

参赛队员姓名：Martin Xiao Yu Su、章晨蔚

中学：广东华南师范大学附属中学

省份：广东省

国家/地区：中国

指导教师姓名：高英教授、桂鹏老师

指导教师单位：华南理工大学计算机科学与工程学院、广东华南师范大学附属中学

论文题目：NavFlow-VL: Memory-Augmented
Dual-System UAV Planner with Vision-
Language Guided Flow-Matching Trajectory
Generation

NavFlow-VL: Memory-Augmented Dual-System UAV Planner with Vision-Language Guided Flow-Matching Trajectory Generation

Martin Xiao Yu Su, Chenwei Zhang

Affiliated High School of South China Normal University

Email: {suxy.martin, zhangcw.vivian}2025@gdhfi.com

Abstract

Driven by the rapid development of robotic intelligence, Unmanned Aerial Vehicles (UAVs) are widely adopted in diverse practical tasks due to their superior agility and flexible deployment. They greatly optimize traditional operation modes and boost the overall efficiency of intelligent sensing and decision-making. However, most existing UAVs still depend heavily on manual remote control. Human intervention brings extra labor costs and latent flight safety risks. Operators cannot fully perceive surrounding environments, so hidden obstacles and sudden hazards are easily overlooked, resulting in delayed responses and navigation failures. In addition, limited onboard sensing coverage further restricts environmental perception, making UAVs hard to adapt to cluttered maze scenes and dynamic obstacle environments. To solve these practical defects, this work proposes a robust learning-based UAV navigation framework, aiming to achieve stable dead-end escape and safe autonomous navigation in complex static and dynamic scenarios.

- This work builds a tailored **closed-loop simulation platform** and **dataset pipeline** for UAV **dead-end escape** and **dynamic obstacle avoidance**. Different from open environments, maze scenarios require strong global perception and long-term environmental memory, which our system fully supports. It also resolves the common simulation defect of **transparent obstacle penetration**.
- We propose a **fast-slow dual-system navigation framework** for UAV navigation in complex maze environments. The high-frequency fast subsystem employs a CNN-based trajectory prediction network for responsive local adjustment and dynamic-obstacle avoidance. The low-frequency slow subsystem is implemented as a **task-specific vision-language model** with approximately 0.5 billion trainable parameters. It jointly processes five-frame multimodal observation history, recurrent memory tokens, the UAV state, and the navigation goal. A constrained text decoder generates either Dead End or Normal, while a conditional **Flow Matching** decoder predicts 30 future waypoints over a planning horizon of 3 s.
- We adopt **ESDF geometric constraints** to build a continuous safety supervision mechanism in high-frequency system and improve the generalization ability of learning-based planners. Unlike conventional imitation learning that heavily relies on expert trajectories and fails easily in new scenes, our constrained framework maintains stable robustness in diverse complex environments.
- We introduce an **auxiliary learning task** for the VLM and Flow Matching decoder to reduce the tendency of the model to become trapped in local optima and improve its dead-end escape capability. In addition, a **simplified semantic annotation strategy** provides scene-level supervision for the vision-language module, helping the model make better use of historical environmental information and identify and recover from trapped states in maze environments.

This project is made open-source on: <https://github.com/martinsu20100623-netizen/VLMBasedE2E-UAV-Planner>.

Keywords: *Unmanned Aerial Vehicle, Euclidean Signed Distance Field, Flow-Matching, Vision-Language-Action, ROS, Gazebo, Motion Primitive, End-to-End Learning Based Trajectory Planner*

NavFlow-VL: 基于记忆增强的快-慢双系统视觉语言引导的 Flow Matching 无人机轨迹规划算法

Martin Xiao Yu Su, 章晨蔚
广东华南师范大学附属中学

{suxy.martin, zhangcw.vivian}2025@gdhfi.com

摘要

随着机器人智能技术的高速发展，无人机（UAV）凭借其优异的机动灵活性与部署便捷性，已广泛应用于各类实际作业场景，大幅优化了传统作业模式，提升了智能感知与决策体系的整体运行效率。然而，现有绝大多数无人机作业仍高度依赖人工远程操控，人工干预不仅增加了人力成本，还带来了潜在的飞行安全隐患。操作人员无法全方位感知周边环境，易忽略隐蔽障碍物与突发危险场景，进而引发响应滞后、导航失效等问题。同时，无人机机载感知设备的探测范围有限，环境感知能力受限，难以适配复杂迷宫场景与动态障碍物环境下的自主作业需求。针对上述现存技术缺陷，本文提出一种鲁棒性强、基于学习的无人机导航框架，旨在实现无人机在静态、动态复杂场景下的稳定死区逃逸（Dead-End Escape）与安全自主导航，本文的核心创新点如下：

- 构建适配无人机脱困逃逸与动态避障任务的闭环仿真平台与数据集 pipeline。相较于开阔环境，迷宫场景对系统的全局感知能力与长时环境记忆能力要求极高，所搭建平台可充分满足该需求，同时解决了传统仿真环境中障碍物透明穿透的共性仿真缺陷，保障仿真实验的真实性与可靠性。
- 提出一种面向复杂迷宫环境的快-慢双系统（Fast-Slow Dual-System）导航框架。高频快速子系统采用基于卷积神经网络（CNN）的轨迹预测网络，实现局部路径实时调整与动态障碍物规避；低频慢速子系统则构建为一个约 5 亿参数规模的任务专用视觉语言模型（Vision-Language Model, VLM），处理五帧历史多模态观测、循环记忆 Token、无人机状态以及导航目标信息。其受约束的文本解码器输出 Dead End 或 Normal 等场景状态判断，而条件 Flow Matching 轨迹解码器则生成未来 3 秒内共 30 个预测航点，为无人机提供全局导航决策
- 引入基于 ESDF（Euclidean Signed Distance Field）的几何约束机制，构建连续安全监督框架。该机制嵌入高频规划系统中，有效提升学习型规划器的泛化能力。与严重依赖专家轨迹、在未知场景中容易失效的传统模仿学习方法相比，所提出的约束学习框架能够在多样化复杂环境中保持稳定的鲁棒性和安全性。
- 设计面向视觉语言模型与 Flow Matching 解码器的辅助学习任务。该机制能够降低模型陷入局部最优策略的倾向，增强其死区识别与逃逸能力。同时，本文提出一种轻量级语义标注策略，为视觉语言模块提供场景级监督信息，帮助模型更有效地利用历史环境记忆，在迷宫环境中准确识别受困状态并完成恢复导航。

该项目代码发布在 <https://github.com/martinsu20100623-netizen/VLMBasedE2E-UAV-Planner>。

关键词：无人机, 欧式符号距离场, Flow-Matching, Vision-Language-Action, ROS, Gazebo, Motion Primitive, 端到端轨迹规划器

Contents

1	Introduction	3
2	Related Work	5
2.1	Classification and Characteristics of Typical UAV Platforms	5
2.2	Traditional Kinodynamic Trajectory Planning Algorithms	7
2.2.1	Heuristic Global Search Based on A* Algorithm	7
2.2.2	Sampling-Based Exploration with RRT Algorithm	8
2.2.3	Dynamically Feasible Path Search via Hybrid A*	8
2.2.4	Kinodynamic RRT Algorithm	9
2.3	Reactive Based Trajectory Planning	9
2.4	End2End Learning Based Trajectory Planning	10
2.5	Benchmark Analysis of Open-Source Simulation Ecosystems	12
3	Simulation Environment	15
3.1	Complex Scene Map Construction for UAV Navigation	16
3.1.1	Structured Dead-Zone Maze Environment	16
3.1.2	Unstructured Forest Obstacle Environment	17
3.1.3	Dynamic Obstacle-Embedded Maze Environment	17
3.2	Closed-Loop Dataset Generation Pipeline for High-Frequency System	18
3.3	VLM Training Data Generation with Simplified Semantic Annotation	19
4	Methodology	22
4.1	Motion Primitive Sampling and Trajectory Optimization	22
4.2	Flow-Matching Based Trajectory Generation	25
4.3	Continuous ESDF Map Representation for UAV Obstacle Avoidance	27
4.4	Dual-Frequency Inference Pipeline	29
5	Experiments	31
5.1	Experiment Environment	31
5.2	Evaluating NavFlow-VL on Maze Environment	32
5.3	Evaluating NavFlow-VL on Forest Dynamic Obstacle Environment	35
5.4	Evaluating NavFlow-VL on Maze Dynamic Obstacle Environment	37
6	Conclusion	39
	References	41
	Acknowledgments	44
	Appendix	46

1 Introduction

UAV technology has been widely applied in practical scenarios such as emergency search and rescue, intelligent warehouse logistics, infrastructure routine inspection, disaster field disposal and even extraterrestrial exploration. In most real-world tasks, drones often need to work in unstructured, dense and dynamically changing environments with incomplete environmental information. Although manual remote control can provide basic high-level operation guidance, heavy reliance on human intervention greatly limits the operational efficiency and large-scale application of UAVs. In addition, affected by limited sensor detection range, signal delay and visual blind areas, human operators can hardly grasp full environmental details in real time. This makes it difficult to make accurate and timely judgments during complex navigation missions. Accordingly, equipping UAVs with the ability of autonomous perception, environmental reasoning and intelligent navigation in complex scenes has become a core research focus in the field of aerial robotics.

As one of the most essential core capabilities of UAV autonomous flight, intelligent navigation has always been a key and difficult research topic. A practical and stable navigation system needs to not only avoid surrounding obstacles in real time, but also continuously predict feasible flight paths to ensure the completion of long-term mission goals. This task becomes extremely challenging in maze-like narrow and enclosed scenarios. In such complex environments, local sensor observation information is too limited to support global optimal decision-making. UAVs are prone to repeatedly hitting dead ends, being trapped in fixed local areas, or failing to record and revisit previously accessible routes. Moreover, randomly moving obstacles bring uncertain interference to the flight process, requiring the navigation system to take both long-term global planning and instant emergency obstacle avoidance into account. For this reason, building a new UAV navigation system that integrates environmental understanding, spatial memory, global logical reasoning and real-time dynamic response capability is of great research significance for upgrading the next generation of autonomous drone platforms.

According to Y. Tian et al.[1], there are different models in the UAV system, which work and achieve certain target together. To begin with, the sensor model is used to collect and analyze data from different sensors, aiming to gain a comprehensive understand of the surrounding environments. This kind of sensors include RGB Camera, Event Camera, Thermal Imagers, 3D Cameras, LiDAR, radar and ultrasonic sensors, etc. In addition, the navigation model is used to turn organized tracks into accurate flying paths, which is achieved through sequential estimation and changing UAVs' position, direction, and speed. This model relies on multiple sensors, for example GPS, IMU, visual odometry and barometer, sensor fusion algorithms are used to enhance the reliability and accuracy of state estimation. Furthermore, the next model is the organization model It is responsible for turning the project target into direct flying tracks and movements, relying on the data from the input sensors to ensure safety. Covering heuristic algorithms and machine learning methods are used to calculate feasible routes and optimize them in this model. Moreover, the control model is used to generate direct orders to control the UAVs' actuators, in order to maintain stability and flying. Finally, the communication model supports the UAVs to communicate with ground control station, satellites and other external entities, ensuring the transportation of essential information. Nowadays, UAVs have been deployed in numerous fields and diverse real-world scenarios. The typical application cases are summarized below:

Monitoring: This is one of the important fields that UAVs are used in, which includes traffic situations, city environments, and supervisory tasks. Traditional monitoring system mainly relied on machine learning techniques. However, using FMs, especially LLMs and VLMs, can significantly improve the UAVs' environmental perception ability and task completion efficiency. **Material Flow:** UAVs can achieve the intelligence of the logistics chain, including decision making, paths organization, and final

delivery. The use of FMs improves the planning of the paths, the experience of human-computer interaction, and the organization of supporting lines. **Emergency Response:** Because of their quick adaptations towards most of the emergency situations, UAVs have advantages in emergency response and disaster assistance projects. Combining FMs can enable the UAVs to make decisions quicker and analyze complex data, leading to further enhance the real-time decision-making capabilities and task execution efficiency of UAVs.

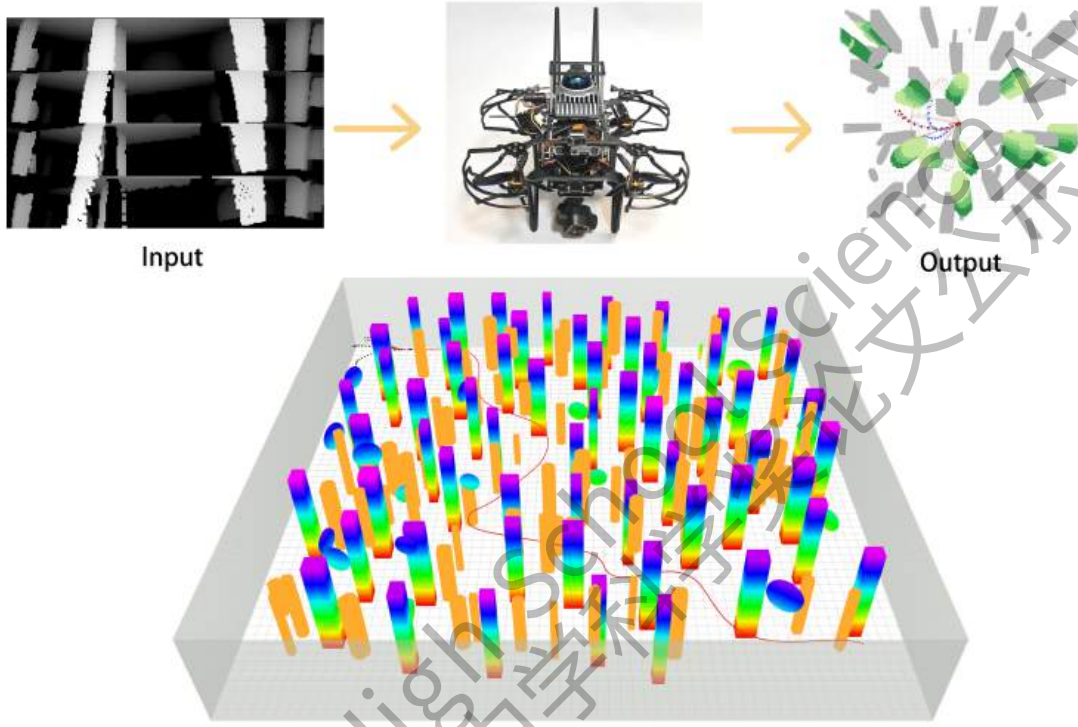


Figure 1: UAV Training in an Obstacle Course

Currently, there is no unified framework for implementing dead zone escape and dynamic, complex scenarios. To make up for this deficiency, we need to design a learning-based map navigation algorithm that can handle both static and dynamic obstacles and maze-like conditions. To strengthen model capability, we built a close-loop simulation environment and a train dataset named pipeline to handle dead-zone escape and avoid dynamic obstacles. For the maze-like training field, which is different to the open environment with static obstacles such as forest, this unique layout requires the model to gain memory and a global perspective, our main contributions can be summarized as follows:

- A dedicated simulation benchmark and dataset generation pipeline were developed for evaluating autonomous navigation under dead-end escape and dynamic obstacle avoidance scenarios. Unlike conventional navigation benchmarks that mainly focus on local collision avoidance, the constructed maze-based environments impose additional requirements on long-horizon decision making, environmental memorization, and route backtracking. The proposed platform provides a complete closed-loop framework for training and evaluation while eliminating unrealistic interactions caused by obstacle penetration.
- This study designs a layered dual-module navigation structure that connects real-time motion control and global path decision-making. Drawing on the operating mechanism of human and biological rapid response and slow logical thinking, this framework organically combines short-range

local trajectory generation with long-distance global path planning. This design allows UAVs to move flexibly and efficiently in dense and complex maze scenes. Meanwhile, it enables the drone to actively get out of trapped dead areas and adaptively cope with real-time dynamic changes in the surrounding environment.

- To make the learning-based navigation model more robust and adaptable for actual UAV deployment, we add safety constraint supervision during the model training process based on geometric environmental information. Traditional imitation learning methods overly depend on high-quality expert demonstration data and are prone to performance degradation due to data distribution differences when facing unfamiliar new scenarios. In contrast, our improved framework shows better environmental adaptability and generalization performance, which can stably adapt to various complex navigation scenes.
- We develop a history- and memory-aware semantic learning strategy for long-horizon navigation and dead-end recovery. The slow subsystem processes five consecutive multimodal observations with temporal embeddings and maintains recurrent memory tokens across planning cycles. A constrained text decoder is trained to generate one of two navigation-state descriptions, Dead End or Normal. This auxiliary text-generation objective provides an interpretable semantic signal for the gating module and helps the model maintain a consistent escape intention after previously observed structures leave the current sensing field.

The rest of the paper is organized as follows: Section 2 reviews the related work on UAV platforms and trajectory planning. Section 3 describes the constructed simulation environment, map settings and data generation pipeline. Section 4 elaborates the overall methodology of our proposed framework. Section 5 presents experimental results across different scenarios. Finally, Section 6 concludes the paper.

2 Related Work

This chapter sorts out and summarizes previous research progress in the field of UAV autonomous navigation. We start by introducing the common features and practical application scenarios of mainstream UAV types, including fixed-wing, multi-rotor, and flapping-wing aircraft. After clarifying the characteristics of different flight platforms, we further organize and compare existing mainstream navigation and planning techniques, covering traditional trajectory planning strategies, heuristic search methods, real-time local obstacle avoidance algorithms, and modern end-to-end learning-based planning frameworks. In addition, we investigate several widely adopted open-source simulation environments for UAV verification. Through comprehensive literature sorting and experimental comparison, we find that current navigation methods still face obvious limitations. Most existing approaches lack stable generalization ability, perform poorly in dynamic changing scenes, and cannot effectively handle complex maze navigation and dead-end escape tasks.

2.1 Classification and Characteristics of Typical UAV Platforms

UAV come in various configurations, each with its unique geometry and application scenarios. Multirotor UAVs constitute the vast majority of civilian UAVs, and have many advantages such as the ability to hovering precisely and the ability to perform delicate maneuvers that cannot be adapted in fixed-wing and flapping-wing UAVs, making them an ideal form for applying algorithms.

Fixed-wing UAVs: As shown in Fig.2, use a fixed airfoil, achieving takeoff and flight by generating propulsion driven by engines and electric motors. According to G. Cai et al.[2] unlike multirotar UAVs that rely on rotor lift, fixed-wing designs depend on forward airspeed to create aerodynamic lift over the

wing surface. Compared to other types of drones—such as quadcopters and hexacopters—fixed-wing UAVs are generally faster, have longer endurance, and are more stable in cruising flight. These characteristics make them suitable for long-duration missions, including large-scale mapping, border surveillance, agricultural crop monitoring, and atmospheric data collection. However, fixed-wing UAVs are the only type of drone that cannot take off and land vertically or hover in place. Similar to commercial airplanes, fixed-wing UAVs also require more space for takeoff and landing—typically a runway. Additionally, due to their reliance on aerodynamic control, they have a larger turning radius, and this larger turning radius demands higher flight skills from the operator or autopilot system, especially during low-altitude maneuvers or in confined environments. Consequently, fixed-wing UAVs are relatively less maneuverable than other types of drones.



