathrm{d}^2\mathbf{p}_i(t)}{\mathrm{d}t^2}\Big|_{t=T_i} = \frac{\mathrm{d}^2\mathbf{p}_{i+1}(t)}{\mathrm{d}t^2}\Big|_{t=0} \\
& \rightarrow (2\mathbf{c}_{i2} + 6\mathbf{c}_{i3}t + 12\mathbf{c}_{i4}t^2 + 20\mathbf{c}_{i5}t^3)\Big|_{t=T_i} \\
& \quad = (2\mathbf{c}_{(i+1)2} + 6\mathbf{c}_{(i+1)3}t + 12\mathbf{c}_{(i+1)4}t^2 + 20\mathbf{c}_{(i+1)5}t^3)\Big|_{t=0} \\
& \rightarrow 2\mathbf{c}_{i2} + 6\mathbf{c}_{i3}T_i + 12\mathbf{c}_{i4}T_i^2 + 20\mathbf{c}_{i5}T_i^3 = 2\mathbf{c}_{(i+1)2} \\
& \rightarrow \begin{bmatrix} 0 & 0 & 2 & 6t & 12t^2 & 20t^3 & 0 & 0 & -2 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{c}_{ixy} \\ \mathbf{c}_{(i+1)xy} \end{bmatrix} = 0
\end{aligned}
$$

The PHR Augmented Lagrange Multiplier method (PHR-ALM) [1] effectively integrates constraints into the optimization problem. Given a set of equality constraints $h(\mathbf{x}) = 0$ and inequality constraints $g(\mathbf{x}) \leq 0$, the method augments the objective function with additional terms that penalize constraint violations. For each equality constraint $h_i(\mathbf{x}) = 0$ and inequality constraint $g_j(\mathbf{x}) \leq 0$, the augmented cost functions $\Lambda(h, \lambda)$ and $\Lambda(g, \mu)$ can be defined as:

$$
\begin{aligned}
\Lambda(h, \lambda) &= h(\lambda + \frac{\rho}{2}h) \\
\Lambda(g, \mu) &= g(\mu + \frac{\rho}{2}g)
\end{aligned}
$$

where λ is the Lagrange multiplier for the equality constraint. It is a vector whose dimension corresponds to the number of equality constraints; μ is the Lagrange multiplier for the

inequality constraint. It is a vector with a size N_{nec} equal to the number of inequality constraints; ρ is the penalty parameter. The gradients of the augmented cost functions with respect to the constraint variables h and g are:

$$\nabla\Lambda(h,\lambda) = \rho h + \lambda$$
$$\nabla\Lambda(g,\mu) = \rho g + \mu$$

For each constraint $h_i(\mathbf{x})$ or $g_j(\mathbf{x})$, first, we compute the augmented cost for each constraint and incorporate these costs into the total objective function. Next, we calculate the gradients of the augmented costs and utilize these gradients to update the trajectory parameters during the optimization process. The Lagrange multipliers for the equality constraints and inequality constraints λ and μ can be updated as:

$$\lambda \leftarrow \lambda + \rho \cdot h(x)$$
$$\mu_j \leftarrow \max\left(\mu_j + \rho \cdot g_j(x), 0\right) \quad \forall j \in \{1, 2, \ldots, N_{nec}\}$$

This approach ensures that the constraints are integrated into the optimization problem. By penalizing any violations, the solution is driven toward feasibility, effectively balancing the original objective and the satisfaction of constraints.

3-3-2 Cost in the Optimization Problem

The objective function includes terrain jerk cost, surface variation cost, and time efficiency cost. The terrain variation cost derives from one of the terrain variables from the terrain pose mapping process. The time efficiency cost represents the total duration of the trajectory. In practice, each segment of the trajectory is allocated an equal portion of the total trajectory time. As mentioned before, the trajectory is in the form of piecewise polynomials, taking dimension x of i-th segment of the trajectory as an example; the trajectory jerk can be represented as:

$$\mathbf{j}_1(t) = \frac{\partial^3 x_i(t)}{\partial t^3} = \frac{\partial^3 (c_0 + c_1 t + c_2 t^2 + c_3 t^3 + c_4 t^4 + c_5 t^5)}{\partial t^3} \tag{3-51}$$
$$= 6c_3 + 24c_4 t + 60c_5 t^2 \tag{3-52}$$

where $\mathbf{j}_1(t)$ is the first element of $\mathbf{j}(t)$; $c_{x_i} = [c_0, c_1, c_2, c_3, c_4, c_5]^\top$. Then $\mathbf{j}_1^\top(t)\mathbf{j}_1(t)$ can be expanded as:

$$\mathbf{j}_1^\top(t)\mathbf{j}_1(t) = \mathbf{j}_1^2(t) = (6c_3 + 24c_4 t + 60c_5 t^2)^2 \tag{3-53}$$
$$= 36c_3^2 + 288c_3c_4 t + 576c_4^2 t^2 + 720c_3c_5 t^2 + 2880c_4c_5 t^3 + 3600c_5^2 t^4 \tag{3-54}$$

Then, the integral of $\mathbf{j}_1^\top(t)\mathbf{j}_1(t)$ with respect to time t is given by

$$\int_0^{T_s} \mathbf{j}_1^\top(t)\mathbf{j}_1(t)dt$$
$$= \int_0^{T_s} (36c_3^2 + 288c_3c_4 t + 576c_4^2 t^2 + 720c_3c_5 t^2 + 2880c_4c_5 t^3 + 3600c_5^2 t^4)dt$$
$$= 36c_3^2 t + 144c_3c_4 t^2 + 192c_4^2 t^3 + 240c_3c_5 t^3 + 720c_4c_5 t^4 + 720c_5^2 t^5$$

In essence, through the creation of terrain pose mapping—establishing a one-to-one correspondence between the car-like robot's 2D spatial position, heading direction, and its 3D spatial location and altitude—an optimization problem is formulated. This optimization problem leverages the 2D space trajectory and trajectory coefficients as control inputs for optimization. Concerning 3D state constraints, each dynamic state uniquely corresponds to the projected 2D state on the trajectory, serving as explicit feasibility constraints in the optimization framework.

3-4 MPC Control

Model Predictive Control (MPC) is an advanced control strategy used to manage systems optimally while respecting certain constraints. It relies on a system dynamics model to predict future behavior and optimize control actions over a specified time horizon. The process involves minimizing an objective function and meeting a set of constraints, producing an optimal series of control inputs for the given horizon. A distinctive feature of MPC is its iterative, receding horizon approach. At each time step, an optimization problem is solved over a prediction horizon, yielding a sequence of control inputs. Only the first input is implemented, and the process repeats at the next time step with updated system states. This continual re-optimization allows MPC to adjust to new information and correct for prediction errors.

In motion planning, once the optimal trajectory is provided by the optimization block, the MPC block transforms this trajectory into a constrained, finite-time optimal control problem. This problem is then solved in real-time at each sampling instance using the most recent data. The state of the vehicle at any time t is represented by $x(t) = \begin{bmatrix} x & y & \theta \end{bmatrix}^\top$. The control inputs are $u(t) = \begin{bmatrix} v_x & \delta \end{bmatrix}^\top$ by using the Ackermann Steering Mode. Here, x and y denote the 2D spatial coordinates, and θ represents the orientation. The variable v_x signifies the longitudinal velocity of the vehicle, δ indicates the steering angle, and L represents the wheelbase of the vehicle.

$$
\begin{aligned}
x_{k+1} &= x_k + v_x \cos(\theta_k)\Delta t \\
y_{k+1} &= y_k + v_x \sin(\theta_k)\Delta t \\
\theta_{k+1} &= \theta_k + \frac{v_x \tan(\delta)}{L}\Delta t
\end{aligned}
$$

We linearize the system around the current state and input to facilitate MPC. The linearized system can be written as:

$$
x_{k+1} = Ax_k + Bu_k + c, \quad k = 0, 1, \ldots, T-1
$$

Where:

- A is the state transition matrix

$$
A = \begin{bmatrix} 1 & 0 & -v_x \sin(\theta)\Delta t \\ 0 & 1 & v_x \cos(\theta)\Delta t \\ 0 & 0 & 1 \end{bmatrix}
$$

- B is the control input matrix

$$B = \begin{bmatrix} \cos(\theta)\Delta t & 0 \\ \sin(\theta)\Delta t & 0 \\ 0 & \frac{\Delta t \tan(\delta)}{L} \\ 0 & \frac{v_x \Delta t}{L \cos^2(\delta)} \end{bmatrix}$$

- c is the constant term due to linearization

$$c = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^\top$$

The objective of the MPC optimization problem is to minimize a cost function while considering system dynamics and constraints. The cost function generally takes the following form:

$$J = \sum_{k=0}^{T-1} \Big((x_k - x_{\text{ref}})^\top Q (x_k - x_{\text{ref}}) + (u_k - u_{\text{ref}})^\top R (u_k - u_{\text{ref}}) \Big)$$

Where x_{ref} and u_{ref} represent desired state and input at time step k respectively; Q and R are weight matrices for state error and control input, respectively. The constraints include:

- State constraints: $x_{\min} \leq x_k \leq x_{\max}$
- Control input constraints: $u_{\min} \leq u_k \leq u_{\max}$

Then the overall steps of the MPC control are:

- **Linearization**: The `LinearModel` function takes the current state x_k and control input u_k, and returns the matrices A, B, and vector c that define the linearized system dynamics:

$$\text{LinearModel}(x_k, u_k) \rightarrow (A, B, c)$$

  This step approximates the nonlinear system around the current operating point (x_k, u_k), simplifying the control problem.

- **State Transition**: Using the A, B, and c obtained from the linear model, the `StateTransition` function computes the next state x_{k+1} based on the current state x_k and control input u_k:

$$\text{StateTransition}(x_k, u_k) \rightarrow x_{k+1} = Ax_k + Bu_k + c$$

  This step uses the linearized model to predict the future state of the system.

- **Predict Motion**: The `PredictMotion` function predicts the sequence of future states $\{x_k\}_{k=1}^{T}$ starting from the initial state x_0 and using a sequence of control inputs $\{u_k\}_{k=0}^{T-1}$:

$$\text{PredictMotion}(x_0, \{u_k\}_{k=0}^{T-1}) \rightarrow \{x_k\}_{k=1}^{T}$$

  This step involves simulating the future behavior of the system over a finite time horizon based on the current state and a sequence of planned control inputs.

- **Solve MPC Optimization Problem**: The `SolveMPC` function solves the Model Predictive Control (MPC) optimization problem. It takes the initial state x_0, the reference trajectory x_{ref}, weight matrices Q and R, and an initial sequence of control inputs $\{u_k\}_{k=0}^{T-1}$, and returns the optimized sequence of control inputs:

$$\text{SolveMPC}(x_0, x_{\text{ref}}, Q, R, \{u_k\}_{k=0}^{T-1}) \rightarrow \{u_k^*\}_{k=0}^{T-1}$$

  This step involves formulating and solving an optimization problem that minimizes a cost function. This function usually incorporates components for tracking error and control effort, while adhering to system dynamics and constraints.

3-5 Conclusion

In this chapter, we introduced the key concepts of vehicle dynamics, terrain pose mapping, trajectory optimization, and MPC control, providing a comprehensive overview of motion planning on uneven terrain and the mathematical foundation for all relevant approaches. In the next chapter, we will focus on the two main contributions of this thesis: improvements in global path planning and the development of enhanced obstacle avoidance methods.

Chapter 4

Proposed Approach and Methodology

As discussed in Chapter 2, motion planning for navigation on uneven terrain involves designing a path for the ego vehicle to travel from a starting point to a target location while avoiding obstacles. This thesis adopts the common structure of Hybrid Motion Planning, which incorporates various obstacle-avoidance strategies. The motion planning system can be broken down into three key components: Global Path Planning, Collision Check, and Local Trajectory Optimization [29].

- Global Path Planning: This process involves generating an initial path for the vehicle to follow from its starting point to its destination. The goal is to create the shortest possible path, typically represented as a piece-wise linear path that is sampled across the vehicle's environment. This initial path accounts for static obstacles in the environment but does not consider moving obstacles at this stage, so it is only free of static obstacles.
- Collision Check: Considering dynamic obstacles, when such an obstacle starts moving, the initial path generated by global path planning may not be sufficient to ensure a safe and collision-free route. The collision check process involves monitoring for potential collisions with dynamic obstacles. If a collision risk is detected, the path is adjusted in real time to avoid the dynamic obstacle, thereby ensuring the safety of the vehicle's movement.
- Local Trajectory Optimization: After establishing a collision-free global path, the next step is to refine this path into a smooth and time-efficient trajectory that ensures safety. The global path, which is a piece-wise linear sequence of waypoints, serves as the initial state for local trajectory optimization. The primary goal is to convert this path into a continuous trajectory that is smoother, more dynamically feasible for the vehicle, minimizes travel time, and ensures safety by optimizing the trajectory parameters.

Together, these three components shown in Figure 4-1 form a cohesive motion planning framework that enables a vehicle to navigate on uneven terrain and avoid both static and dynamic obstacles.

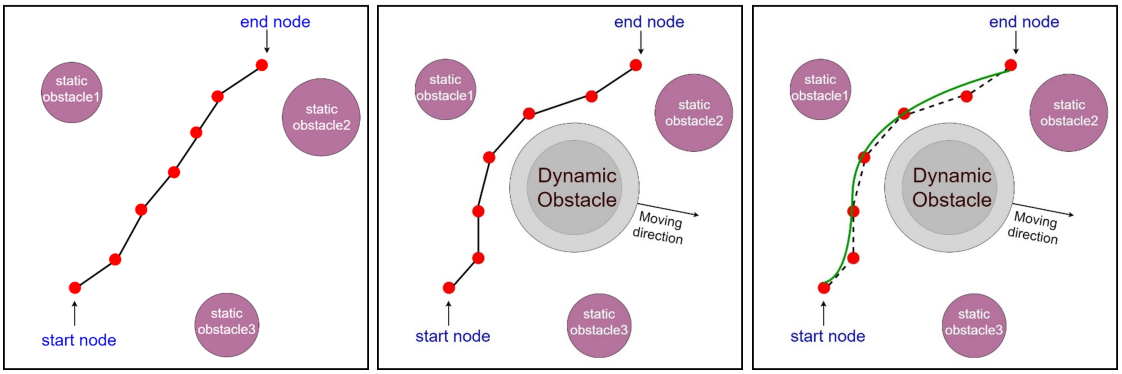

Figure 4-1: Three components in Motion Planning (a) Global Path Planning. (b) Collision Check. (c) Local Path Optimization.

4-1 Traversability Maps in Motion Planning

Traversability maps provide critical information on how easily a vehicle can pass through different areas. This information is essential for autonomous vehicles navigating uneven terrain, as such terrain varies significantly in its properties. By identifying smoother, more navigable areas and highlighting regions that are difficult to traverse, these maps enable vehicles to optimize their routes, ensuring safer and more efficient travel.

Common traversability maps of uneven terrain could be defined according to terrain slope, roughness, and curvature [22]. Most of them only consider the static properties of terrain geometric characteristics and materials. The common metrics of traversability, including definition, mathematical expression, and whether to use in our traversability definition, are shown in Table 4-1.

4-2 Dynamic Traversability-Based Global Path Planning

In Chapter 3-2, Terrain Pose Mapping is introduced as a powerful tool that leverages point cloud data to establish a direct correspondence between 2D spatial positions and 3D locations and orientations. This technique simplifies the complex task of 3D global path planning by transforming it into a 2D problem. Essentially, the goal of finding a 3D global path is converted to finding an equivalent 2D global path, which then maps back to the 3D space. This transformation not only reduces computational complexity but also enables more efficient and effective path planning in uneven terrain environments. Accordingly, Terrain Pose Mapping provides detailed 3D terrain information by mapping 2D spatial locations to their corresponding 3D orientations. This data enables the creation of traversability maps, which summarize the terrain's traversability in a 3D context and represent this information in a 2D grid map format. By encapsulating 3D properties of the terrain, such as curvature and static obstacles, within the 2D grid, these maps facilitate a more comprehensive planning process.

Traditional global path-planning algorithms often prioritize the safety of the planned paths, which means they try to schedule the path to avoid hard-to-traverse regions according to traversability maps. While ensuring the plan's safety, it may obtain a conservative path route

Table 4-1: Common criterion for terrain traversibility

Metrics	Description	Mathematical Expression
Slope	measures the steepness of the local terrain surface	local terrain slope around $\mathbf{x}$: φ_x/φ_m, where φ_x represents the inclination angle of local plane around $\mathbf{x}$, φ_m represents the maximum inclination angle that the vehicle can overcome [19].
Curvature	measures the deviation of the terrain surface from being flat	In the ellipsoidal region near $\mathbf{x}$, surface variation [31] can be computed as $\sigma(\mathbf{x}) = \lambda_0/(\sum_{i=0}^{2}\lambda_i)$, where λ_0, λ_1, and λ_2 are the eigenvalues of the covariance matrix for the ellipsoidal region. These eigenvalues satisfy $\lambda_0 \leq \lambda_1 \leq \lambda_2$.
Roughness	quantifies irregularities or unevenness of the terrain surface	Fit a smooth surface model around $\mathbf{x}$ using k neighborhood points. Then, calculate the sum of the absolute residuals $\frac{\sum_{k=1}^{n}\lvert d_k\rvert}{r_{max}}$, where r_{max} represents the maximum roughness value, d_k represents the distance from point k to the fitted plane [19].
Sparsity	represents the extent of vacant or unoccupied areas within a given terrain, indicating the potential presence of defects such as pits and depressions.	$\lambda = \frac{s-s_{min}}{s_{max}-s_{min}}$, where s_{min} and s_{max} represent the lowest and highest acceptable vacancy ratios respectively [13]. r indicates the fraction of the plane that has defects.

that has a longer path length and lower time efficiency, for example, designing a detour to follow the edge of a flat hill region instead of going directly across the flat hill region.

When a vehicle traverses on uneven terrain, the vehicle with significant momentum can more effectively traverse rough regions and steep slopes on uneven terrain due to its greater kinetic energy. This increased momentum allows the vehicle to overcome the resistance posed by rough surfaces and inclines, enabling it to maintain stability and traction even in challenging regions. As a result, such a vehicle is better equipped to handle abrupt changes in elevation and rough patches that are recognized as hard-to-traverse regions in the traversability map. If we consider the kinetic properties of the vehicle, like velocity and acceleration, in the context of terrain traversability, the traversability of uneven terrain could dynamically change based on the vehicle's velocity and acceleration.

The concept of dynamic traversability is then introduced: it is defined based on factors of slope and curvature. Table 4-1 presents the criteria for terrain classification based on slope

and curvature. The terrain slope is expressed as the ratio of the inclination of the local plane to the maximum inclination angle that the vehicle can navigate. An increase in velocity or acceleration may enhance the vehicle's ability to traverse specific regions, thereby increasing the maximum allowable inclination angle. Thus, by considering the vehicle's momentum, the terrain slope criteria are dynamic, depending not only on the static properties of the terrain but also on the current states of velocity and acceleration that the vehicle possesses. In contrast, terrain curvature is a static term derived directly from terrain pose mapping.

It is assumed that the terrain slope is constant of local planes on uneven terrain and t The terrain pose map [47] has been obtained which describes the corresponding correspondence from location and orientation on X-Y plane $\mathbf{x} = [x, y, \theta]^\top$ to height z, representing spatial coordinates long z-axis and pose of vehicle $\mathbf{R} = [\mathbf{x}_b, \mathbf{y}_b, \mathbf{z}_b] \in SO(3)$ in X-Y-Z global frame. Also, the map offers terrain information, including the inclination and curvature of the local plane corresponding to point $\mathbf{x} = [x, y, \theta]^\top$. Figure 4-3 illustrates a specific local plane, shaded in grey, showing the relative position of the body axes and spatial coordinates. The direction x_{pro}, which lies on the X-Y plane, represents the projected heading direction of the vehicle on this plane, corresponding to its actual heading in the X-Y plane. The angle between the body axis $\mathbf{x}_b$ and x_{pro} is interpreted as the slope of the local plane.

To better describe the dynamic traversability considering different velocities and acceleration of the vehicle, the force analysis of the vehicle on a certain slope is given according to analysis [2] and is shown as follows:

$$\vec{F} = \vec{F}_{air} + \frac{P}{v}\vec{T} + M\vec{g} + N\vec{n} - \mu N\vec{T} \tag{4-1}$$

where $\vec{F}_{air}$ is the aerodynamic force, P is the vehicle's power output, v is the speed along the body axis $\mathbf{x}_b$, $M\vec{g}$ is the gravitational force with g denoting the acceleration due to gravity, N is the normal force along the body axis $\mathbf{z}_b$, and μ is the dynamic friction coefficient. The schematic diagram of the forces on the slop is shown in Figure 4-2.

Within a short time interval on uneven terrain, the vehicle's traversal route can be approximated to be a straight line with a constant inclination angle, assuming we ignore aerodynamic forces and the inclination angle of the local plane is φ_x. By analyzing the vehicle's traversal in uphill and downhill scenarios separately, and under three cases of acceleration a_{max}, 0, and $-a_{max}$, we can determine the maximum inclination angle that the vehicle is capable of traversing.

According to Newton's second law, the net force along the direction of motion is equal to the mass M of the vehicle times its acceleration

$$\vec{F} = M\frac{\mathrm{d}v}{\mathrm{d}t} \tag{4-2}$$

When the vehicle is going uphill, the term $Mg\sin(\varphi_x)$ acts as a negative force component, opposing the vehicle's motion and thus reducing the net force. By combining (4-2), (4-1) can be represented as:

$$M\frac{\mathrm{d}v}{\mathrm{d}t} = \frac{P}{v} - Mg\sin(\varphi_x) - \mu Mg\cos(\varphi_x) \tag{4-3}$$

By considering three different acceleration scenarios and the vehicle's maximum available power P_{max}, we can calculate the maximum inclination angle, denoted as ϕ_f, that the vehicle can traverse based on this power. The calculation is as follows:

- When $a = a_{max}$, (4-3) can be represented as:

$$\begin{aligned} M\frac{\mathrm{d}v}{\mathrm{d}t} =& Ma_{max} = \frac{P}{v} - Mg\sin(\varphi_x) - \mu Mg\cos(\varphi_x) \\ \Rightarrow& Ma_{max} = \frac{P}{v} - Mg\sin(\varphi_x) - \mu Mg\cos(\varphi_x) \\ \Rightarrow& \frac{P}{v} - Ma_{max} = Mg(\sin(\varphi_x) + \mu\cos(\varphi_x)) \\ \Rightarrow& \frac{P}{Mgv} - \frac{a_{max}}{g} = \sqrt{1+\mu^2}\cdot\left(\frac{\sin(\varphi_x)}{\sqrt{1+\mu^2}} + \frac{\mu\cos(\varphi_x)}{\sqrt{1+\mu^2}}\right) \\ \Rightarrow& \frac{1}{\sqrt{1+\mu^2}}\left(\frac{P}{Mgv} - \frac{a_{max}}{g}\right) = \sin\left(\varphi_x + \beta\right), \\ & \tan\left(\beta\right) = \mu \\ \Rightarrow& \varphi_x = \arcsin\left(\frac{1}{\sqrt{1+\mu^2}}\left(\frac{P}{Mgv} - \frac{a_{max}}{g}\right)\right) - \arctan\left(\mu\right) \end{aligned}$$

  The maximum inclination angle determined by the maximum motion power of the vehicle in this scenario can be represented as

$$\phi_f = \varphi_x = \arcsin\left(\frac{1}{\sqrt{1+\mu^2}}\left(\frac{P_{max}}{Mgv} - \frac{a_{max}}{g}\right)\right) - \arctan\left(\mu\right)$$

  using maximum vehicle power P_{max}.

- When $a = 0$, (4-3) can be then represented as:

$$\begin{aligned} M\frac{\mathrm{d}v}{\mathrm{d}t} =& M\cdot 0 = \frac{P}{v} - Mg\sin(\varphi_x) - \mu Mg\cos(\varphi_x) \\ \Rightarrow& \frac{P}{v} - Mg\sin(\varphi_x) - \mu Mg\cos(\varphi_x) = 0 \\ \Rightarrow& \frac{P}{v} = Mg\left(\sin(\varphi_x) + \mu\cos(\varphi_x)\right) \\ \Rightarrow& \frac{P}{Mgv} = \sqrt{1+\mu^2}\sin\left(\varphi_x + \beta\right), \quad \tan\left(\beta\right) = \mu \\ \Rightarrow& \varphi_x = \arcsin\left(\frac{P}{\sqrt{1+\mu^2}Mgv}\right) - \arctan\left(\mu\right) \end{aligned}$$

  The maximum inclination angle determined by the maximum motion power of the vehicle in this scenario can be represented as

$$\phi_f = \varphi_x = \arcsin\left(\frac{P_{max}}{\sqrt{1+\mu^2}Mgv}\right) - \arctan\left(\mu\right)$$

  with maximum vehicle power P_{max}.