Figure 2: Fixed-Wing Drone (Left) Multirotor Drone (Right)

Multirotor UAVs: A multirotor drone is an unmanned aerial vehicle with three or more rotor shafts. In terms of configuration, multirotors typically have an even number of rotors to counteract spin.[3] The most common configuration is a quadcopter, but hexacopter and octacopter configurations also exist. Multirotor UAVs do not rely on complex mechanical structures for flight. Instead, they achieve flight by adjusting the rotational speed of each propeller through the flight control system inside the fuselage. As the propellers rotate, they exert a torsional force in the opposite direction. Therefore, multirotors must ensure that its adjacent propellers have to rotate in opposite directions to counteract this force. This is why multirotor UAVs often have an even number of rotors. Compared to fixed-wing or flapping-wing UAVs, the advantage of multirotor aircraft is their control flexibility and extremely low barrier to entry. The characteristic of multirotor UAVs are small and simple, which makes them easy to maintain and hence with a low accident rate. They can also take off and land stably and vertically, requiring little space, thus becoming the first choice for many drone users.

Flapping-wing UAVs: The wing of Flapping-wing UAVs, unlike fixed-wing UAVs, can actively twist and bend during flapping to dynamically adjust aerodynamic performance.[4] During flight, the wings flapping upward and downward periodically in a short time, generating lift and thrust simultaneously. Its main advantage is that it produce little noise during flapping, and at a micro scale, its aerodynamic efficiency is better than that of rotors and fixed wings. Its appearance and flapping motion are also highly similar to those of real birds or insects, making it difficult to distinguish with the naked eye from hundreds of meters away. The combination of low noise, high efficiency, and visual similarity gives flapping-wing UAVs a unique advantage in covert operations where detection by sight or sound must be avoided. However, The flight of flapping-wing UAVs involves unsteady aerodynamics.

During flapping, the wings continuously generate vortices. Therefore, even small changes in the flapping frequency, amplitude, and torsional angle can lead to drastic changes in flight status, making them particularly prone to loss of control, especially when hovering or flying at low speeds.

2.2 Traditional Kinodynamic Trajectory Planning Algorithms

2.2.1 Heuristic Global Search Based on A* Algorithm

The A* algorithm is a heuristic search algorithm commonly used in path planning and graph traversal, developed by P.E.Hart et al.[5] in 1968. The A* algorithm combines the characteristics of Dijkstra's algorithm and the greedy algorithm, ensuring effective finding of the shortest path while also improving search efficiency through the greedy function. Compare to Dijkstra algorithm, A* algorithm can output a reasonable path in a short time, hence A* algorithm is better than Dijkstra in Computational efficiency. This was demonstrated in experiments by Iqbal et al [6].With this approach, the A* algorithm, during runtime, does not need to traverse the entire map like BFS, making it faster. In addition, it can guarantee that the path found is optimal under certain conditions, and the search process is traceable. The A* algorithm is widely applied in game AI, robot navigation, map navigation, and other scenarios. Fan et al. [7] introduced an enhanced use of the A* algorithm for civilian UAV path planning under 3G communication constraints. Unlike traditional distance-optimal A* applications, their method prioritizes signal quality by avoiding base station dead ends, achieving 2.44 times better communication quality with only 1.56 times longer flight distance—demonstrating A*'s flexibility in multi-objective scenarios.

The core of the A* algorithm is the evaluation function, which is the weight of all points along its path. When using A* to traverse a graph, the algorithm marks points connected to already traversed points with the current cost, which is the number of steps from the starting point to this point, and the estimated cost, which is the Manhattan distance from this point to the endpoint. By traversing points one by one, it finds the point with the minimum sum of current cost and estimated cost and then determines the next point to traverse. The macroscopic manifestation of this process is shown in Fig. 3. However, it has considerable drawbacks: since each iteration calculates all weights of all points adjacent to the traversed points, it consumes a lot of memory, has slow computation speed, and is prone to fragmentation. In practical applications, the optimal solution given by the A* algorithm may not always be smooth, sometimes the route generated by the traditional A* algorithm is constrained by the resolution of the map, just as Song et al. [8] mentioned. This defect will be magnified when a path requires sharp turns for a drone.

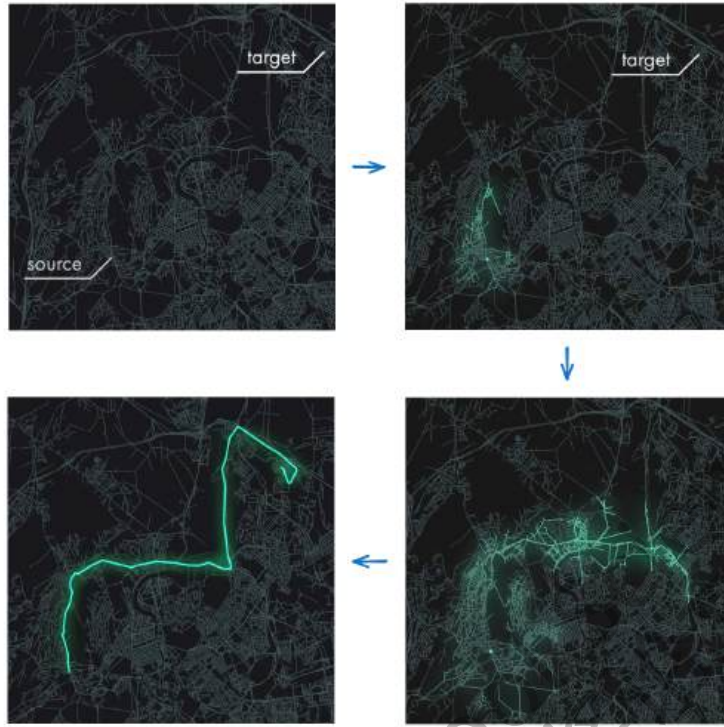


Figure 3: A Algorithm Pathfinding Graph*

2.2.2 Sampling-Based Exploration with RRT Algorithm

In 2001, S.LaValle et al.[9] developed the Rapidly-Exploring Random Tree Algorithm (RRT) algorithm, also known as Rapidly-exploring Random Tree, which is a sampling-based path planning algorithm. It is commonly used in robots, drones, autonomous driving, and in extended fields such as high-dimensional space pathfinding. It generates a continuously expanding tree structure through random sampling, covering the search space and finding a path from the starting point to the endpoint.

The RRT algorithm involves three points: near, next, and rand. At the beginning of each traversal iteration, a random point is selected in the search space, called rand. Then, all previously traversed points, namely the starting point and the points obtained in each iteration (next), are considered to find the point closest to rand, called near. Next, near is connected to rand, and the point obtained by moving near a certain length k along the line is the next point for this iteration, after which the iteration ends. If the distance from the endpoint to near is less than k , it is considered that R has found a continuous polyline segment from the start point to the endpoint. Zhang et al. [10] applied an improved RRT algorithm to inspection tasks using multirotor UAVs. Their improved RRT method demonstrates that RRT-based methods can effectively adapt to real-world storage scenarios. However, The RRT algorithm, due to its characteristic as a rapidly exploring random tree method, has some significant drawbacks. The path output by the RRT algorithm is a random polyline, winding and tortuous, far from optimal. Since the algorithm focuses on quickly exploring the space rather than optimizing the path, its paths often contain a large amount of redundancy. In addition, the RRT algorithm lacks guidance and has no features to direct it toward finding the endpoint, making it very time-consuming.

2.2.3 Dynamically Feasible Path Search via Hybrid A*

The Hybrid A* algorithm, also known as the mixed A* algorithm, developed by D. Dolgov et al.[11] in 2008, is one of the core path planning algorithms for autonomous UAV navigation. It combines the shortest path search of the A* algorithm with features of flight dynamics that allow reasonable turning

for aircraft, addressing the shortcomings of the traditional A* algorithm where the aircraft cannot follow the path directly. The Hybrid A* algorithm was proposed by Stanford University in 2010 and has been successfully applied in various fields such as autonomous vehicles and UAVs. Saeid Sedighi et al. [12] proposed that the Hybrid A* algorithm can be operated in a continuous-discrete space, and takes nonholonomic constraints into account to generate acceptable paths for vehicles with restricted mobility.

The application of Hybrid A* in rotor UAVs is mainly reflected in the design of the state space and the generation of motion primitives. For multirotor UAVs, although lateral flight is possible, considerations such as the quality of the sensor view—like a forward-facing camera mounted on a gimbal—and energy efficiency prioritize the alignment of the UAV's heading with the direction of speed during turns. Hybrid A* prevents the UAV from moving in straight-line, angular steps dictated by the 4-neighborhood or 8-neighborhood of the standard A* algorithm, instead planning 3D curved segments consistent with UAV dynamics, considering the minimum turning radius and maximum climb angle.

Similar to the A* algorithm, the Hybrid A* algorithm is a heuristic algorithm. Fine-tuning the path from A* means it cannot guarantee finding the global optimum. Since it must account for the yaw angle, if the A* path passes through a narrow space, the Hybrid A* algorithm can only make adjustments within the limited space. If its optimal solution conflicts with obstacles, Hybrid A* may even be forced to deviate from the nearby A* path and find a longer alternative route.

2.2.4 Kinodynamic RRT Algorithm

The Kinodynamic RRT algorithm is an important algorithm in the field of motion planning.[13] It combines the global search capability of RRT with dynamic constraints, and is used to solve system motion planning problems that involve both dynamics and kinematic constraints.

The idea of Kinodynamic RRT is similar to RRT; both perform random sampling in the search space to construct a tree-like network structure. However, Kinodynamic RRT involves more features, including not only position but also a series of variables such as velocity and orientation. In terms of simulation methods, the Kinodynamic RRT algorithm no longer simply connects points like the RRT algorithm, but instead calculates the edge transition problem between two points and forward simulates inputs to obtain trajectory segments that satisfy dynamic constraints. Ge et al. [14] apply the kinodynamic RRT algorithm for fixed-wing UAV trajectory planning. Unlike standard RRT, kinodynamic RRT uses the UAV's flight dynamics model using feedback linearization, ensuring that the generated trajectory is dynamically feasible for the actual UAV.

Because the Kinodynamic RRT algorithm is similar to RRT, its drawbacks are also similar: the Kinodynamic RRT algorithm not only has the shortcomings of RRT such as long paths and high time consumption, but also has lower computational efficiency because optimizing the path generated by the RRT algorithm requires precise solving, which poses greater disadvantages when moving in rough terrain.

2.3 Reactive Based Trajectory Planning

Traditional trajectory planning usually relies on global discrete search or random sampling, which is high computational cost in dynamic conditions and has difficulties in responding to obstacle changes quickly. However, reactive based trajectory planning can achieve low latency obstacle avoidance and flying control through using local sensing information to adjust the tracks. Boyu Zhou et al.[15] introduced a UAV trajectory planning framework based on Kinodynamic Search and B-spline Optimization, known as the **Fast-Planner**. It focused on the problem of UAVs' autonomous navigation in unknown environments, aiming to achieve autonomous flights which are safe and quick. Its advantages are that the speed of planning is fast and the generated trajectory is smooth. Furthermore, it can achieve high

speed and safety at the same time. However, using it needs to contribute and maintain Euclidean Signed Distance Field (ESDF), which will cost a lot of calculating resources. In addition, when the environment changes frequently, the updating of ESDF will affect the instantaneity of the system, as a result, it is not suitable in complex and dynamic situations. Xin Zhou et al.[16] suggested using gradient-based Local Trajectory Optimization, B-spline Optimization, and ESDF-free obstacle solving strategy to reduce calculating costs, known as the **EGO-Planner**. Its planning speed is much more quicker than traditional ESDF ways, which is very suitable for conditions with limited calculating resources. However, it belongs to local planner, when facing a large complex situation, it relies on additional instructions from repetitive re-planning or global planner. Additionally, the time reallocation strategy may lead to excessive conservation in path selection and limited adaptation towards dynamic conditions.

Zhepei Wang et al.[17] suggested using minimum control quantity optimization theory and polynomial trajectory representation method to achieve smooth and safe flying tracks that fulfill dynamical constraints, known as **MINCO**. Because of its high efficiency of calculating, high quality of tracks, small amounts of parameters, and the ability of optimizing flying tracks and allocating time at the same time, it is very suitable for UAVs' real-time planning. However, MINCO can not generate paths on its own, which means that it usually needs an additional front-end algorithm to offer path point or safety corridor. Furthermore, because of being affected by local optimization concept, the final results will be affected by the quality of the original tracks, which means that the globally optimal solution might not be found. Jesus Tordesillas et al.[18] chose to use MIQP and safety backup trajectory mechanism together to achieve UAVs' high speed and safe navigation in unknown environments, known as **FASTER**. It can reach a higher speed when promising UAVs' safety. Furthermore, the paths generated using FASTER is more flexible. However, it requires frequent reorganization and maintenance of backup tracks, which means that when the known safe zone is small or the environment is extremely small, the result might be rather conservative. Ihab S. Mohamed et al.[19] suggested using MPPI and SGP together to achieve UAVs' autonomous navigation in unknown environments, known as **GP-MPPI**. It does not require the contribution of the whole map or offline training, which means that it can adapt to unknown environments. However, it also relies on local optimization. As a result, its performance relies on the design of cost function, collecting efficiency, and the quality of information the sensors collected from the environments.

2.4 End2End Learning Based Trajectory Planning

Traditional trajectory planning needs cost functions of explicit mapping and manual design, which might be limited by modeling error and calculating efficiency in complex environments. However, End2End learning based trajectory planning can change sensor input into flying control or tracks output directly through neural networks, which means that it can learn implicit programming strategies and improve generalization ability in complex situations.

As Ursula Challita et al.[20] suggested using ESN, is a method that trains the output layer to predict the result according to the input, to achieve maximizing the use of energy and minimizing the delay at the same time. Through training the ESN can help the UAV to form an action instruction through the prediction of the future when it receives an input. Many UAVs are able to be used together and communicate with each other through this method. It can improve the communication quality between the UAVs, including decreasing the delay, and improving the efficiency. Abhik Singla et al.[21] suggested using RNN, which has memory ability, Attention, which can select key points, and DRL, which can learn decision making through trials and errors together. They enable UAVs to memorize the paths, select the key barrier and learn how to make decisions when facing barrier. Through using this idea, UAVs can solve POMDP, which is avoiding obstacles according to training and memories in limited scope when

there is only limited information known, making the UAVs' distance of collision-free light longer. However, the process of training is rather complex and there is a reliance on the quality of the data provided to train it. Omar et al.[22] suggested using DDPG-based DRL, which means that there is an operating system that provides specific actions and an organ to judge whether itself made a good decision. It can provide a sequence of continuous output, as a result, the UAVs' tracks will be rather stable. However, because of its sensitiveness towards its judging standard, the judging organ might not be stable, leading to a lack of stability in its result and incapability of providing the best solution.

Julius Rückin et al.[23] suggested using RL and Tree Search together. Tree Search is a method using every different possible solution of each step to form a decision tree, testing the paths, and selecting the best one. RL has the ability to identify which one is the best decision in a specific condition, however, it can not predict or analyze the future. The information collected can be maximized through using this method, which is very suitable for exploring missions. Furthermore, its calculating efficiency is higher than all the methods in the past. However, the prediction of the future made by the decision tree might not be accurate because the future in the reality is changing, not stable. Yifan Guo et al.[24] suggested using DQN model, a method using a neural network to calculate the degree of whether a decision is good in a condition, and find out the best decision in the provided condition. It is stable and rather easy to achieve comparing with other methods. However, DQN can only make decisions between several limited choices, contradicts with fact that the best decision in the realistic world might be changing, leading to a lack of ability to solve complex conditions. This is due to that the process of training DQN is to train it to decide the degree of whether the decision is good, but not to provide a new decision. Lihan Xu et al.[25] suggested using DQN and PPO/DDPG/TD3 together, which all can control the UAVs continuously. DDPG is good at controlling continuously but is lack of stability. TD3 is stable but faced problem implementing. PPO is the most stable one, but the training process is rather slow. The advantage of this method is that DQN makes decision quickly while one of the other three models produce continuous actions.

The two most popular algorithms recently are the followings, YOPO and NavRL. J. LU et al.[26] introduced a learning-based one-stage quadrotor planner that integrates perception, apping, path searching, and trajectory optimization into a single neural network, as shown in Fig.4, is known as **YOPO**. It uses motion primitives as anchors to predict trajectory offsets and scores in one forward pass, achieving very low latency for high-speed obstacle-dense flight. However, its performance heavily depends on training data distribution and sim-to-real generalization. It can generate many paths at once, but the diversity of the adjustment results is insufficient. Furthermore, the reality is not as ideal as the training conditions. The perception system might be affected by the light, which leads to mispercept and might cause wrong decisions.

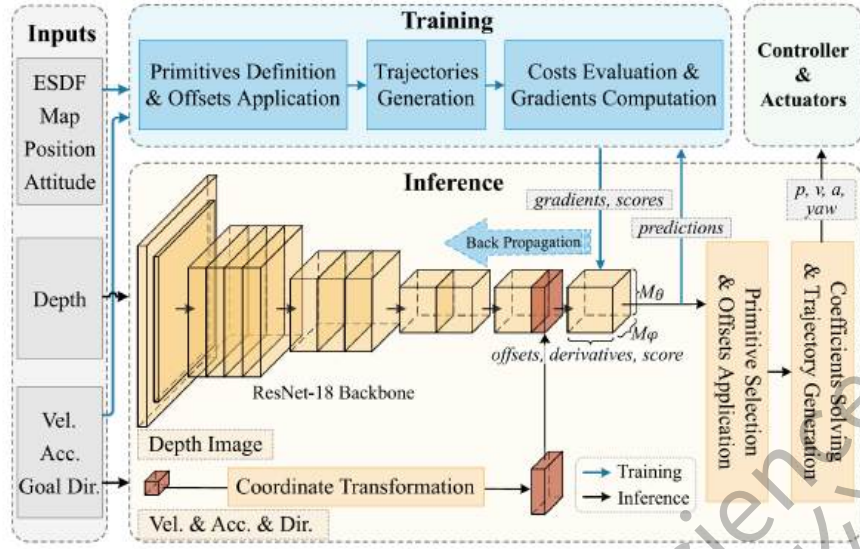


Figure 4: The Framework of YOPO[26]

Z. Xu et al.[27] introduced a reinforcement learning framework for safe navigation in dynamic environments, known as **NavRL**. It learns a reactive policy using PPO, supported by a safety shield based on velocity obstacles to reduce unsafe actions. It reduces manual rules and generalizes beyond UAVs. Limitations include weaker interpretability, dependence on training quality, and no global optimality guarantee. However, the training process is quite slow, this is because that trial and error needs a lot of attempts, which leads to large amount of computation. Furthermore, the reward system relies on the design of the reward rules, which might have problem suiting in new environments.

2.5 Benchmark Analysis of Open-Source Simulation Ecosystems

AirSim is a tool that can be used to simulate drones, cars and other vehicles. It is built on Unreal Engine. It is a open source app. It can be used in different platforms and support using PX4 and ArduPilot for software-in-loop simulation. Furthermore, it can use PX4 for hardware-in-loop, making the simulation more physically and visually realistic. It is an Unreal plugin that can be dropped into any Unreal environment easily. Similarly, the company also has a experimental plugin version that focus on Unity engine. The company's aim is to develop Airsim into a platform for AI research, using for experimenting deep learning, computer visual and reinforcement learning algorithms, and to be used in the field of autonomous vehicles. For this purpose, AirSim also provides APIs to search for data on the platforms and control the vehicles, as shown in Fig.5. The advantages of Airsim are that it is designed for UAVs, meaning that when simulating UAVs, it does not rely much on the computer configuration. In addition, the visual effect, including the light, the shadow, and the materials is great since it is based on unreal engine, which is quite important in training visual navigation. However, it is not good at running many environments at the same time, which is required in many reinforcement learning project. Furthermore, it is lack of simulation of wind, noise and all sorts of things in the realistic world that may disturb the UAVs.

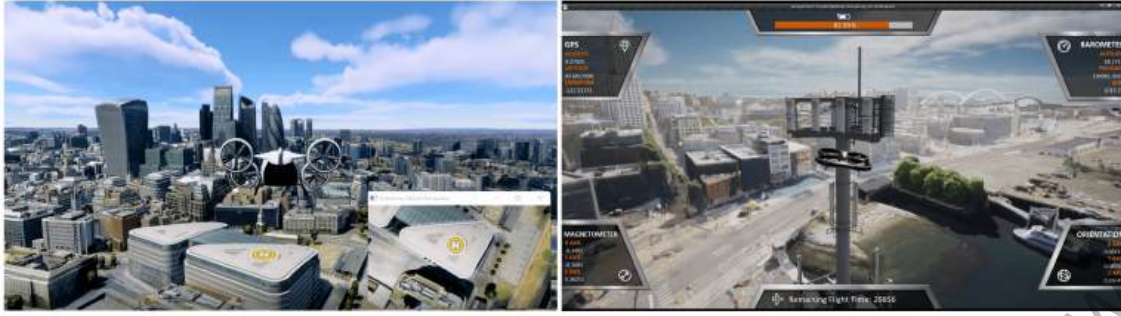


Figure 5: Example of AirSim Simulation Platform

CARLA is aimed to support the development, training and validation of autonomous driving systems. Despite open-source code and protocols, CARLA also provides open digital resources (city layouts, buildings, cars) which are created just for this aim, that can be freely used, as shown in Fig.9. This simulation platform supports configure sensor suites, environment conditions, full control of all static and moving characters, maps generations and more flexibly. CARLA is good at simulating cities, including roads, crossroads, pedestrians, traffic lights, and vehicle flows, which are quite useful and essential in Low-altitude urban inspection and air-ground collaborative navigation. Furthermore, it can fulfill many kinds of sensors, including RGB Camera, Depth Camera, LIDAR, Semantic Segmentation, and Event Camera. Additionally, CARLA can simulate many agents together, for example, UAV and cars, UAV and pedestrians, UAV and UAV. However, it is not designed for UAVs, meaning that its flight dynamics part is rather weak, and the environments it simulated focus on cities, not forests or mountains.



Figure 6: An Example of CARLA Simulation Platform

NVIDIA Isaac Sim™ is an open source reference structure which is based on NVIDIA Omniverse™ libraries. It is used for robots simulation, testing and data generation in virtual environments that are based on physics, as shown in Fig.7. NVIDIA Isaac Sim has full expandability, as a result, the developers can build custom simulators that are based on OpenUSD, or integrate framework function into training and testing progresses. One of the advantages of NVIDIA Isaac Sim is that it can use the computer's GPU to quicken its operation, allowing it to simulate thousands of environments at the same time. In addition, the accuracy of the physical simulation is high, including Rigid body dynamics, collision detection, contact mechanics, shortening the gap between simulation and real world. Furthermore, it offers high-fidelity vision, which is suitable for Foundation Model, Vision Transformer, and Diffusion

model training. However, NVIDIA Isaac has high requirements for computer configuration, meaning that it will be nearly impossible for normal computers to run it.



Figure 7: An Example of NVIDIA Isaac Simulation Platform

Gazebo is an open-source simulation platform which can simulate 3D environments and many intelligent agents at the same time. There are many physical engines in Gazebo, for example ODE and Bullet, helping it to simulate the physical movements of the UAVs accurately, making the simulation environment more realistic than simple mathematic models. It supports simulations of all kinds of sensors, for example cameras, LIDAR, and IMU. Furthermore, there are rich plugin system and APIs in Gazebo, allowing developers to design their own environments, models, and controllers. Additionally, Gazebo and ROS are seamlessly integrated. This is shown in many different parts, including ROS plugin, sensor simulation, controller interface, the seamless switch between simulation and the real world, and data collecting and reviewing. However, the visual effect of Gazebo is rather simple, which means that its modeling ability is limited and there is big gap between its simulation environment and realistic world in visual perspective. Fig.8 shows a preliminary Gazebo scene constructed for future cross-simulator evaluation. This scene was not used to produce the quantitative results reported in Section 5.

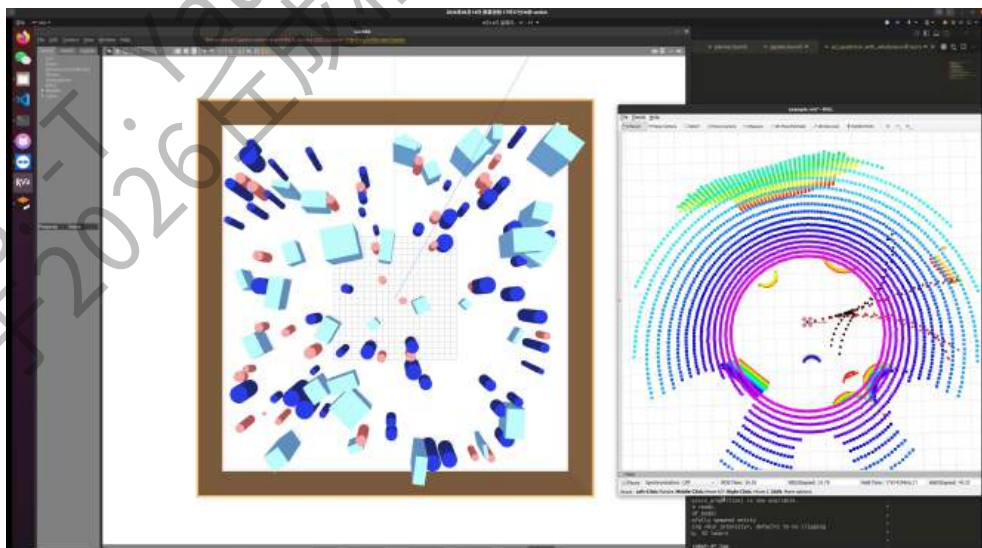


Figure 8: An Example of the Forest We Built by Gazebo Simulation Platform

3 Simulation Environment

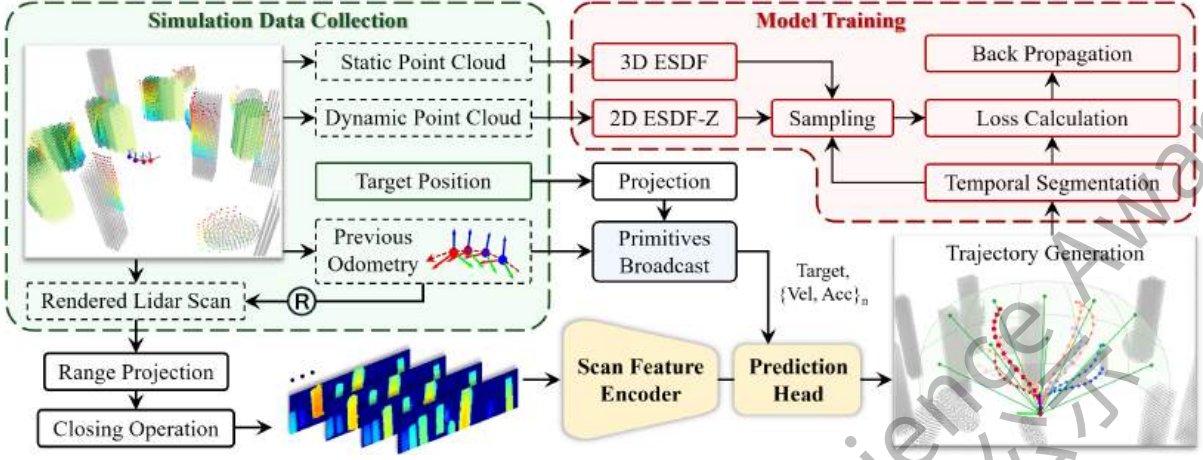


Figure 9: Fast System's Simulation and Data Generation Pipeline for UAV

A reliable simulation environment is a prerequisite of validating the learning planner and generating training data. As showing in Fig.9, this system is based on ego-planner ecosystem construction 3D quadrotor simulation platform, using its map generation, local perception, and controlling model. The simulation platform has three layers: the scene layer could generate 3D point cloud map that includes static obstacles and dynamic obstacles, and set visual walls at the edges of the map to limit the flying area; the perception layer is based on the CUDA voxel rendering simulation of airborne LiDAR, real-time generating local point cloud according to the UAV's postures; the controlling layer uses the identical position controller, changing the position-speed-acceleration order that the planner outputted into odometer feedback, forming a closed loop. The detailed process could be both shown in the algorithm below and the following part.

Algorithm 1 Simulation Procedure of the Proposed Planner

Require: Static map P_s , dynamic map P_d , target p_g , odometry O , construct static and dynamic ESDF maps

- 1: **while** Simulation **do**
 - 2: Render LiDAR scan S
 - 3: Generate range image R from S
 - 4: Generate motion primitives from O
 - 5: Encode scan features F
 - 6: Predict trajectory parameters using F , p_g , and primitives
 - 7: Generate candidate trajectories $\mathcal{T}$
 - 8: Sample ESDF values along $\mathcal{T}$
 - 9: Compute loss $\mathcal{L}$ and update network
 - 10: **end while**
 - 11: **return** simulator planner
-

The training data is collected through multi-aircraft parallel autonomous flight: Multiple UAVs that loaded with models would explore by themselves in a randomly generated map and record LiDAR rendering of point clouds, dynamic obstacle states, and odometry sequences at the same time. Each sequence corresponds to a complete flight episode, including the static map, frame-by-frame dynamic

obstacle states, synchronized sensor observations, and UAV odometry sequences. Collecting scale is decided by the number of the UAVs, sensor frequency, and collecting time, which could generate thousands of samples at once.

The fast and slow subsystems are trained using different supervision strategies. The high-frequency fast subsystem follows a self-supervised training paradigm without trajectory annotations. During training, the simulator provides complete map information for constructing a static 3D ESDF and a dynamic 2D ESDF. The fast subsystem generates multiple candidate trajectories from the current observation and UAV state, and samples the ESDF values along each trajectory to evaluate its collision risk. The training objective combines static and dynamic obstacle-avoidance losses with trajectory smoothness and goal-guidance losses. In this way, the fast subsystem learns local trajectory adjustment and obstacle avoidance directly from geometric constraints rather than imitating reference trajectories.

The low-frequency slow subsystem is trained with both trajectory and semantic supervision, as described in Section 3.3. Reference trajectories generated by the kinodynamic planner supervise the Flow Matching decoder, while the simplified binary text labels Dead End and Normal provide semantic supervision for the VLM. The two forms of supervision help the slow subsystem learn long-horizon trajectory generation and distinguish trapped states from normal navigation states. All dataset generation and closed-loop evaluations reported in this work are performed using the same lightweight simulation pipeline. The maze geometry, dynamic-obstacle states, and sensor observations therefore remain synchronized throughout data collection, training, and evaluation. Reproducing the complete environment in Gazebo and conducting cross-simulator evaluation are left for future work.



Figure 10: An example of a procedurally generated maze. The colored wall point clouds represent the maze geometry, while the orange cylinders denote dynamic obstacles placed in the free corridors

3.1 Complex Scene Map Construction for UAV Navigation

3.1.1 Structured Dead-Zone Maze Environment

The maze environments used in this study are generated by a unified procedural pipeline within a $50\text{m} \times 50\text{m}$ workspace. The generator first divides the horizontal plane into an irregular grid and applies random perturbations to the grid vertices. Each grid cell is treated as a graph node, and randomized Depth First Search (DFS) [28] is used to select the connections between adjacent cells. Shared edges selected by DFS are removed to form traversable corridors, while the remaining internal and boundary edges form the maze walls. The generated wall segments are assigned randomized thicknesses and fixed heights before being discretized into point clouds. Their axis-aligned bounding boxes are retained for collision detection between the walls, the UAV, and the moving obstacles. The same maze generator supports both the static and dynamic experimental configurations. In the static-maze configuration, the dynamic

obstacle layer is disabled so that dead-end recognition and route recovery can be evaluated independently. In the dynamic-maze configuration, cylindrical moving obstacles are enabled within the free corridors. No separate maze is reconstructed for these two configurations. Figure 10 shows an example with the dynamic obstacle layer enabled, where the orange cylinders represent moving obstacles.

3.1.2 Unstructured Forest Obstacle Environment

The method for generating a forest-like map involves a large amount of static obstacles, which are the "trees" in Fig.11, and then voxelization to point clouds based on their resolution. To elaborate, we randomly sampled the position, scale, and rotation of each tree instance. To avoid a potential risk that obstacles are too close to each other, and evade some start and finish area at the same time, we load the point cloud of the "tree" as a template, and the program performs voxel downsampling on the template point cloud. Then, randomly sample the positions of multiple trees within the map, and randomly assign a scale and a rotation angle around the Z-axis to each tree. During sampling, the minimum distance is still used to ensure that trees are not placed too close to one another. Subsequently, each tree obstacle is converted into a point-cloud representation according to its sampled position, scale, and rotation. In addition, ring-shaped obstacles are generated by randomly sampling their center positions, radii, heights, and orientations. Points are then sampled along the circumference of each ring and added to the point cloud. At last, points along the circumference are added to the point cloud.

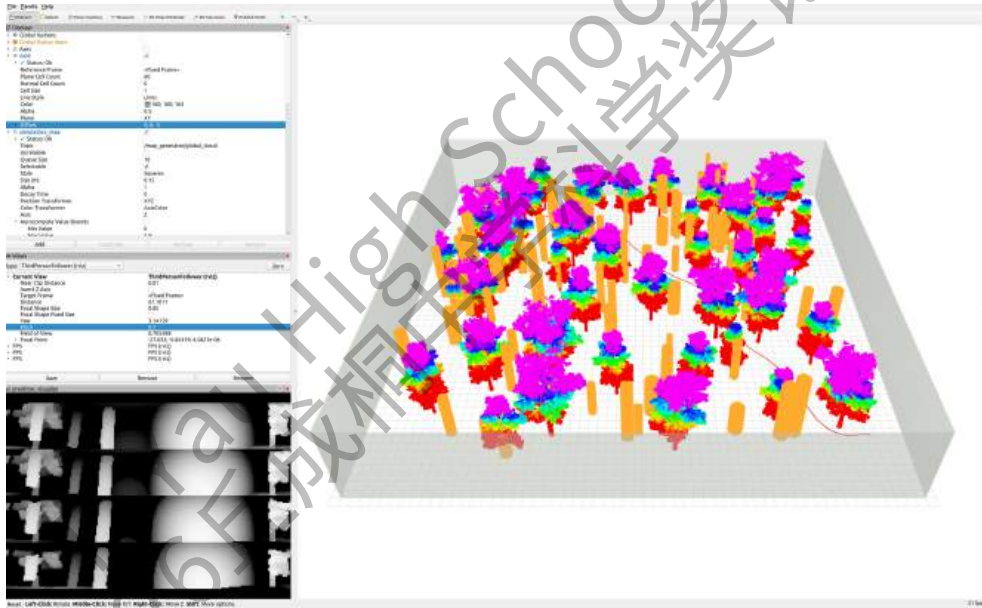


Figure 11: An Example of a Forest-like Map

3.1.3 Dynamic Obstacle-Embedded Maze Environment

Dynamic obstacles are implemented as an optional layer of the unified maze generator rather than as a separately reconstructed environment. When this layer is enabled, cylindrical obstacles are initialized at collision-free positions inside the maze corridors. Their planar motion states are updated at each simulation step, with random perturbations introduced to produce non-deterministic trajectories. The obstacle velocity is limited to 1.25 m/s. When an obstacle reaches a maze wall or the map boundary, its direction is adjusted to keep it within the valid flight area.

After each state update, the cylindrical obstacles are discretized into a dynamic point cloud according to their current positions and dimensions. The dynamic point cloud is synchronized with the static maze point cloud and is used for LiDAR rendering, dynamic ESDF construction, trajectory supervision,

and closed-loop evaluation. Disabling this layer produces the static-maze configuration used in Section 5.2, whereas enabling it produces the dynamic-maze configuration evaluated in Section 5.4. Both configurations therefore share the same map generator, collision model, and perception pipeline.

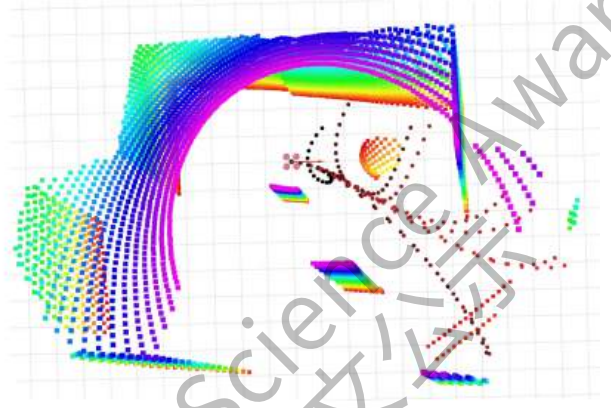
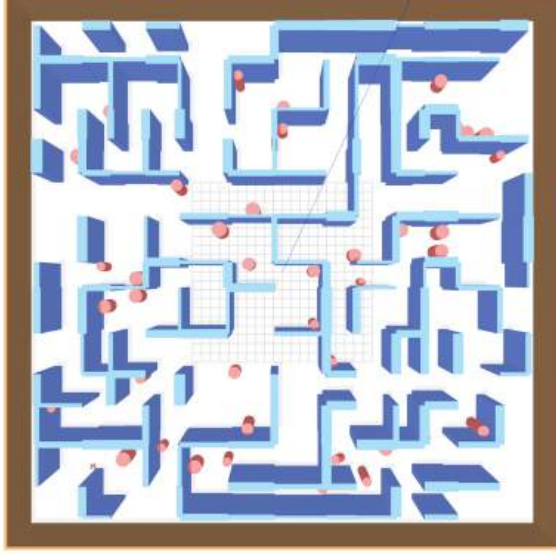


Figure 12: An Example of a Dynamic Obstacle Maze

3.2 Closed-Loop Dataset Generation Pipeline for High-Frequency System

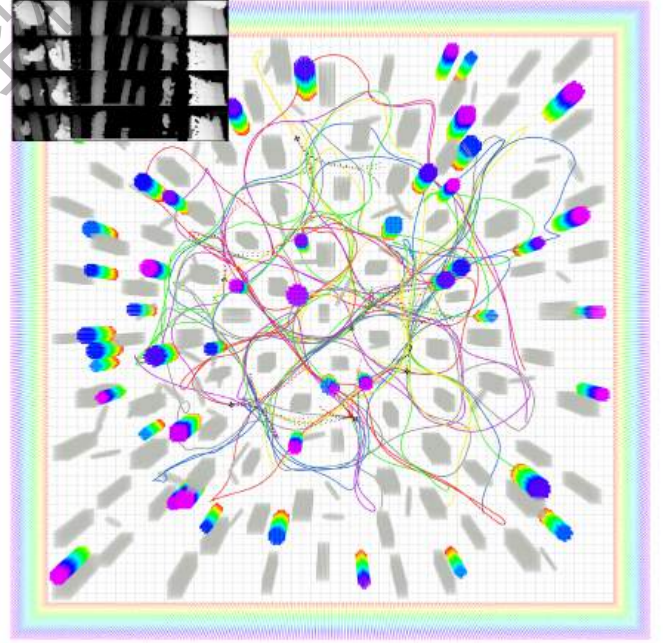
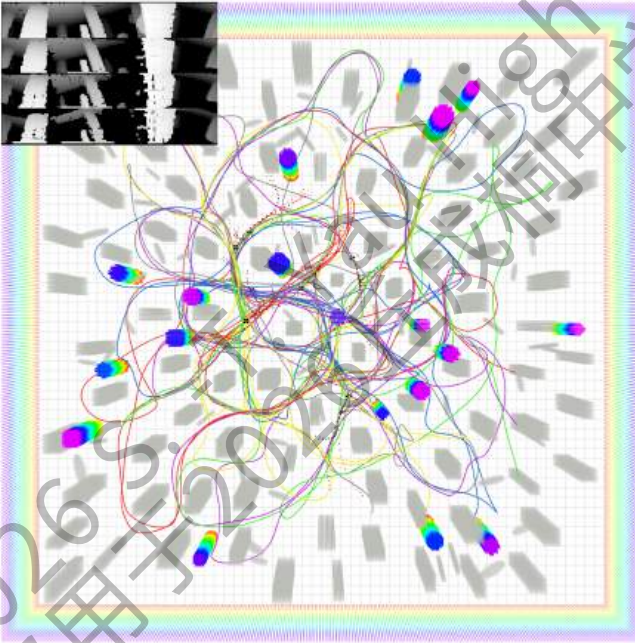


Figure 13: Examples of Closed-loop Data Collection for the High-frequency Subsystem

Figure 13 shows representative examples of the closed-loop data-collection process used for the high-frequency subsystem. Multiple UAVs perform autonomous rollouts in randomized forest-like environments containing both static structures and moving obstacles. In the figure, the gray structures represent static obstacles, the colored short traces indicate the positions of moving obstacles sampled at

different times, and the colored curves represent UAV flight trajectories collected from different navigation episodes. The inset images show representative LiDAR range images rendered at the corresponding UAV states. The static occupancy map and the current dynamic-obstacle occupancy are also retained as privileged training information for constructing the static 3D ESDF and dynamic 2D ESDF. Accordingly, a high-frequency training sample can be represented as

$$\mathcal{D}_{\text{fast}} = \left\{ \mathbf{R}_{t-4:t}, \mathbf{x}_t, \mathbf{g}_t, \mathbf{P}^s, \mathbf{P}_t^d \right\}, \quad (1)$$

where $\mathbf{R}_{t-4:t}$ denotes the five-frame range-image history, $\mathbf{x}_t$ is the current UAV state, $\mathbf{g}_t$ is the world-frame navigation goal position, and $\mathbf{P}^s$ and $\mathbf{P}_t^d$ denote the static and dynamic occupancy information, respectively. The colored rollout trajectories in Figure 13 therefore visualize the coverage of the collected observation sequences and are not used as expert trajectory labels.