- When $a = -a_{max}$, (4-3) can be represented as:

$$\begin{aligned} M\frac{\mathrm{d}v}{\mathrm{d}t} =& - Ma_{max} = \frac{P}{v} - Mg\sin(\varphi_x) - \mu Mg\cos(\varphi_x) \\ \Rightarrow& \frac{P}{v} + Ma_{max} = Mg(\sin(\varphi_x) + \mu\cos(\varphi_x)) \\ \Rightarrow& \frac{1}{\sqrt{1+\mu^2}}\left(\frac{P}{Mgv} + \frac{a_{max}}{g}\right) = \sin\left(\varphi_x + \beta\right), \\ & \tan\left(\beta\right) = \mu \\ \Rightarrow& \varphi_x = \arcsin\left(\frac{1}{\sqrt{1+\mu^2}}\left(\frac{P}{Mgv} + \frac{a_{max}}{g}\right)\right) - \arctan\left(\mu\right) \end{aligned}$$

  The maximum inclination angle determined by the maximum motion power of the vehicle in this scenario can be represented as

$$\phi_f = \arcsin\left(\frac{1}{\sqrt{1+\mu^2}}\left(\frac{P_{max}}{Mgv} + \frac{a_{max}}{g}\right)\right) - \arctan\left(\mu\right)$$

  with maximum vehicle power P_{max}.

When the vehicle is going downhill, the term $Mg\sin(\varphi_x)$ acts as a positive force component, aligning with the vehicle's motion and thus increasing the net force. By combining (4-2), (4-1) can be represented as:

$$M\frac{\mathrm{d}v}{\mathrm{d}t} = \frac{P}{v} + Mg\sin(\varphi_x) - \mu Mg\cos(\varphi_x) \tag{4-4}$$

In a similar way, the maximum inclination angle determined by the maximum motion power of the vehicle under three different accelerations can be represented as:

- When $a = a_{max}$, the maximum inclination angle determined by the maximum motion power of the vehicle in this scenario can be represented as

$$\phi_f = \arccos\left(\frac{1}{\sqrt{1+\mu^2}}\left(\frac{P}{Mgv} - \frac{a_{max}}{g}\right)\right) - \arctan\left(\frac{1}{\mu}\right)$$

  using maximum vehicle power P_{max}.

- When $a = 0$, the maximum inclination angle determined by the maximum motion power of the vehicle in this scenario can be represented as

$$\phi_f = \arccos\left(\frac{P}{\sqrt{1+\mu^2}Mgv}\right) - \arctan\left(\gamma\right)$$

  with maximum vehicle power P_{max}.

- When $a = -a_{max}$, the maximum inclination angle determined by the maximum motion power of the vehicle in this scenario can be represented as

$$\phi_f = \arccos\left(\frac{1}{\sqrt{1+\mu^2}}\left(\frac{P_{max}}{Mgv} + \frac{a_{max}}{g}\right)\right) - \arctan\left(\gamma\right)$$

  with maximum vehicle power P_{max}.

For a detailed description of the calculation steps, please refer to Appendix A-1.

The maximum inclination angle that the vehicle can overcome is determined by the maximum motion power of the vehicle and traction loss threshold:

$$\phi_m = \min(\phi_f, \phi_s) \tag{4-5}$$

where ϕ_m represents the maximum angle of inclination that the mobile vehicle can successfully overcome; ϕ_f represents the maximum inclination angle determined by the maximum motion power of the vehicle; To avoid an-isotropic traction loss, the inclination angle should have the traction loss threshold [38],

$$\phi_s = \arctan(\mu_s - \mu) \tag{4-6}$$

where μ_s is static friction at the contact point; μ is dynamic friction coefficient. Considering that defining the maximum inclination angle when upper bounding by ϕ_s in (4-5) is too conservative, and at the same time, we want to ensure the vehicle can move within safe limits, especially when encountering sudden changes in slope, the safe extension factor α can be applied to (4-5):

$$\phi_m = \min(\phi_f, \alpha \cdot \phi_s) \tag{4-7}$$

where α is a user-defined constant value. By combining surface variation [31], which is used to approximate terrain curvature, the dynamic terrain variance can be further depicted as:

$$\tau(\mathbf{x}, v, a) = m_1\sigma(\mathbf{x}) + m_2 s(\mathbf{x}, v, a), \quad 0 \leq \tau(\mathbf{x}, v, a) \leq 1 \tag{4-8}$$

where $\tau(\mathbf{x}, v, a)$ is the terrain traversability of point $\mathbf{x} = [x, y, \theta]^\top$; $\sigma(\mathbf{x})$ represent the surface variation on the point $\mathbf{x}$, the smaller of surface variation can lead to flatter regions which are easier to traverse (low traversability); $s(\mathbf{x}, v, a)$ represents slope compatibility of point $\mathbf{x}$ with velocity v and acceleration a, it assessed terrain slope relative to the maximum inclination angle (beyond this the vehicle is unable to move or rollover); m_1 and m_2 are weighting vectors.

$$s(\mathbf{x}, v, a) = \begin{cases} \gamma \left(\frac{2}{1+e^{-\frac{2\varphi_x}{\phi_m}}} - 1 \right) & \text{if } \varphi_x \leq \phi_m \\ 1 & \text{if } \varphi_x > \phi_m \end{cases} \tag{4-9}$$

where $\gamma = (1 + e^{-2})/(1 - e^{-2})$ is a scaling factor used to adjust the dynamic traversability, ensuring it lies within the range $[0, 1]$ in the case $\varphi_x \leq \phi_m$. In function $s(\mathbf{x}, v, a)$, the point $\mathbf{x}$ defines φ_x and velocity v and acceleration a define ϕ_m. When φ_x is closer to ϕ_m, meaning the slope is closer to the upper bound slope, then larger slope compatibility is generated, which increases the terrain traversability (hard to traverse).

4-2-1 Combination of Dynamic Traversability and Global Path Planning

In this thesis, we adopt the Hybrid A* algorithm [3] as the baseline for Global Path Planning. Unlike traditional A* algorithms, Hybrid A* accounts for the kinematic constraints of the vehicle. It expands the search tree using steering functions based on vehicle kinematics, allowing it to plan feasible, smooth, and realistic paths that avoid sharp turns and satisfy the non-holonomic constraints of the vehicles. This approach addresses the limitations of the A* algorithm, particularly in scenarios where the generated path may not be feasible for tracking due to the car's kinematic model.

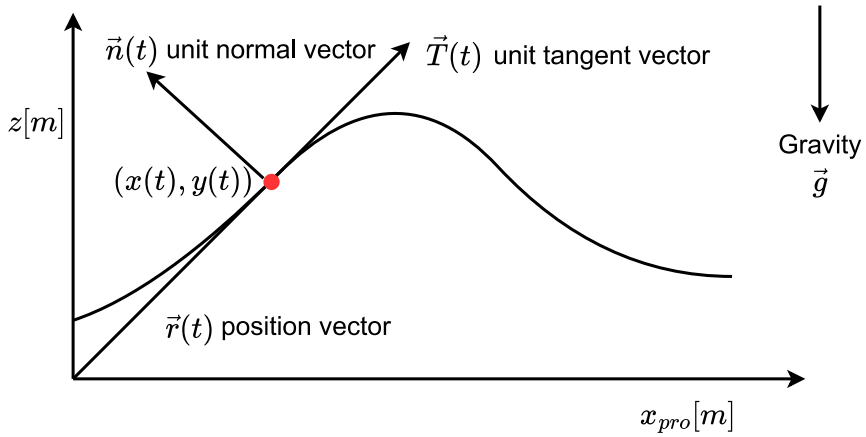

Figure 4-2: Force analysis on a certain slope.

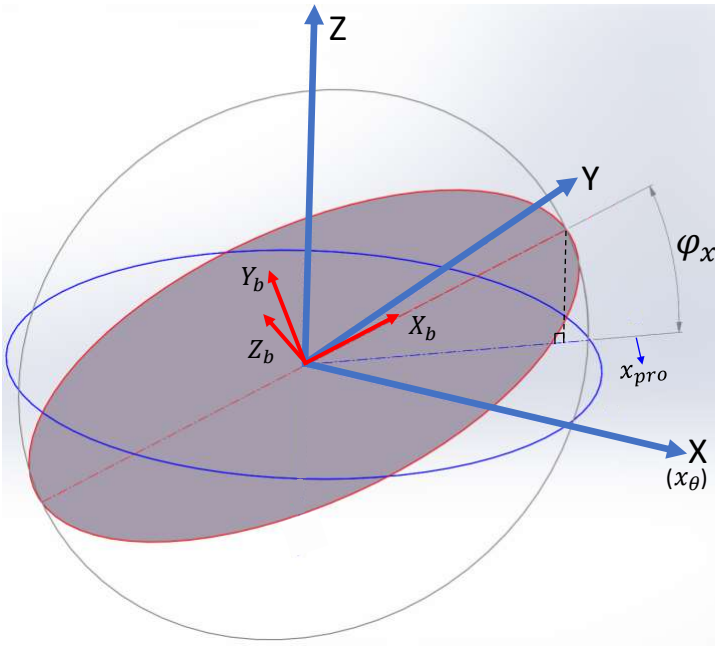

Figure 4-3: 3D representation of body frame $\mathbf{x}_b$-$\mathbf{y}_b$-$\mathbf{z}_b$, global frame X-Y-Z and inclination angle φ_x of local plane.

Search Space

In the Hybrid A* algorithm, the search is conducted within a 3-dimensional grid defined by the variables X, Y and θ. This grid represents the vehicle's position (x, y) in the 2D plane and its orientation θ. The trajectory generated by Hybrid A* is discretized into $N+1$ nodes and denoted as $\chi = \{\sigma_0, \sigma_1 \ldots, \sigma_N\}$.

Each node σ_i within this trajectory corresponds to a specific position, orientation, and time stamp of the vehicle and is expressed as:

$$\sigma_i = [x(t_i), y(t_i), \theta(t_i), t_i] = [\mathbf{x}(t_i), t_i]$$

where $\mathbf{x}(t_i) = [x(t_i), y(t_i), \theta(t_i)]$ represents the state vector of the vehicle at time t_i, including its position $(x(t_i), y(t_i))$ and orientation $\theta(t_i)$. The search space thus includes not only the spatial and angular dimensions but also the temporal dimension, allowing the algorithm to generate time-parameterized trajectories that satisfy the vehicle's kinematic constraints.

Motion Primitives

Most terrain cars are four-wheeled drive vehicles and have the advantage of maneuverability to rotate easily in place. Also, the timestamp of the generated trajectory is vital because the time stamp allows the path planner to continuously update the planned path based on the current positions and velocities of the dynamic obstacle. If we use a simplified bicycle model that takes velocity and steering inputs, it is important to note that this model does not account for the vehicle's ability to rotate in place. Additionally, when using this model, uniform time sampling cannot be used as a timestamp. This is because the fixed time intervals between consecutive nodes only allow for branching between them, without providing accurate time information. In other words, there is no guarantee that the vehicle's velocity remains constant between successive nodes or that the vehicle will traverse the same distance in any fixed time interval.

Based on these considerations, the bicycle kinematic model introduced in Section 3-1, which uses acceleration and steering as inputs, is employed to describe motion primitives. The control input $\mathbf{u} = (a, \phi)$ is represented as a combination of acceleration a which is longitudinal control input, and steering input ϕ as lateral control input. The vehicle state space, including location x and y, orientation θ and velocity v, can be represented as $\mathbf{x}_v = [x, y, \theta, v]$. The bicycle kinematic model (introduced in Section 3-1) is used to generate the steering function. The travel length d between successive vehicle states is constrained by minimum value d_{min} and maximum value d_{max} [12]. Thus the time duration Δt between the branching of a node σ_i and its subsequent node is regulated by d_{min} and maximum value d_{max}. The d can be computed as

$$d = v_i \Delta t_0 + a \Delta t_0^2 / 2 \tag{4-10}$$

where Δt_0 is a fixed time interval.

- If $d < d_{min}$, we assign d_{min} to d, the Δt can be represented as:

$$\left(\sqrt{(2 a d_{min} + v_i^2)} - v_i \right) / a \tag{4-11}$$

  (this result is obtained from solving the equation $d = d_{min} = v_i \Delta t + a \Delta t^2 / 2$.)

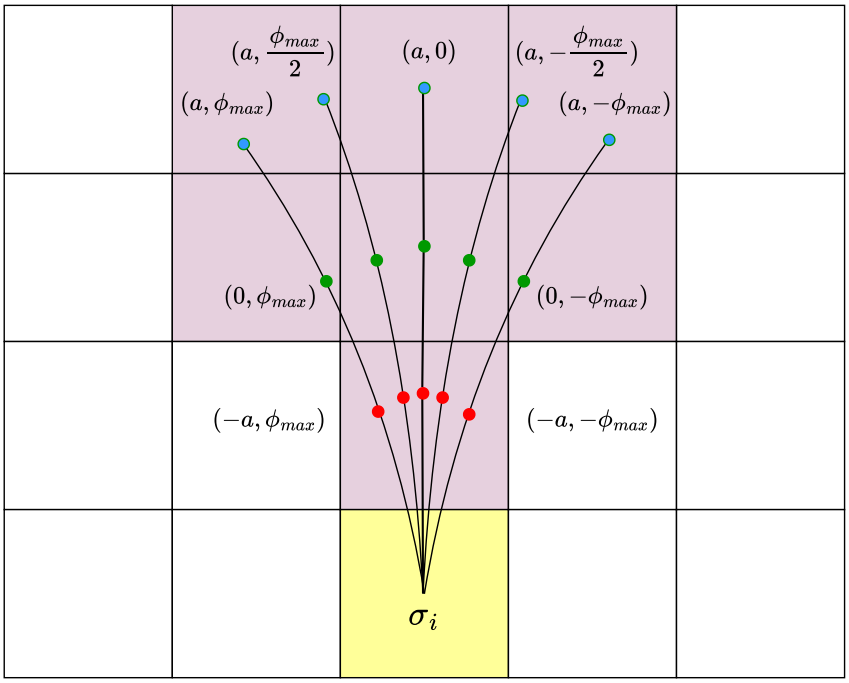

Figure 4-4: Motion primitives from node σ_i.

- If $d > d_{max}$, we assign d_{max} to d, similarly Δt can be represented as

$$\left(\sqrt{(2ad_{max} + v_i^2)} - v_i\right)/a \tag{4-12}$$

- If $d_{min} \leq d \leq d_{max}$, the time duration Δt between the branching of a node σ_i and its subsequent node is $\Delta t = \Delta t_0$.

Here it is assumed that the time interval for generating a path between adjacent nodes will be adjusted during the optimization process. Initiating expansion from the current node σ_i to meet the kinematic constraint, expansion is constructed considering both acceleration and steering inputs, with acceleration bounds $a \in [a_{min}, a_{max}]$ and steering bounds $\phi \in [-\phi_{max}, \phi_{max}]$. Expansion is possible from any acceleration input within the set $\{-a_{max}, 0, a_{max}\}$, where $-a_{max}$ and a_{max} denote the minimum and maximum acceleration respectively; $a = 0$ means the vehicle keeps the constant velocity. Additionally, expansion can occur from any steering input within the set

$$\{-\phi_{max}, -\phi_{max}/2, 0, \phi_{max}/2, \phi_{max}\} \tag{4-13}$$

where $-\phi_{max}$ and ϕ_{max} represent the minimum and maximum curvature for turning left and right respectively; while $\phi = 0$ signifies maintaining a straight line.

In Figure 4-4, the Motion primitives from node σ_i are shown. The shortest movement between any two nodes σ_i and σ_{i+1} can be analytically computed with Dubins path [30] which represent feasible trajectories for a vehicle with a fixed turning radius [5].

It is worth mentioning that a higher penalty is put on turning movements, the higher curvature will lead to higher movement cost. Also the penalty can be put on current velocity and the velocity change. The higher the current velocity and the greater the magnitude of velocity change compared with the previous velocity, the higher the energy costs. Additionally, the time at the goal node represents the time efficiency, the shorter the time to achieve the goal node, the greater the time efficiency.

Search Strategy

The algorithm employs three distinct states for nodes: "open," "closed," and "abandoned." During the exploration of the X-Y-θ grid field, the algorithm expands by opening adjacent grids around the current node to generate new potential paths [4]. Nodes that have been processed and whose neighbors have been explored are marked as "closed" and are not revisited. Nodes that are yet to be processed are considered "open" and are available for exploration in future steps. Nodes that encounter static obstacles or exceed the map boundaries are marked as "abandoned" and are excluded from further expansion. At each step, the algorithm combines previously visited nodes to generate new paths and selects the optimal, obstacle-free path based on cost functions.

Cost functions

The cost functions of hybrid A* consist of cost-so-far and cost-to-goal. The cost-so-far function represents the lowest cost between the start state σ_0 and the current state σ_i. When a vehicle navigates on uneven terrain, the terrain properties and equality of the resultant 3D trajectory are implemented into the cost-to-go functions. The cost-to-go functions consist of path length, steering cost, energy cost, and traversability cost.

It would be computationally consuming to directly calculate the length of a stretched 3D trajectory on a terrain surface from a generated 2D trajectory by geometry transformation. To approximate the 3D path length, we compute the squared Euclidean distance between points on uneven terrain, which are projected onto the X-Y-θ grid field. For node σ_i with coordinates x_i, y_i, and orientation θ_i, the corresponding height z_i can be determined using the terrain pose mapping as described in (3-6). Specifically, the height z_i is calculated as $z_i = f_1(\mathbf{s}_p)$, with $\mathbf{s}_p = [x_i, y_i, \theta_i]^\top$. The 3D path length is then calculated as:

$$l(\sigma_0, \sigma_i) = \sum_{j=0}^{i-1} (x_{j+1} - x_j)^2 + (y_{j+1} - y_j)^2 + (z_{j+1} - z_j)^2 \tag{4-14}$$

The steering cost indicates the smoothness of the path. A path with small steering changes is generally smoother, as it avoids sharp turns in direction. The steering cost can be computed as $\sum_{j=0}^{i-1} w_1 \cdot (\theta(t_{j+1}) - \theta(t_j))^2$, where w_1 is a constant weighting value. This cost imposes a higher penalty on turning movements, with curvature $-\varphi_{max}$ or φ_{max} compared to the straight movement with 0 curvature.

Then, the energy cost can be described based on acceleration and current velocity. Greater acceleration within a motion primitive, and larger current velocity increase energy consumption. The energy cost can be represented as $\sum_{j=0}^{i-1} w_2 \cdot a_j^2 + w_3 \cdot v_{j+1}^2$, where w_2 and w_3 are constant weighting values.

Finally, the traversability cost is accumulating terrain traversability along the motion primitives to evaluate path feasibility,

$$\tau\left(\sigma_0, \sigma_i\right) = \sum_{j=0}^{i} s(\mathbf{x}(t_j), v_j, a_j) \tag{4-15}$$

Overall, the cost-so-far function can be represented as:

$$\begin{aligned} f\left(\sigma_0, \sigma_{i-1}\right) =& f_0 + \sum_{j=0}^{i-1} w_1 \cdot (\theta(t_{j+1}) - \theta(t_j))^2 + w_2 \cdot a_j^2 + w_3 \cdot v_{j+1}^2 \qquad (4\text{-}16) \\ &+ w_4 \cdot ((x_{j+1} - x_j)^2 + (y_{j+1} - y_j)^2 + (z_{j+1} - z_j)^2) + w_5 \cdot s(\mathbf{x}(t_j), v_j, a_j) \qquad (4\text{-}17) \end{aligned}$$

where w_1, w_2, w_3, w_4 and w_5 are the constant weighting values and

$$f_0 = w_3 v_0^2 + w_5 s(\mathbf{x}(t_0), v_0, a_0)$$

The cost-to-goal function represents the current cost and the heuristic difference between the current node and the end node. According to thesis [33], the cost-to-goal function contains 2D path length heuristic and time heuristic. The time heuristic measures the predicted time from the current node to the end node to reduce the time cost of searching.

The 2D path length heuristic aims to optimize the search tree by accurately measuring the shortest Dubins curve length from the current node to the destination, prioritizing efficient traversal without consideration of static or dynamic obstacles. If we assume the first three elements of the current node's state are $(0, 0, 0)$, and those of the end state are (x, y, θ), the Dubins curve length can be represented as s_g. This heuristic ensures precise representation of Dubins curve which represent the shortest distance between current and end nodes [30], and thus helps expand the search tree with the heading towards the goal node.

The time heuristic evaluates the time cost from the current node to the end node by following the obtained Dubins curve based on the velocities of the two nodes and the maximum velocity constraint. Assume that the vehicle at the current node has the velocity v_0, and the vehicle is expected to reach the end node with velocity v_g. Simplifying the calculation of time along the Dubins curve, even though the curve exists in two dimensions, can be achieved by treating it as a one-dimensional problem, that is, using a trapezoidal speed profile to calculate time spent traversing from $(v_0, 0)$ to (v_g, s_g). There are 6 modes of trapezoidal speed planning if we consider the velocities of the current and end nodes based on the same acceleration or deceleration in Figure 4-5. Each of these six modes can be utilized depending on the values of v_0, v_g, and s_g.

The second heuristic, denoted as t_g, represents the minimum time required for the vehicle to travel from the current node to the end node, ensuring that it approaches the goal node at the desired velocity v_g.

$$h(\sigma(t_i)) = h_l s_g + h_t t_g \tag{4-18}$$

where h_l and h_t are constant heuristic parameters.

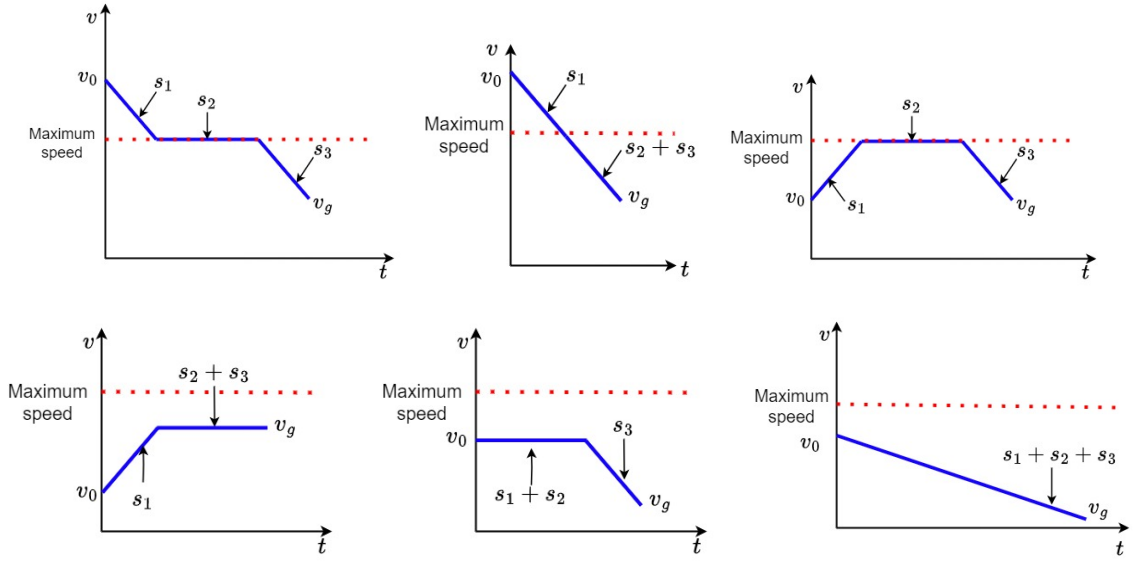

Figure 4-5: 6 modes of trapezoidal speed planning (a) $v_0 > v_{max}$, $\frac{v_0^2 - v_g^2}{2a} < s_g$. (b) $v_0 > v_{max}$, $\frac{v_0^2 - v_g^2}{2a} \geq s_g$. (c) $\frac{2v_{max}^2 - v_0^2 - v_g^2}{2a} < s_g$. (d) $v_0 < v_g$, $\frac{2v_{max}^2 - v_0^2 - v_g^2}{2a} > s_g$, $\frac{v_g^2 - v_0^2}{2a} < s_g$. (e) $v_0 > v_g$, $\frac{2v_{max}^2 - v_0^2 - v_g^2}{2a} > s_g$, $\frac{v_0^2 - v_g^2}{2a} < s_g$. (f) $v_0 > v_g$, $\frac{v_0^2 - v_g^2}{2a} \geq s_g$

Termination Condition

Most terrain cars are four-wheeled drive vehicles and have the advantage of maneuverability to rotate easily in place. The termination condition is to find a collision-free trajectory from the start node to the goal node. Here only the current position serves as a reference point for measuring the distance to the goal state.

$$\text{len}(p_i, p_{goal}) < l_{min} \tag{4-19}$$

where $p_i = [x_i, y_i]$ and $p_{goal} = [x_{goal}, y_{goal}]$; l_{min} is a constant value to define the safety distance. If the distance between the current node and the goal node is smaller than shotrange, the current node is viewed as close to goal and the termination of the algorithm occurs.

Overall, integrating dynamic traversability into the Hybrid A* algorithm as its cost function to evaluate feasibility of path brings the Dynamic Traversability-based Hybrid A* algorithm (DT Hybrid A*).