3.3 VLM Training Data Generation with Simplified Semantic Annotation

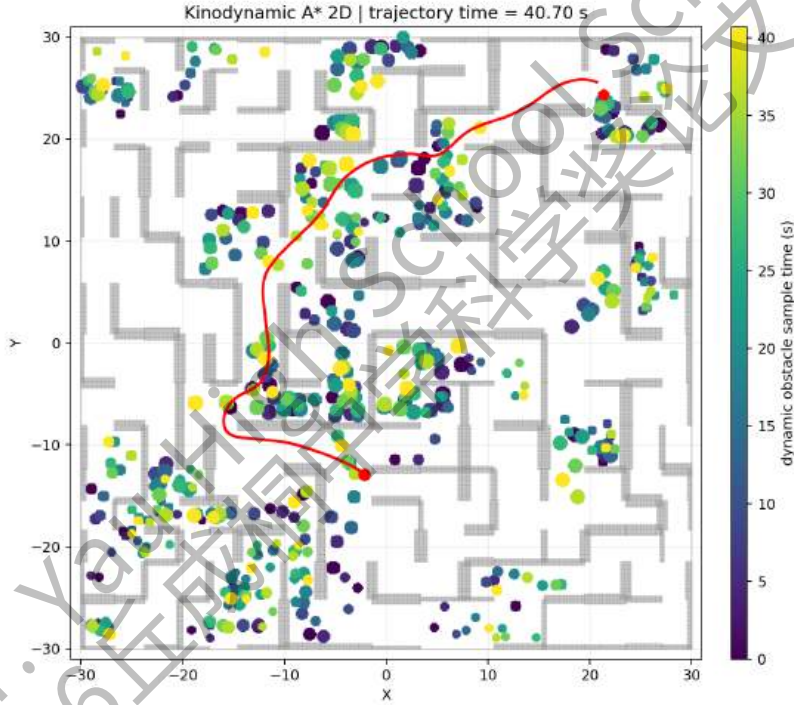


Figure 14: Kinodynamic A*-based Reference-trajectory Generation for the Low-frequency Subsystem

Figure 14 illustrates the offline reference-trajectory generation procedure used for the low-frequency subsystem. The gray segments represent the static maze walls, the red curve denotes the reference trajectory generated by the kinodynamic A* planner, and the colored markers show the time-varying positions of moving obstacles. The color of each marker represents its sampling time along the dynamic-obstacle sequence.

The kinodynamic A* planner uses the static maze geometry together with the synchronized dynamic-obstacle occupancy to generate collision-free reference trajectories. We generate two types of training cases. For normal navigation, the start and goal are sampled from valid free-space regions. For dead-end recovery, we first select a terminal branch and deliberately place the initial UAV state inside it, while sampling the goal outside the branch. Kinodynamic A* then plans from this initial state to the goal. Since the UAV starts inside a dead end, the reference path has to leave the branch through its entrance

before proceeding toward the goal. These trajectories provide direct examples of how the UAV should recover after entering an unfavorable branch.

The complete reference trajectory and its synchronized observation sequence are sampled at 10Hz, corresponding to a sampling interval of $\Delta t = 0.1$ s. At each valid planning step t , where t denotes the discrete sample index, we extract the next $N = 30$ reference waypoints:

$$\tau_t = \{\mathbf{p}_{t+1}, \mathbf{p}_{t+2}, \dots, \mathbf{p}_{t+30}\}. \quad (2)$$

This future trajectory segment is used to supervise the conditional Flow Matching decoder. Reference trajectories generated by the kinodynamic A* planner are used only for the low-frequency subsystem and are not provided as trajectory annotations to the high-frequency subsystem.

The low-frequency subsystem receives the latest five synchronized multimodal observations,

$$\mathcal{O}_t^H = \{\mathcal{O}_{t-4}, \mathcal{O}_{t-3}, \mathcal{O}_{t-2}, \mathcal{O}_{t-1}, \mathcal{O}_t\}, \quad H = 5, \quad (3)$$

where each observation contains the LiDAR range image, the simulator-provided UAV state, including position, velocity, and acceleration, and the navigation goal position expressed in the world coordinate frame. The range-image features, UAV-state features, and goal features are projected into a shared token space. The five observations are arranged in chronological order and assigned temporal embeddings before being processed by the Transformer. This temporal sequence allows the model to capture changes in the local environment and relate the current observation to recently observed structures.

During offline data generation, the simulator has access to the complete maze topology, the current UAV position, and the navigation goal. A training step is assigned the target text `Dead End` when the UAV is located inside a terminal maze branch that does not contain the goal and reaching the goal requires returning through the same branch entrance. All other valid navigation states are assigned the target text `Normal`. The complete maze topology is used only to generate the training text and reference trajectories and is not provided to the model during inference.

Consequently, each labeled slow-system training sample is represented as

$$\mathcal{D}_{\text{slow}} = \{\mathcal{O}_t^H, \mathbf{x}_t, \mathbf{g}_t, \tau_t, y_t\}, \quad y_t \in \{\text{Dead End}, \text{Normal}\}, \quad (4)$$

where $\mathcal{O}_t^H$ denotes the five-frame LiDAR observation history, $\mathbf{x}_t = [\mathbf{p}_t, \mathbf{v}_t, \mathbf{a}_t]$ denotes the current UAV state consisting of position, velocity, and acceleration, $\mathbf{g}_t$ denotes the navigation goal position expressed in the world coordinate frame, τ_t is the future reference-trajectory segment, and y_t is the target navigation-state label. In addition to the fixed five-frame history, the slow subsystem maintains recurrent memory tokens across consecutive planning steps:

$$\mathbf{M}_{t-1} \in \mathbb{R}^{N_m \times d}, \quad N_m = 16, \quad d = 1280. \quad (5)$$

The memory state is an internal model state rather than a manually generated dataset annotation. At the beginning of each navigation episode, the memory tokens are initialized using a learnable initial state. During sequential training, the previous memory state $\mathbf{M}_{t-1}$ is concatenated with the current multimodal tokens and jointly processed by the Transformer. The updated memory state $\mathbf{M}_t$ is passed to the next planning step and is reset only when a new navigation episode begins.

The semantic branch is implemented as a constrained autoregressive text decoder. The target text is tokenized as a short sequence containing a beginning-of-sequence token, the navigation-state tokens,

and an end-of-sequence token. During training, teacher forcing is used to optimize

$$p_{\theta}(\mathbf{y}_t \mid \mathcal{O}_t^H, \mathbf{x}_t, \mathbf{g}_t, \mathbf{M}_{t-1}) = \prod_{\ell=1}^{L_y} p_{\theta}(y_{t,\ell} \mid y_{t,<\ell}, \mathbf{h}_t^{\text{text}}), \quad (6)$$

where $\mathbf{h}_t^{\text{text}}$ denotes the text-query representation produced by the multimodal Transformer. The corresponding text-generation loss is

$$\mathcal{L}_{\text{text}} = - \sum_{\ell=1}^{L_y} \log p_{\theta}(y_{t,\ell} \mid y_{t,<\ell}, \mathbf{h}_t^{\text{text}}). \quad (7)$$

At inference time, decoding is restricted to the two valid text outputs, Dead End and Normal. The generated text likelihood provides an interpretable semantic signal for the fast-slow gating module. Meanwhile, the future reference-trajectory segment supervises the conditional Flow Matching decoder. Joint text and trajectory supervision enables the recurrent memory representation to associate historical observations with both the current navigation state and the corresponding escape behavior.

4 Methodology

The overall architecture of NavFlow-VL is illustrated in Figure 15. The framework combines a high-frequency reactive subsystem with a low-frequency history and memory aware vision-language subsystem [29]. Given the current LiDAR observation, UAV state, and navigation goal, the fast subsystem generates local motion primitives for responsive obstacle avoidance. In parallel, the slow subsystem processes five consecutive multimodal observations and maintains recurrent memory tokens across planning cycles. The memory tokens preserve task-relevant information after previously observed structures leave the current sensing field. The slow subsystem contains two output branches. A constrained autoregressive text decoder generates the navigation-state text `Dead End` or `Normal`, while a conditional Flow Matching decoder generates an escape-oriented trajectory consisting of 30 three-dimensional waypoints over the next 3 s. The gating module calculates the dead-end confidence from the likelihoods of the two candidate text outputs. Under normal navigation conditions, the high-frequency trajectory is executed. When the generated text indicates a dead-end state with sufficient confidence, the system switches to the trajectory generated by the slow subsystem. This design connects temporal perception, recurrent memory, semantic text generation, and trajectory generation within a unified closed-loop navigation pipeline.

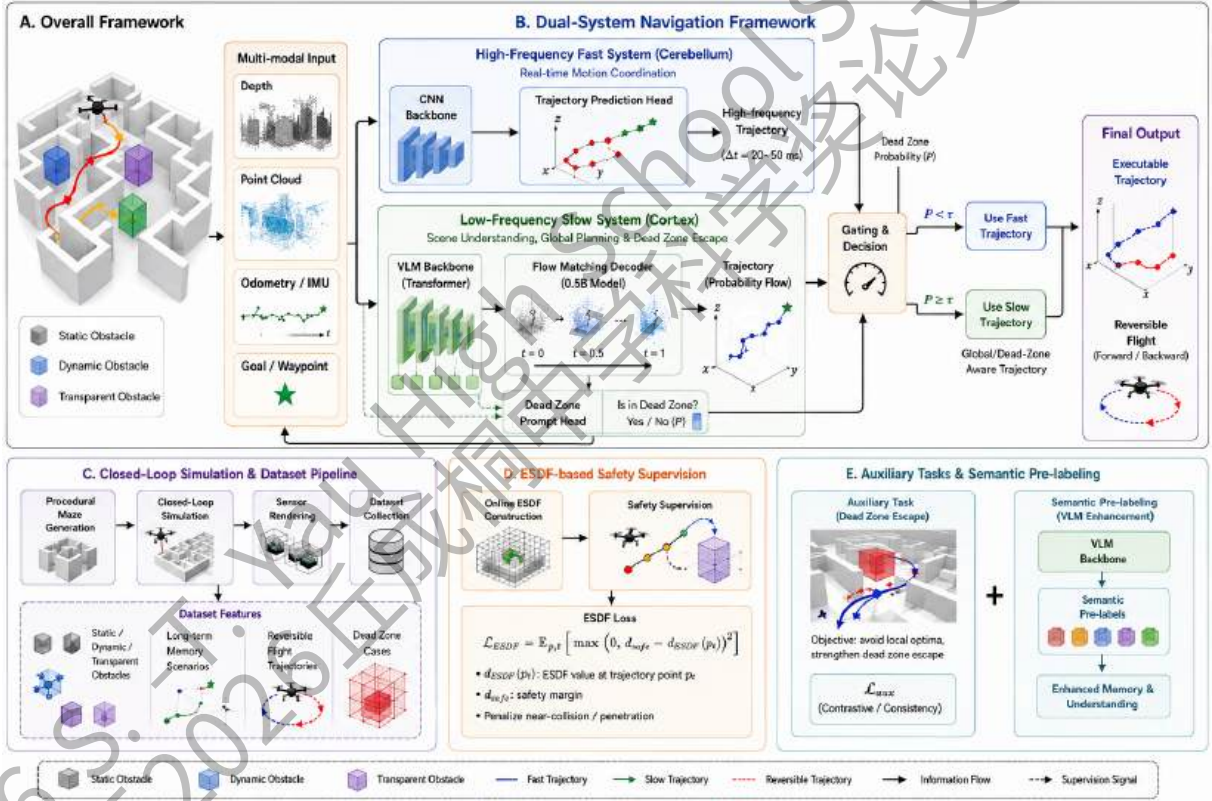


Figure 15: Overall Architecture and Data Flow Diagram of the Proposed NavFlow-VL Pipeline

Motion Primitive Sampling and Trajectory Optimization

4.1 Motion Primitive Sampling and Trajectory Optimization

The high-frequency subsystem generates a set of short-horizon trajectory candidates rather than directly regressing a single trajectory. This design is motivated by the fact that several locally feasible directions may exist at the same time. For example, when an obstacle blocks the center of the LiDAR field of view, the UAV may pass it from either side. Predicting only one trajectory makes it difficult to preserve these different possibilities.

Following the motion-primitive representation in [30] and the anchor-based formulation used in YOPO [26], we divide the LiDAR field of view into a 5×2 angular grid. Each grid cell is associated with one primitive anchor, giving $N = 10$ candidate regions. The anchors do not directly represent the final trajectories. Instead, they provide spatially separated initial directions from which the network predicts locally refined terminal states.

Let ϕ_j and θ_j denote the azimuth and elevation associated with the j -th primitive, and let r denote the nominal planning radius. The anchor endpoint in the UAV body frame is

$$\bar{\mathbf{p}}_j^B = r \begin{bmatrix} \cos \theta_j \cos \phi_j \\ \cos \theta_j \sin \phi_j \\ \sin \theta_j \end{bmatrix}, \quad j = 1, \dots, N. \quad (8)$$

These anchors divide the visible space into different local search regions. The trajectory-prediction head then predicts an offset for each anchor, together with the terminal velocity, terminal acceleration, and a trajectory cost. In practice, the endpoint correction can be expressed using offsets in radius and direction,

$$r_j = r + \Delta r_j, \quad \phi'_j = \phi_j + \Delta \phi_j, \quad \theta'_j = \theta_j + \Delta \theta_j, \quad (9)$$

which gives the refined terminal position

$$\mathbf{p}_{T,j}^B = r_j \begin{bmatrix} \cos \theta'_j \cos \phi'_j \\ \cos \theta'_j \sin \phi'_j \\ \sin \theta'_j \end{bmatrix}. \quad (10)$$

The offsets are bounded around their corresponding anchors so that different primitives remain associated with different parts of the visible space. Therefore, the ten outputs do not collapse into ten predictions of the same trajectory. The network also predicts terminal velocity $\mathbf{v}_{T,j}$ and acceleration $\mathbf{a}_{T,j}$ for each candidate.

Once the terminal state has been predicted, a continuous trajectory is constructed from the current UAV state. For the j -th candidate, each spatial dimension is represented by a fifth-order polynomial,

$$\mathbf{p}_j(\tau) = \sum_{k=0}^5 \mathbf{a}_{j,k} \tau^k, \quad \tau \in [0, T], \quad (11)$$

where τ denotes time within the planning horizon. The polynomial coefficients are uniquely determined by the initial and terminal boundary conditions,

$$\begin{aligned} \mathbf{p}_j(0) &= \mathbf{p}_t, & \dot{\mathbf{p}}_j(0) &= \mathbf{v}_t, & \ddot{\mathbf{p}}_j(0) &= \mathbf{a}_t, \\ \mathbf{p}_j(T) &= \mathbf{p}_{T,j}, & \dot{\mathbf{p}}_j(T) &= \mathbf{v}_{T,j}, & \ddot{\mathbf{p}}_j(T) &= \mathbf{a}_{T,j}. \end{aligned} \quad (12)$$

Here, the predicted terminal state is transformed from the body frame to the world frame before the boundary-value problem is solved. In this way, every network output corresponds to a dynamically smooth trajectory rather than an independent waypoint.

ESDF-based trajectory supervision. The high-frequency subsystem is trained without expert trajectory labels. Instead, each predicted trajectory is evaluated directly using the geometry available in the simulator. For candidate j , we define the trajectory cost as

$$J_j = \lambda_{\text{sm}} J_{\text{sm}}^j + \lambda_{\text{goal}} J_{\text{goal}}^j + \lambda_{\text{stat}} J_{\text{stat}}^j + \lambda_{\text{dyn}} J_{\text{dyn}}^j. \quad (13)$$

The smoothness term penalizes rapid changes in motion. For the quintic trajectory used here, it is written using the trajectory jerk as

$$J_{\text{sm}}^j = \int_0^T \left\| \mathbf{p}_j^{(3)}(\tau) \right\|_2^2 d\tau. \quad (14)$$

The goal term encourages the endpoint of the local trajectory to make progress toward the navigation goal,

$$J_{\text{goal}}^j = \left\| \mathbf{p}_j(T) - \mathbf{g}_t \right\|_2^2, \quad (15)$$

where $\mathbf{g}_t$ is the navigation goal position expressed in the world coordinate frame.

Obstacle safety is evaluated by sampling points along the candidate trajectory. Let $d_s(\mathbf{p})$ denote the distance obtained from the static 3D ESDF and let $d_d(\mathbf{p}, \tau)$ denote the corresponding distance to dynamic obstacles. The two collision costs are written as

$$J_{\text{stat}}^j = \frac{1}{K} \sum_{k=1}^K \rho(d_s(\mathbf{p}_j(\tau_k))), \quad (16)$$

$$J_{\text{dyn}}^j = \frac{1}{K} \sum_{k=1}^K \rho(d_d(\mathbf{p}_j(\tau_k), \tau_k)), \quad (17)$$

where $\rho(\cdot)$ is a collision penalty that increases when the obstacle clearance becomes smaller than the desired safety margin. The static term therefore penalizes trajectories close to fixed structures, while the dynamic term penalizes trajectories that conflict with moving obstacles over the planning horizon.

The important point is that the ESDF maps are used as privileged information during training rather than as network inputs. Since the polynomial trajectory is a differentiable function of the terminal state predicted by the network, the gradient of J_j can be propagated through the trajectory construction and back to the network. The network therefore learns how the terminal state of each primitive should be adjusted to improve smoothness, obstacle clearance, and progress toward the goal.

In addition to predicting the terminal state, the network predicts a scalar cost $\hat{J}_j$ for each primitive. The cost head is trained to reproduce the geometric cost evaluated above,

$$\mathcal{L}_{\text{score}} = \frac{1}{N} \sum_{j=1}^N \text{SmoothL1}(\hat{J}_j, J_j). \quad (18)$$

Thus, geometric information is required to supervise the planner during training, but it does not need to be queried for every candidate during inference.

At inference time, a single network forward pass produces the refined terminal states and predicted costs of all ten primitives. Candidates that violate the hard kinematic limits are discarded, and the

remaining candidate with the lowest predicted cost is selected,

$$j^* = \arg \min_{j \in \mathcal{V}} \hat{J}_j, \quad \tau_t^f = \tau_{j^*}, \quad (19)$$

where $\mathcal{V}$ denotes the set of dynamically valid candidates. Only a short initial portion of the selected trajectory is executed before the next high-frequency planning cycle. The planner therefore continuously replans from the latest LiDAR observation and UAV state, allowing it to respond quickly to changes in the local environment.

4.2 Flow-Matching Based Trajectory Generation

To generate smooth, feasible flight trajectories for our dual-system navigation framework, we adopt conditional Flow Matching (FM) to implement the trajectory planning module of our low-frequency slow subsystem. Different from traditional regression-based planning models that only output a single fixed trajectory, the Flow Matching approach learns a continuous probability flow field. It gradually converts simple Gaussian distribution samples into reasonable trajectory distributions that adapt to complex navigation scenarios [31]. This learning mechanism effectively fits the multimodal characteristics of UAV navigation, where multiple feasible flight paths usually exist in cluttered maze environments.

In our dual-system architecture, the entire navigation planning process depends on comprehensive contextual information further constructed. The conditioning representation of the Flow Matching decoder is constructed from the current fast-system feature, the history-aware slow-system representation, the updated recurrent memory, the UAV state, and the navigation goal:

$$\mathbf{c}_t = \text{Fuse} \left(\mathbf{z}_t^f, \mathbf{z}_t^s, \mathbf{M}_t, \mathbf{x}_t, \mathbf{g}_t \right), \quad (20)$$

where $\mathbf{z}_t^f$ denotes the local obstacle-aware feature produced by the fast subsystem, $\mathbf{z}_t^s$ denotes the history-aware representation produced by the slow subsystem, and $\mathbf{M}_t$ denotes the updated recurrent memory tokens. The same memory-conditioned Transformer representation is shared by the constrained text decoder and the conditional Flow Matching decoder. $\mathbf{g}$ is the preset navigation target position, and $\mathbf{x}$ stands for the current flight state of the UAV. Based on this comprehensive contextual condition, our model aims to learn the conditional distribution of feasible future trajectories. We define the navigation trajectory τ composed of N consecutive flight waypoints, and the corresponding conditional probability distribution is formulated as:

$$p(\tau|\mathbf{c}) \quad \{\tau = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_N\}\} \quad (21)$$

We construct intermediate noisy trajectory states through linear interpolation between pure noise and the real target trajectory:

$$\mathbf{z}_t = (1-t)\mathbf{z}_0 + t\tau \quad \{\mathbf{z}_0 \sim \mathcal{N}(0, \mathbf{I}), t \in [0, 1]\} \quad (22)$$

In this formula, t represents the continuous flow time step. At the initial stage ($t = 0$), the sampled data is pure Gaussian noise without any trajectory information. As t increases gradually, the sample is continuously optimized under the guidance of the learned vector field, and slowly forms the spatial structure of a feasible flight path. When the flow process converges at $t = 1$, the final sample becomes a complete and effective trajectory connecting the UAV's current position and the navigation target. We use a lightweight Transformer decoder as the core network f_θ to parameterize the whole flow transportation

process and predict the real-time velocity field:

$$\mathbf{u}_\theta = f_\theta(\mathbf{z}_t, t, \mathbf{c}) \quad (23)$$

This decoder takes noisy trajectory samples, flow time parameters and our dual-system navigation context as joint input. It comprehensively considers the local dynamic obstacle avoidance requirement of the fast subsystem and the global dead-end escape and long-distance planning goals of the slow subsystem, ensuring the generated trajectories adapt to complex maze scenarios. During model training, we optimize the network by minimizing the error between the predicted flow direction and the real trajectory transformation direction. The Flow Matching loss function is defined as:

$$\mathcal{L}_{FM} = \mathbb{E} \left[\|\mathbf{u}_\theta(\mathbf{z}_t, t, \mathbf{c}) - (\tau - \mathbf{z}_0)\|_2^2 \right] \quad (24)$$

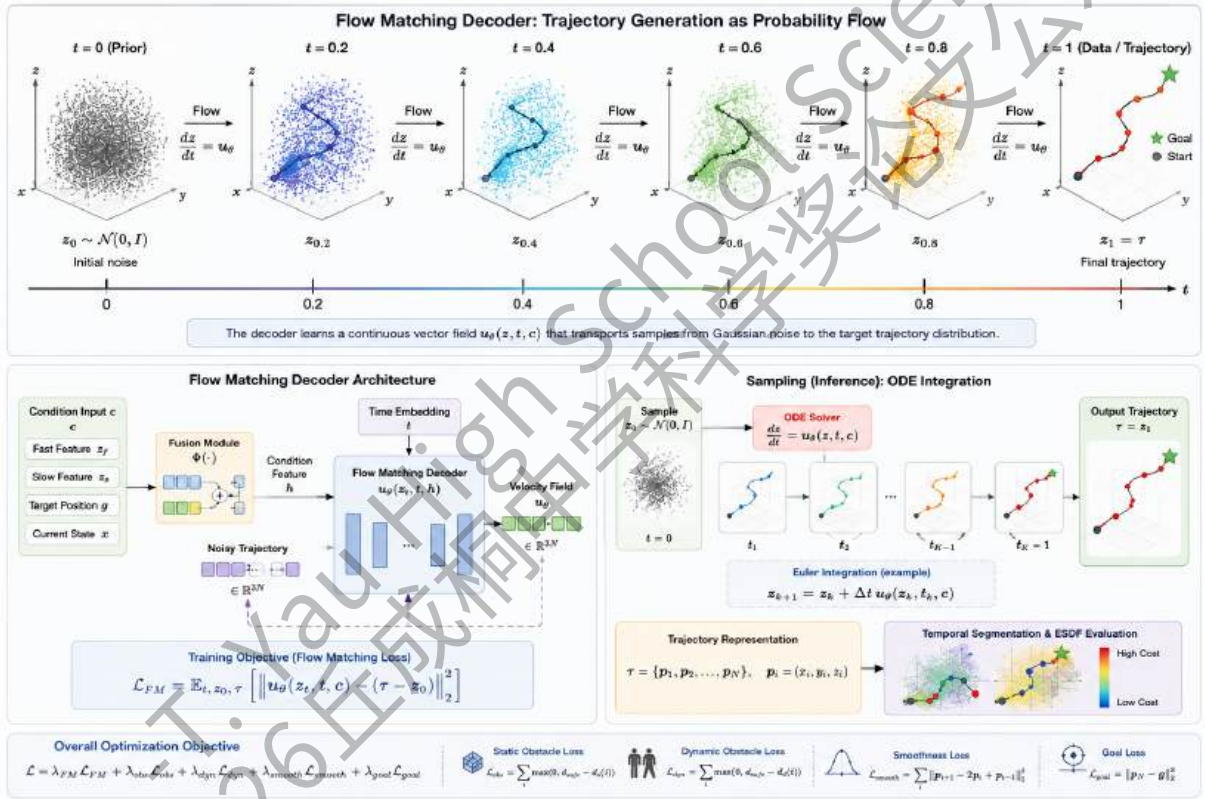


Figure 16: Flow Matching Trajectory Pattern Diagram

Compared with mainstream diffusion-based trajectory generation algorithms, our Flow Matching scheme directly learns the continuous vector field of trajectory transformation, rather than relying on iterative reverse denoising. This design avoids the high computational cost of multiple denoising iterations, greatly reduces the sampling steps required for model inference, and significantly improves planning efficiency. This advantage is crucial for UAV real-time navigation tasks with limited onboard computing resources. In the model inference stage, we solve the ordinary differential equation (ODE) learned by the network to complete trajectory generation:

$$\frac{d\mathbf{z}}{dt} = \mathbf{u}_\theta(\mathbf{z}, t, \mathbf{c}) \quad \{\mathbf{z}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})\} \quad (25)$$

The final usable navigation trajectory is obtained when the ODE integration is completed:

$$\tau = \mathbf{z}_1 \quad (26)$$

To further guarantee flight safety and dynamic feasibility, we verify and constrain all generated trajectories through the hybrid 2D/3D ESDF safety supervision mechanism mentioned above. We integrate multiple optimization objectives to construct the final composite loss function, including Flow Matching fitting loss, static obstacle avoidance loss, dynamic obstacle avoidance loss, trajectory smoothness loss, the slow subsystem is trained by jointly optimizing trajectory generation and constrained text generation:

$$\begin{aligned} \mathcal{L}_{\text{slow}} = & \lambda_{\text{FM}} \mathcal{L}_{\text{FM}} + \lambda_{\text{text}} \mathcal{L}_{\text{text}} + \lambda_{\text{obs}} \mathcal{L}_{\text{obs}} \\ & + \lambda_{\text{dyn}} \mathcal{L}_{\text{dyn}} + \lambda_{\text{smooth}} \mathcal{L}_{\text{smooth}} + \lambda_{\text{goal}} \mathcal{L}_{\text{goal}}. \end{aligned} \quad (27)$$

The text-generation loss encourages the model to summarize the current navigation state from historical observations and recurrent memory. The Flow Matching and geometric losses supervise the generation of a feasible future trajectory. Because the two output branches share the same multimodal Transformer and memory representation, the semantic navigation-state objective also provides auxiliary supervision for trajectory generation.

In parallel, the high-frequency fast system generates short-horizon trajectories based on local obstacle observations and immediate environmental changes. An adaptive gating module determines whether the UAV should continue following the reactive fast-system trajectory or switch to the escape-oriented trajectory generated by the slow system. This design preserves rapid responses to nearby dynamic obstacles during normal navigation while providing history-aware recovery when the UAV is identified as being inside a goal-excluding terminal maze branch. It is therefore particularly suitable for maze navigation and complex dynamic environments in which multiple feasible trajectories may exist.

4.3 Continuous ESDF Map Representation for UAV Obstacle Avoidance

The Euclidean Signed Distance Field (ESDF) is a map representation that stores the distance to the nearest obstacle for every points in space [32]. It is a crucial aspect of UAV's of obstacle avoidance and trajectory optimization. For Occupancy Grid Map (OGM), it only shows whether the UAV can reach the point or not [33], but ESDF can tell the distance between the UAV and the nearest obstacle and show the possible directions that are more safer for UAV to take action by leveraging continuous values and gradients. For every points in the 3-dimensional space, its ESDF value is defined as:

$$\Phi(p) = \begin{cases} +d(p, \partial O), & p \text{ in free space.} \\ 0, & p \text{ is on the surface of the obstacle.} \\ -d(p, \partial O), & p \text{ is inside the obstacle.} \end{cases} \quad (28)$$

where $d(p, \partial O)$ denotes the Euclidean distance from p to the boundary of the nearest obstacle

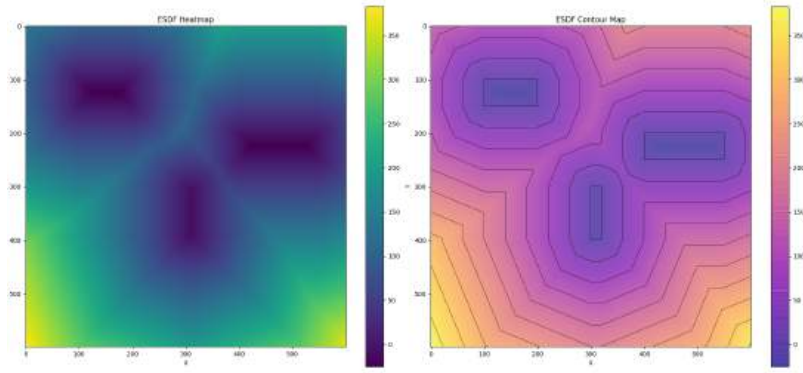


Figure 17: ESDF

UAV flight has three crucial requirements, especially in expressway, indoor and complex environment. The first requirement is safe obstacle avoidance, in another word, distance perception. Compare to the OGM, which can only provide binary judgment and need to check the distance repeatedly, the ESDF can pre-store the distance directly at each point and check only once when necessary. Secondly, trajectory optimization. Modern UAVs commonly employ optimization-based trajectory planning, such as Fast-Planner and TEB. However, ESDF is continuously differentiable, and can provides a gradient direction that steers away from obstacles, allowing direct trajectory optimization to creates a path that is both smooth and safe. The Last requirement is real-time capability. Most of the mainstream ESDF construction algorithms like the two-pass scan or EDT method is $O(n)$ complexity, which means they're linear in complexity. This characteristic makes them faster than repeatedly performing nearest-neighbor searches and is suitable for the real-time onboard computation, as shown in Fig. 17. ESDF has an elaborate construction process based on a grid map such as OGM, which is created by LiDAR and depth cameras. Then, transform the binary value grid into a distance field, each point in this field storages its distance from its nearest obstacle, and use sign to notify whether the point is in the free space, that is, the space where UAV can reach. In this distance field, points in free space are positive, points on the obstacle surface are zero, and points inside the obstacle are negative. Subsequently, calculate the gradient of the distance field, in another word, taking the derivative, to determine the direction for UAV to evade obstacles at each point, therefore deriving a safe route.

To simultaneously guarantee flight safety, positioning accuracy and real-time computational efficiency during UAV navigation, this study adopts a hybrid ESDF modeling strategy to separately characterize static and dynamic obstacles. As illustrated in Fig. 18, we first fuse LiDAR sensing data to construct a three-dimensional occupancy grid for all static environmental structures. On this basis, we calculate the 3D Euclidean signed distance field, which records the spatial clearance from each voxel unit to the nearest static obstacle surface. We further adopt trilinear interpolation to acquire continuous distance values and spatial gradient information throughout the entire space. This enables gradient optimization to generate safe, collision-free and smooth 3D flight trajectories. Unlike static obstacles, dynamic obstacles are modeled using a lightweight 2D ESDF under a planar-motion assumption. In our simulation, moving obstacles are vertically extended objects whose motion is restricted to the horizontal plane.

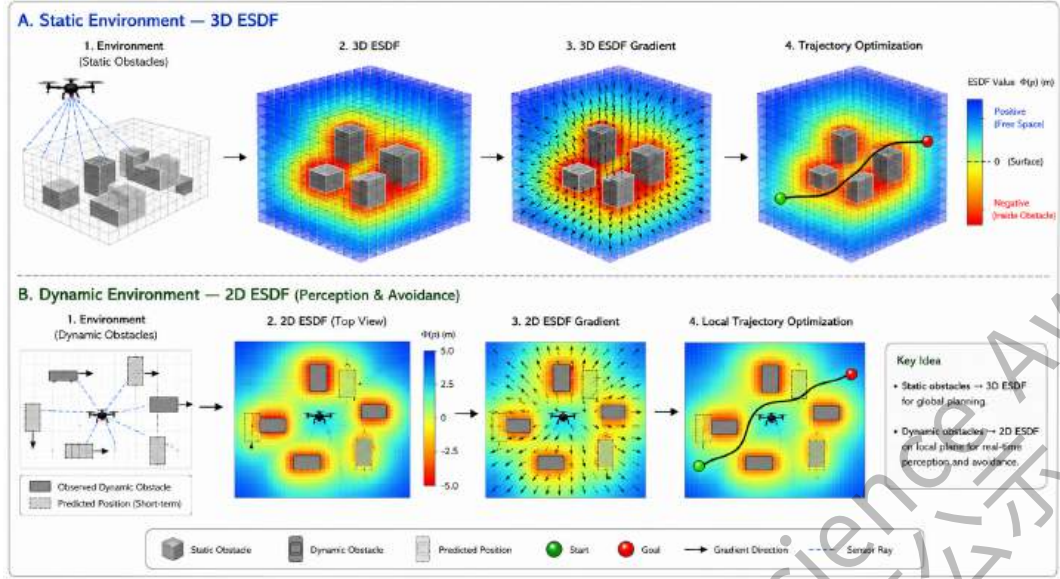


Figure 18: Illustration of ESDF-based obstacle avoidance. A binary occupancy map is transformed into a continuous signed distance field, where positive, zero, and negative values represent free space, obstacle boundaries, and obstacle interiors, respectively. Spatial ESDF gradients provide repulsive directions for deforming an unsafe initial trajectory into a smooth collision-free trajectory with sufficient obstacle clearance.

Since the LiDAR observations are relatively sparse along the vertical direction, explicitly estimating the vertical motion of dynamic obstacles is both unreliable and unnecessary under this setting. We therefore project their occupied regions onto the local horizontal plane and assume that each obstacle occupies the relevant flight-height range above its projected footprint. The resulting 2D ESDF is updated at each planning step according to the obstacle motion states and is used for high-frequency local avoidance. This approximation reduces memory and computation, at the cost of not explicitly modeling dynamic obstacles with significant vertical motion. The two-dimensional modeling method effectively cuts down memory usage and computational delay, while retaining key spatial features for fast local obstacle avoidance. The derived 2D distance and gradient data support high-frequency trajectory adjustment to evade sudden dynamic obstacles, and the matched 3D ESDF ensures stable and consistent geometric constraint effects on static environments. The combination of dual-dimensional ESDF structures finally forms an efficient hybrid safety modeling scheme, which well adapts to the actual navigation requirements of UAVs in complex mixed scenarios.

4.4 Dual-Frequency Inference Pipeline

The dual-frequency planner runs the fast and slow subsystems at different update rates. The fast subsystem operates at 10, Hz for local obstacle avoidance, while the slow subsystem runs at 2, Hz to detect dead-end states and provide longer-horizon trajectory guidance. At each planning cycle, the current LiDAR point cloud is projected into a range image. Five consecutive range images are used as the temporal observation, together with the current UAV state and navigation goal. At each fast-system update, the CNN predicts the terminal states and costs of the $N = 10$ motion primitives defined in Section 4.1. The corresponding trajectories are generated from the current UAV state using the quintic-polynomial formulation described previously. Candidates that violate the motion constraints are discarded, and the valid trajectory with the lowest predicted cost is selected as the fast-system output τ_t^f . The selected trajectory is then passed to the low-level controller, while the fast subsystem replans from the latest observation at the next update.