4-3 Safety and Direction Elements based Velocity Obstacle(Velocity Obstacle (VO)) Algorithm in Collision Check

In this thesis, a collision detection mechanism is integrated into the global path planning algorithm to ensure that the system can replan if a potential future collision is detected. The process begins with the global path planner generating an initial path that avoids all static obstacles. As dynamic obstacles move, if a potential collision is identified, the vehicle discards

the current global path and its associated optimal path, initiating a replanning process from its current location. To ensure the new path is free of dynamic obstacles, the collision detection mechanism incorporates Safety [9] and Direction Elements based on a Velocity Obstacle (VO) cost term into the global path algorithm. The Safety element, based on VO, represents the minimal distance between the current velocity of the ego vehicle and the boundary of the VO region, indicating the safety of the chosen velocity. The Direction element, introduced in this thesis, assesses whether the chosen velocity will lead to a collision in a longer time frame than the one considered in the VO model. More details about replan mechanism can refer to Section 4-5.

Most approaches use Euclidean distance as a penalty cost for dynamic obstacles relative to the global path to identify whether it would lead to a collision. While this method is simple and straightforward, it can overlook the velocity and direction of dynamic obstacles, potentially leading to paths that are not truly collision-free. This is because Euclidean distance only accounts for the current locations of obstacles and without accounting for their future movement or trajectory.

In contrast, the VO method introduces a penalty term based on the distance to the VO region. This region represents the set of velocities that could result in a collision, taking into account both the speed and direction of the obstacles [9]. By penalizing velocities that are close to or within the VO region, the ego vehicle is effectively guided to select safer trajectories that proactively avoid potential collisions. A simple graphical example is shown in Figure 4-6 to illustrate this concept. In this figure, two velocities, v_i' and v_i'', are considered at the ith node. Both velocities result in the same Euclidean distance to the obstacle. Therefore, during node selection, branches corresponding to both velocities are retained. In the next step of node branching, the endpoints of these branches become new nodes for further expansion. By not discarding nodes that may later result in a collision, there's a risk of generating an infeasible global path that, when the obtained nodes on the global path are connected in a straight line, could collide with a dynamic obstacle at a future time step. This issue is demonstrated in the simulation results where Euclidean distance is used as the baseline for collision check, as shown in Section 5-3-2. In contrast, the VO based method would discard the branch corresponding to v_i'' at this step, preventing further expansion in a direction that could lead to a future collision.

In the collision check module, it is assumed that the location, orientation, and velocity of the dynamic obstacle can be continuously monitored and updated in real-time, and are represented as

$$\mathbf{x}^{ob}(t) = \left[x^{ob}(t), y^{ob}(t), \theta^{ob}(t)\right] \tag{4-20}$$

and $v^{ob}(t)$ at time t.

According to the VO algorithm [9], the velocity vector is built to make sure the vehicle remains in the region VO where there is no collision between the vehicle and dynamic obstacle.

$$VO = \left\{v | \exists t : (p + vt) \cap (p^{ob} + v^{ob}t) \neq 0\right\} \tag{4-21}$$

where $p = [x, y]$ and $p^{ob} = \left[x^{ob}, y^{ob}\right]$ represent the location of the vehicle and dynamic obstacle, respectively. It is worth mentioning that the velocity v and v^{ob} are vectors and contain orientation θ and θ^{ob} as direction.

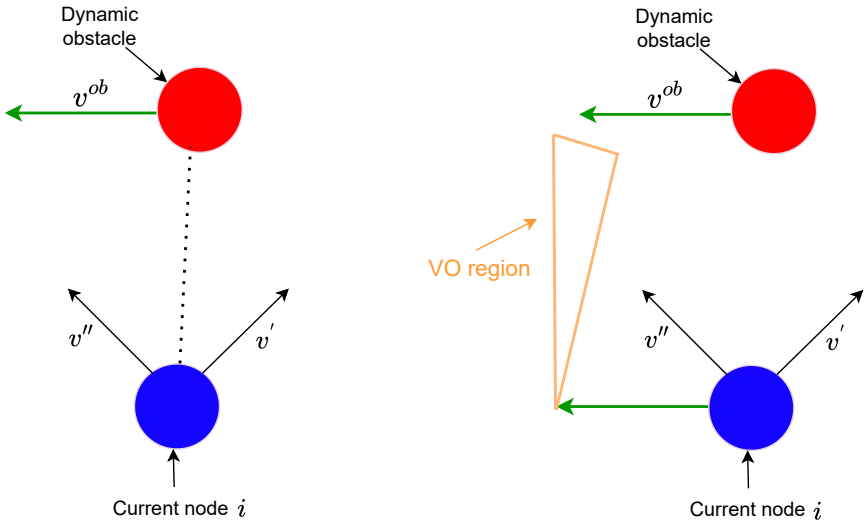

Figure 4-6: An example demonstrating the effectiveness of the VO algorithm in comparison to the Euclidean distance method. (a) Euclidean based penalty term, v_i' or v_i'' have the same Euclidean distance at the current node i. During node selection, branches corresponding to both velocities are retained for future node expansion. (b) VO based penalty term, v_i' has a larger distance to VO region than v_i''. The VO based method discards the branch corresponding to v_i''.

According to node expansion in Hybrid A* discussed in Section 4-2-1, the set of acceleration a and steering ϕ inputs from current node σ_i is determined and could be represented as pairs of (a, ϕ).

Assume the vehicle is currently traversing from node σ_{i-1} to σ_i with velocity v and orientation θ. The VO region can be constructed according to the definition (4-21). Through the steering function, with different inputs (steering and acceleration), we can get different poses and velocities at a set from which to choose node σ_i. To ensure the velocity at node σ_i not lying in VO region, the safety element (SA) is adopted from [9]. The robot is deemed completely safe if it cannot reach the nearest obstacle within the specified time frame. In this case, increasing the distance further would not enhance its safety status. The safety element (SA) can then be represented as:

$$\mathrm{SA}(v_i) = \min(1, \frac{\min_{v_{VO} \in VO} \|v_i - v_{VO}\|}{v_{max} \cdot T_{max}}) \tag{4-22}$$

Here, T_{max} denotes a predefined parameter that specifies the maximum time interval, while v_{VO} represents the nearest point to velocity v_i at node σ_i within the VO cone, as illustrated in Figure 4-7.

If we only use the SA element as an evaluation of the cost function to avoid dynamic obstacles,

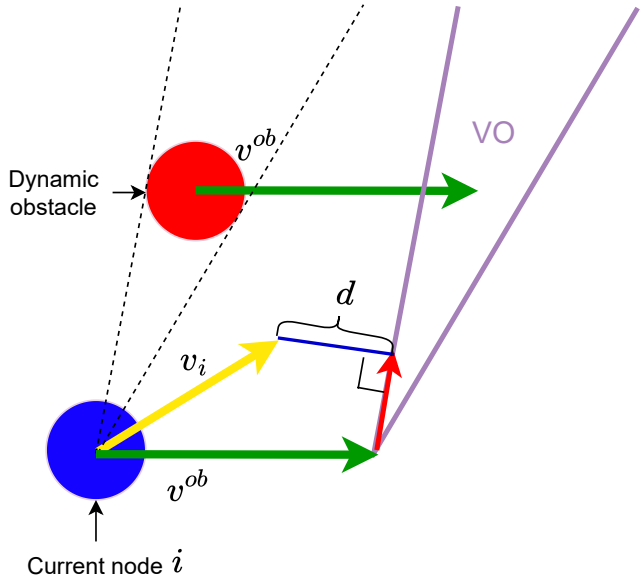

Figure 4-7: Graphic representation of SA. For the current node i, the SA value of the planned velocity v_i is represented by d, which denotes the minimum distance between v_i and the VO region of a dynamic obstacle moving with velocity v^{ob}.

it could cause a possible collision in a longer time span compared with T_{max}. Here is one example in Figure 4-8, from current node σ_i, the steering function could help to generate two velocities v_i' and v_i''. The distance between the VO cone and the velocity v_i' and v_i'' is the same $d' = d''$. If we only rely on SA as our cost function to avoid the dynamic obstacle, it would cost us equally to choose whether v_i' or v_i''. However, the case could become different when having different velocity options. Figure 4-9 shows choosing velocity v_i' can cause a collision while choosing v_i'' can help avoid the dynamic obstacle.

We assume the dynamic obstacle moves in a straight line at a constant velocity. In Figure 4-10, four different cases show that chosen v_i can easily cause a future collision with respect to the current location of the velocity of the dynamic obstacle. d_{eg} denotes the relative position vector from the ego vehicle's current location toward the goal position, while d_{eo} represents the relative position vector from the ego vehicle's current location toward the obstacle. A logic table is made in Figure 4-11 to summarize these four scenarios. The four logic conditions are then listed as:

- $a = d_{eo} \times d_{eg} > 0$, if d_{eo} is to the the left of d_{eg}, this condition is true. ($\times$ is cross-product operation)
- $b = d_{eg} \times v^{ob} > 0$, if d_{eg} is to the the left of v^{ob}, this condition is true.
- $c = \theta_1 < \theta_2$, if the dynamic obstacle is closer to the goal than the ego vehicle, c is true.

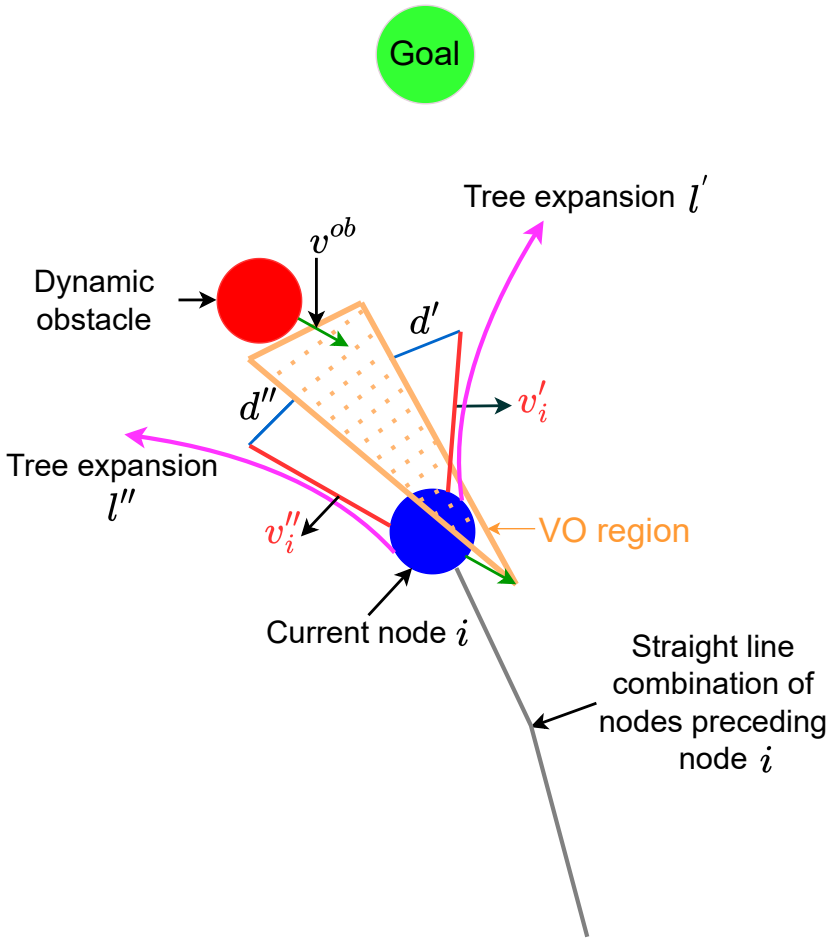

Figure 4-8: One example to show two velocities having the same SA values. For the current node i, two tree expansions l' and l'' can be generated using motion primitives. The velocities at current node i, v_i', and v_i'' are obtained by selecting different motion primitives. Both velocities have the same minimal distance to VO region, denoted as d' and d'' respectively. The straight line combination of nodes preceding node i defines how the nodes are connected leading up to node i and the direction of node i.

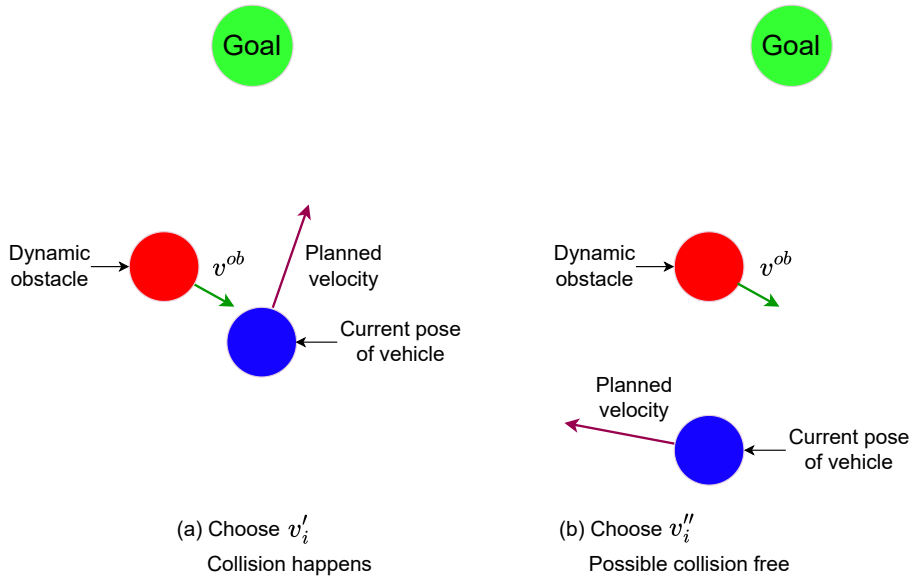

Figure 4-9: Choosing different velocities v_i' and v_i'' leads to different collision outcomes at the same future time.

- $d = d_{eg} \times v_i > 0$, if d_{eg} is to the the left of v_i, this condition is true.

Overall the collision direction element can be defined as:

$$\mathrm{CD(v_i)} = (\mathrm{a} == \mathrm{b}) \wedge (\mathrm{c} == (\mathrm{a} == \mathrm{d})) \tag{4-23}$$

If CD is true, the possible collision would happen. The cost functions for the safety element (SA) and collision direction element (CD), derived from a VO, are integrated into the global path algorithm (DT Hybrid A*) to avoid dynamic obstacles. This replanning algorithm is referred to as DSVO. The cost functions of this algorithm use the safety element (SA) and collision direction element (CD) to select the node σ_i:

$$J = \begin{cases} e^{-\mathrm{SA(v_i)}}, & \text{if } \mathrm{CD(v_i)} = \text{false} \\ e^{\mathrm{SA(v_i)}}, & \text{if } \mathrm{CD(v_i)} = \text{true} \end{cases} \tag{4-24}$$

In (4-24), if a velocity is detected to potentially cause a collision in the future, the collision direction element is set to true. Regardless of the corresponding SA value, this velocity is assigned a relatively high cost, making it less likely to be selected during the node selection process in the DT Hybrid A* algorithm. In this process, nodes with higher costs are unlikely to be chosen for expansion from the available node options. Conversely, if a velocity is deemed beneficial for avoiding future collisions, it is assigned a lower cost. When the SA value is higher, the cost is reduced further, indicating that the velocity is safer and more likely to be selected. This approach not only avoids dynamic obstacles over a short time span but also in a longer time interval by selecting an optimal direction based on the predicted motion of

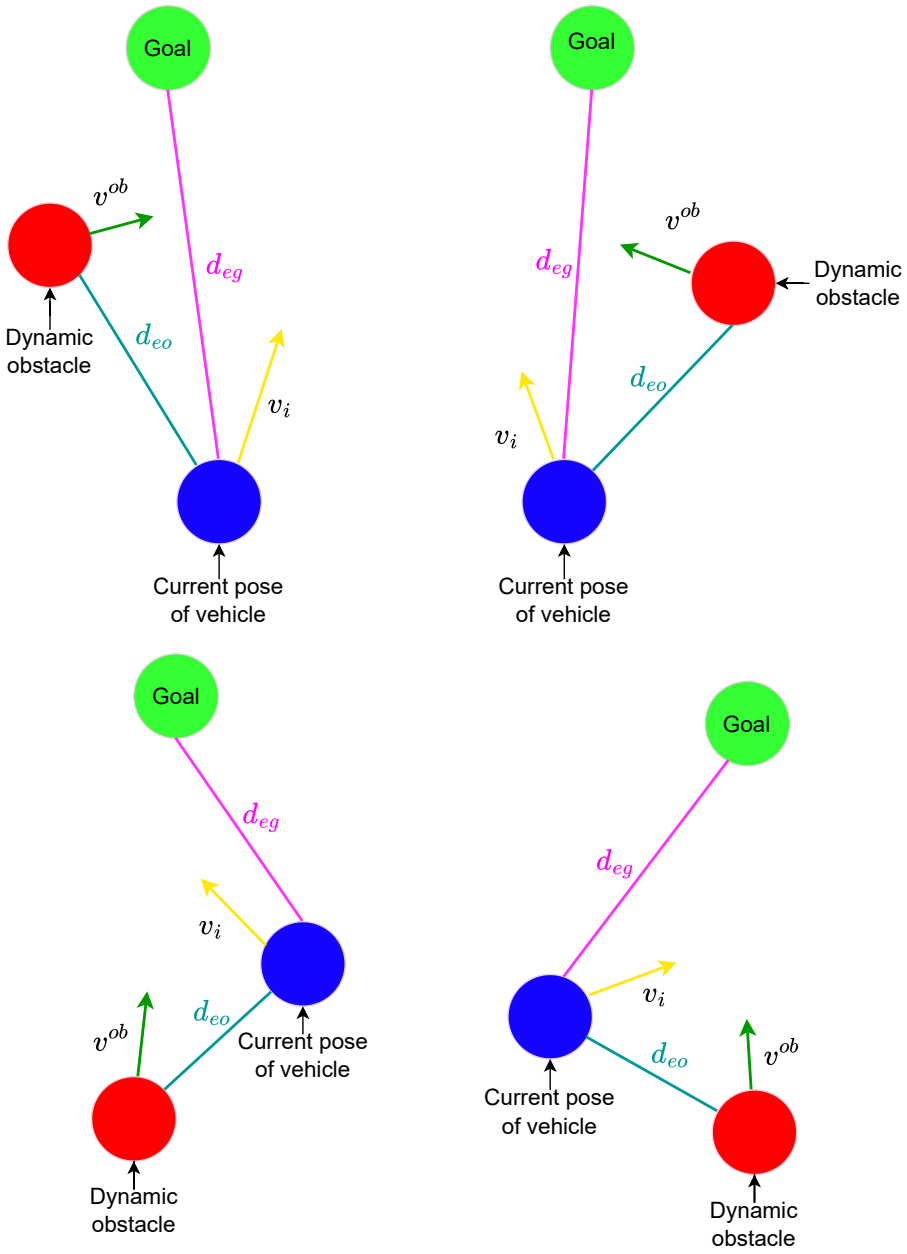

Figure 4-10: 4 different cases to show chosen v_i can easily cause future collision.

Case	$d_{eo} \times d_{eg} > 0$	$d_{eg} \times v^{ob} > 0$	$\theta_1 < \theta_2$	$d_{eg} \times v_i > 0$
Easy collide 1	True	True	True	True
Easy collide 2	False	False	True	False
Easy collide 3	True	True	False	False
Easy collide 4	False	False	False	True

Figure 4-11: A logic table to summarize four different easy-collision cases.

the obstacle. Tree expansion is guided more effectively by assigning higher costs to velocities that are more likely to lead to future collisions. As a result, nodes associated with these risky velocities are less likely to be selected during the node expansion process, effectively filtering out less desirable paths and prioritizing safer, collision-free options.

4-4 Artificial Potential Field based Optimization

In Chapter 3-3, the structure of the trajectory optimization problem is discussed, including the representation of the trajectory, the formulation of the optimization problem, and the development of both inequality and equality constraints.

The cost function of the initial optimization block in Chapter 3-3 includes terrain curvature, trajectory jerk, and total time, which helps generate a static optimal trajectory. While this approach ensures that the optimal path is similar to any given initial path with the same start and end points, it primarily focuses on avoiding static obstacles. However, it is crucial to incorporate a dynamic obstacle-related cost term to achieve a truly dynamic obstacle-free trajectory. This addition is essential to account for moving obstacles, ensuring that when the optimization block receives the dynamic obstacle-free global path, the optimized trajectory not only avoids collisions with static elements but also converges to the collision-free global path and adapts to avoid the dynamic obstacle, thereby enhancing the safety and reliability of the path-planning process.

Then the Artificial Potential Field (APF) [20] concept is introduced and explaining how it generates repulsive forces from dynamic obstacles and attractive forces from the global path to guide the vehicle toward an optimal trajectory.

The cost functions of trajectory optimization of each piece of trajectory in this project can be designed as

$$\tau = \mathbf{j}^\top(t)\mathbf{j}(t) + \rho_{ter} \cdot \sigma(\mathbf{x}(t)) + U_{rep}(\mathbf{p}(t), \mathbf{p}_{ob}) + U_{att}(\mathbf{p}(t), \mathbf{l}_g) \tag{4-25}$$

Here $\mathbf{j}(t) = \mathbf{x}^{(3)}(t)$ represents the third derivative of the trajectory state $\mathbf{x}(t)$ which is the jerk of the trajectory, as derived from the terrain pose mapping in the perception model; U_{rep} is the repulsive force field that penalizes the distance between the current trajectory state $\mathbf{p}(t) = [x(t), y(t)]^\top$ and the dynamic obstacle position $\mathbf{p}_{ob} = [x^{ob}(t), y^{ob}(t)]^\top$, helping to avoid collisions. The dynamic obstacle's position is predicted using a model that assumes straight-line motion at a constant velocity; U_{att} is the attractive force field to minimize the distance

between optimal trajectory and global path; $\mathbf{l}_g$ is the global path from DT Hybrid A*. It is worth mentioning that when integrating the global path into the optimization process as an initial guess, the number of trajectory states that will be optimized is the same as the waypoints of the global path. Therefore, we can use the index to find the corresponding waypoint on the global path of the current trajectory state.

Then the optimization problem can be constructed as follows:

$$\begin{aligned}
\min_{\mathbf{c}_{xy},\mathbf{c}_\theta,\mathbf{T}_{xy},\mathbf{T}_\theta} & \int_0^{T_s} (\mathbf{j}^\top(t)\mathbf{j}(t) + \rho_{ter}\sigma(\mathbf{x}(t)) + U_{att}(\mathbf{p}(t),\mathbf{l}_g))dt + \sum_{i=1}^{N} U_{rep}(\mathbf{p}(t_i),\mathbf{p}_{ob}) \\
& + \rho_T T_s \\
s.t. \quad & \dot{x}\sin\theta - \dot{y}\cos\theta = 0 \\
& \mathbf{G}_{xy}\mathbf{c}_{xy} = \mathbf{b}_{xy}, \quad \mathbf{G}_\theta\mathbf{c}_\theta = \mathbf{b}_\theta \\
& \mathbf{T}_{xy} \succeq \mathbf{0}, \quad \mathbf{T}_\theta \succeq \mathbf{0} \\
& v_x^2 - v_{max}^2 \leq 0 \\
& a_x^2 - a_{mlon}^2 \leq 0 \\
& a_y^2 - a_{mlat}^2 \leq 0 \\
& \frac{\omega_z^2}{v_x^2 + \delta_+} - \frac{\tan^2\delta_{max}}{L_w^2} \leq 0 \\
& c_{min} - \cos\xi \leq 0 \\
& \sigma(\mathbf{x}) - \sigma_{max} \leq 0
\end{aligned} \tag{4-26}$$

here N is the number of segments in the trajectory, and T_s represents the total duration of the trajectory; The term $\sum_{i=1}^{N} U_{rep}(\mathbf{p}(t_i),\mathbf{p}_{ob})$ calculates the sum of repulsive forces exerted by obstacles at the endpoints of each trajectory segment, relative to the obstacle positions at those time points. The integral $\int_0^{T_s} U_{att}(\mathbf{p}(t),\mathbf{l}_g)dt$ represents the time integral of the attractive force field, penalizing the distance from the current trajectory position; The repulsive forces are summed rather than integrated over time because their expression, which involves the reciprocal of a polynomial of the highest order raised to the power of ten, makes time integration computationally challenging; $\mathbf{c}_{xy}$, $\mathbf{c}_\theta$ are the coefficient matrix of trajectory segments; $\mathbf{T}_{xy} \in \mathbb{R}^M$ and $\mathbf{T}_\theta \in \mathbb{R}^\Omega$ are the time vectors; Dynamic inequalities conditions include limits on v_x, a_x, a_y, κ, $\cos\xi$ and $\sigma(\mathbf{x})$ with constants v_{max}, a_{mlon}, a_{mlat}, δ_{max}, c_{min}, σ_{max}; t_i with $i \in \{1, 2, \cdots, N\}$, corresponds to the time intervals associated with each trajectory segment. It is important to note that when the dynamic obstacle remains stationary, the cost function still operates, but the objective function becomes static. However, if the dynamic obstacle is in motion, the repulsive force exerted by the obstacle on the optimal path changes based on the obstacle's location.