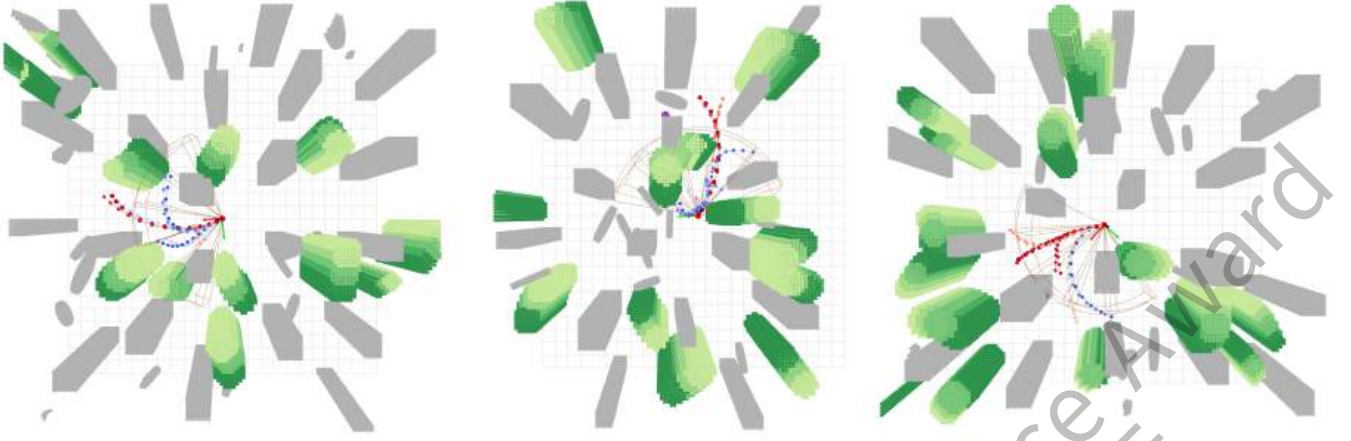


Figure 19: Real-time Drone Path Inference and Trajectory Visualization

The low-frequency slow subsystem is implemented as a task-specific, history- and memory-aware vision-language model. It receives five consecutive multimodal observations, the current UAV state, and the world-frame goal position. Temporal embeddings preserve the order of the five observations, while 16 recurrent memory tokens carry navigation information across consecutive slow-system updates. The complete slow subsystem, including the modality-projection layers, 24-layer multimodal Transformer, constrained text decoder, and conditional Flow Matching decoder, contains approximately 0.5 billion trainable parameters. The constrained text decoder generates one of two valid text outputs, `Dead End` or `Normal`. Rather than treating this output as an isolated auxiliary prediction, NavFlow-VL uses its sequence likelihood as the semantic input to the gating module. Under a normal navigation state, the gate selects the high-frequency fast trajectory. When the text likelihood of `Dead End` exceeds the switching threshold, the gate selects the memory-conditioned slow trajectory, allowing the UAV to maintain a consistent escape direction after the entrance or boundary of the dead-end branch is no longer visible in the current observation.

The overall dual-system reasoning progress could be conclude as: environment perception code $\rightarrow$ high frequency fast system local trajectory prediction + low frequency slow system global semantic perception and Flow Matching trajectory generation $\rightarrow$ VLM dead zone state determination $\rightarrow$ Gate-controlled adaptive trajectory selection $\rightarrow$ trajectory collecting and kinetic improvements $\rightarrow$ bottom layer controller execution tracking. This builds a E2E "Perception—Semantic Decision—Trajectory Generation—Execution" closed loop navigation system w=that could fulfill real-time dynamic obstacle avoidance and global dead zone escape at the same time. Furthermore, ESDF constraints are fully embedded in the dual-system cost evaluation framework, continuously supervises the ability of dynamic obstacle avoidance, and further improves the safeness of dual-trajectory output and environmental generalization ability.

Formally, the inference process can be expressed as:

$$\tau_t^f = \arg \min_{\tau \in \mathcal{G}(F_\theta(\mathbf{R}_{t-4:t}, \mathbf{o}_t), \mathcal{P})} s_\tau, \quad (29)$$

For the slow subsystem, the normalized sequence score of a candidate text $\mathbf{y}$ is defined as

$$S_t(\mathbf{y}) = \frac{1}{|\mathbf{y}|} \sum_{\ell=1}^{|\mathbf{y}|} \log p_\theta(y_\ell | y_{<\ell}, \mathbf{h}_t^{\text{text}}). \quad (30)$$

The dead-end confidence is then calculated from the two valid text candidates:

$$q_t = \frac{\exp S_t(\text{Dead End})}{\exp S_t(\text{Dead End}) + \exp S_t(\text{Normal})}. \quad (31)$$

The gating module maintains a discrete planning mode $m_t \in \{f, s\}$, corresponding to the fast and slow subsystems, respectively. The mode is updated using a hysteresis rule:

$$m_t = \begin{cases} s, & q_t \geq \delta_{\text{on}}, \\ f, & q_t \leq \delta_{\text{off}}, \\ m_{t-1}, & \delta_{\text{off}} < q_t < \delta_{\text{on}}, \end{cases} \quad \delta_{\text{on}} > \delta_{\text{off}}, \quad (32)$$

where q_t denotes the confidence that the UAV is currently in a dead-end state. The two thresholds define a hysteresis interval: switching from the fast subsystem to the slow subsystem requires q_t to exceed δ_{on} , whereas switching back to the fast subsystem requires q_t to fall below δ_{off} . When q_t lies between the two thresholds, the previously selected planning mode is retained.

The executable trajectory is then selected according to the current planning mode:

$$\tau_t^{\text{exec}} = \begin{cases} \tau_t^s, & m_t = s, \\ \tau_t^f, & m_t = f, \end{cases} \quad (33)$$

where τ_t^f and τ_t^s denote the trajectories produced by the fast and slow subsystems, respectively. The hysteresis prevents frequent switching when the dead-end confidence fluctuates near the decision boundary.

5 Experiments

5.1 Experiment Environment

All training and evaluation experiments are carried out on two distinct computing resources for different model tasks. Our NavFlow-VL is trained and validated on an on-campus laboratory GPU server equipped with 64 CPU cores, 2 NVIDIA RTX 3090Ti GPUs and 256 GB system RAM. To support the training of large vision-language multimodal models, we additionally rented an 4×NVIDIA H20 cloud server from Tencent Cloud for a period of two months; each H20 GPU carries 96 GB dedicated video memory to accommodate the large parameter size of the VLM backbone. Both servers run Ubuntu 20.04 LTS as the operating system. The unified software stack for model training consists of Python 3.10, PyTorch 2.0.1 and CUDA 11.7, a widely validated stable combination compatible with both RTX 30 series and H20 GPUs.

For data collection and simulation-based evaluation, we use a lightweight closed-loop simulator developed from the EGO-Planner ecosystem. All quantitative results reported in Section 5 are obtained using this simulator. Gazebo was used only for preliminary scene construction and visualization and was not included in the reported data-generation or quantitative-evaluation pipeline. Cross-simulator validation in Gazebo is left for future work. The simulated environment covers 50m × 50m and contains maze walls, static obstacles, and moving obstacles. The dynamic-obstacle velocity is set to 1.25m/s, while the nominal UAV flight velocity is 3.0m/s. EGO-Planner is included as an optimization-based baseline. The dataset-generation pipeline runs on a local Ubuntu workstation, where pre-built maps and

a kinodynamic A* planner are used to generate reference trajectories for maze navigation and dynamic-obstacle avoidance.

The proposed planner consists of a high-frequency fast subsystem and a low-frequency vision-language slow subsystem. A trajectory-prediction head estimates the terminal position offset, velocity, acceleration, and trajectory cost of each primitive. Each primitive covers a 3 s horizon and is represented by 30 trajectory points. The fast subsystem contains approximately 11.2 million trainable parameters and is scheduled at 10Hz in simulation. The slow subsystem receives five consecutive multimodal observations containing range images, local point-cloud representations, UAV state measurements, and the navigation goal. Lightweight modality-specific projection layers map these inputs into a shared token space with a hidden dimension of 1280. Temporal embeddings preserve the ordering of the five observations, while 16 recurrent memory tokens transfer task-relevant information across consecutive planning cycles. The multimodal Transformer contains 24 layers, 20 attention heads, and a feed-forward dimension of 5120. A constrained autoregressive text decoder generates Dead End or Normal, and a four-layer conditional Flow Matching decoder generates 30 three-dimensional waypoints over the next 3 s using 10 fixed-step ODE integration steps. The complete slow subsystem contains approximately 0.5 billion trainable parameters and is scheduled at 2 Hz in simulation.

5.2 Evaluating NavFlow-VL on Maze Environment

In order to compare the advantage and disadvantage of different path-finding algorithm, we designed a maze to test UAVs that apply some prevalent algorithm such as FASTER, MINCO and MPPI. By removing specific percentage of walls and obstacles randomly to creates different maps in order to regulate the variability of the test environment properly and eliminate the possibility of existing element of chance in experimental samples.

In many selective environment, the maze-like map can show difference directly and it is also easy to build. Fig.20 shows the performance of our samples. In this graph, UAVs with FASTER, MINCO, MPPI and our algorithm were tested inside a maze-like map that is reduced by 10% walls, reduced by 20% walls, and reduced by 30% walls. FASTER can generate feasible trajectories in relatively simple maze environments. However, because its JPS-based path search operates on the currently available local occupancy map, its planning decisions are limited by the sensing horizon and partial environmental observations. In long, winding corridors or complex maze layouts where the complete route is not visible, the planner may repeatedly select locally favorable branches and replan only after encountering a blockage. This local-map-based planning mechanism can therefore result in myopic decisions and increase the risk of entering dead ends or becoming trapped in locally unfavorable regions.

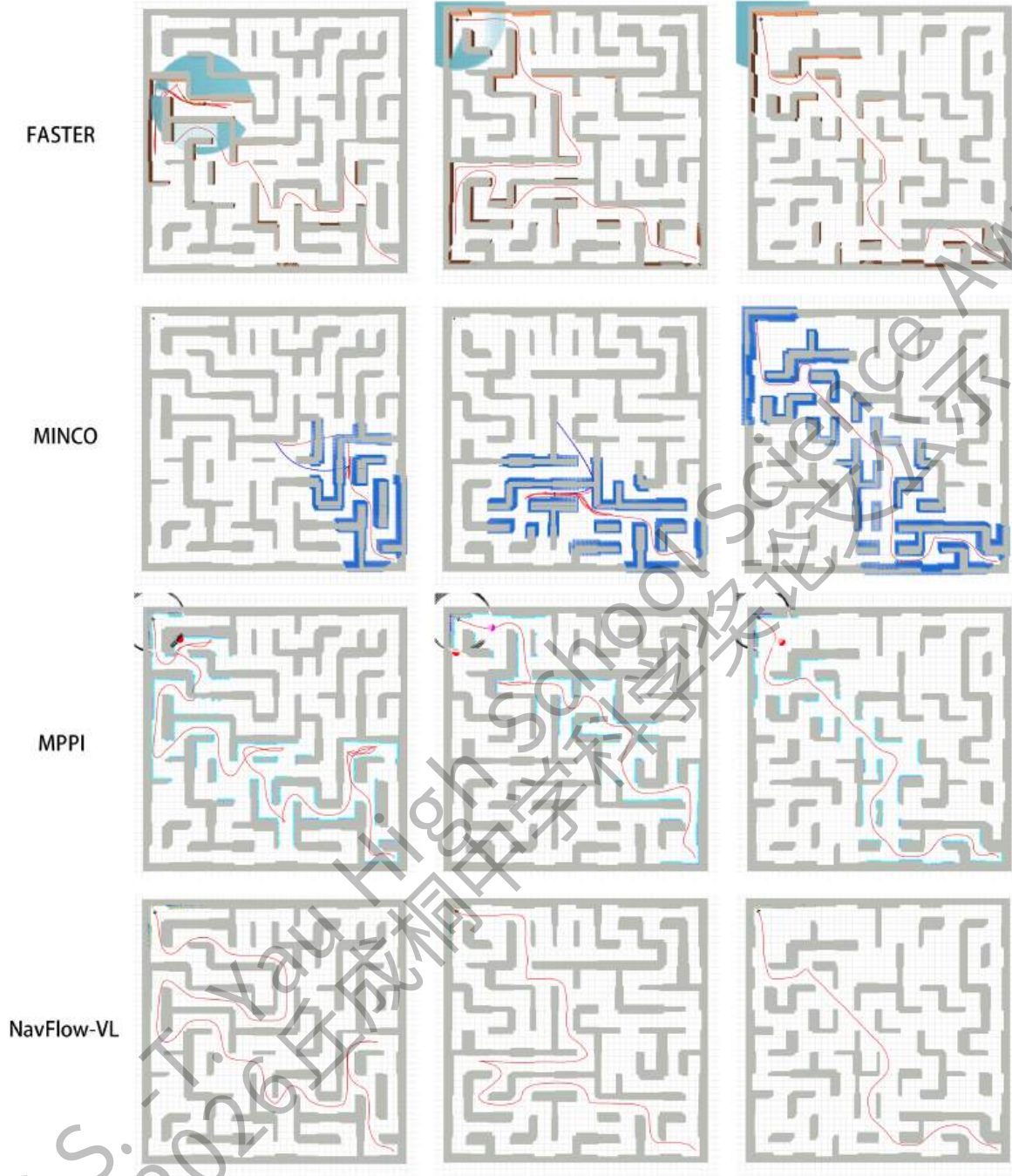


Figure 20: Evaluation of FASTER, MINCO, GP-MPPI, and NavFlow-VL in maze environments. The UAV navigates between two diagonally opposite corners under wall-removal probabilities of 10%, 20%, and 30%.

In our maze experiments, the MINCO-based baseline exhibited more frequent planning oscillations and early navigation failures than FASTER. This behavior is not primarily caused by MINCO's preference for smooth trajectories, since MINCO mainly provides an efficient polynomial trajectory representation and optimization framework. Instead, the limitation mainly arises from the local A*-based front end used to generate the guiding path in our implementation. Because A* is repeatedly executed on a local occupancy grid that changes with the UAV's field of view, the resulting guidance path may vary significantly between consecutive replanning cycles. In winding corridors, intersections, and dead-end

regions, newly observed obstacles can cause the planner to switch repeatedly between nearby candidate directions, resulting in oscillatory replanning, repeated backtracking, or hesitation near the entrance of a dead end. Although the optimized trajectory may remain locally smooth, it lacks persistent long-horizon guidance and may therefore fail to establish an effective escape direction. In several difficult cases, the mismatch between the discrete A* guidance path and the continuously optimized trajectory also produced insufficient obstacle clearance. Under limited map resolution or inadequately weighted collision penalties, this could lead to trajectories that pass excessively close to, or even collide with, obstacle boundaries.

GP-MPPI performs better than FASTER and the MINCO-based baseline in several maze scenarios. Rather than selecting a navigation direction through purely random exploration, GP-MPPI first uses a Sparse Gaussian Process to model the locally observed navigable space and recommend a promising subgoal. The selected subgoal is then passed to the sampling-based MPPI controller, which generates a locally feasible control trajectory while accounting for obstacle avoidance and vehicle dynamics. However, the subgoal recommendation remains dependent on the currently available local observations. In our maze experiments, when the UAV turned around or moved away from a dead-end boundary, previously observed structures could leave the sensing field of view. The updated Gaussian-process model could then recommend a different locally favorable subgoal, causing the planner to alternate between nearby directions. This behavior resulted in oscillatory steering, repeated backtracking, or delayed escape in some enclosed and winding regions. In a few difficult cases, the locally selected trajectory also provided insufficient clearance from obstacle boundaries. **In contrast, NavFlow-VL retains historical semantic information after previously observed regions leave the current field of view. Its slow system can therefore maintain a more consistent escape intention, while the fast system handles local trajectory execution and obstacle avoidance. As a result, NavFlow-VL produces smoother trajectories with fewer unnecessary detours and demonstrates more reliable dead-end escape across mazes of different difficulty levels.**

Table 1 concentrates on two indicators: **Success Rate** and **Average Trajectory Length**. These two indicators are used to demonstrate the performance of different algorithms when faced with maze-like map with different difficulty. We consider maze-like maps with wall reduction ratios $r \in \{0.1, 0.2, 0.3\}$, which represent wall reductions of 10%, 20% and 30%. Among the previous algorithms, FASTER can maintain a relatively high success rate and short trajectory length. In terms of success rate, FASTER can always stays over 50%, in comparison, GP-MPPI performs a perfect rate when facing simple map, but as the difficulty increases, its success rate drop rapidly and cannot stays in a specific range like FASTER does. In terms of trajectory length, FASTER and MPPI achieve comparable performance, although MPPI generates slightly longer trajectories. However, MINCO exhibits poor performance in both success rate and trajectory length. For success rate, while MINCO shows a trend similar to MPPI, its overall success rate remains at a low level. Due to its instability in avoiding obstacles and navigating, MINCO's trajectory length planning is also unsatisfactory, even when facing the simplest maze, it's trajectory length also exceeds 100. The last row is our proposed algorithm, NavFlow-VL. According to the sheet, NavFlow-VL did a great job in both success rate and trajectory length, and surprisingly, its success rate has not appear considerable decay when facing mazes with different r value, which all of the previous 3 algorithms appears linear decay when r decreases. In Trajectory length, NavFlow-VL exhibits competitive performance compared with existing path-finding algorithms. The trajectory length generated by our method stays consistently under 100 meters, which verifies the effectiveness of NavFlow-VL.

Table 1: Comparison of Success Rate [%] and Average Trajectory Length [m] across Different Environmental Complexities.

Method	$r : 0.3$		$r : 0.2$		$r : 0.1$	
	Suc. Rate [%]	Traj. Len. [m]	Suc. Rate [%]	Traj. Len. [m]	Suc. Rate [%]	Traj. Len. [m]
FASTER [18]	91	94.33	77	98.40	64	107.66
MINCO [17]	63	108.91	41	112.38	26	135.24
GP-MPPI [19]	84	99.67	56	109.64	35	111.32
Ours	100	86.35	98	89.75	99	96.21

5.3 Evaluating NavFlow-VL on Forest Dynamic Obstacle Environment

The performance of different path-finding algorithms can also be observed in forest-like experimental field. Fig.21 shows the trajectory generated by different algorithms in a forest-like map. In this figure, purple, green, blue, yellow, and red line represents Ego-Planner, MINCO, NavRL, YOPO, and NavFlow-VL algorithm, respectively. In our dynamic-obstacle experiments, EGO-Planner occasionally produced trajectories with sharp local turns due to frequent collision detection and replanning. These abrupt changes increase the control effort and may reduce tracking accuracy. MINCO exhibited different failure modes in maze and dynamic-obstacle environments. In maze-like dead-end regions, repeated replanning on a locally maintained map could lead to oscillation and prolonged wandering when no consistent feasible direction was found. In dynamic environments, failures were mainly caused by the collision-detection and replanning delay: replanning was triggered only after a moving obstacle intersected the current planned trajectory, which sometimes left insufficient time for avoidance when the obstacle approached rapidly. By comparison, NavRL and YOPO's trajectory is more stable when passing among obstacles. Because of the high calculating efficiency, YOPO generated relatively smooth trajectories in the evaluated scenarios. However, because it performs planning from single-frame observations, it lacks explicit temporal modeling of dynamic obstacle motion and cannot anticipate their future trajectories. As a result, avoidance is often triggered only after a moving obstacle enters the current observation or conflicts with the predicted path, which may lead to delayed reactions and abrupt local trajectory adjustments. NavRL demonstrated strong active avoidance capability in dynamic environments, benefiting from its training in scenarios containing a large number of moving obstacles. Unlike the other baseline methods, it could proactively adjust the UAV's motion before an imminent collision occurred. However, because NavRL directly outputs control actions through reinforcement learning, its behavior sometimes became conservative in densely cluttered regions. The UAV occasionally slowed down, stopped, or moved backward while searching for a safe passage, which increased the overall navigation time.