As mentioned before, the trajectory is in the form of piecewise polynomials, taking dimension x of i-th segment of the trajectory as an example. Repulsive force field U_{rep} can be defined as the reciprocal of the squared Euclidean distance,

$$U_{rep}(\mathbf{p}(t),\mathbf{p}_{ob}) = \begin{cases} \frac{1}{(\mathbf{p}(t)-\mathbf{p}_{ob})^\top(\mathbf{p}(t)-\mathbf{p}_{ob})} & \|\mathbf{p}(t)-\mathbf{p}_{ob}\|_2 < d_0 \\ 0 & \|\mathbf{p}(t)-\mathbf{p}_{ob}\|_2 \geq d_0 \end{cases} \tag{4-27}$$

Since the trajectory has the same number of segments in both the x and y dimensions, the

first two dimensions of the trajectory can be combined as:

$$\begin{bmatrix} x_i(t) \\ y_i(t) \end{bmatrix} = \mathbf{p}_{xy}(t) = \mathbf{c}_{ixy}\gamma(t) = \mathbf{c}_0 + \mathbf{c}_1 t + \mathbf{c}_2 t^2 + \mathbf{c}_3 t^3 + \mathbf{c}_4 t^4 + \mathbf{c}_5 t^5 \tag{4-28}$$

where $\mathbf{c}_* \in \mathbb{R}^{2\times 1}$ is coefficient vector taking from coefficient matrix $\mathbf{c}_{xy}$, with $* \in \{0, 1, 2, \ldots, 5\}$ .The squared Euclidean distance can be firstly written as:

$$\begin{aligned}
&\text{Euclidean } \langle \mathbf{p}(\mathrm{t}), \mathbf{p}_{\mathrm{ob}} \rangle^2 = (\mathbf{p}(\mathrm{t}) - \mathbf{p}_{\mathrm{ob}})^\top (\mathbf{p}(\mathrm{t}) - \mathbf{p}_{\mathrm{ob}}) \\
&= (\mathbf{c}_0 t + \mathbf{c}_1 t + \mathbf{c}_2 t^2 + \mathbf{c}_3 t^3 + \mathbf{c}_4 t^4 + \mathbf{c}_5 t^5 - \mathbf{p}_{ob})^\top (\mathbf{c}_0 t + \mathbf{c}_1 t + \mathbf{c}_2 t^2 + \mathbf{c}_3 t^3 \\
&+ \mathbf{c}_4 t^4 + \mathbf{c}_5 t^5 - \mathbf{p}_{ob}) = \underbrace{\mathbf{c}_0^\top \mathbf{c}_0 - \mathbf{c}_0^\top \mathbf{p}_{ob} - \mathbf{p}_{ob}^\top \mathbf{c}_0 + \mathbf{p}_{ob}^\top \mathbf{p}_{ob}}_{m_0} \\
&+ (\underbrace{\mathbf{c}_1^\top \mathbf{c}_0 + \mathbf{c}_0^\top \mathbf{c}_1 - \mathbf{p}_{ob}^\top \mathbf{c}_1 - \mathbf{c}_1^\top \mathbf{p}_{ob}}_{m_1}) t + (\underbrace{\mathbf{c}_2^\top \mathbf{c}_0 + \mathbf{c}_1^\top \mathbf{c}_1 + \mathbf{c}_0^\top \mathbf{c}_2 - \mathbf{p}_{ob}^\top \mathbf{c}_2 - \mathbf{c}_2^\top \mathbf{p}_{ob}}_{m_2}) t^2 \\
&+ (\underbrace{\mathbf{c}_3^\top \mathbf{c}_0 + \mathbf{c}_2^\top \mathbf{c}_1 + \mathbf{c}_1^\top \mathbf{c}_2 + \mathbf{c}_0^\top \mathbf{c}_3 - \mathbf{p}_{ob}^\top \mathbf{c}_3 - \mathbf{c}_3^\top \mathbf{p}_{ob}}_{m_2}) t^3 \\
&+ (\underbrace{\mathbf{c}_4^\top \mathbf{c}_0 + \mathbf{c}_3^\top \mathbf{c}_1 + \mathbf{c}_2^\top \mathbf{c}_2 + \mathbf{c}_1^\top \mathbf{c}_3 + \mathbf{c}_0^\top \mathbf{c}_4 - \mathbf{p}_{ob}^\top \mathbf{c}_4 - \mathbf{c}_4^\top \mathbf{p}_{ob}}_{m_4}) t^4 \\
&+ (\underbrace{\mathbf{c}_5^\top \mathbf{c}_0 + \mathbf{c}_4^\top \mathbf{c}_1 + \mathbf{c}_3^\top \mathbf{c}_2 + \mathbf{c}_2^\top \mathbf{c}_3 + \mathbf{c}_1^\top \mathbf{c}_4 + \mathbf{c}_0^\top \mathbf{c}_5 - \mathbf{p}_{ob}^\top \mathbf{c}_5 - \mathbf{c}_5^\top \mathbf{p}_{ob}}_{m_5}) t^5 \\
&+ (\underbrace{\mathbf{c}_5^\top \mathbf{c}_1 + \mathbf{c}_4^\top \mathbf{c}_2 + \mathbf{c}_3^\top \mathbf{c}_3 + \mathbf{c}_2^\top \mathbf{c}_4 + \mathbf{c}_1^\top \mathbf{c}_5}_{m_6}) t^6 + (\underbrace{\mathbf{c}_5^\top \mathbf{c}_2 + \mathbf{c}_4^\top \mathbf{c}_3 + \mathbf{c}_3^\top \mathbf{c}_4 + \mathbf{c}_2^\top \mathbf{c}_5}_{m_7}) t^7 \\
&+ (\underbrace{\mathbf{c}_5^\top \mathbf{c}_3 + \mathbf{c}_4^\top \mathbf{c}_4 + \mathbf{c}_3^\top \mathbf{c}_5}_{m_8}) t^8 + (\underbrace{\mathbf{c}_5^\top \mathbf{c}_4 + \mathbf{c}_4^\top \mathbf{c}_5}_{m_9}) t^9 + \underbrace{\mathbf{c}_5^\top \mathbf{c}_5}_{m_{10}} t^{10} \\
&= m_0 + m_1 t + m_2 t^2 + \cdots + m_9 t^9 + m_{10} t^{10} = e(t)
\end{aligned}$$

where $m_1, m_2, \ldots, m_{10}$ are constant values. Then, the gradients of $1/(m_0 + m_1 t + m_2 t^2 + \cdots + m_9 t^9 + m_{10} t^{10})$ with respect to coefficient vectors $\mathbf{c}_0$, $\mathbf{c}_1$, $\mathbf{c}_2$, $\mathbf{c}_3$, $\mathbf{c}_4$ and $\mathbf{c}_5$ and time t can be calculated. Taking the gradient to coefficient vector $\mathbf{c}_1$ and time t as examples, the gradient of $1/(m_0 + m_1 t + m_2 t^2 + \cdots + m_9 t^9 + m_{10} t^{10})$ with respect to coefficient vector $\mathbf{c}_1$ is represented as:

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}\mathbf{c}_1}\left(\frac{1}{m_0 + m_1 t + m_2 t^2 + \cdots + m_9 t^9 + m_{10} t^{10}}\right) = \frac{\mathrm{d}}{\mathrm{d}\mathbf{c}_1}\left(\frac{1}{e(t)}\right) \\
&= -\frac{1}{e^2(t)}\frac{\mathrm{d}e(t)}{\mathrm{d}\mathbf{c}_1} = -\frac{1}{e^2(t)}\left(\frac{\mathrm{d}m_0}{\mathrm{d}\mathbf{c}_1} + \frac{\mathrm{d}m_1}{\mathrm{d}\mathbf{c}_1} t + \frac{\mathrm{d}m_2}{\mathrm{d}\mathbf{c}_1} t^2 + \cdots + \frac{\mathrm{d}m_9}{\mathrm{d}\mathbf{c}_1} t^9 + \frac{\mathrm{d}m_{10}}{\mathrm{d}\mathbf{c}_1} t^{10}\right) \\
&= -\frac{1}{e^2(t)}(2\mathbf{c}_0 t - 2\mathbf{p}_{ob} t + 2\mathbf{c}_1 t^2 + 2\mathbf{c}_2 t^3 + 2\mathbf{c}_3 t^4 + 2\mathbf{c}_4 t^5 + 2\mathbf{c}_5 t^6)
\end{aligned}$$

the gradients $1/(m_0 + m_1 t + m_2 t^2 + \cdots + m_9 t^9 + m_{10} t^{10})$ to $\mathbf{c}_0$, $\mathbf{c}_2$, $\mathbf{c}_3$, $\mathbf{c}_4$ and $\mathbf{c}_5$ can be calculated in a similar way. The gradient of $1/(m_0 + m_1 t + m_2 t^2 + \cdots + m_9 t^9 + m_{10} t^{10})$ with respect to time t is represented as

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{1}{m_0 + m_1 t + m_2 t^2 + \cdots + m_9 t^9 + m_{10} t^{10}}\right) = -\frac{1}{e^2(t)}\left(m_1 + 2m_2 t + \cdots + 9m_9 t^8 + 10m_{10} t^9\right)$$

U_{att} is the attractive force field to minimize the distance between optimal trajectory and global path,

$$U_{att}((\mathbf{p}(t), \mathbf{l}_g)) = \text{Euclidean}\,\langle \mathbf{p}(\mathrm{t}), \mathbf{l}_{\mathrm{g}} \rangle^2 = (\mathbf{p}(\mathrm{t}) - \mathbf{l}_{\mathrm{g}})^\top (\mathbf{p}(\mathrm{t}) - \mathbf{l}_{\mathrm{g}}) \tag{4-29}$$

where $\mathbf{l}_g$ is the global path point from DT Hybrid A* having the same index as $\mathbf{p}(t)$. Similarly, we can obtain $U_{att}((\mathbf{p}(t), \mathbf{l}_g))$ with the same form as $m_0 + m_1 t + m_2 t^2 + \cdots + m_9 t^9 + m_{10} t^{10}$. And the time integral can be obtained as the form $m_0 t + m_1 t^2/2 + m_2 t^3/3 + \cdots + m_9 t^{10}/10 + m_{10} t^{11}/11$. The gradients of $m_0 t + m_1 t^2/2 + m_2 t^3/3 + \cdots + m_9 t^{10}/10 + m_{10} t^{11}/11$ with respect to coefficient vectors $\mathbf{c}_1$ can be calculated as follows:

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}\mathbf{c}_1}\left(m_0 t + m_1 t^2/2 + m_2 t^3/3 + \cdots + m_9 t^{10}/10 + m_{10} t^{11}/11\right) \\
&= \frac{\mathrm{d}m_0}{\mathrm{d}\mathbf{c}_1} t + \frac{\mathrm{d}m_1}{\mathrm{d}\mathbf{c}_1} t^2/2 + \frac{\mathrm{d}m_2}{\mathrm{d}\mathbf{c}_1} t^3/3 + \cdots + \frac{\mathrm{d}m_9}{\mathrm{d}\mathbf{c}_1} t^{10}/10 + \frac{\mathrm{d}m_{10}}{\mathrm{d}\mathbf{c}_1} t^{11}/11 \\
&= \mathbf{c}_0 t^2 - \mathbf{p}_{ob} t^2 + 2\mathbf{c}_1 t^3/3 + \mathbf{c}_2 t^4/2 + 2\mathbf{c}_3 t^5/5 + \mathbf{c}_4 t^6/3 + 2\mathbf{c}_5 t^7/7
\end{aligned}$$

The gradients of $m_0 t + m_1 t^2/2 + m_2 t^3/3 + \cdots + m_9 t^{10}/10 + m_{10} t^{11}/11$ with respect to $\mathbf{c}_0$, $\mathbf{c}_2$, $\mathbf{c}_3$, $\mathbf{c}_4$ and $\mathbf{c}_5$ can be calculated similarly. The gradient of $m_0 t + m_1 t^2/2 + m_2 t^3/3 + \cdots + m_9 t^{10}/10 + m_{10} t^{11}/11$ with respect to time t can be represented as:

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}t}\left(m_0 t + m_1 t^2/2 + m_2 t^3/3 + \cdots + m_9 t^{10}/10 + m_{10} t^{11}/11\right) \\
&= m_0 + m_1 t + m_2 t^2 + \cdots + m_9 t^9 + m_{10} t^{10}
\end{aligned}$$

4-5 Replan Mechanism

Initially, the global path planner generates a path upon receiving the goal pose on the Rviz platform, ensuring all static obstacles are avoided. This initial path is then passed to the optimization block, where an optimal trajectory is produced. Once optimization is complete, MPC commands are generated to guide the ego vehicle along the planned trajectory.

Simultaneously, a collision detection mechanism continuously monitors potential collisions between the ego vehicle and dynamic obstacles. The collision condition is primarily based on the Euclidean distance between the ego vehicle and dynamic obstacles—if the distance falls below a defined safety threshold, a collision is predicted. Additionally, the condition checks if the vehicle's velocity lies within the VO region of a dynamic obstacle and assesses whether the dynamic obstacle is dangerously close to any waypoints on the current global path.

If a potential collision is detected, the current global path and its corresponding optimal trajectory are discarded, triggering a replanning process from the ego vehicle's current position. The replanning algorithm, DSVO, integrates the cost function for safety and directional elements based on the VO to effectively avoid dynamic obstacles, as illustrated in Figure 4-12. The replanned path is then sent to the optimization block to generate a new optimal trajectory, which provides the reference for the MPC block to obtain updated commands. This process continues until the ego vehicle reaches the goal pose.

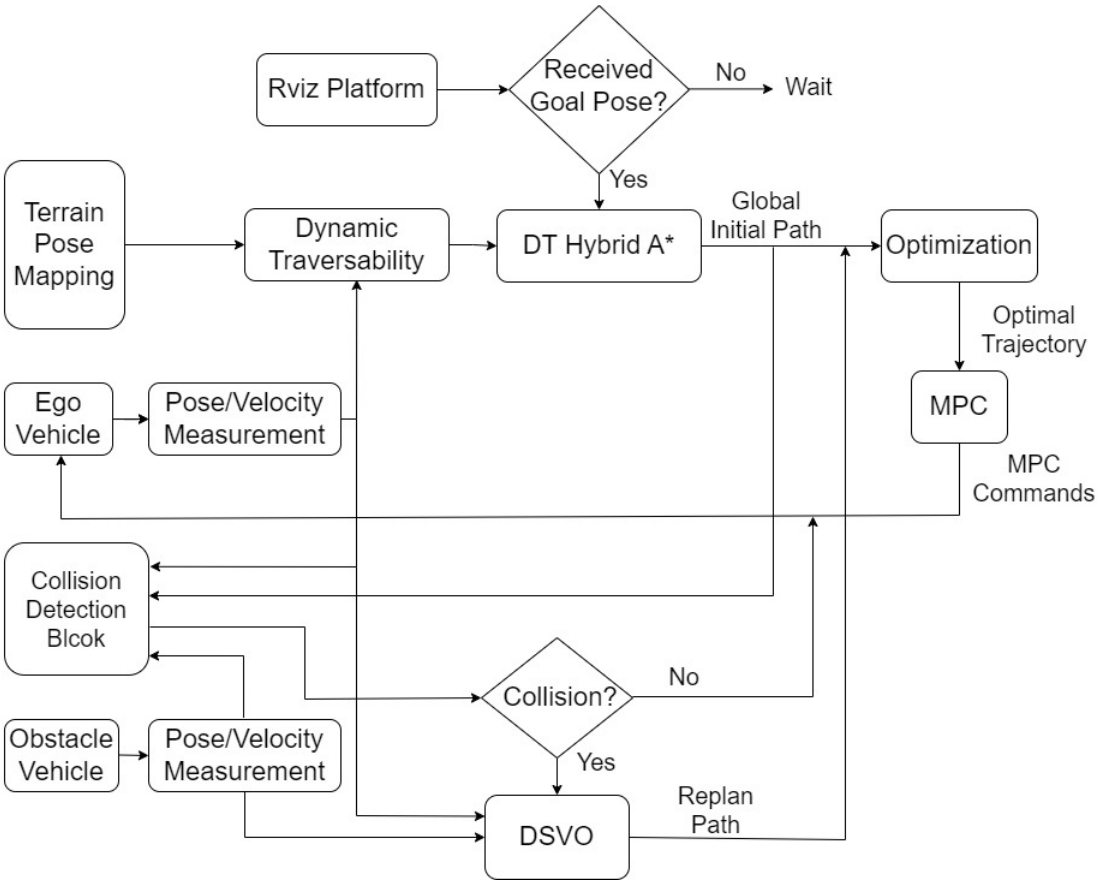

Figure 4-12: Replan mechanism.

4-6 Conclusion

In this Chapter, two novel algorithms are proposed the Dynamic Traversability-based Hybrid A* (DT Hybrid A*) algorithm and the Safety Direction Velocity Obstacle (DSVO) replanning method. The DT Hybrid A* incorporates vehicle momentum into terrain traversability in order to more accurately reflect the vehicle's capability to navigate uneven terrain. Meanwhile, the DSVO method combines Safety and Direction Elements with a Velocity Obstacle (VO) cost term. This approach not only helps avoid dynamic obstacles in the short term but also considers their predicted motion to optimize path planning over a longer time horizon. In the next chapter, we will describe the simulation environment setup and present the simulation results to verify the effectiveness of the two proposed approaches.

Chapter 5

Simulation Environment Setup and Results

In preceding chapters, the trajectory planner has been devised utilizing a terrain pose mapping-based perception model through the LIDAR-based SLAM algorithm. An iterative planning fitting strategy has been employed as a pipeline for determining the elevation height, denoted as z, and the robot's pose in contact with irregular terrain. The entire experimental setup is established on ROS (Robot Operating System) Noetic, utilizing C++ and Python programming languages. The integration of Point Cloud Library (PCL) packages enhances the efficiency of Point Cloud data processing and offers a wide range of algorithms and tools for point cloud manipulation from raw data. Moreover, PCL could facilitate the conversion of processed data into Msg (message) ROS format combined with `sensor_msgs` library, ensuring compatibility with ROS's message-passing framework.

5-1 Simulator

In this thesis, conducting real-world tests on a car-like robot is impractical due to time constraints and limited physical equipment. Therefore, a high-fidelity simulation framework has been developed to evaluate the effectiveness of the proposed algorithms.

5-1-1 Gazebo

In this thesis, given the presence of dynamic obstacles, two car-like robots employing bicycle kinematic models are considered: one serving as the ego vehicle, while the other is regarded as a dynamic obstacle. Both robots utilize the same `robot_description.xacro` file, which describes their models and captures the kinematic and geometric characteristics in XML format, adhering to the URDF (Unified Robot Description Format). This file is processed by Gazebo during the launch process, generating and visualizing the robot models within the simulation environment. Furthermore, to differentiate the two robots' state publisher

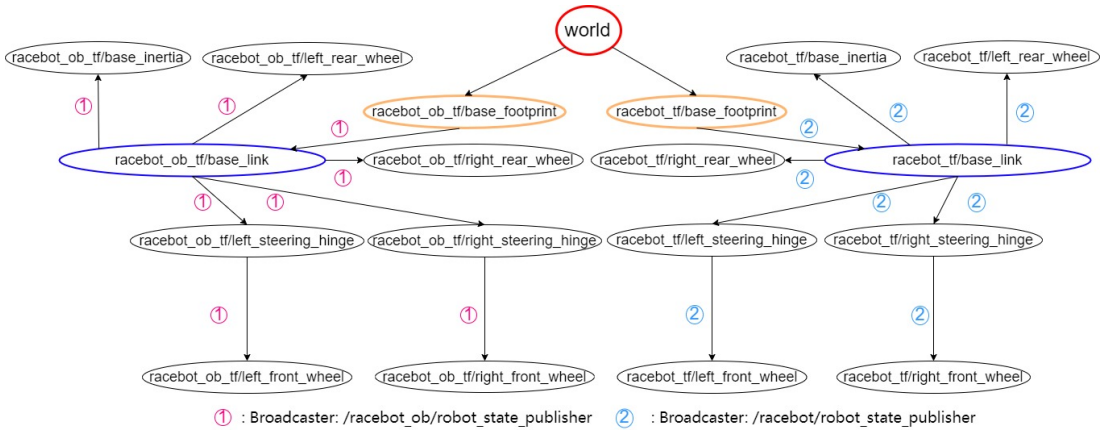

Figure 5-1: The tf tree of the simulation environment. The ego vehicle and obstacle vehicle have different `tf_prefix` parameter values, `racebot_tf` and `racebot_ob_tf` respectively. The `base_footprint` serves as the base link of the robot, acting as the reference frame for other links and connecting to the simulation frame header 'world'. `base_link` represents the main body of the robot and includes a visual representation using an STL model. `base_inertia` defines the inertia properties of the robot's base, specifying mass and inertia values. `left_steering_hinge` and `right_steering_hinge` represent the steering mechanism for the left and right wheels, allowing them to rotate around the z-axis. The `left_front_wheel`, `right_front_wheel`, `left_rear_wheel`, and `right_rear_wheel` represent the robot's front and rear wheels, featuring visual and collision geometry, as well as inertial properties. `left_steering_hinge_joint` and `right_steering_hinge_joint` are revolute joints connecting the steering hinges to the base link, with defined limits for the range of motion.

and controller and show the motion of the two robots synchronously, the `<group ns>` tag is used to organize nodes into separate namespaces, in this case, namespaces `\racebot` and `\racebot_ob` respectively. At the same time, `tf_prefix` is configured differently for the two robots to uniquely prepend a prefix to all frames published by the tf (transform) library which represents the spatial relationship between different coordinate frames in the simulation environment.

Then under different namespaces and `tf_prefix` parameters, `robot_state_publisher` publishes the state of a robot's joint configuration to the corresponding ROS tf tree, shown in Figure (5-1). Finally, the robot controller configures PID gains for the right rear velocity controller, left front velocity controller, right front velocity controller, left rear velocity controller, right front steering position controller, and left front steering position controller. These gains are employed to regulate the velocity of individual wheels and the position of steering joints. This configuration is facilitated by the Gazebo ROS Control package, which seamlessly integrates ROS control functionalities with the Gazebo robot simulation environment.

5-1-2 RViz

In the RViz (ROS Visualization) platform, the trajectories can be published as `nav_msgs::Path` messages, allowing for visualization of trajectories within a 3D space. These messages contain a timestamp, frame ID, and elements of `PoseStamped` which could represent 3D pose (position and orientation) along with its header information. Then to directly visualize the

point cloud data in a 3D space in RViz, `sensor_msgs::PointCloud2` message type is used to store and transmit data between ROS nodes or over ROS topics.

5-1-3 Perception Model

In this thesis, the terrain pose mapping model is adopted as the perception model upon which the implemented algorithms are based. Within this model, five distinct world maps representing diverse terrains—hills, forests, deserts, mountains, and volcanoes—have been constructed and stored as `pcd` type files. These maps are then processed using the integrated PCD reader function in PCL packages to convert into point cloud data. By using spatial queries for 2D search by sampling 2D coordinates, the corresponding 3D point cloud data point could be obtained by performing a nearest neighbor search using KD-trees. Moreover, employing terrain pose mapping algorithms, the perception model is constructed to enable direct conversion from any provided spatial coordinates and heading direction to a comprehensive 3D pose and elevation height. This conversion encompasses terrain gradients, spatial angles, terrain geometry variables, and grid map indexing.

5-2 Motion Planner

Once the simulation environment is established, the task involves determining an appropriate method for navigating the car-like robot within the simulated environment. In this thesis, manual control using external keyboards is used to control the two robots' movements. Considering the operator's driving proficiency can influence the quality of generating trajectory of dynamic obstacles and the operator's control skills may not ensure adherence to expected trajectories, particularly over extended paths, the switch control mode can be chosen as autonomous control to make the obstacle robot follow the user-intended trajectory. Also for the ego vehicle, the autonomous movement is expected to follow the optimized trajectory derived from optimization processes, the autonomous control is equally applicable for the ego vehicle.

5-2-1 Keyboard Control

In manual keyboard control, the current control node can be switched between the ego vehicle and obstacle vehicle by using keyboard inputs '1' and '2' respectively. This ensures that the control command is stored under the namespace of the currently controlled vehicle. Once the robot to be controlled is specified, the keys 'w', 'a', 's', and 'd' correspond to providing the vehicle with maximum velocity, negative velocity, positive steering, and negative steering angles, respectively. The control command is stored as a Twist message, encapsulating both linear and angular velocity along the x, y, and z axes. Lastly, pressing 'q' quits the keyboard control mode. The Tkinter library is used to create a GUI (Graphical User Interface) window with a frame containing a label, shown in Figure (5-2).

5-2-2 Autonomous Control

As previously discussed, both the ego vehicle and obstacle vehicle require autonomous control to meet distinct needs: the ego vehicle necessitates autonomous navigation while ensuring the

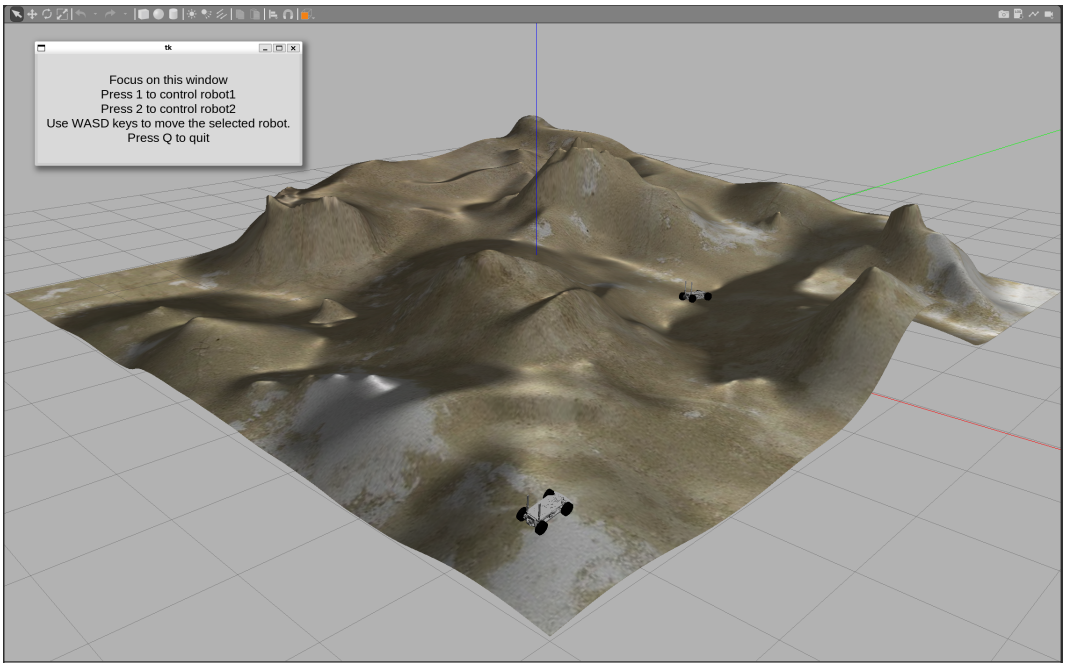

Figure 5-2: GUI window of Keyboard control in Gazebo environment.

obstacle vehicle is easily controllable to generate trajectories of high quality aligned with user intent.

User-intended Trajectory

After the user sets the control model as autonomous and designs the desired trajectory of an obstacle, the commands to control the obstacle switch to listening to different topics. A set of commands is designed to dictate the duration of control actions for the obstacle vehicle, ensuring it adheres to the user-intended trajectory. This is achieved by utilizing the `rospy.Timer()` function in conjunction with `rospy.Duration()` to establish specific time intervals. The first function initializes a ROS timer to continuously publish control commands with a certain interval, accomplished by invoking `rospy.Timer()` with the duration of `rospy.Duration(0.2)`. If the vehicle is expected to stop after a certain time interval, setting `oneshot=True` can trigger the robot stop function (both linear and angular velocity being set to zero) once after the designated duration.

2D Navigation Goal in RViz

The functionality of "2D Navigation Goal" in RViz can help set a goal position and orientation for navigation in a 2D environment. Once a goal is set using the tool of "2D Navigation Goal", RViz sends the corresponding navigation goal message to the `/move_base_simple/goal` topic. In `maneger node`, the callback function of receiving the message to the topic of navigation goal contains the functions of generating global path planning, location path optimization, and Model Predictive Control (MPC) controller with position feedback enabling the ego

robot to accurately track a given trajectory. Global path planning is triggered once the goal's position and orientation are received. By transmitting the global path to location path optimization, location path optimization is subsequently activated. After Obtaining the optimized trajectory, the MPC controller is then triggered accordingly. Finally, The MPC function publishes the Twist command for the ego vehicle. This command is then remapped to the same node as the keyboard control node to control the movement of ego vehicle.

5-3 Simulation Results

In this section, we present simulation tests to evaluate the effectiveness of the DT Hybrid A* Algorithm and Dynamic Obstacle Avoidance. Additionally, an analysis of the results from these tests is provided. The parameter values used in the simulations are listed in Table 5-1.

Table 5-1: Values of parameters used in the simulation experiments.

Parameter	Value	Unit	Parameter	Value	Unit
v_{max}	0.5	m/s	w_1	10.0	-
a_{max}	5.0	m/s^2	w_2	0.2	-
P_{max}	5.0	W	w_3	1.0	-
μ	1.0	-	w_4	25.0	-
μ_s	1.2	-	w_5	20.0	-
α	1.2	-	h_l	1.0	-
m_1	0.7	-	h_t	1.0	-
m_2	0.3	-	l_{min}	1.0	m
L_w	2.6 $\times 10^{-1}$	m	T_{max}	1.0	s
d_{min}	0.0	m	ρ_{ter}	1.0	-
d_{max}	37.5$\times 10^{-2}$	m	ρ_T	100000.0	-
Δt_0	0.3	s	d_0	0.8	m
ϕ_{max}	0.5	rad	-	-	-

5-3-1 Effectiveness of the DT Hybrid A* Algorithm

To verify the effectiveness of the proposed DT Hybrid A* algorithm, the comparative test is made in the absence of obstacles. One lightweight Hybrid A* algorithm adopted in Thesis [47] is chosen as the baseline of the Hybrid A* algorithm, which use simplified bicycle model as its steering function.

The baseline Hybrid A* cost functions account for steering effort, path length, velocity changes, variations in steering angle and heuristic cost. Given that the search tree is expanding from the start pose σ_s towards the goal pose σ_g, the current node is denoted as σ_i with inputs $[v_i, \delta_i]$, the cost functions can be represented as:

$$f(\sigma_s, \sigma_i, \sigma_g) = g(\sigma_s, \sigma_i) + \lambda h(\sigma_i, \sigma_g) \tag{5-1}$$

where $g(\sigma_s, \sigma_i)$ is the cost-so-far function and $h(\sigma_i, \sigma_g)$ is the cost-to-goal function; the weighted sum of g and h functions comprise the overall cost $f(\sigma_s, \sigma_i, \sigma_g)$. The DT Hybrid A*

algorithm differs from the standard Hybrid A* algorithm in several key aspects. Firstly, the DT Hybrid A* algorithm utilizes a kinematic bicycle model for steering, while the baseline Hybrid A* employs a simplified bicycle model. Additionally, the DT Hybrid A* algorithm incorporates the 2D Dubins curve length and time-based heuristic costs, whereas the baseline Hybrid A* relies on a heuristic that stretches the 2D straight-line distance as its cost estimate.

To more accurately assess the effectiveness of dynamic traversability, an intermediate algorithm was introduced, which replaces the static traversability in the baseline Hybrid A* with dynamic traversability while keeping the rest of the baseline structure and parameters unchanged. This allows for a clear comparison to determine the impact of dynamic traversability on performance and whether the structural modifications in the DT Hybrid A* further enhance the planning behavior.

The simulation test is conducted by selecting the same start and end nodes in RViz and applying consistent physical limitations, maximum longitudinal velocity 0.5m/s, maximum longitude and latitude acceleration 5m/s^2, maximum steering angle 0.5. The control inputs $[v, \delta]$ of steering functions of Baseline Hybrid A* and Baseline Hybrid A* with dynamic traversability take values under the bounds of maximum longitudinal velocity and maximum steering angle, respectively, $v \in \{-0.5, 0.5\}\,\text{m/s}$ and $\delta \in \{-0.5, 0.5\}\,\text{rad}$. Then the DT Hybrid A* algorithm adopts acceleration $a \in \{-5, 5\}\,\text{m/s}^2$ and steering angle $\delta \in \{-0.5, 0.5\}\,\text{rad}$ as the control inputs of steering function.

In Figure (5-3), a simulation result in a hilly environment is displayed on the corresponding 2D traversability map in RViz. The terrain traversability ranges from $[0, 1]$; the smaller the traversability value at a certain location on the traversability map, the easier it is for the vehicle to traverse that point. A color bar represents the terrain traversability range. Regions with traversability between 0 and 0.1 are shaded light red, between 0.1 and 0.3 are vivid orange, and between 0.3 and 0.5 are lime green. Areas shown in white on the map indicate non-traversable regions with terrain traversability 1, typically corresponding to the steepest parts of the hills. The green, blue, and red lines indicate the global paths generated by the Baseline Hybrid A* Algorithm, the Baseline Hybrid A* Algorithm with dynamic traversability, and the DT Hybrid A* Algorithm, respectively. To display global paths three-dimensionally, the corresponding gazebo simulation result is shown with a solid hill environment, including terrain material information in Figure (5-4). This allows for a clear view of how the global paths interact with the terrain surface.

Then, four metrics, Mean Static Terrain Traversability, Path Length, Path Planning Completion Time, and CPU Time Cost, are used to evaluate the effectiveness of the three global path algorithms. The mean static terrain traversability takes the mean of the static traversability of waypoints on the global paths. It reflects the global paths' overall static smoothness; the smaller the mean static terrain traversability, the smoother the path. Path Length calculates the sum of 3D Euclidean distance between adjacent waypoints on the global path; planning completion Time measures the total duration required to complete the planned path by following the steering functions, and CPU time cost evaluates the computational expense of the algorithm. It is worth mentioning, that the Baseline Hybrid A* only uses the static terrain curvature as its terrain traversability, which is the static component of dynamic traversability. Meanwhile, the Baseline Hybrid A* and DT Hybrid A* use dynamic terrain traversability, which may result in a slight decrease in terrain traversability compared to using static terrain traversability. At some locations of the terrain traversability map, the dynamic terrain

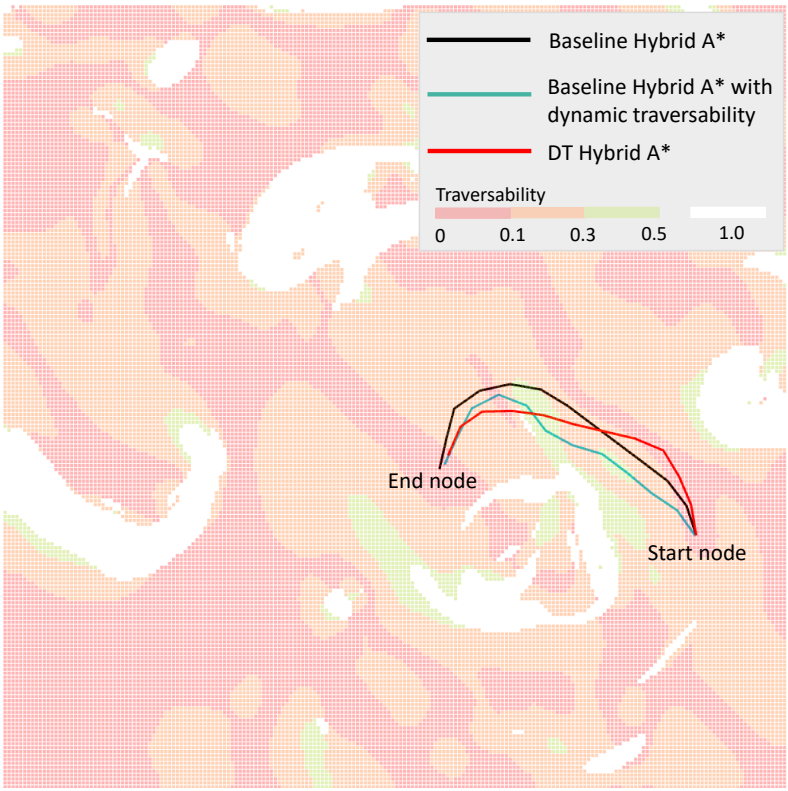

Figure 5-3: Test1: Comparison of the three algorithms in a hilly environment on the 2D traversability map.

traversability can be smaller than the static terrain traversability. That means for one global path, the mean static terrain traversability may be larger than the mean dynamic terrain traversability, and smaller mean static terrain traversability could only verify better static smoothness of the path when we consider the movement of the ego vehicle, the difference between static smoothness and dynamic traversability can be possibly compensated. Table 5-2 shows the comparison results of three algorithms based on metrics.

Table 5-2: Test1: Comparison of four metrics: Mean Static Terrain Traversability $\mathrm{Tra_m}$, Path Length Len, Path Planning Completion Time $\mathrm{T_{pc}}$, CPU Time Cost $\mathrm{T_{cpu}}$ across three algorithms: Baseline (Baseline Hybrid A*), Baseline with DT (Baseline Hybrid A* with dynamic traversability), and DT Hybrid A*.

Algorithm	$\mathrm{Tra_m}\,(\mathrm{m^{-1}})$	$\mathrm{Len}\,(\mathrm{m})$	$\mathrm{T_{pc}}\,(\mathrm{s})$	$\mathrm{T_{cpu}}\,(\mathrm{ms})$
Baseline	8.2300×10^{-4}	6.2257	12.8621	43
Baseline with DT	1.4033×10^{-3}	6.0827	12.0834	187
DT Hybrid A*	3.2857×10^{-3}	5.6698	5.3655	18

First, we compare the Baseline Hybrid A* with and without dynamic traversability. By incorporating dynamic traversability, the Baseline Hybrid A* with dynamic traversability achieves a slightly shorter overall path length. As shown in Figure (5-4), the global path generated with dynamic traversability tends to move closer to the hill region. This results in

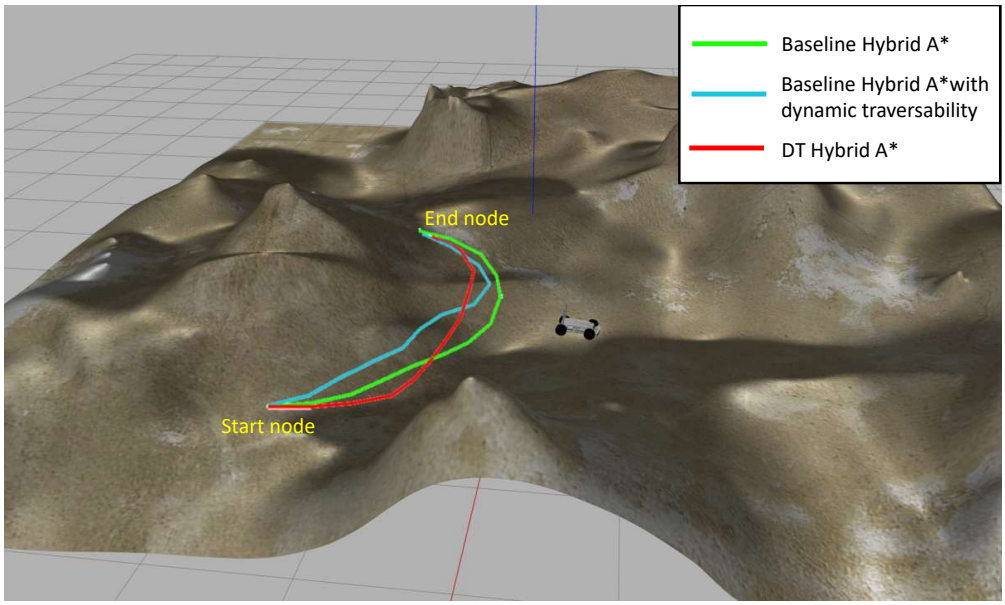

Figure 5-4: Test1: Comparison of the three algorithms on the 3D hill terrain. The 3D path length of the DT Hybrid A* is the shortest. This path closely aligns with the hill edge and minimizes elevation changes. In contrast, the paths generated by the Baseline Hybrid A* with and without dynamic traversability take detours that do not align as closely with the hill edge or experience greater elevation changes.

a slightly shorter path planning completion time, indicating that the ego vehicle would reach its goal more quickly.

However, the Baseline Hybrid A* with dynamic traversability incurs a higher CPU time cost. This is because the algorithm requires additional computational resources to assess traversability dynamically based on the planned movement of the ego vehicle at the current node. Consequently, the CPU time cost for the Baseline Hybrid A* with dynamic traversability is significantly higher than that of the standard Baseline Hybrid A*.

Moreover, when using static traversability to evaluate these algorithms, the mean static terrain traversability of the Baseline Hybrid A* with dynamic traversability is found to be higher than that of the standard Baseline Hybrid A*. This aligns with the earlier discussion on the evaluation of dynamic traversability-based algorithms using static mean terrain traversability.

The DT Hybrid A* algorithm significantly improves performance across several metrics, including Path Length, Path Planning Completion Time, and CPU Time Cost. The Path Length achieved by DT Hybrid A* is shorter than that of the Baseline Hybrid A* with dynamic traversability, and its Path Planning Completion Time is less than half that of the Baseline Hybrid A* with dynamic traversability.

This improvement can be attributed to the use of the kinematic bicycle model as the steering function in DT Hybrid A*. Unlike the simplified bicycle model, which assumes constant velocity across branches due to its lack of acceleration information, the kinematic bicycle model allows for acceleration and deceleration between adjacent nodes. This capability helps significantly reduce the overall planning completion time.

Additionally, the CPU Time Cost for DT Hybrid A* is dramatically lower than both the

Baseline Hybrid A* with dynamic traversability and the standard Baseline Hybrid A*. This efficiency gain is likely due to the heuristic cost structure employed by DT Hybrid A*. By using the 2D Dubins Curve to calculate the distance heuristic and incorporating trapezoidal velocity planning to estimate time heuristic cost, the algorithm can optimize the time required to move from the current node to the goal while respecting constraints on acceleration, velocity, and travel distance based on the planned current node's movement state.

This heuristic structure effectively guides the search tree by considering the current node's state to plan an optimal path and generate a realistic velocity profile that smoothly transitions between adjacent nodes. In contrast, the simplified bicycle model-based steering function assumes that the vehicle maintains a constant velocity to reach the next node, which results in abrupt and unrealistic velocity changes when transitioning to a new node.

Moreover, the DT Hybrid A* does not rely on Spatial Parameter Discretization, meaning that the travel length of each segment can vary. This flexibility allows the algorithm to generate path segments of different lengths, adapting to the specific needs of the terrain. In contrast, the Baseline Hybrid A* uses Spatial Parameter Discretization, where each segment is of equal length. This uniform segment length can lead to inefficiencies, particularly if a segment intersects a non-traversable region. In such cases, the entire segment must be discarded, and planning must restart from the previous node, increasing computation time. On the other hand, DT Hybrid A* can more effectively navigate around obstacles by choosing shorter segments, reducing the likelihood of encountering non-traversable regions, and optimizing the overall path-planning process.

Finally, the Mean Static Terrain Traversability of DT Hybrid A* is higher than that of the other two algorithms. This suggests that DT Hybrid A* tends to choose steeper, less conservative paths, indicating a more aggressive approach to terrain navigation.

There are two other tests that verify the effectiveness of DT Hybrid A*. Test 2, shown in Figure (5-5) and (5-6), a simple example of global path planning where the direct connection between the start and end poses does not cross a large non-traversable region, and the distance between the start and end poses is relatively short. The global path generated by the DT Hybrid A* is the most straight compared with the other two algorithms. In Table 5-3, the mean static terrain traversability of the DT Hybrid A* is even smaller than that of the Baseline Hybrid A*, which means in this case, DT Hybrid A* can achieve the smoothest global path based on static terrain traversability. This is likely because, in smooth regions, the static traversability is relatively small. When using dynamic traversability, which accounts for the vehicle's momentum, the traversability can become even smaller, to the point where it has little impact on the cost functions. As a result, the cost term associated with path length becomes more significant, and the global path generated by DT Hybrid A* tends to be shorter. Moreover, the Baseline with DT Hybrid A* using dynamic traversability consistently produces shorter global paths compared to the Baseline without it, further verifying this characteristic. The path length and path planning completion time for the three algorithms are ranked in the following order: DT Hybrid A*, Baseline Hybrid A* with dynamic traversability, and Baseline Hybrid A*. This demonstrates the same characteristics in both metrics as observed in Test 1. Although the Baseline Hybrid A* has the shortest CPU time, with DT Hybrid A* also performing efficiently at just 2 ms, this suggests that when the planned global path is relatively short and lies in a smooth region, the computation cost of dynamic traversability can be significant. This is evident as the Baseline Hybrid A* runs extremely fast, nearly

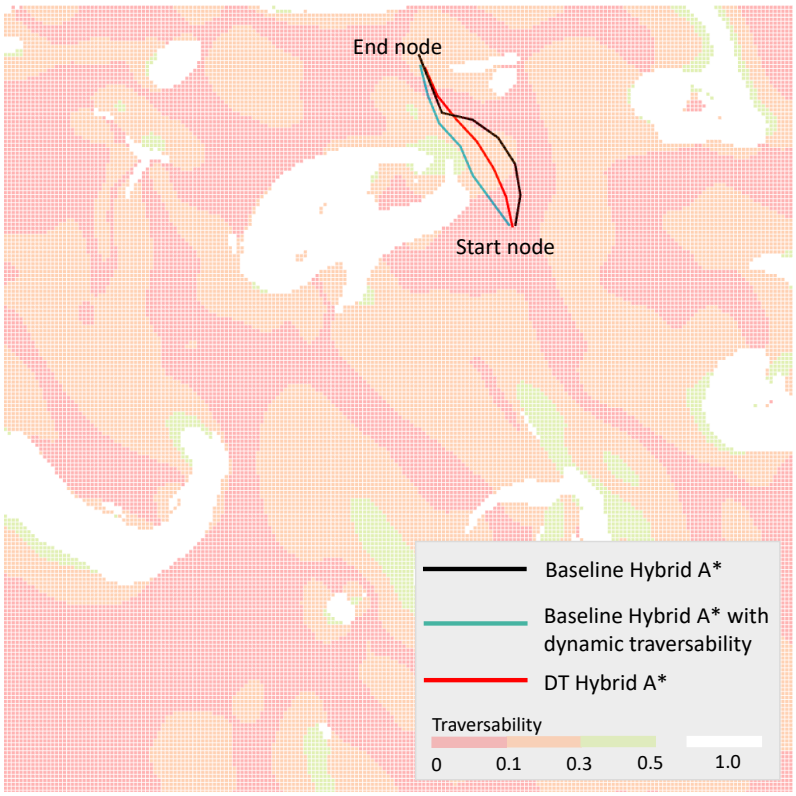

Figure 5-5: Test2: Comparison of the three algorithms in a hilly environment on the 2D traversability map.

approaching zero computation time.

Test3 is shown in Figure (5-7) and (5-8). On the 2D traversability map, the global path generated by the DT Hybrid A* does not pass closest to the hill region. The Baseline Hybrid A*, incorporating dynamic traversability, circumnavigates the hill with the shortest distance, closely following the hill's edge. However, as shown in Table 5-4, the DT Hybrid A* results in a shorter path overall compared to the other algorithms because the shortest path in 2D does not necessarily correspond to the shortest path in 3D. The global path generated by the Baseline Hybrid A* with dynamic traversability involves more waypoints with high static traversability but also carries more risks. In (5-8), the significant changes in the z-axis of the path indicate substantial elevation variations, leading to large ups and downs along the global path. These elevation changes are reflected in the mean static terrain traversability value, where the DT Hybrid A* shows a much lower value compared to the Baseline Hybrid A* with dynamic traversability. This suggests that the Baseline Hybrid A* with dynamic traversability selects more waypoints that are not statically smooth. This indicates that the DT Hybrid A* tends to converge towards the 3D shortest path by considering both static terrain traversability, which captures the smoothness of the local terrain, and the 2D path length, which is one of the cost functions in the DT Hybrid A* algorithm.

In summary, the effectiveness of dynamic traversability is demonstrated by comparing the Baseline Hybrid A* with and without dynamic traversability, using the same cost functions, search algorithms, and parameters. The Baseline Hybrid A* with dynamic traversability

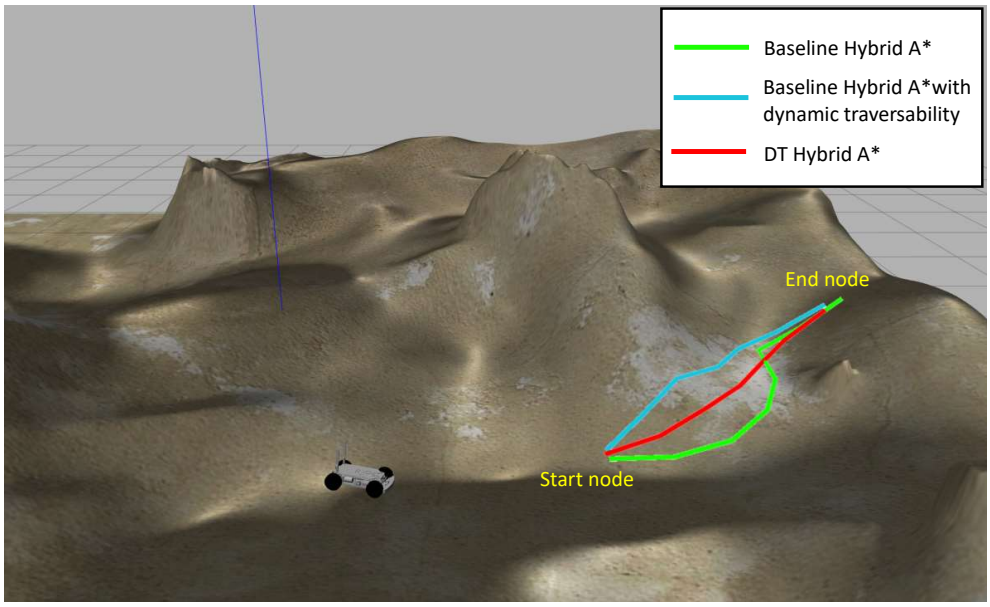

Figure 5-6: Test2: Comparison of the three algorithms on the 3D hill terrain. The DT Hybrid A* algorithm chooses a more direct and straightforward path compared to the other two algorithms. It avoids detours and maintains minimal elevation changes.