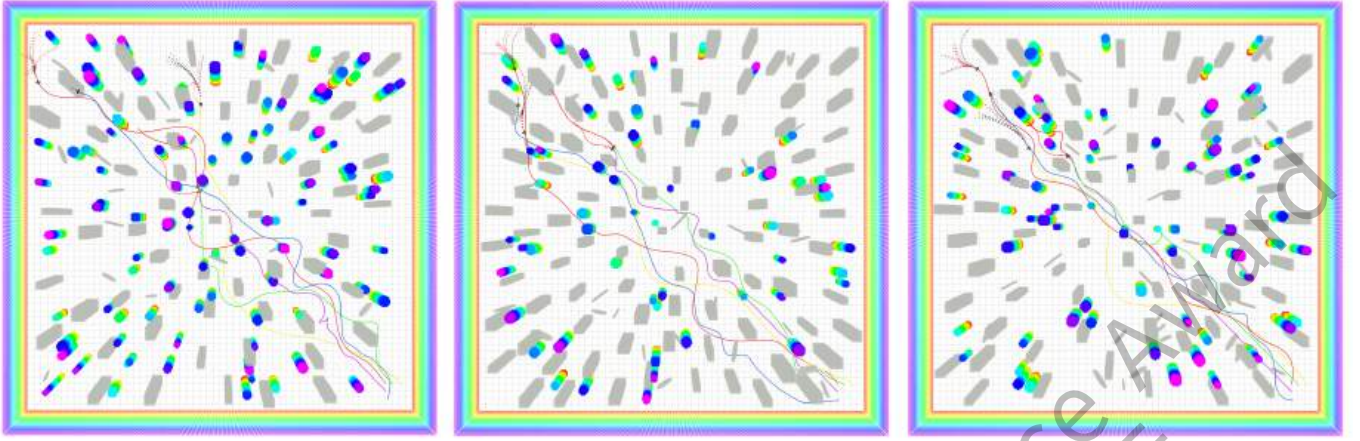


Figure 21: Evaluating Different Algorithm on Forest Dynamic Obstacle Environment (Purple: Ego-Planner [16], Green: MINCO[17], Blue: NavRL[27], Yellow: YOPO[26], Red: NavFlow-VL)

Table 2: Performance Comparison under Varying Environmental Dynamism Levels

Dynamism	Method	Success Rate [%]	Safe Dist. [m]	Traj. Length [m]	Curvature [1/m]	Flight Time [s]
120 static 60 dynamic	EGO [16]	28	0.46	78.34	358.51	35.81
	MINCO [17]	34	0.40	69.21	183.76	30.57
	YOPO [26]	51	0.65	80.83	105.53	37.16
	NavRL [27]	92	0.72	73.59	156.82	32.24
	Ours	97	0.76	75.88	83.07	31.98
100 static 80 dynamic	EGO [16]	11	0.34	77.85	426.37	38.43
	MINCO [17]	14	0.51	71.28	162.02	34.67
	YOPO [26]	36	0.62	78.90	127.61	40.35
	NavRL [27]	67	0.59	70.63	192.44	33.82
	Ours	84	0.75	74.86	79.31	32.80
80 static 100 dynamic	EGO [16]	0	—	—	—	—
	MINCO [17]	4	0.41	70.94	149.02	31.75
	YOPO [26]	23	0.67	82.53	126.83	38.30
	NavRL [27]	45	0.63	74.28	170.55	33.57
	Ours	68	0.78	76.31	86.45	31.62

Table.2 shows different algorithms' performance in different ratio of static and dynamic obstacle environment. We judge the algorithms' ability level on 5 aspects: **Success Rate**, a possibility that a sample that will not have an accident while passing through this environment; **Safe Distance**, the smallest average distance between the obstacles and the sample during the flying process; **Trajectory Length**, the length of the trajectory designed by these algorithm; **Curvature**, refers to the degree of tortuous of the trajectory; and the **Flight Time**, is defined as the time required for the UAV to travel from the starting point to the destination. Among the previous four algorithms, NavRL algorithm received the highest success rate whether the environment condition is hard, and lead an interruption layer among all previous algorithms. In determining the safety distance, YOPO and NavRL algorithms are neck and neck, when dynamic obstacles are rare, NavRL achieved better overall performance than YOPO in environments with moving obstacles. Among the evaluated baseline methods, NavRL was the only planner that consistently demonstrated active avoidance of dynamic obstacles, benefiting from its reinforcement-learning policy trained in highly dynamic scenarios. YOPO occasionally produced smoother trajectories because its polynomial trajectory parameterization favors motion continuity. However, its single-frame prediction mechanism lacks explicit temporal modeling of obstacle motion, which limits its ability to anticipate moving obstacles. Therefore, although YOPO may achieve better trajectory smoothness in

some cases, NavRL remains more reliable in environments with a high density of dynamic obstacles. Trajectory length is also an essential indicator, algorithms with less trajectory length means it can save time. MINCO gain the shortest trajectory time in most environment factor, and also be awesome in the remaining environment.

Although trajectory smoothness is a common objective in optimization-based planning methods, YOPO produced smoother trajectories mainly because its polynomial trajectory prediction tended to generate more conservative local motions. In contrast, EGO-Planner and MINCO lacked explicit modeling of dynamic obstacle motion in our implementation and therefore could not respond reliably to moving obstacles. YOPO's conservative predictions reduced abrupt trajectory changes, but its single-frame planning mechanism still limited its ability to anticipate future obstacle motion. According to the data shown in the table, YOPO is the only one among the 4 previous algorithm with a curvature under 150 per meter. When UAVs have similar cruising speed, the flight time is directly related to trajectory length. In fact, MINCO, who gained the No.1 in trajectory length, has great advantage in managing its flight time, and is also ranked at the top this term. Our algorithm, NavFlow-VL, behaves much better and more stable compared to previous 4 algorithms. It has the highest successful rate, and can even stays over 50% when facing complex environment with many dynamic obstacles. Additionally, Our algorithms usually stays over 0.7 meters from the nearest obstacles, ensuring the UAV's safety when shuttling in a weird terrain. Although we can't find the shortest path, our algorithm is the only one with a curvature under 100 per meter in every environment, the great advantage on curvature help guarantee that the UAV won't deviate too much, so that our algorithm still act a great score in the ultimate flight time.

5.4 Evaluating NavFlow-VL on Maze Dynamic Obstacle Environment

To evaluate navigation robustness under combined geometric and dynamic challenges, we enabled the dynamic-obstacle layer in the same procedurally generated maze environment described in Section 3.1. In this setting, the UAV must avoid dynamic obstacles while simultaneously navigating through narrow passages, intersections, and dead-end regions. The task therefore tests both short-term collision avoidance and long-horizon escape capability. In our implementation, FASTER and GP-MPPI relied on locally constructed occupancy representations without explicit modeling of dynamic obstacle motion, and thus could not respond reliably to moving obstacles. MINCO had already shown limited performance in the forest environment and also lacked an effective mechanism for dead-end recognition and escape. Consequently, these methods often failed to reach the destination in the combined maze and dynamic obstacle scenario. The red trajectory in Fig.22 and Fig.23 is created by NavFlow-VL algorithm, because of the interference of dynamic obstacles, the trajectory has some small sharp turn, especially in narrow space such as corner. Although several local turns can be observed in the trajectory, they mainly occur when the UAV responds to moving obstacles in narrow passages and are therefore expected under the tested conditions. These adjustments remain within the vehicle's feasible motion range and do not lead to sustained oscillation or loss of control. Overall, the trajectory maintains reasonable continuity and obstacle clearance. Since the planner relies on local observations, it may initially enter an unfavorable passage before the dead-end structure becomes visible; however, after detecting the blockage, NavFlow-VL can use its historical context to maintain a consistent escape direction and avoid repeatedly returning to the same failed route.

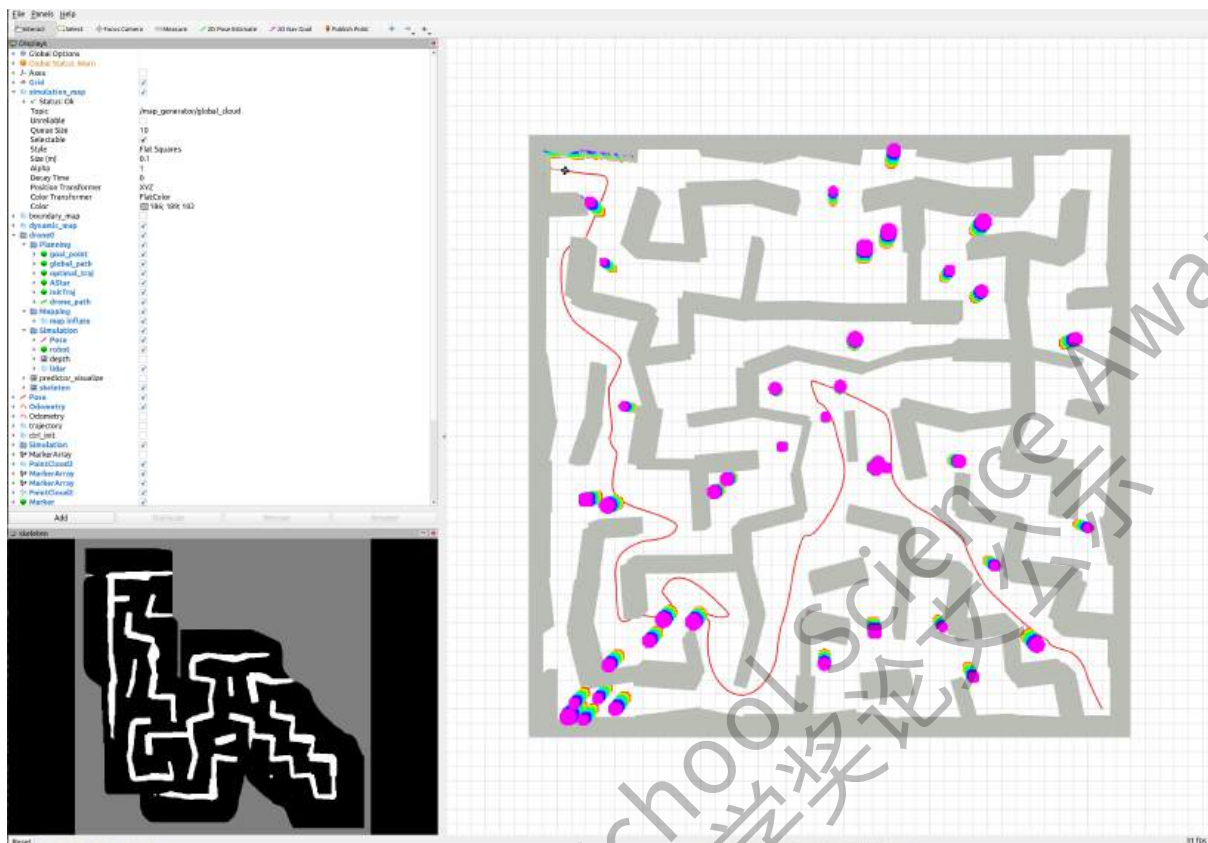


Figure 22: An Example of Evaluating NavFlow-VL on Irregular Geometry Maze Dynamic Obstacle Environment

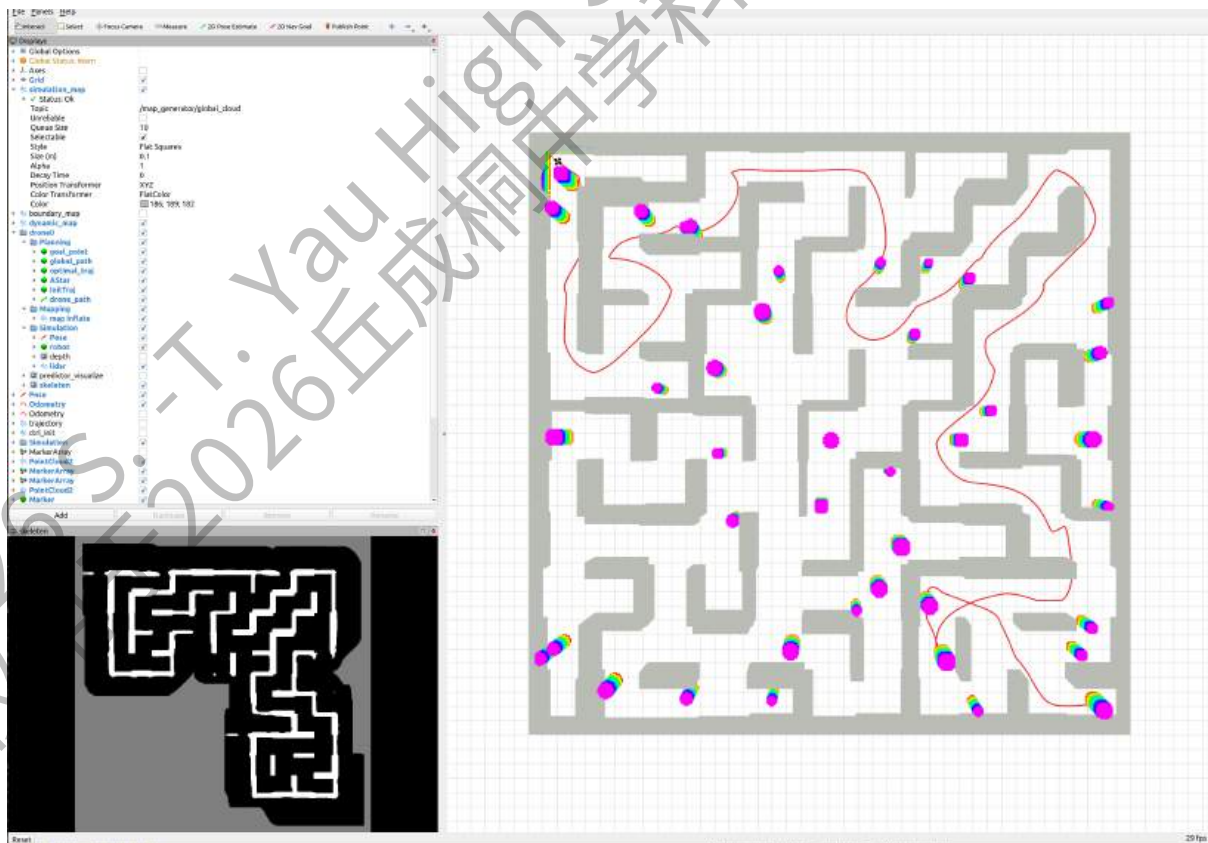


Figure 23: An Example of Evaluating NavFlow-VL on Regular Geometry Maze Dynamic Obstacle Environment

6 Conclusion

This paper presents NavFlow-VL, a dual-system navigation framework for UAVs operating in maze-like environments containing static structures and moving obstacles. The work is motivated by a limitation of many local planners: although they can respond effectively to nearby obstacles, they may repeatedly select locally feasible directions when only partial observations are available. This behavior becomes particularly problematic in long corridors, enclosed branches, and dead ends, where successful navigation requires both immediate collision avoidance and the use of historical environmental information. To address these requirements, NavFlow-VL combines a high-frequency reactive planner with a low-frequency reasoning and trajectory-generation module.

The high-frequency fast subsystem uses a modified ResNet-18 backbone to process five consecutive LiDAR range images together with the current UAV state and world-frame goal direction. It predicts the endpoint states and costs of 10 motion primitives, each covering a future horizon of 3 s. Running at 10 Hz, this subsystem is responsible for local trajectory adjustment and rapid obstacle avoidance. The low-frequency slow subsystem is implemented as a task-specific vision-language model with approximately 0.5 billion trainable parameters. It processes five consecutive multimodal observations with temporal embeddings and maintains 16 recurrent memory tokens across planning cycles. A constrained autoregressive text decoder generates the navigation-state text `Dead End` or `Normal`, while a conditional Flow Matching decoder generates 30 three-dimensional waypoints over the next 3 s. The likelihood of the generated text is used by the gating module to select between the fast reactive trajectory and the memory-conditioned slow trajectory.

The two subsystems are trained with different forms of supervision. The fast subsystem does not require reference-trajectory labels. Instead, static 3D ESDF and dynamic 2D ESDF maps constructed from privileged simulator information are used to evaluate the safety of candidate trajectories during training. Collision, smoothness, and goal-guidance costs are back-propagated through the trajectory prediction network. The slow subsystem is trained using reference trajectories generated by a kinodynamic A* planner. In addition to trajectory supervision, a simplified semantic annotation procedure assigns each training sequence one of two text labels, `Dead End` or `Normal`. The resulting auxiliary semantic objective is optimized together with the trajectory-generation objective. This design encourages the slow subsystem to associate temporal observation patterns with trapped states and corresponding recovery behavior without requiring detailed natural-language descriptions of every scene. The complete map and other privileged simulator information are used only for offline data generation and supervision and are not provided to the planner at inference time.

A closed-loop simulation and data-generation platform was developed to support training and evaluation in structured mazes, unstructured forests, and environments containing moving obstacles. The platform includes narrow passages, terminal branches, irregular maze walls, and dynamic obstacles with randomized motion. It also maintains physical collision boundaries for visually transparent structures, preventing unrealistic obstacle penetration during data collection and evaluation. These settings allow the fast and slow subsystems to be examined under both local collision-avoidance tasks and navigation problems that require dead-end recognition and route reversal. In the static-maze benchmark, NavFlow-VL achieved success rates of 100%, 98%, and 99% at wall-removal ratios of $r = 0.3$, 0.2, and 0.1, respectively. The corresponding average trajectory lengths were 86.35 m, 89.75 m, and 96.21 m. In the dynamic-forest experiments containing 60, 80, and 100 moving obstacles, the method achieved success rates of 97%, 84%, and 68%, while maintaining average minimum obstacle clearances of 0.76 m, 0.75 m, and 0.78 m. The experiments in dynamic maze environments further provide qualitative examples in which the UAV enters an unfavorable branch, recognizes the trapped state from its recent obser-

vation history, and maintains a consistent escape direction while responding to moving obstacles. Taken together, the simulation results suggest that separating rapid local response from slower history-aware planning can improve navigation consistency as environmental complexity increases.

Future work will focus first on deploying NavFlow-VL on the candidate quadrotor platform equipped with an NVIDIA Jetson Orin NX and a Livox Mid-360 LiDAR. The initial tests will examine sensor calibration and synchronization, onboard inference latency, controller integration, and trajectory-tracking accuracy under protected indoor conditions. The evaluation will then be extended to outdoor environments after the reliability of the complete system has been established. We also plan to expand the simplified semantic annotation scheme, improve the temporal memory and gating mechanisms, and reduce the computational cost of the slow subsystem through model compression and inference optimization. These steps are necessary to determine whether the advantages observed in simulation can be retained under perception noise, control error, and limited onboard computing resources.