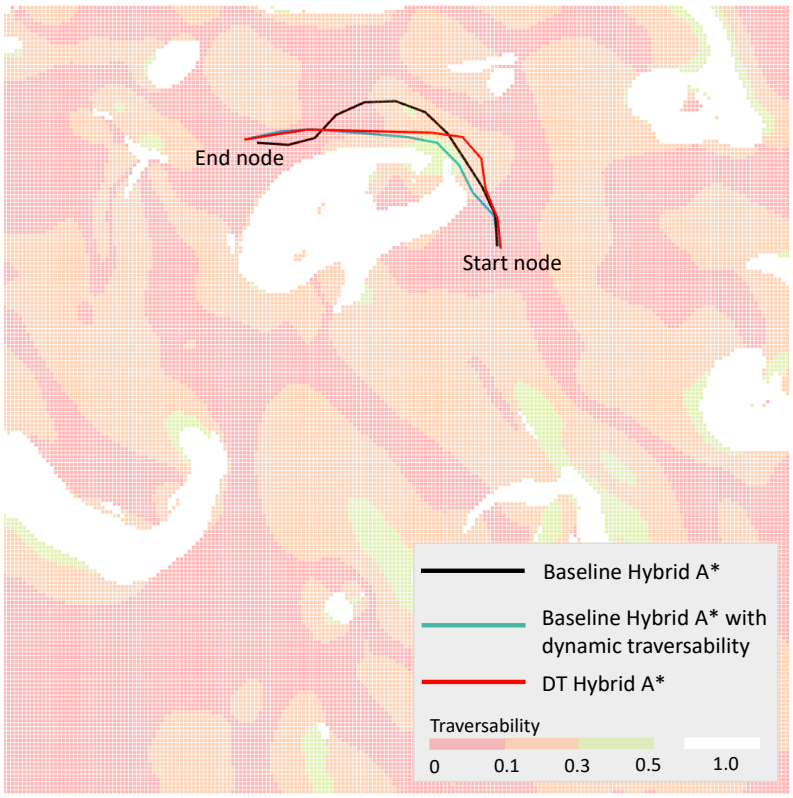

Figure 5-7: Test3: Comparison of the three algorithms in a hilly environment on the 2D traversability map.

Table 5-3: Test2: Comparison of four metrics: Mean Static Terrain Traversability $\mathrm{Tra_m}$, Path Length Len, Path Planning Completion Time $\mathrm{T_{pc}}$, CPU Time Cost $\mathrm{T_{cpu}}$ across three algorithms: Baseline (Baseline Hybrid A*), Baseline with DT (Baseline Hybrid A* with dynamic traversability), and DT Hybrid A*.

Algorithm	$\mathrm{Tra_m}\,(\mathrm{m}^{-1})$	$\mathrm{Len}\,(\mathrm{m})$	$\mathrm{T_{pc}}\,(\mathrm{s})$	$\mathrm{T_{cpu}}\,(\mathrm{ms})$
Baseline	1.2438×10^{-3}	7.5592	7.9078	<1
Baseline with DT	2.7616×10^{-3}	7.1573	7.2754	49
DT Hybrid A*	1.10646×10^{-3}	6.8866	3.5477	2

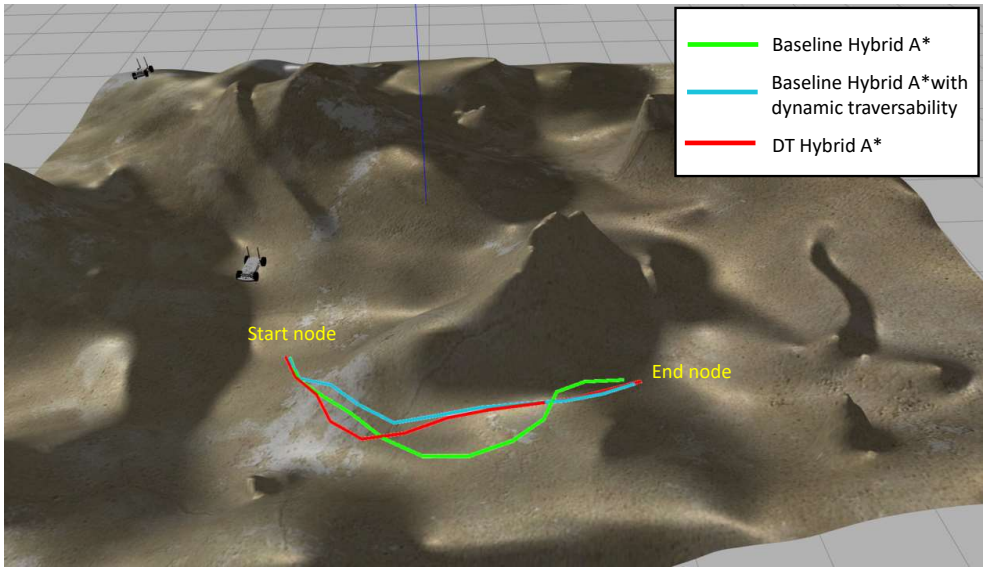

Figure 5-8: Test3: Comparison of the three algorithms on the 3D hill terrain. On the 2D traversability map, the Baseline Hybrid A* with dynamic traversability has the shortest 2D path length. However, the path by DT Hybrid A* connects the points with similar elevation heights when navigating around the hill region from the start node side to the end node side. However, Baseline Hybrid A*, with dynamic traversability, chooses a steeper path with greater elevation changes, resulting in a longer 3D path.

outperforms the version without it in terms of path length and path planning completion time. However, the introduction of dynamic traversability incurs additional computational costs.

To address this drawback, the DT Hybrid A* leverages a kinematic bicycle model as its steering function, allows for variable travel lengths between adjacent nodes, and incorporates both distance and time heuristics. These enhancements significantly reduce CPU time costs, producing shorter path lengths and faster planning times.

In straightforward global path planning scenarios, where the path between start and end poses is relatively short and avoids large non-traversable areas, the DT Hybrid A* generates the most direct route compared to other algorithms. However, it incurs higher CPU time compared to the Baseline Hybrid A*. This indicates that, even for short and smooth paths, the computational cost of dynamic traversability can be significant, as reflected by the Baseline Hybrid A*'s minimal computation time.

Table 5-4: Test3: Comparison of four metrics: Mean Static Terrain Traversability $\mathrm{Tra_m}$, Path Length Len, Path Planning Completion Time $\mathrm{T_{pc}}$, CPU Time Cost $\mathrm{T_{cpu}}$ across three algorithms: Baseline (Baseline Hybrid A*), Baseline with DT (Baseline Hybrid A* with dynamic traversability), and DT Hybrid A*.

Algorithm	$\mathrm{Tra_m}\,(\mathrm{m}^{-1})$	$\mathrm{Len}\,(\mathrm{m})$	$\mathrm{T_{pc}}\,(\mathrm{s})$	$\mathrm{T_{cpu}}\,(\mathrm{ms})$
Baseline	1.93615×10^{-3}	9.0549	12.2372	31
Baseline with DT	1.2846×10^{-3}	8.8250	11.0779	359
DT Hybrid A*	9.4836×10^{-4}	8.7822	3.8674	13

Furthermore, while the Baseline Hybrid A* with dynamic traversability may opt for more waypoints that are not statically smooth, potentially increasing elevation variations and then 3D path length, the DT Hybrid A* is designed to converge more effectively towards the 3D shortest path. It achieves this by balancing static terrain traversability, which addresses local terrain smoothness, with the 2D path length, one of its key cost functions.

5-3-2 Effectiveness of the Dynamic Obstacle Avoidance

In this section, the effectiveness of the Dynamic Obstacle Avoidance method is validated by comparing the Safety and Direction Elements-based VO algorithm (DSVO) against the Safety Element-based VO algorithm and the Euclidean distance method for collision check. In Section 4-3, the differences among the three algorithms are introduced, highlighting how they incorporate various cost functions into the DT Hybrid A* algorithm to address potential collisions with dynamic obstacles during path replanning. DSVO incorporates an exponential function based on the shortest distance to the dynamic obstacle's VO (Velocity Obstacle) region, adjusted by the sign of the direction element. This adjustment helps in assessing the collision risk more accurately by considering both the safety for velocity of each waypoints not in the VO region and the direction to be away from the dynamic obstacle in a longer time span than that to construct VO region and. Safety Element-based VO Algorithm only adds an exponential function to the shortest distance to the VO region of the dynamic obstacle. This approach is exemplified by the existing SATG algorithm [9], which applies this concept to assess and mitigate collision risks. Euclidean Distance Method calculates the Euclidean distance from the dynamic obstacle to each waypoint on the replanned path. It directly measures the spatial distance for collision risk assessment.

In this thesis, we focus exclusively on the straight-line motion of the dynamic obstacle at a constant velocity. To isolate the impact of the dynamic obstacle's movement on the replanning process, we use the same initial and goal positions and the same initial pose for the dynamic obstacle. The dynamic obstacle remains stationary at the start, ensuring it does not influence the generation of the initial global path, since it is positioned outside the vicinity of any potential connections between the start and goal poses. This setup allows the initial global path to be free of dynamic obstacles. The obstacle then begins moving at time $t = 1s$, with different velocities and directions tested in various scenarios.

As the maximum longitudinal velocity is set to be 0.5m/s, three different velocity magnitudes of 0.2m/s, 0.3m/s, and 0.4m/s are selected for testing at heading directions -1.81rad and -1.51rad respectively.

Then, four metrics are used to evaluate the effectiveness of the three replanning algorithms: Path Length, Re-planning Completion Time, CPU Time Cost, and Success Rate. Path Length measures the total 3D Euclidean distance by summing the distances between consecutive waypoints on the replanned path, reflecting how efficient the replanned path is. Re-planning Completion Time is the estimated total duration required to follow the replanned path, indicating how quickly the vehicle can reach its destination. CPU Time Cost assesses the computational resources consumed by the algorithm. Finally, the Success Rate determines whether the vehicle successfully avoids collisions with dynamic obstacles while following the replanned path, highlighting the algorithm's ability to ensure safe navigation.

Test 1: Dynamic Obstacle Moving at 0.2m/s in the Direction -1.81rad

In Test 1, the dynamic obstacle starts moving at time $t = 1s$ with a velocity of 0.2m/s in the direction of -1.81rad. Figures (5-9) show the replan results of the three replan algorithms: Baseline (Euclidean distance-based replan algorithm), SATG, and DSVO. It can be observed that the replanned path generated by the Baseline algorithm fails to avoid the dynamic obstacle, whereas both SATG and DSVO successfully navigate around it. The likely reason is that the Euclidean distance-based replanning algorithm considers only the sum of the Euclidean distances from the dynamic obstacle to all waypoints on the replanned path. As a result, it may generate a path that minimizes the distance to the obstacle but fails to account for the obstacle's velocity. Consequently, the algorithm might produce a path that still intersects with the moving obstacle. By comparing the replanned paths of SATG and DSVO, it is observed that the path generated by DSVO is shorter than that of SATG, while the path replanning time for SATG is shorter than that of DSVO. This difference may be attributed to DSVO using a relatively smaller acceleration during the node branching process when safety and directional elements cannot be optimized simultaneously, requiring a balance between the two, resulting in a more conservative node branching strategy. In terms of CPU time, the Baseline and DSVO algorithms take the same amount of time $2ms$, which is slightly less than the time required by SATG $3ms$ shown in Table 5-5.

Table 5-5: Test 1: Comparison of four metrics: Replanned Path Length Len, Path Re-planning Completion Time $\mathrm{T_{pc}}$, CPU Time Cost $\mathrm{T_{cpu}}$ and Success Condition across three algorithms: Baseline (Euclidean distance based replan algorithm), SATG, and DSVO.

Algorithm	Len (m)	$\mathrm{T_{pc}}$ (s)	$\mathrm{T_{cpu}}$ (ms)	Success Condition
Baseline	4.8055	2.6105	2	Fail
SATG	4.9431	1.2314	3	Succeed
DSVO	4.4912	1.7215	2	Succeed

Test 2: Dynamic Obstacle Moving at 0.3m/s in the Direction -1.81rad

Test 2 sets the dynamic obstacle to start moving at time $t = 1s$ with a velocity of 0.3m/s in the direction of -1.81rad. Figures (5-10) show the replan results of Baseline (Euclidean distance-based replan algorithm), SATG, and DSVO.

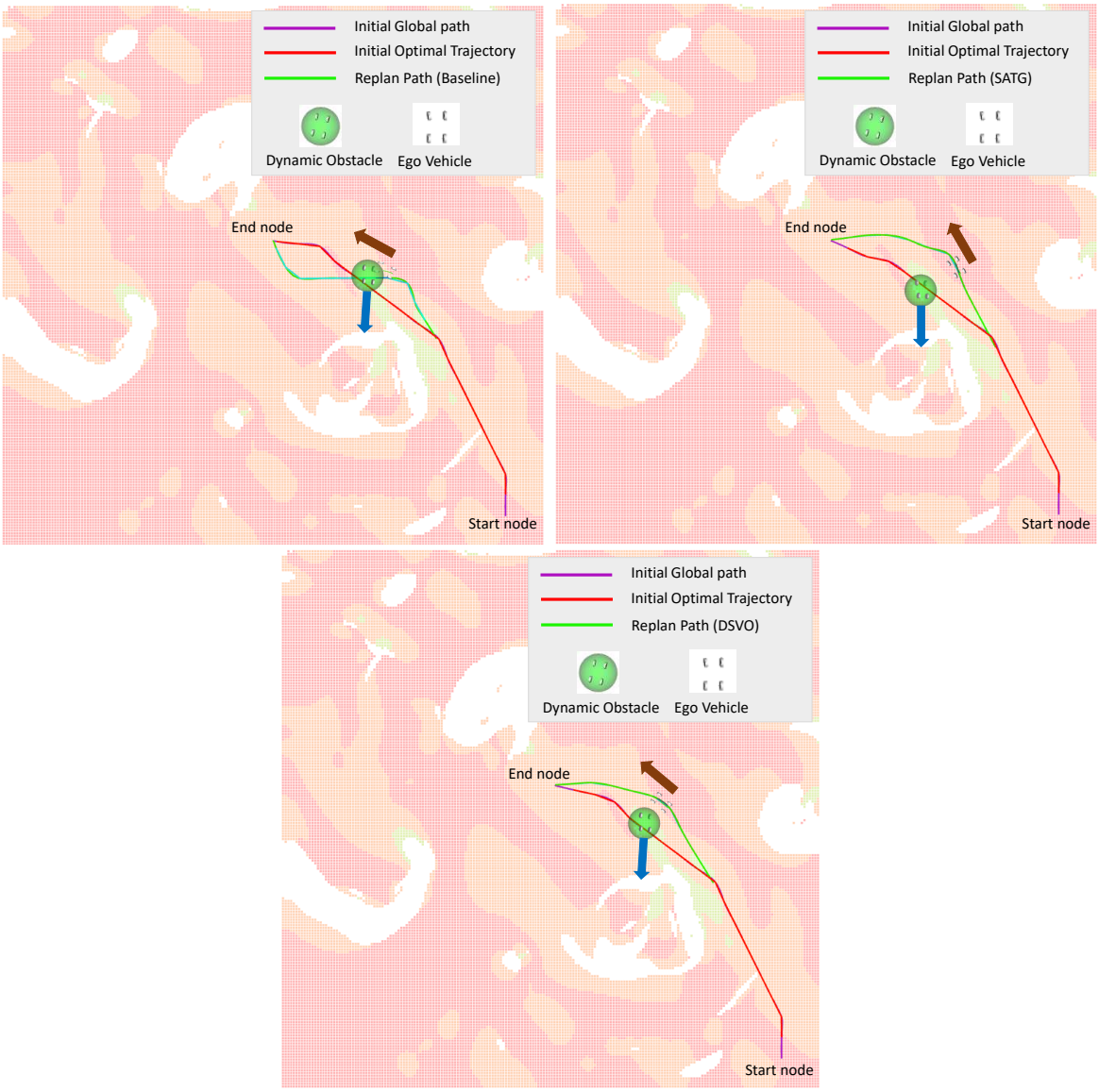

Figure 5-9: Test 1: Replan results in a hilly environment on the 2D traversability map (a)Baseline (Euclidean distance based) replan algorithm. (b) SATG. (c) DSVO. The blue and brown arrows indicate the direction of movement of the dynamic obstacle and ego vehicle, respectively.

In this case, the replanning process for all three algorithms successfully produces a collision-free path. Among them, DSVO generates the shortest route and has the quickest path replanning completion time shown in Table 5-6. The CPU time for DSVO is slightly higher than that of the other two algorithms, but this difference is minimal and can be considered negligible.

Table 5-6: Test2: Comparison of four metrics: Replanned Path Length Len, Path Re-planning Completion Time T_{pc}, CPU Time Cost T_{cpu} and Success Condition across three algorithms: Baseline (Euclidean distance based replan algorithm), SATG, and DSVO.

Algorithm	Len (m)	T_{pc} (s)	T_{cpu} (ms)	Success Condition
Baseline	4.7703	2.7521	5	Succeed
SATG	4.8056	1.2013	5	Succeed
DSVO	4.4973	1.1730	6	Succeed

Test 3: Dynamic Obstacle Moving at 0.4m/s in the Direction -1.81rad

In Test 3, the dynamic obstacle starts moving at time $t = 1s$ with a velocity of 0.4m/s in the direction of -1.81rad. Figures (5-11) show the replan results of Baseline (Euclidean distance-based replan algorithm), SATG, and DSVO.

In this case, the three algorithms' replanned process successfully achieves a collision-free replan path. DSVO achieves the shortest replanned path length and requires the least CPU time compared to the other two algorithms. Table 5-7 shows that SATG has a shorter path replanning time than DSVO. This discrepancy can be attributed to DSVO's use of smaller acceleration during the node branching process. When safety and direction elements cannot be simultaneously optimized, DSVO adopts a more conservative strategy, balancing these factors and leading to the observed difference in replanning time.

Consider the following example: a larger acceleration results in a higher velocity, which in turn leads to a larger SA value. Conversely, a smaller acceleration leads to a lower velocity and a correspondingly smaller SA value. According to Equation (4-24), when the collision direction boolean element is false, a higher SA value results in a lower cost, favoring the selection of a higher velocity. However, the situation changes when the collision direction boolean element is true. The lower velocity with a smaller SA value becomes more likely to be selected. So, based on this case, when the direction boolean element is true, a smaller acceleration leading to lower velocity is easily chosen, which would further cause a longer path replanning completion time.

Test 4: Dynamic Obstacle Moving at 0.2m/s in the Direction -1.51rad

In Test 4, the dynamic obstacle moves at a velocity of 0.2m/s in the direction of -1.51rad. Figures (5-12) show the replan results of Baseline (Euclidean distance-based replan algorithm), SATG, and DSVO. It is evident that the replanned paths generated by both the Baseline algorithm and SATG fail to avoid the dynamic obstacle, while only DSVO successfully navigates around it. Table 5-8 demonstrates that, although DSVO results in the longest

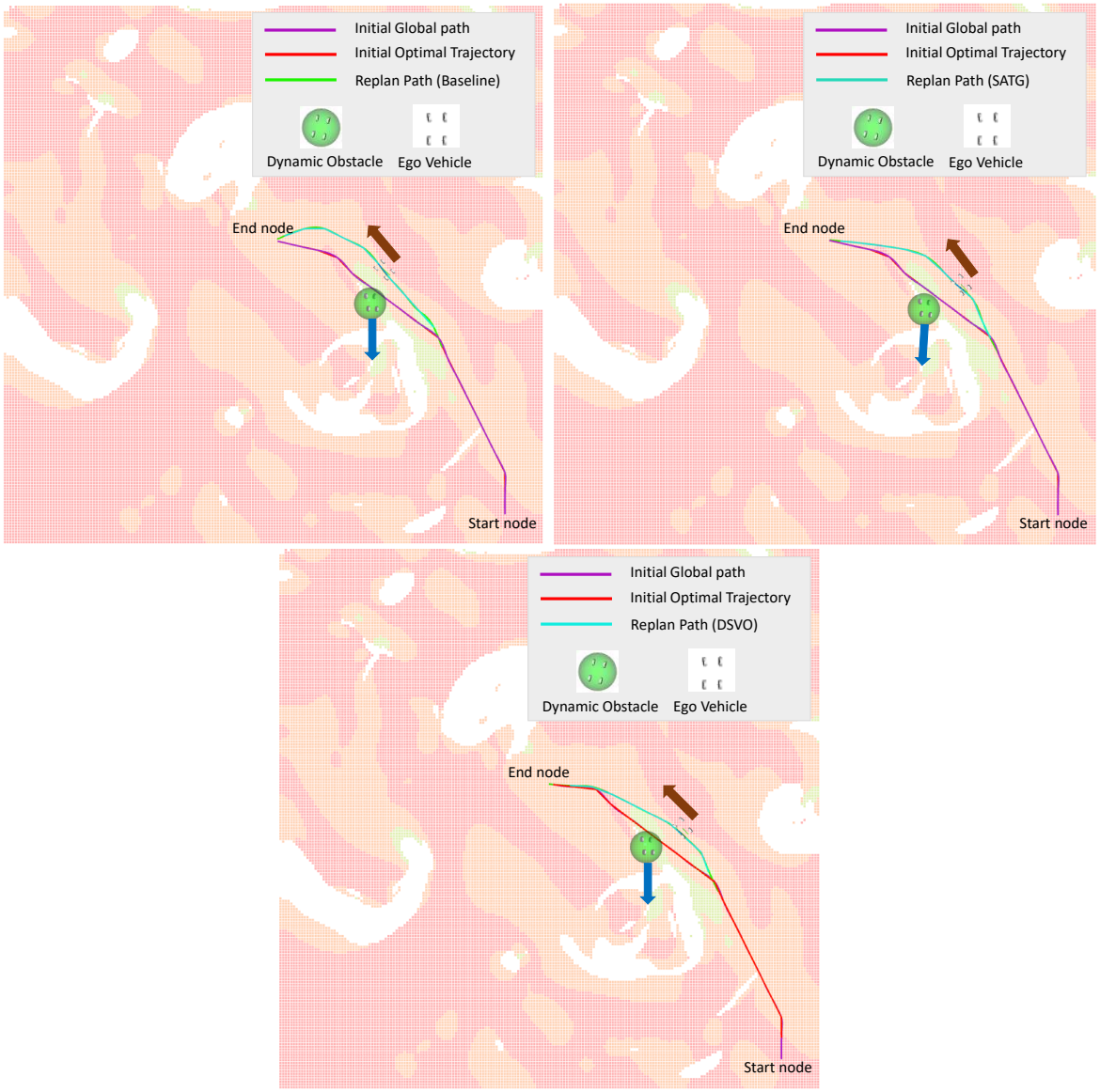

Figure 5-10: Test 2: Replan results in a hilly environment on the 2D traversability map (a)Baseline (Euclidean distance based) replan algorithm. (b) SATG. (c) DSVO. The blue and brown arrows indicate the direction of movement of the dynamic obstacle and ego vehicle, respectively.

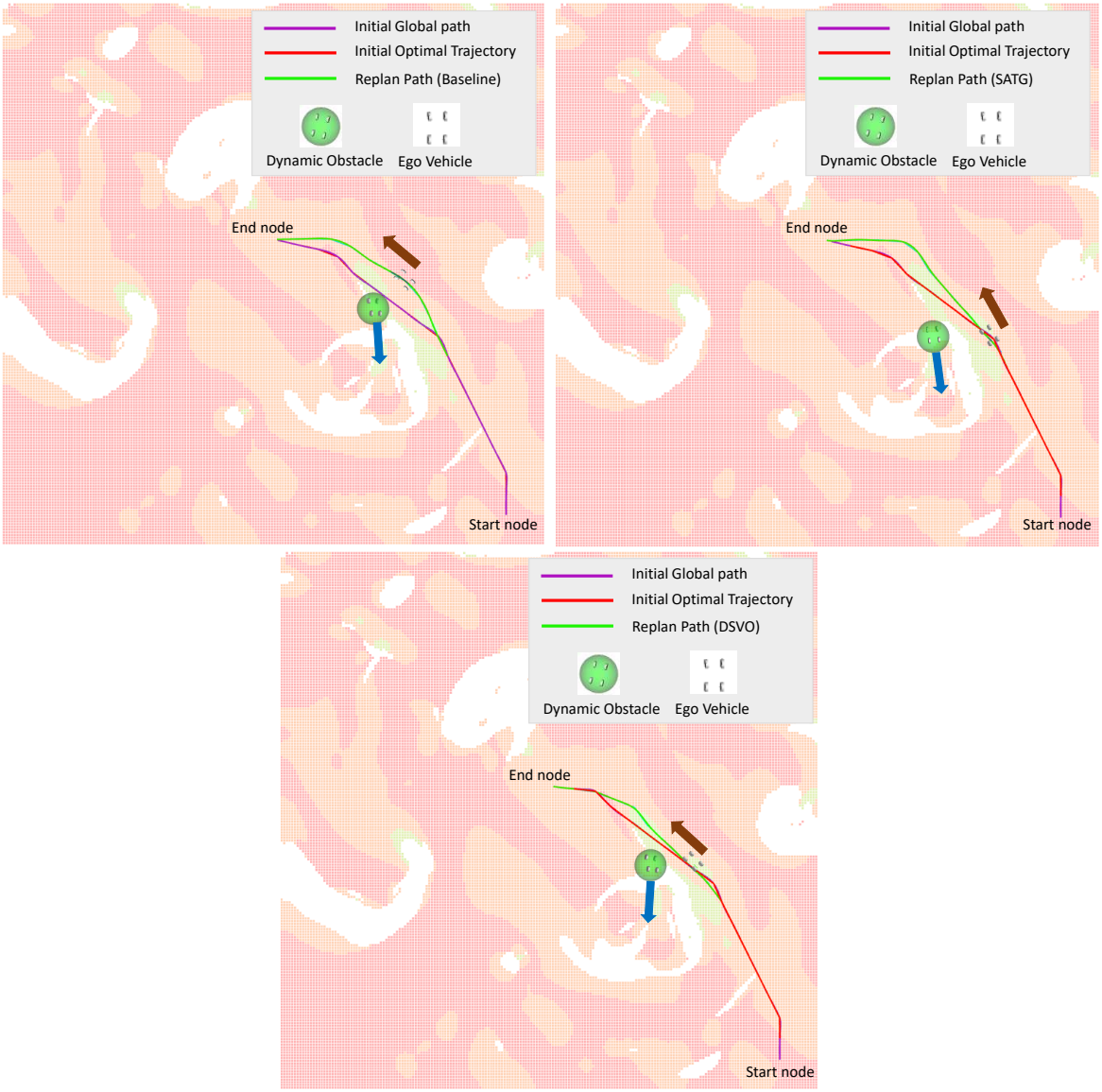

Figure 5-11: Test 3: Replan results in a hilly environment on the 2D traversability map (a)Baseline (Euclidean distance based) replan algorithm. (b) SATG. (c) DSVO. The blue and brown arrows indicate the direction of movement of the dynamic obstacle and ego vehicle, respectively.

Table 5-7: Test3: Comparison of four metrics: Replanned Path Length Len, Path Re-planning Completion Time T_{pc}, CPU Time Cost T_{cpu} and Success Condition across three algorithms: Baseline (Euclidean distance based replan algorithm), SATG, and DSVO.

Algorithm	Len (m)	T_{pc} (s)	T_{cpu} (ms)	Success Condition
Baseline	4.7472	1.6688	9	Succeed
SATG	5.0587	1.2670	7	Succeed
DSVO	4.4455	1.4497	7	Succeed

path length, it achieves the fastest path replanning completion time, the lowest CPU time cost and the only success replanning process among the three algorithms. This highlights the advantage of the DSVO algorithm: the portion of the path near the replanned section stays well clear of the dynamic obstacle, adapting to the obstacle's movement by bending away from it. Meanwhile, the portion of the replanned path farther from the obstacle takes the most direct route toward the goal position. This approach ensures safety by curving the path around the dynamic obstacle in the opposite direction of its movement while also selecting the shortest route that maintains a safe distance from the obstacle.

To better visualize the process of the ego vehicle following the replanned path and either successfully avoiding or failing to avoid dynamic obstacles, , Figures 5-13, 5-14 and 5-15 capture snapshots of the vehicle's movement at some time points. Time is measured starting from the moment the ego vehicle begins moving, designated as $t = 0$s. In these pictures, the blue arrow indicates the direction of the moving obstacle, while the brown arrow shows the moving direction of the ego vehicle at the time the snapshot was taken.

Table 5-8: Test4: Comparison of four metrics: Replanned Path Length Len, Path Re-planning Completion Time T_{pc}, CPU Time Cost T_{cpu} and Success Condition across three algorithms: Baseline (Euclidean distance based replan algorithm), SATG, and DSVO.

Algorithm	Len (m)	T_{pc} (s)	T_{cpu} (ms)	Success Condition
Baseline	4.2590	2.6692	8	Fail
SATG	4.7944	15.6039	7	Fail
DSVO	4.8296	1.2049	7	Succeed

Test 5: Dynamic Obstacle Moving at 0.3m/s in the Direction -1.51rad

Test 2 sets the dynamic obstacle to start moving at time $t = 1s$ with a velocity of 0.3m/s in the direction of -1.51rad. Figures (5-16) show the replan results of Baseline (Euclidean distance-based replan algorithm), SATG, and DSVO.

In this scenario, both the Baseline and DSVO algorithms successfully generate collision-free replanned paths, whereas the SATG algorithm fails to avoid the dynamic obstacle. As shown in Table 5-9, although DSVO has a slightly higher CPU usage compared to the other two algorithms, it consistently produces the shortest collision-free path. Moreover, it completes the path replanning process in the shortest time.

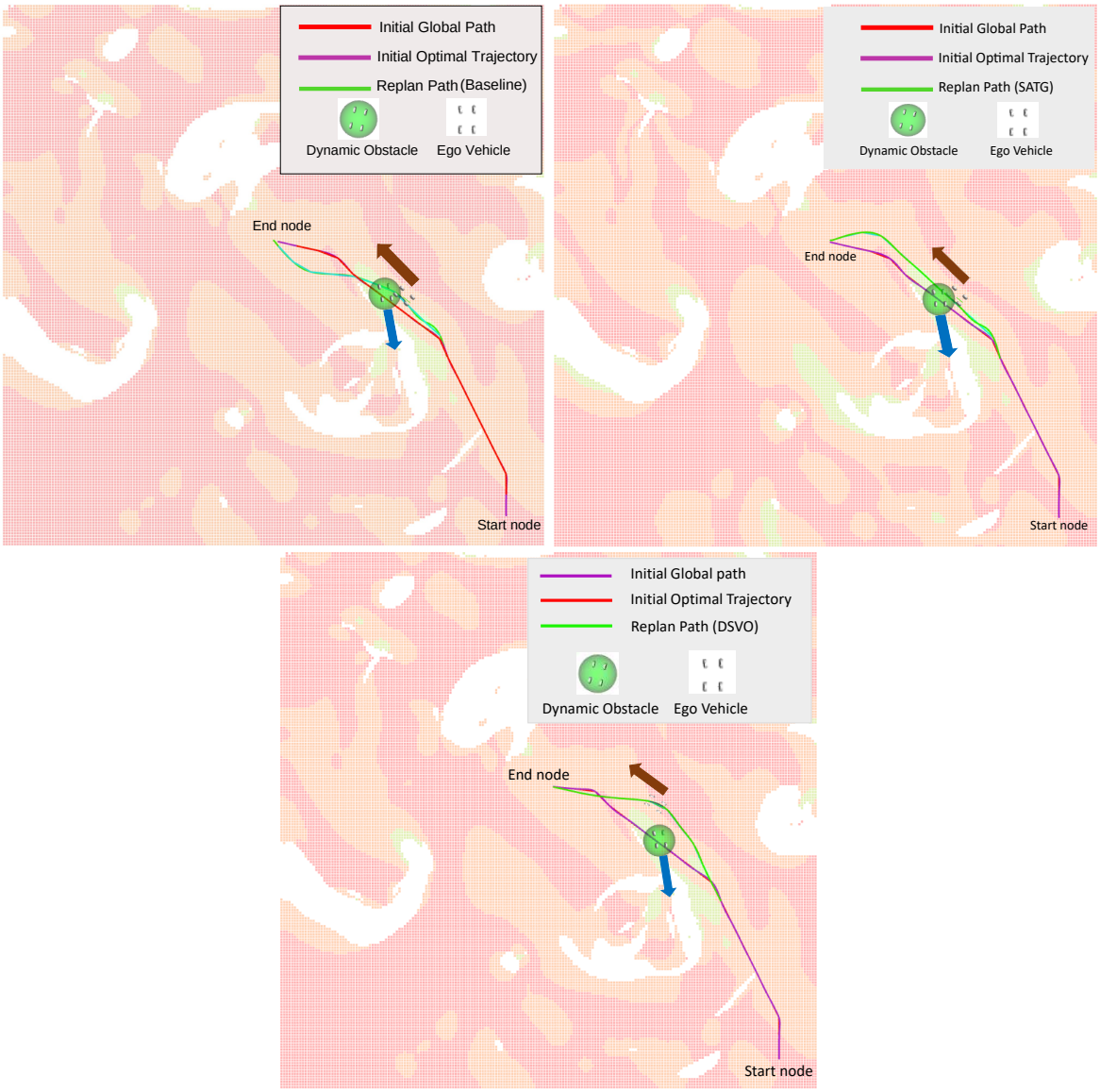

Figure 5-12: Test 4: Replan results in a hilly environment on the 2D traversability map (a)Baseline (Euclidean distance based) replan algorithm. (b) SATG. (c) DSVO. The blue and brown arrows indicate the direction of movement of the dynamic obstacle and ego vehicle, respectively.

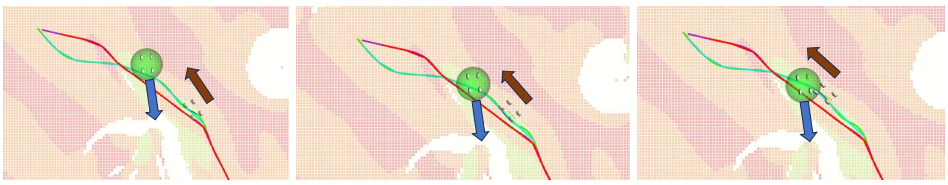

Figure 5-13: Test4: Snapshots of the vehicle's movement as it follows the replanned path generated by the Baseline replanning algorithm. The snapshots are taken at the following time points: (a) $t = 19s$ (b) $t = 21s$ (c) $t = 23s$. The blue and brown arrows indicate the movement directions of the dynamic obstacle and ego vehicle at the time the snapshot was taken, respectively.

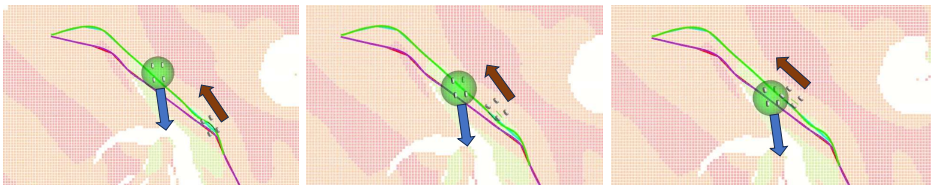

Figure 5-14: Test4: Snapshots of the vehicle's movement as it follows the replanned path generated by the SATG replanning algorithm. The snapshots are taken at the following time points: (a) $t = 17s$.(b) $t = 20s$ (c) $t = 23s$. The blue and brown arrows indicate the movement directions of the dynamic obstacle and ego vehicle at the time the snapshot was taken, respectively.

Table 5-9: Test5: Comparison of four metrics: Replanned Path Length Len, Path Re-planning Completion Time $\mathrm{T_{pc}}$, CPU Time Cost $\mathrm{T_{cpu}}$ and Success Condition across three algorithms: Baseline (Euclidean distance based replan algorithm), SATG, and DSVO.

Algorithm	Len (m)	$\mathrm{T_{pc}}$ (s)	$\mathrm{T_{cpu}}$ (ms)	Success Condition
Baseline	5.0777	6.5861	3	Succeed
SATG	4.6483	5.8943	3	Fail
DSVO	4.6898	1.2539	5	Succeed

Test 6: Dynamic Obstacle Moving at 0.4m/s in the Direction -1.51rad

Test 6 sets the dynamic obstacle to start moving at time $t = 1s$ with a velocity of 0.4m/s in the direction of -1.51rad. Figures (5-17) show the replan results of Baseline (Euclidean distance-based replan algorithm), SATG, and DSVO.

In this test, all three algorithms successfully generated collision-free replanned paths. As shown in Table 5-10, the DSVO algorithm produced the shortest replanned path length, with the smallest completion time and CPU usage. It can be observed that, due to the relatively fast movement of the dynamic obstacle, DSVO effectively predicts that the obstacle will cross the original path from the other side. As a result, the replanned path does not need to deviate significantly from the initial global path, unlike the paths generated by Baseline or SATG, both of which involve noticeable detours that take them farther away from the original route. This demonstrates that DSVO, by considering the dynamic obstacle's movement and its relative position to the waypoints on the replanned path, can generate a more efficient path with a shorter overall length, faster completion time, and lower CPU usage.

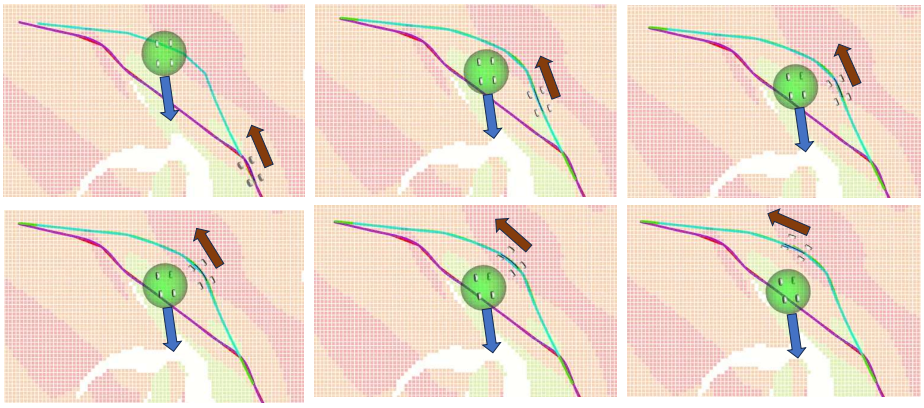

Figure 5-15: Test4: Snapshots of the vehicle's movement as it follows the replanned path generated by the DSVO replanning algorithm. The snapshots are taken at the following time points: (a) $t = 16s$ (b) $t = 17s$ (c) $t = 18s$ (d) $t = 19s$ (e) $t = 21s$ (f) $t = 22s$. The blue and brown arrows indicate the movement directions of the dynamic obstacle and ego vehicle at the time the snapshot was taken, respectively.

Table 5-10: Test 6: Comparison of four metrics: Replanned Path Length Len, Path Re-planning Completion Time $\mathrm{T_{pc}}$, CPU Time Cost $\mathrm{T_{cpu}}$ and Success Condition across three algorithms: Baseline (Euclidean distance based replan algorithm), SATG, and DSVO.

Algorithm	$\mathrm{Len}\,(\mathrm{m})$	$\mathrm{T_{pc}}\,(\mathrm{s})$	$\mathrm{T_{cpu}}\,(\mathrm{ms})$	Success Condition
Baseline	4.9398	6.8223	13	Succeed
SATG	5.0788	4.5422	5	Succeed
DSVO	4.7249	2.0128	5	Succeed

Test of Replan Success Rate

The replan success rates of the three algorithms were compared through a series of controlled tests. In each test, an obstacle was placed at a fixed location and began moving at time $t = 1\mathrm{s}$. To introduce variability in the goal position, a uniformly distributed random value within the range $[-0.5, 0.5]\mathrm{m}$ was added to both the x and y coordinates. The obstacle's movement direction was set at a fixed angle of $-1.51\mathrm{rad}$, with its velocity magnitude randomly selected from a uniform distribution in the range $[0.2, 0.4]\,\mathrm{m/s}$. A total of 100 test trials were conducted, and the replan success rates $\mathrm{s_r}$ for each algorithm are summarized in Table 5-11. The replan success rate $\mathrm{s_r}$ is the ratio of the number of successful avoidance instances by the ego vehicle during the replan process to the total number of tests. The results in Table 5-11

Table 5-11: Replan Success Rate $\mathrm{s_r}$ of three algorithms: Baseline (Euclidean distance based replan algorithm), SATG, and DSVO.

Algorithm	Baseline	SATG	DSVO
$\mathrm{s_r}$	53%	61%	74%

demonstrate that the DSVO algorithm achieved a replan success rate of 74%, significantly outperforming the SATG algorithm, which had a success rate of 61%, and the Baseline,

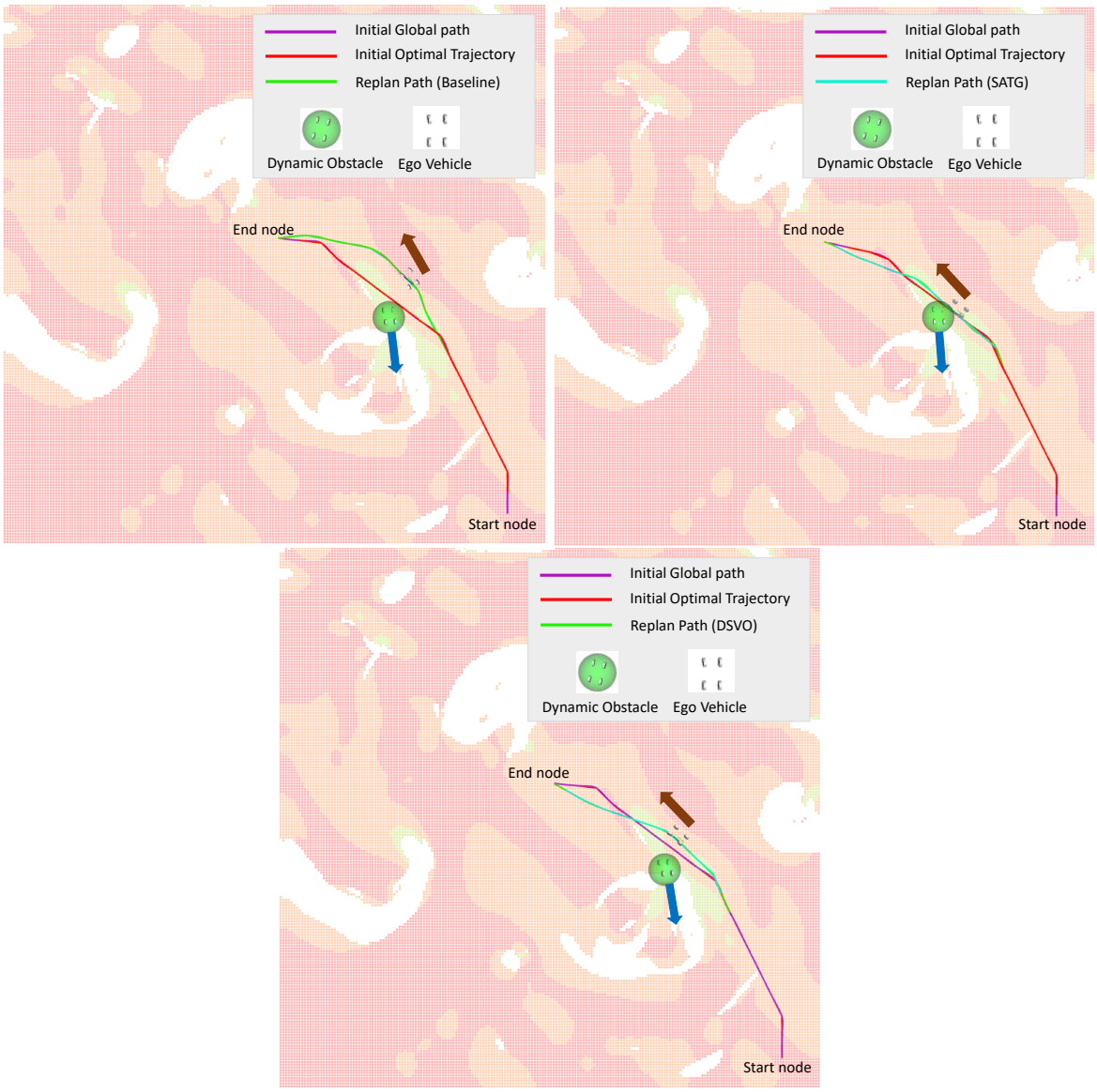

Figure 5-16: Test 5: Replan results in a hilly environment on the 2D traversability map (a)Baseline (Euclidean distance based) replan algorithm. (b) SATG. (c) DSVO. The blue and brown arrows indicate the direction of movement of the dynamic obstacle and ego vehicle, respectively.

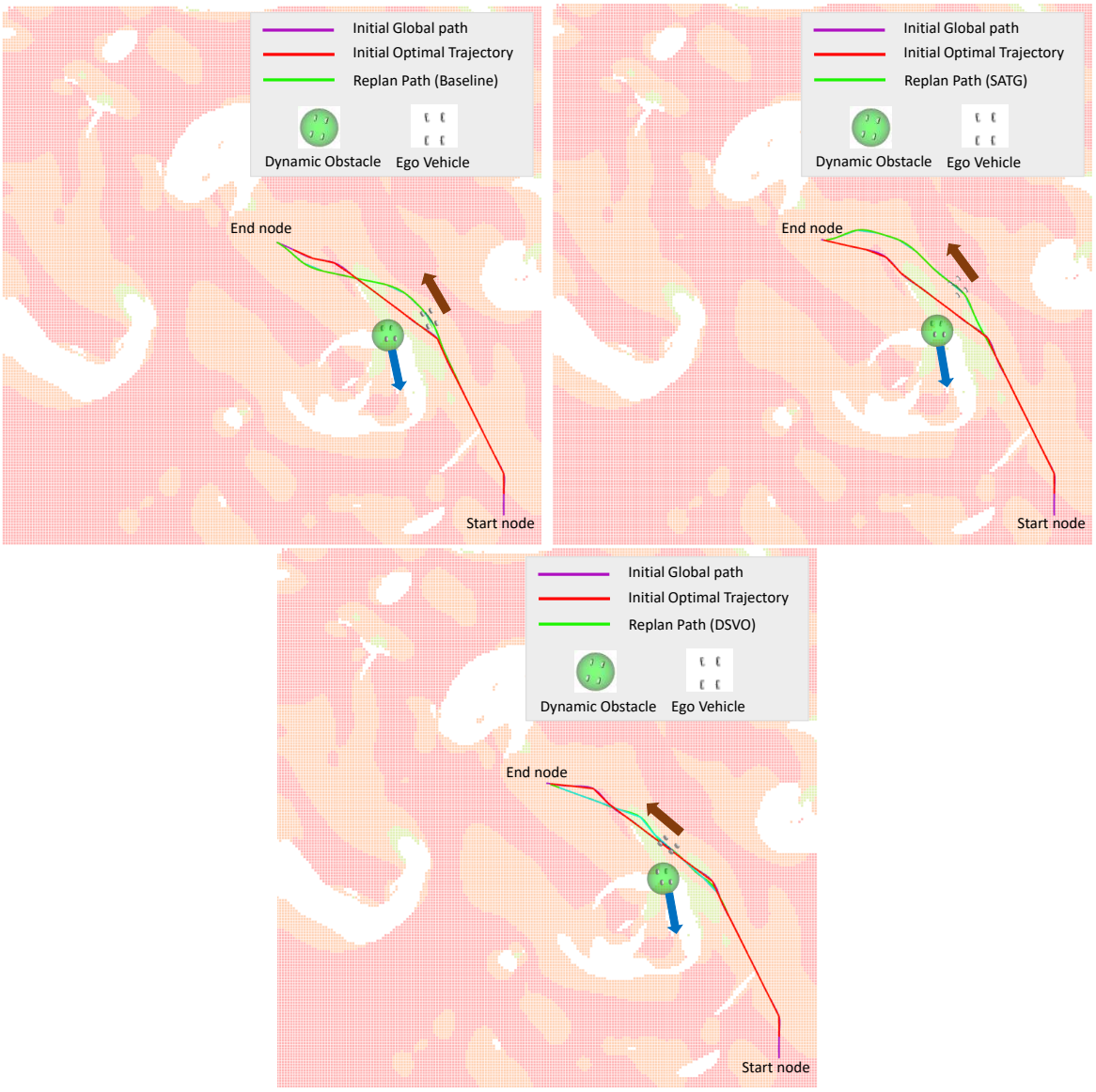

Figure 5-17: Test 6: Replan results in a hilly environment on the 2D traversability map (a)Baseline (Euclidean distance based) replan algorithm. (b) SATG. (c) DSVO. The blue and brown arrows indicate the direction of movement of the dynamic obstacle and ego vehicle, respectively.

which achieved a rate of 53%. This indicates that the DSVO algorithm is more effective at generating safe, dynamic obstacle-free replan paths compared to the other tested algorithms.

In Tests 1, 2, 3, and 6, both SATG and DSVO successfully generated dynamic, obstacle-free replanned paths. The replanned path lengths of DSVO were 6.9%, 12.1%, 6.4%, and 9.1% shorter than those of SATG, respectively, while maintaining a similar CPU time cost. Additionally, the estimated time for the vehicle to complete DSVO's replan path was generally shorter or slightly longer than SATG's. The slightly higher completion time for DSVO's replanning in certain cases may be due to its use of lower acceleration during node branching when safety and direction elements cannot be optimized simultaneously, leading to a more conservative node branching strategy.