References

- [1] Y. Tian, F. Lin, Y. Li, T. Zhang, Q. Zhang, X. Fu, J. Huang, X. Dai, Y. Wang, C. Tian, *et al.*, “Uavs meet llms: Overviews and perspectives towards agentic low-altitude mobility,” *Information Fusion*, vol. 122, p. 103158, 2025.
- [2] G. Cai, K.-Y. Lum, B. M. Chen, and T. H. Lee, “A brief overview on miniature fixed-wing unmanned aerial vehicles,” *IEEE ICCA 2010*, pp. 285–290, 2010.
- [3] R. Rashad, J. Goerres, R. Aarts, J. B. Engelen, and S. Stramigioli, “Fully actuated multirotor uavs: A literature review,” *IEEE Robotics & Automation Magazine*, vol. 27, no. 3, pp. 97–107, 2020.
- [4] D. Bie, D. Li, J. Xiang, H. Li, Z. Kan, and Y. Sun, “Design, aerodynamic analysis and test flight of a bat-inspired tailless flapping wing unmanned aerial vehicle,” *Aerospace Science and Technology*, vol. 112, p. 106557, 2021.
- [5] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.
- [6] A. Ardiansyah, A. M. Nasution, and M. Iqbal, “Comparative analysis of dijkstra and a* algorithms for determining the shortest route,” *bit-Tech*, vol. 8, no. 2, pp. 2974–2983, 2025.
- [7] F. H. Tseng, T. T. Liang, C. H. Lee, L. Der Chou, and H. C. Chao, “A star search algorithm for civil uav path planning with 3g communication,” in *2014 Tenth International Conference on Intelligent Information Hiding and Multimedia Signal Processing*, pp. 942–945, IEEE, 2014.
- [8] R. Song, Y. Liu, and R. Bucknall, “Smoothed a* algorithm for practical unmanned surface vehicle path planning,” *Applied Ocean Research*, vol. 83, pp. 9–20, 2019.
- [9] S. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” *Research Report 9811*, 1998.
- [10] J. Zhang, Y. An, J. Cao, S. Ouyang, and L. Wang, “Uav trajectory planning for complex open storage environments based on an improved rrt algorithm,” *IEEE Access*, vol. 11, pp. 23189–23204, 2023.
- [11] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, “Practical search techniques in path planning for autonomous driving,” *ann arbor*, vol. 1001, no. 48105, pp. 18–80, 2008.
- [12] S. Sedighi, D.-V. Nguyen, and K.-D. Kuhnert, “Guided hybrid a-star path planning algorithm for valet parking applications,” in *2019 5th international conference on control, automation and robotics (ICCAR)*, pp. 570–575, IEEE, 2019.
- [13] S. M. LaValle and J. J. Kuffner, “Rapidly-exploring random trees: Progress and prospects: Steven m. lavalley, iowa state university, a james j. kuffner, jr., university of tokyo, tokyo, japan,” *Algorithmic and computational robotics*, pp. 303–307, 2001.
- [14] J. Ge, L. Liu, X. Dong, and W. Tian, “Trajectory planning of fixed-wing uav using kinodynamic rrt algorithm,” in *2020 10th International Conference on Information Science and Technology (ICIST)*, pp. 44–49, IEEE, 2020.

- [15] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, "Robust and efficient quadrotor trajectory generation for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3529–3536, 2019.
- [16] X. Zhou, Z. Wang, H. Ye, C. Xu, and F. Gao, "Ego-planner: An esdf-free gradient-based local planner for quadrotors," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 478–485, 2020.
- [17] Z. Wang, X. Zhou, C. Xu, and F. Gao, "Geometrically constrained trajectory optimization for multicopters," *IEEE Transactions on Robotics*, vol. 38, no. 5, pp. 3259–3278, 2022.
- [18] J. Tordesillas, B. T. Lopez, M. Everett, and J. P. How, "Faster: Fast and safe trajectory planner for navigation in unknown environments," *IEEE Transactions on Robotics*, vol. 38, no. 2, pp. 922–938, 2021.
- [19] I. S. Mohamed, M. Ali, and L. Liu, "Gp-guided mppi for efficient navigation in complex unknown cluttered environments," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7463–7470, IEEE, 2023.
- [20] U. Challita, W. Saad, and C. Bettstetter, "Interference management for cellular-connected uavs: A deep reinforcement learning approach," *IEEE Transactions on Wireless Communications*, vol. 18, no. 4, pp. 2125–2140, 2019.
- [21] A. Singla, S. Padakandla, and S. Bhatnagar, "Memory-based deep reinforcement learning for obstacle avoidance in uav with limited environment knowledge," *IEEE transactions on intelligent transportation systems*, vol. 22, no. 1, pp. 107–118, 2019.
- [22] O. Bouhamed, H. Ghazzai, H. Besbes, and Y. Massoud, "A uav-assisted data collection for wireless sensor networks: Autonomous navigation and scheduling," *IEEE Access*, vol. 8, pp. 110446–110460, 2020.
- [23] J. Rückin, L. Jin, and M. Popović, "Adaptive informative path planning using deep reinforcement learning for uav-based active sensing," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 4473–4479, IEEE, 2022.
- [24] Y. Guo and Z. Liu, "Uav path planning based on deep reinforcement learning," *Monitoring and Control (ANMC) Cooperate: Xi'an Technological University (CHINA) West Virginia University (USA) Huddersfield University of UK (UK)*, p. 81, 2023.
- [25] L. Xu and W. Zhang, "Survey on path planning based on deep reinforcement learning," in *International Conference on Machine Learning and Intelligent Computing*, pp. 685–695, PMLR, 2025.
- [26] J. Lu, X. Zhang, H. Shen, L. Xu, and B. Tian, "You only plan once: A learning-based one-stage planner with guidance learning," *IEEE Robotics and Automation Letters*, vol. 9, no. 7, pp. 6083–6090, 2024.
- [27] Z. Xu, X. Han, H. Shen, H. Jin, and K. Shimada, "Navrl: Learning safe flight in dynamic environments," *IEEE Robotics and Automation Letters*, 2025.
- [28] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM journal on computing*, vol. 1, no. 2, pp. 146–160, 1972.

- [29] K. Qian, Z. Ma, Y. He, Z. Luo, T. Shi, T. Zhu, J. Li, J. Wang, Z. Chen, X. He, *et al.*, “Fasionad: Fast and slow fusion thinking systems for human-like autonomous driving with adaptive feedback,” *arXiv preprint arXiv:2411.18013*, 2024.
- [30] M. W. Mueller, M. Hehn, and R. D’Andrea, “A computationally efficient motion primitive for quadcopter trajectory generation,” *IEEE transactions on robotics*, vol. 31, no. 6, pp. 1294–1310, 2015.
- [31] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, *et al.*, “A vision-language-action flow model for general robot control,” *arXiv preprint arXiv:2410.24164*, 2024.
- [32] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, “Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1366–1373, IEEE, 2017.
- [33] H. Moravec and A. Elfes, “High resolution maps from wide angle sonar,” in *Proceedings. 1985 IEEE international conference on robotics and automation*, vol. 2, pp. 116–121, IEEE, 1985.

Acknowledgments

This project could not be completed successfully without the guidance of our instructors, support from our teachers and parents, and academic resources from the research community. From topic selection, data collection, experimental debugging and data analysis to paper writing and revision, we encountered many difficulties. Every progress we made relies on the help and encouragement of people around us.

First, we sincerely thank Professor Ying Gao from the School of Computer Science and Engineering, South China University of Technology. As our university supervisor, she provided professional guidance throughout our research. She helped us refine our research framework, optimize experimental designs, and standardize model training and evaluation. Whenever our research logic was unclear or experimental results were unsatisfactory, she patiently corrected our mistakes and guided us to improve our work, which guaranteed the smooth progress of this project. We sincerely appreciate Mr. Peng Gui from the Affiliated High School of South China Normal University, our school research advisor. We had learned Mr Gui since our junior high school also well-known as The Guangdong Olympic School. At that time, we both attended mathematics Olympiad training at Guangdong Olympic School, where Mr. Gui served as a coach. His rigorous logical training and patient guidance back then laid a solid foundation for our scientific research thinking. As a professional mathematics teacher, First-Class Mathematical Olympiad Coach, and Director of the Olympic School Affairs Office, he has rich experience in academic training and project guidance. During this research, we often encountered confusing experimental problems and repeated model failures. Mr. Gui always spared time to communicate with us, analyze the causes of our experimental difficulties, reorganize our paper logic, and offer valuable suggestions on experiment arrangement and academic writing. His encouragement helped us overcome research bottlenecks and persist in completing this project. We also thank our teachers and parents for providing a stable research environment, assisting us in material collection and model testing, and constantly encouraging us during difficult times.

We are grateful to researchers who have open-sourced datasets, models and codes in UAV navigation, trajectory planning, computer vision and machine learning. Their work provided important references for our project. We also thank Andrew Ng's open courses on machine learning and deep learning, which helped us build a solid theoretical foundation. In addition, GPT 5.6 Sol assisted us in polishing the schematic diagrams of this paper, including Fig.15, Fig.16 and Fig.18, making our experimental presentation clearer.

Most importantly, we specify the detailed division of labor and problem-solving experience of the two core members throughout the research. This project was fully completed by the two authors through continuous discussion, repeated experiments and self-exploration. In the early stage, we jointly reviewed existing studies and found that most current navigation algorithms struggle with maze dead-end escape and dynamic obstacle avoidance. After extensive literature research and repeated discussion, we finally confirmed our research topic: a learning-based UAV navigation algorithm adapted to complex maze scenarios.

Chenwei Zhang was mainly responsible for literature investigation, background analysis, abstract and introduction writing, as well as experimental conclusion sorting and paper polishing. She systematically sorted out domestic and foreign research progress, summarized the defects of existing algorithms, and clarified the innovation and research significance of our work. When our literature arrangement was disordered and our core innovation was unclear, she reorganized massive research materials, compared different algorithms, and finally clarified the advantages of our hierarchical dual-system navigation framework. Martin Xiao Yu Su mainly undertook the core work of experimental design, simulation en-

environment construction, dataset sorting, model debugging, data analysis, and methodology writing. He faced the most difficult and repetitive experimental tasks. In the research, we met serious problems: the model easily fell into local optima, failed to escape dead zones, and could not stably avoid dynamic obstacles, leading to continuous experiment failures. Facing these setbacks, he kept optimizing model structures, tried semantic enhancement strategies and geometric safety constraints, and conducted dozens of iterative training and comparison experiments. Finally, he effectively improved the model's dead-zone escape ability and dynamic adaptation, and completed all data analysis and result verification.

Furthermore, all key research decisions, scheme adjustments, result verification and final revisions were completed and discussed by both authors. We cooperated closely, complemented each other's strengths, and solved various difficulties together. When data was messy or paper logic was confusing, we sorted out rules, revised content and optimized the overall structure side by side to ensure the quality and integrity of this work.

Finally, we sincerely thank all teachers, families and researchers who have supported this project directly or indirectly. All the guidance and encouragement we received were essential to the successful completion of our research.

Appendix

A. Kinodynamic A* Based Data Generation

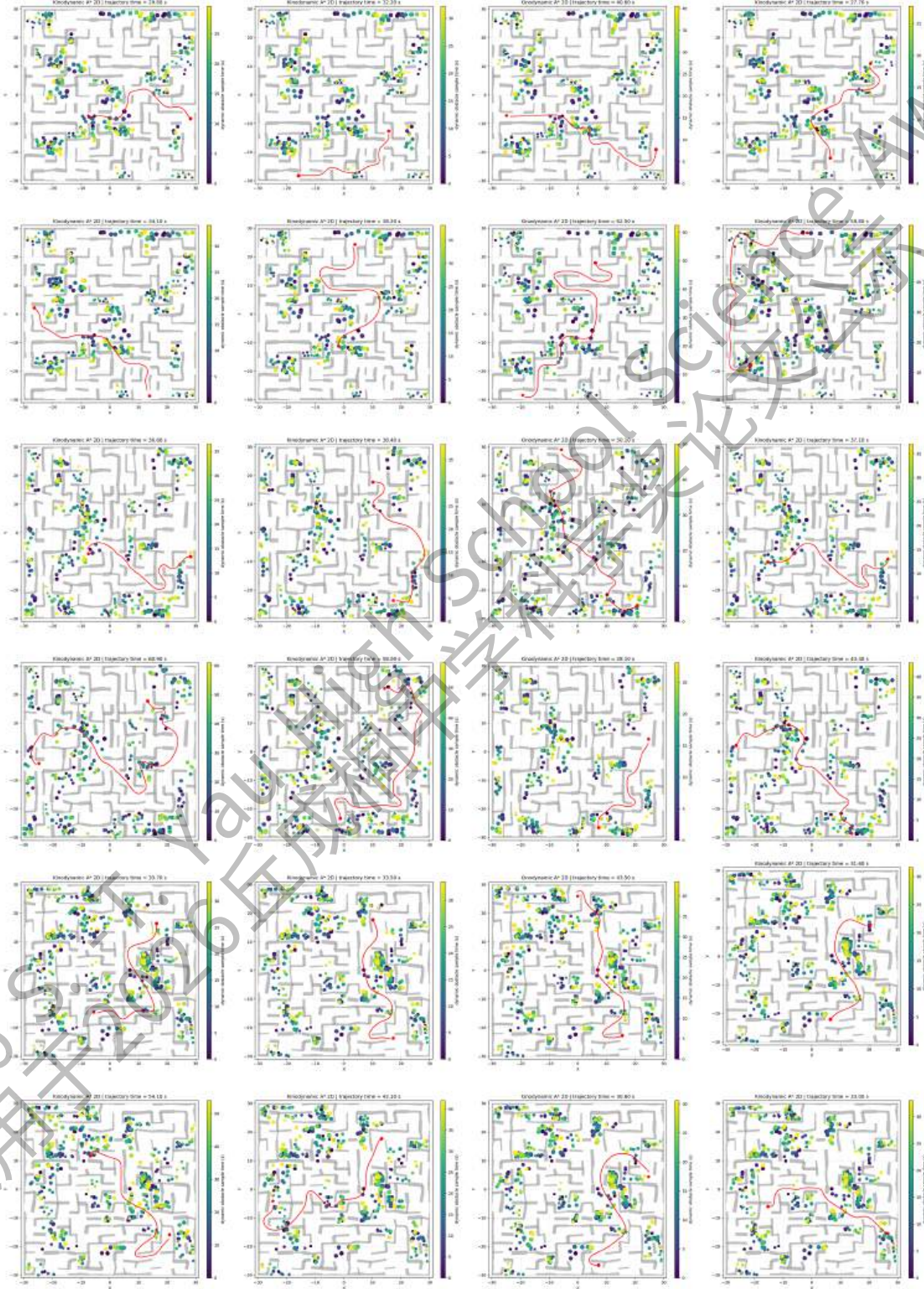


Figure 24: Examples of Kinodynamic A* Based Data Generation I

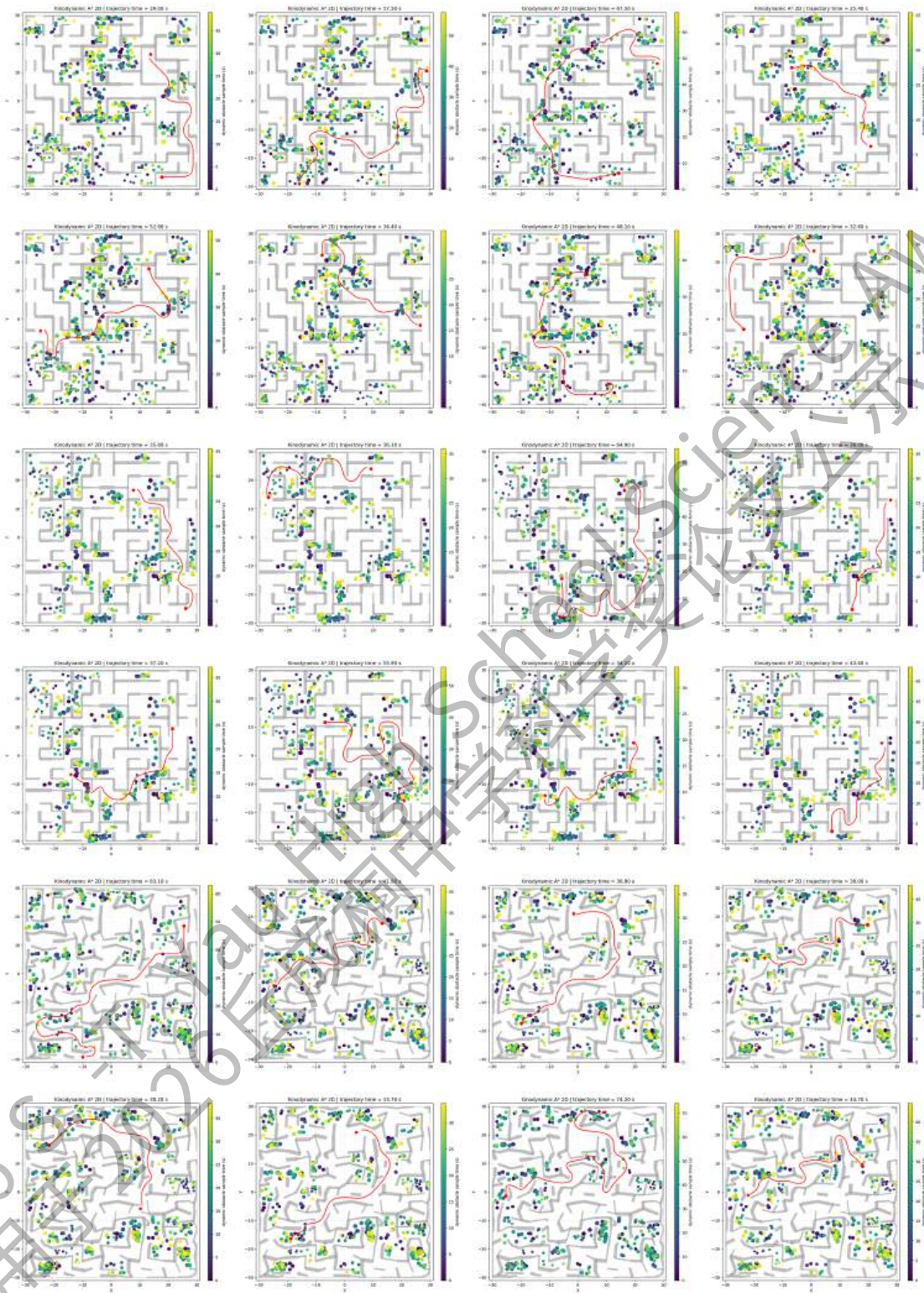


Figure 25: Examples of Kinodynamic A* Based Data Generation2

MARTIN XIAO YU SU(苏晓渝)

The Affiliated High School of South China Normal University

+86 18922198466 martinsu20100623@gmail.com

From robotic cleaners to drones that used in large-scale events like ceremonies and rituals, Artificial Intelligence permeates almost every corner of our life. In this world where AI technology is flourishing, it is essential to have the ability to reach specific goals though AI technology. This aim encourages me to continuously study in order to use this ability appropriately.



HONORS & ACADEMICS

<i>AP Computer Science A: 5/5</i>	<i>2026</i>
<i>AMC 10, Top 5% among all testers worldwide</i>	<i>2025</i>
<i>USACO Gold Award</i>	<i>2024-25</i>
<i>Second Prize of 2024 CSP-J</i>	<i>2024</i>
<i>2nd Prize of Guangdong in National Astronomy Olympiad</i>	<i>2023</i>
<i>Grand Honor in JHU Center for Talented Youth</i>	<i>2023</i>

ACTIVITIES & PROJECTS

<i>The Young Computer Scientist Summer Experience in UoL</i>	<i>2025</i>
<i>Astrophysics and Data Analysing Activities in Johns Hopkins University (JHU)</i>	<i>2023</i>

ABILITIES & STRENGTHS

Basic C++/ Java programming skills

*An astrophile,a book **Space** written in ten years old*

Intermediate Chess player

章晨蔚 VIVIAN ZHANG

The Affiliated High School of South China Normal University

+86 15918519056 zhangcw.vivian2025@gdhfi.com

In the school, I can see UAVs moving back and forth between the buildings in various activities, such as school sports day and graduation ceremony. In daily activities, artificial intelligence appears in almost every aspects in my life. All of these things sparked me of the idea to study and develop what would burst out when they meet together.



HONORS & ACADEMICS

TOEFL iBT score: 5.5/6.0 2026

Y10 GPA: 5.00/5.00 2026

First prize of 2023CSP-J 2023

Second prize of 2023CSP-S 2023

Top 5% globally in AMC10 2025

ABILITIES & STRENGTHS

Basic C++ programming skills

Basic Photoshop techniques

INTERESTS

Keep painting since five years old

Learned playing piano for five years

Reading different kinds of books, including novels, history, legends, etc.

Highly focus on what I love, once read a eight-hundred-page book in three days

Writing and Designing original characters

Traveling and taking photos

Visiting museums and learning about the stories about the exhibits, especially sculptures and paintings