Tests 2, 3, 5, and 6 demonstrate successful dynamic obstacle avoidance by both Baseline and DSVO. The replanned path lengths of DSVO were 5.7%, 6.4%, 8.5%, and 4.4% shorter than those of Baseline. Additionally, the estimated time for the vehicle to complete DSVO's replanned path was significantly shorter than Baseline's, while maintaining a similar CPU time cost.

Tests 1 and 4 show the failure of the Baseline and show relect the similar character and verify the comparison of the Euclidean distance method and Velocity Obstacle (VO) method in Figure 4-6. This figure demonstrates that the VO method helps discard nodes that may lead to future collisions during the node selection process. It prevents the risk of generating an infeasible global path, where connecting nodes in a straight line could cause a collision with a dynamic obstacle at a later time step. VO stops further expansion in directions that may lead to a collision. In contrast, the Euclidean distance method may retain nodes based solely on their distance to the goal, allowing them to be expanded and potentially leading to future collisions. This behavior is evident in the simulation tests, where sections of the replanned path near obstacles align with the movement of dynamic obstacles, intersecting their movement trajectory and causing collisions.

Tests 4 and 5 reveal the failure of the SATG method. Unlike the Baseline, which fails because it retains unwanted nodes based solely on Euclidean distance, SATG fails because it does not maintain sufficient distance from moving dynamic obstacles. Although SATG discards nodes that could lead to future collisions and directs the replanned path away from the moving obstacle, it does not adequately consider the relative positions and velocities of the nodes and dynamic obstacles. As a result, SATG fails to generate a path that effectively curves around dynamic obstacles.

Tests of Replan Success Rate show that DSVO has a significantly higher replan success rate compared to the other two algorithms, Baseline and SATG. While SATG has a replan success rate of 61%, the DSVO achieves a rate of 74%, which represents a 21.3% increase. The reason DSVO does not achieve an even higher replan success rate is related to the time input in the obstacle prediction model. In this model, the time input is based on the predicted duration between adjacent nodes in the global path planning algorithm. We estimate the ego vehicle will reach each node in a certain number of seconds and assume the dynamic obstacle will follow a similar timeline. However, due to limitations in Optimization and the MPC, we did not implement velocity profile planning. This means the ego vehicle does not adjust its velocity according to the reference path to ensure that the predicted and actual time durations between nodes align. This can lead to inaccurate pose estimates of dynamic obstacles, which in turn may cause the ego vehicle to fail in avoiding them. Additionally, there is a discrepancy

between the velocity and pose of the obstacle as received from ROS messages and the actual dynamic obstacle movement.

In summary, DSVO shows a significant advantage by achieving a high replan success rate and generating the shortest collision-free paths while maintaining low completion time and CPU usage. In the replanned section of DSVO, the path carefully bends away from the dynamic obstacle, adapting to its movement, while the portion farther from the obstacle takes the most direct route to the goal. This approach ensures safety by curving around the obstacle while also selecting the shortest safe route.

When a dynamic obstacle moves rapidly, DSVO accurately anticipates its trajectory, allowing the replanned path to stay close to the original route with minimal deviation. In contrast, other methods, such as Baseline or SATG, result in larger detours. DSVO's effectiveness lies in its ability to adapt to the obstacle's movement and its position relative to key waypoints, optimizing both path length and overall performance.

5-4 Conclusion

This chapter introduces the simulation setup and results for two proposed algorithms: global path planning DT Hybrid A* and replan algorithm DSVO.

Dynamic traversability enhances the Baseline Hybrid A* by improving path length and planning time, though at the cost of increased computation. DT Hybrid A* addresses this by using a kinematic bicycle model, variable travel lengths, and time-distance heuristics, reducing CPU time and generating shorter, faster paths.

DSVO achieves high replan success rates and short, safe paths while adapting efficiently to moving obstacles. Compared to the Baseline, which relies on Euclidean distance, and the SATG algorithm, it takes shorter detours, resulting in more efficient replanning.

Chapter 6

Conclusions and Future Directions

This thesis makes two significant contributions: first, the development of the Dynamic Traversability-based Hybrid A* (DT Hybrid A*) algorithm for global path planning, and second, the introduction of the Safety Direction Velocity Obstacle (DSVO) algorithm for replanning. The subsequent sections present conclusions drawn from simulation tests and discuss future research directions. These future directions address the limitations and assumptions associated with both algorithms, aiming to guide further advancements in path planning and replanning techniques.

6-1 Conclusions

The DT Hybrid A* algorithm deviates from traditional global path-planning approaches, which often prioritize safety by avoiding hard-to-traverse areas based on traversability maps. Instead, the DT Hybrid A* employs a less conservative path selection strategy. It adopts the principle that a vehicle with substantial momentum can more effectively navigate uneven terrain, including rough surfaces and steep slopes, due to its increased kinetic energy. By incorporating dynamic traversability, which takes into account the vehicle's kinetic properties such as velocity and acceleration, the dynamic traversability more accurately reflects the vehicle's ability to handle challenging terrain.

This thesis incorporates a collision detection mechanism within a global path planning framework, enabling the system to replan if a potential collision is detected in the future. The replanning process initializes the global path to avoid all static obstacles. With the movement of dynamic obstacles, if a potential conflict is recognized, the vehicle discards the current global path and replans from its current location. To ensure that the replanned path is free of dynamic obstacles, the collision detection mechanism incorporates Safety and Direction Elements based on a Velocity Obstacle (Velocity Obstacle (VO)) cost term into the global path algorithm (DSVO).

DSVO surpasses methods that use Euclidean distance as a penalty for dynamic obstacles because it accounts for the obstacle's velocity and direction over time. It also outperforms

the Safety Element based on a VO cost term, which only ensures collision avoidance within a short time span and provides the optimal velocity for that specific period, but does not account for longer-term safety. This limitation means that collisions could still occur over longer time durations. Additionally, since these are cost terms within global path planning algorithms, there may be instances where avoiding obstacles conflicts with achieving the goal.

The simulation results highlight the advantages of these two algorithms by comparing the quality of the generated paths, which are summarized as follows:

- The effectiveness of dynamic traversability is assessed by comparing the Baseline Hybrid A* algorithm with and without the dynamic traversability feature, ensuring that the algorithm's structure and parameters remain unchanged in both cases. The version of Baseline Hybrid A* that includes dynamic traversability shows improvements over the one without it, in terms of both path length and planning completion time. However, incorporating dynamic traversability results in increased computational costs. To address this drawback, the DT Hybrid A* leverages a kinematic bicycle model as its steering function, allows for variable travel lengths between adjacent nodes, and incorporates both distance and time heuristics. These enhancements significantly reduce CPU time costs, producing shorter path lengths and faster planning times.

- In typical global path planning scenarios where the route between the start and end points is relatively short and free of significant non-traversable areas, the DT Hybrid A* algorithm tends to produce the most efficient path compared to alternative methods. However, it comes with higher computational demands compared to the Baseline Hybrid A*. This suggests that even when dealing with straightforward and smooth paths, the computational overhead associated with dynamic traversability can be substantial, as evidenced by the Baseline Hybrid A*'s lower computational costs.

- Additionally, while the Baseline Hybrid A* with dynamic traversability might select more waypoints that can lead to greater elevation changes and an increased 3D path length, the DT Hybrid A* algorithm is designed to more effectively converge on the 3D shortest path. This is achieved by striking a balance between static terrain smoothness and the 2D path length, which are key factors in its cost function.

- DSVO demonstrates notable advantages by achieving a high rate of successful replans and producing the shortest collision-free paths with small computational overhead and completion time. The DSVO algorithm skillfully adjusts the path to avoid dynamic obstacles, steering away from them while ensuring the route remains as direct as possible toward the goal. This method effectively balances safety and efficiency by navigating around obstacles and maintaining a streamlined path.

- When a dynamic obstacle moves rapidly, DSVO effectively predicts its future trajectory, enabling the replanned path to remain close to the original route with minimal deviation. This is because by the time the ego vehicle follows the replanned path, the obstacle has already moved out of the area where the vehicle is traveling, reducing the need for significant path adjustments. In contrast, other methods like Baseline or SATG result in larger detours. DSVO's effectiveness lies in its ability to adapt to the obstacle's movement and its position relative to key waypoints, optimizing both path length and overall performance.

6-2 Future Directions

Building on the strengths of the DT Hybrid A* and DSVO algorithms, as well as addressing their limitations and assumptions, future research should concentrate on several key areas to further enhance these two techniques.

- More Accurate depiction of dynamic traversability: the dynamic traversability in this thesis is built on the kinematic force analysis of a car climbing on uneven terrain without considering the turning dynamics such as lateral forces, torque, and inertia. This means that dynamic traversability would only describe how easily the vehicle can travel in a straight line, without accounting for more complex maneuvers like turning or navigating curves. In the future, a more detailed dynamic force analysis could be incorporated, allowing the traversability map to be represented as a transfer function that accounts for factors such as initial velocity, longitudinal and lateral acceleration, turning radius and varying heading directions. With this approach, the traversability can be dynamically adjusted based on the initial pose and the target pose, considering different heading directions, turning radius and motion dynamics. As a result, the traversability assessment would more accurately reflect the difficulties of reaching a target location through various motion strategies.
- Considering different dynamic obstacle movement strategies, this thesis assumes that dynamic obstacles move in a straight line. In future work, various movement patterns could be explored, and the dynamic avoidance method, DSVO, could be validated for effectiveness or further optimized as needed like including detailed terrain information and related uncertainty.
- Future work could focus on advancing the MPC framework by integrating detailed terrain information and incorporating additional costs such as rollover risk and bump costs [10]. This enhancement aims to produce smoother and safer control inputs, ultimately enabling the generation of feasible and reliable motion plans for the vehicle.

Appendix A

The Back of the Thesis

A-1 Maximum Inclination Angle in Downhill Case

$$M\frac{\mathrm{d}v}{\mathrm{d}t} = \frac{P}{v} + Mg\sin(\varphi_x) - \mu Mg\cos(\varphi_x) \tag{A-1}$$

- When $a = a_{max}$, the (A-1) can be represented as:

$$\begin{aligned}
M\frac{\mathrm{d}v}{\mathrm{d}t} =& Ma_{max} = \frac{P}{v} + Mg\sin(\varphi_x) - \mu Mg\cos(\varphi_x)\\
\Rightarrow& Ma_{max} = \frac{P}{v} + Mg\sin(\varphi_x) - \mu Mg\cos(\varphi_x)\\
\Rightarrow& \frac{P}{v} - Ma_{max} = Mg(-\sin(\varphi_x) + \mu\cos(\varphi_x))\\
\Rightarrow& \frac{1}{\sqrt{1+\mu^2}}\left(\frac{P}{Mgv} - \frac{a_{max}}{g}\right) = \cos\left(\varphi_x + \gamma\right),\\
& \tan\left(\gamma\right) = \frac{1}{\mu}\\
\Rightarrow& \varphi_x = \arccos\left(\frac{1}{\sqrt{1+\mu^2}}\left(\frac{P}{Mgv} - \frac{a_{max}}{g}\right)\right) - \arctan\left(\frac{1}{\mu}\right)
\end{aligned}$$

The maximum inclination angle determined by the maximum motion power of the vehicle in this scenario can be represented as

$$\phi_f = \varphi_x = \arccos\left(\frac{1}{\sqrt{1+\mu^2}}\left(\frac{P}{Mgv} - \frac{a_{max}}{g}\right)\right) - \arctan\left(\frac{1}{\mu}\right)$$

using maximum vehicle power P_{max}.

- When $a = 0$, (A-1) can be represented as:

$$\begin{aligned} M\frac{\mathrm{d}v}{\mathrm{d}t} =& M \cdot 0 = \frac{P}{v} + Mg\sin(\varphi_x) - \mu Mg\cos(\varphi_x) \\ \Rightarrow& \frac{P}{v} + Mg\sin(\varphi_x) - \mu Mg\cos(\varphi_x) = 0 \\ \Rightarrow& \frac{P}{v} = Mg\left(-\sin(\varphi_x) + \mu\cos(\varphi_x)\right) \\ \Rightarrow& \frac{P}{Mgv} = \sqrt{1+\mu^2}\cos\left(\varphi_x + \gamma\right), \quad \tan\left(\gamma\right) = \frac{1}{\mu} \\ \Rightarrow& \varphi_x = \arccos\left(\frac{P}{\sqrt{1+\mu^2}Mgv}\right) - \arctan\left(\gamma\right) \end{aligned}$$

  The maximum inclination angle determined by the maximum motion power of the vehicle in this scenario can be represented as

$$\phi_f = \varphi_x = \arccos\left(\frac{P}{\sqrt{1+\mu^2}Mgv}\right) - \arctan\left(\gamma\right)$$

  with maximum vehicle power P_{max}.

- When $a = -a_{max}$, (A-1) can be represented as:

$$\begin{aligned} M\frac{\mathrm{d}v}{\mathrm{d}t} =& -Ma_{max} = \frac{P}{v} + Mg\sin(\varphi_x) - \mu Mg\cos(\varphi_x) \\ \Rightarrow& \frac{P}{v} + Ma_{max} = Mg(-\sin(\varphi_x) + \mu\cos(\varphi_x)) \\ \Rightarrow& \frac{1}{\sqrt{1+\mu^2}}\left(\frac{P}{Mgv} + \frac{a_{max}}{g}\right) = \cos\left(\varphi_x + \gamma\right), \\ & \tan\left(\gamma\right) = \frac{1}{\mu} \\ \Rightarrow& \varphi_x = \arccos\left(\frac{1}{\sqrt{1+\mu^2}}\left(\frac{P}{Mgv} + \frac{a_{max}}{g}\right)\right) - \arctan\left(\gamma\right) \end{aligned}$$

  The maximum inclination angle determined by the maximum motion power of the vehicle in this scenario can be represented as

$$\phi_f = \arccos\left(\frac{1}{\sqrt{1+\mu^2}}\left(\frac{P_{max}}{Mgv} + \frac{a_{max}}{g}\right)\right) - \arctan\left(\gamma\right)$$

  with maximum vehicle power P_{max}.

Bibliography

[1] Roberto Andreani, Alberto Ramos, Ademir A Ribeiro, Leonardo D Secchin, and Ariel R Velazco. On the convergence of augmented Lagrangian strategies for nonlinear programming. *IMA Journal of Numerical Analysis*, 42(2):1735–1765, 04 2021.

[2] Serge D'Alessio. A mathematical model of the velocity of a cyclist riding over uneven terrain. *European Journal of Physics*, 42(2):025804, 2021.

[3] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Practical search techniques in path planning for autonomous driving. *International Symposium on Combinatorial Search, SoCS 2008*, 2008.

[4] Dmitry Dolgov, Sebastian Thrun, Michael Montemerlo, and Joshua Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. *The International Journal of Robotics Research*, 29(5):485–501, 2010.

[5] Lester E. 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.

[6] Pedro F. Felzenszwalb. Distance transforms of sampled functions. *Theory of Computing*, 8(1):415–428, 2012.

[7] Nuwan Ganganath, Chi-Tsun Cheng, and Chi K. Tse. Multiobjective path planning on uneven terrains based on NAMOA. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1846–1849, 2016.

[8] Fei Gao, William Wu, Wenliang Gao, and Shaojie Shen. Flying on point clouds: Online trajectory generation and autonomous navigation for quadrotors in cluttered environments. *Journal of Field Robotics*, 36(4):710–733, 2019.

[9] Zoltán Gyenes, Barnabás Pajkos, Ladislau Bölöni, and Emese Gincsainé Szádeczky-Kardoss. Motion planning for mobile robots using uncertain obstacle estimation. *IEEE Access*, 12:16856–16867, 2024.

[10] Tyler Han, Alex Liu, Anqi Li, Alex Spitzer, Guanya Shi, and Byron Boots. Model predictive control for aggressive driving over uneven terrain. *arXiv preprint arXiv:2311.12284*, 2024. Accepted to R:SS 2024.

[11] Taylor A. Howell, Brian E. Jackson, and Zachary Manchester. ALTRO: A fast solver for constrained trajectory optimization. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7674–7679, 2019.

[12] Jiaming Hu, Yuhui Hu, Chao Lu, Jianwei Gong, and Huiyan Chen. Integrated path planning for unmanned differential steering vehicles in off-road environment with 3D terrains and obstacles. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):5562–5572, 2022.

[13] Zhuozhu Jian, Zihong Lu, Xiao Zhou, Bin Lan, Anxing Xiao, Xueqian Wang, and Bin Liang. PUTN: A plane-fitting based uneven terrain navigation framework. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7160–7166, 2022.

[14] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[15] Inyoung Ko, Beobkyoon Kim, and Frank Chongwoo Park. VF-RRT: Introducing optimization into randomized motion planning. *2013 9th Asian Control Conference, ASCC*, 2013.

[16] Shu-peng Lai, Meng-lu Lan, Ya-xuan Li, and Ben Chen. Safe navigation of quadrotors with jerk limited trajectory. *Frontiers of Information Technology Electronic Engineering*, 20:107–119, 01 2019.

[17] Chuan-Che Lee and Kai-Tai Song. Path re-planning design of a cobot in a dynamic environment based on current obstacle configuration. *IEEE Robotics and Automation Letters*, 8(3):1183 – 1190, 2023.

[18] Changliu Liu, Chung-Yen Lin, and Masayoshi Tomizuka. The convex feasible set algorithm for real time optimization in motion planning. *SIAM Journal on Control and Optimization*, 56(4):2712 – 2733, 2018.

[19] Jiayang Liu, Xieyuanli Chen, Junhao Xiao, Sichao Lin, Zhiqiang Zheng, and Huimin Lu. Hybrid map-based path planning for robot navigation in unstructured environments. *arXiv preprint arXiv:2303.05304*, 2023.

[20] Shang Liu, Guoqing Zhang, Ge Guo, and Jiqiang Li. Adaptive output-feedback event-triggered formation control for underactuated ships with obstacle avoidance mechanism. *Ocean Engineering*, 299, 2024.

[21] Carlos Mastalli, Rohan Budhiraja, Wolfgang Merkt, Guilhem Saurel, Bilal Hammoud, Maximilien Naveau, Justin Carpentier, Ludovic Righetti, Sethu Vijayakumar, and Nicolas Mansard. Crocoddyl: An efficient and versatile framework for multi-contact optimal control. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2536–2542, 2020.

[22] Stephen McCrory, Bhavyansh Mishra, Jaehoon An, Robert Griffin, Jerry Pratt, and Hakki Erhan Sevil. Humanoid path planning over rough terrain using traversability assessment. *arXiv preprint arXiv:2203.00602*, 2022.

[23] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *Proc. IEEE Int. Conf. Robot. Autom.*, pages 2520–2525, 2011.

[24] J-P. Merlet. A local motion planner for closed-loop robots. *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3088–3093, 2007.

[25] Heramb Nemlekar, Ziang Liu, Suraj Kothawade, Sherdil Niyaz, Barath Raghavan, and Stefanos Nikolaidis. Robotic lime picking by considering leaves as permeable obstacles. *IEEE International Conference on Intelligent Robots and Systems*, page 3278 – 3284, 2021.

[26] Helen Oleynikova, Michael Burri, Zachary Taylor, Juan Nieto, Roland Siegwart, and Enric Galceran. Continuous-time trajectory optimization for online UAV replanning. *IEEE International Conference on Intelligent Robots and Systems*, 2016-November:5332 – 5339, 2016.

[27] Helen Oleynikova, Zachary Taylor, Roland Siegwart, and Juan Nieto. Safe local exploration for replanning in cluttered unknown environments for microaerial vehicles. *IEEE Robotics and Automation Letters*, 3(3):1474 – 1481, 2018.

[28] Timothy Overbye and Srikanth Saripalli. Fast local planning and mapping in unknown off-road terrain. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5912–5918, 2020.

[29] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55, 2016.

[30] Gianfranco Parlangeli, Daniela De Palma, and Rossella Attanasi. A novel approach for 3PDP and real-time via point path planning of dubins' vehicles in marine applications. *Control Engineering Practice*, 144:105814, 2024.

[31] Mark Pauly, Markus Gross, and Leif P. Kobbelt. Efficient simplification of point-sampled surfaces. In *IEEE Visualization, 2002. VIS 2002.*, pages 163–170, 2002.

[32] Chao-Chung Peng and Wang Cheng-Yu. Design of constrained dynamic path planning algorithms in large-scale 3D point cloud maps for UAVs. *Journal of Computational Science*, 67, 2023.

[33] Yao Qi, Binbing He, Rendong Wang, Le Wang, and Youchun Xu. Hierarchical motion planning for autonomous vehicles in unstructured dynamic environments. *IEEE Robotics and Automation Letters*, 8(2):496–503, 2023.

[34] Ahmed Hussain Qureshi and Yasar Ayaz. Potential functions based sampling heuristic for optimal path planning. *Autonomous Robots*, 40(6):1079 – 1093, 2016.

[35] Rajesh Rajamani. *Vehicle Dynamics and Control*. Springer, Boston, MA, 2nd edition, 2011.

[36] Charles Richter, Adam Bry, and Nicholas Roy. *Polynomial Trajectory Planning for Aggressive Quadrotor Flight in Dense Indoor Environments*, pages 649–666. Springer International Publishing, Cham, 2016.

[37] Gauthier Rousseau, Cristina Stoica Maniu, Sihem Tebbani, Mathieu Babel, and Nicolas Martin. Minimum-time b-spline trajectories with corridor constraints. application to cinematographic quadrotor flight plans. *Control Engineering Practice*, 89:190 – 203, 2019.

[38] N.C. Rowe and R.S. Ross. Optimal grid-free path planning across arbitrarily contoured terrain with anisotropic friction and gravity effects. *IEEE Transactions on Robotics and Automation*, 6(5):540–553, 1990.

[39] Lander Vanroye, Ajay Sathya, Joris De Schutter, and Wilm Decré. FATROP: A fast constrained optimal control problem solver for robot trajectory optimization and control. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10036–10043, 2023.

[40] Robin Verschueren, Gianluca Frison, Dimitris Kouzoupis, Jonathan Frey, Niels van Duijkeren, Andrea Zanelli, Branimir Novoselnik, Thivaharan Albin, Rien Quirynen, and Moritz Diehl. ACADOS—a modular open-source framework for fast embedded optimal control. *Mathematical Programming Computation*, 14(1):147–183, 2022.

[41] Jingping Wang, Long Xu, Haoran Fu, Zehui Meng, Chao Xu, Yanjun Cao, Ximin Lyu, and Fei Gao. Towards efficient trajectory generation for ground robots beyond 2D environment. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7858–7864, 2023.

[42] Yinghan Wang, Jianmin Dong, Yanan Wang, and Bingyang Sun. Multi-view 3D human pose estimation based on multi-scale feature by orthogonal projection. *E3S Web of Conferences*, 522, 2024.

[43] Jian Wen, Xuebo Zhang, Haiming Gao, Jing Yuan, and Yongchun Fang. E3MoP: Efficient motion planning based on heuristic-guided motion primitives pruning and path optimization with sparse-banded structure. *IEEE Transactions on Automation Science and Engineering*, 19(4):2762–2775, 2022.

[44] Tingting Xie, Hui Chen, Wanquan Liu, Rongyu Zhou, and Qilin Li. 3D surface segmentation from point clouds via quadric fits based on DBSCAN clustering. *Pattern Recognition*, 154, 2024.

[45] Jingren Xu, Yukiyasu Domae, Weiwei Wan, and Kensuke Harada. An optimization-based motion planner for a mobile manipulator to perform tasks during the motion. *2022 IEEE/SICE International Symposium on System Integration, SII 2022*, page 519 – 524, 2022.

[46] Jingren Xu, Yukiyasu Domae, Weiwei Wan, and Kensuke Harada. An optimization-based motion planner for a mobile manipulator to perform tasks during the motion. In *2022 IEEE/SICE International Symposium on System Integration (SII)*, pages 519–524, 2022.

[47] Long Xu, Kaixin Chai, Zhichao Han, Hong Liu, Chao Xu, Yanjun Cao, and Fei Gao. An efficient trajectory planner for car-like robots on uneven terrain. *arXiv:2309.06115*, Sep 2023.

[48] Hongji Yang, Qingzhong Jia, and Weizhong Zhang. An environmental potential field based RRT algorithm for UAV path planning. *Chinese Control Conference, CCC*, 2018-July:9922 – 9927, 2018.

[49] Jingze Zhong, Mengjie Zhang, Zonghai Chen, and Jikai Wang. Dynamic obstacle avoidance for mobile robots based on 2D differential euclidean signed distance field maps in park environment. *World Electric Vehicle Journal*, 15(7), 2024.

Glossary

List of Acronyms

ESDF	Euclidean Signed Distance Functions
OCP	Nonlinear optimal control problems
SQP	Sequential Quadratic Programming
DDP	Differential Dynamic Programming
MPC	Model Predictive Control
iLQR	iterative Linear-Quadratic Regulator
FDDP	Feasibility-driven Differential Dynamic Programming
VO	Velocity Obstacle