Planning and Decision Making Report

Davi Alves Oliva - 6397188
John Stewardson - 6420613
Pablo Ocelotl Gómez Colombón - 6564100
Panagiotis Sarikas - 6553745

***Abstract*— This report summarizes our project of autonomous drone navigation. We investigate a hierarchical planning framework combining probabilistic roadmaps, grid-based global planning, and model predictive path integral control, and evaluate its performance in simulated indoor environments. The corresponding code implementation in Python can be accessed on GitHub: `https://gitlab.com/pdm_group/pdm_group_project`**

I. INTRODUCTION

In this project, we address the problem of autonomous navigation of an aerial robot in a known indoor environment containing walls and obstacles. The objective is to compute a collision-free path between a given start position and one or more goal positions, assuming that the environment is static and known at the global planning level. This type of navigation problem is central to many real-world applications, including search and rescue in structured indoor environments, automated inspection of buildings and industrial facilities, and logistics in hospitals or factories. Such applications often involve navigating between multiple predefined locations, which motivated the use of a multi-query global planning method.

The task considered in this project consists of navigating the robot safely through environments of increasing complexity, ranging from open spaces to maze-like layouts with narrow passages. While the global planner operates on a static map, additional obstacles unknown to the global planner may appear during execution, representing dynamic or unmodeled objects such as people or temporary equipment. These disturbances are handled by a local planner with access to more up-to-date environment information, allowing the robot to react online while following the nominal global path.

From a methodological perspective, this project follows a common paradigm in robotic motion planning, in which global path planning is decoupled from local motion control. State-of-the-art approaches for indoor UAV navigation typically rely on discrete environment representations combined with graph-based or sampling-based planners to compute a global path, while local controllers handle trajectory execution and collision avoidance [1], [2]. In particular, graph-search methods such as A* and Dijkstra, often combined with probabilistic roadmaps (PRM), are widely used in static and safety-critical environments due to their predictability and computational efficiency [1]. Recent comparative studies have shown that PRM-based planners perform especially well in environments of moderate complexity, offering fast planning times while producing safe paths [3].

Motivated by these observations, our system adopts a hierarchical planning framework composed of a PRM- and grid-based global planner and a local control layer based on PID and Model Predictive Path Integral (MPPI) control. This design choice aligns with current state-of-the-art practices and provides a clear separation between global path computation and local, real-time obstacle avoidance.
[1] [2]

II. PROBLEM DEFINITION

The robot is modeled as a rigid body with simplified dynamics (differential flatness), where motion is described at the level of three-dimensional position, while orientation, velocity, and low-level actuation are handled by a local controller.

The robot operates in a workspace $W \subset \mathbb{R}^3$, while its configuration is described by its position and orientation. Hence, the configuration space is defined as $\mathcal{C} \subset \mathbb{R}^3 \times S^1$.

The configuration space is divided into the free space $\mathcal{C}_{\text{free}}$ and the obstacle space $\mathcal{C}_{\text{obs}}$, corresponding to collision-free and colliding configurations, respectively. For simplicity in visualization, we adopted a planar formulation by setting $z = 1$ and neglecting orientation. The workspace is reduced to $W \subset \mathbb{R}^2$, and the configuration space is approximated by $\mathcal{C} \subset \mathbb{R}^2$. Accordingly, the robot configuration is represented only by its planar position $\mathbf{p} = (x, y) \in \mathbb{R}^2$.

Global planning is performed using a discrete abstraction of the environment. Two complementary representations are employed: a uniform grid and a probabilistic roadmap (PRM). Both are constructed from a static map of known obstacles. The resulting planning space is a graph $G = (V, E)$, where vertices correspond to collision-free configurations in $\mathcal{C}_{\text{free}}$, and edges represent feasible straight-line

[1] Statement of author contributions: John Stewardson implemented the code for the PRM and Sukharev grid algorithm and PID control (integrated it with the simulator), analyzed their performance, and wrote the corresponding sections in this report. Davi Oliva implemented the code for the collision checker code, the custom environments, access levels for obstacles for different planners, and wrote the related sections of the report. Also wrote the README.md and handled version control issues. Panagiotis Sarikas implemented the code for the A* and Dijkstra algorithm and MPPI local motion planner, analyzed their performance, and wrote the corresponding sections in this report. Pablo Gómez implemented the graphic visualization of the simulations, wrote the state-of-the-art comparison, problem definition, and part of the discussion.

[2] Statement of use of Generative AI: We used generative AI for code debugging.

motions validated through collision checking. The graph-search algorithms A* or Dijkstra are used to compute a path.

Given a start configuration $\mathbf{p}_{\text{start}} \in \mathcal{C}_{\text{free}}$ and a goal configuration $\mathbf{p}_{\text{goal}} \in \mathcal{C}_{\text{free}}$, the planning problem consists of finding a collision-free path $\sigma : [0, 1] \to \mathcal{C}_{\text{free}}$ that connects the start to the goal while minimizing the path length.

The following assumptions are made: The static environment is known a priori; the robot operates indoors without external disturbances; the robot is holonomic at the planning level; localization is accurate; and dynamic obstacles are handled exclusively by the local planner. [3]

III. MOTION PLANNING METHOD

Algorithm 1 Hierarchical planning for multiple tasks

```
G ← BUILDROADMAP(M)
p ← C[0]                                  ▷ first checkpoint is start
for i ← 1 to len(C) do
    ADDCHECKPOINT(C[i], G)
    τ ← GRAPHSEARCH(G, C[i−1], C[i])
    while ‖C[i] − p‖ > threshold do
        p ← LOCALPLANNER(τ, p)
    end while
end for
```

A. Hierarchical Planning Architecture

The proposed quadrotor navigation and control framework follows a hierarchical planning and control architecture, decomposed into three distinct layers: 1) Global planning layer, responsible for computing a collision-free reference path in the environment, 2) Local planning layer, implemented using Model Predictive Path Integral (MPPI) control, which generates optimal translational acceleration commands in flat space. 3) Low-level control layer, which converts the commanded accelerations into attitude references and stabilizes the quadrotor

B. Global Planning

We plan in R^D, with D = 2 in the planar case (x,y), and D = 3 for three-dimensional space (x,y,z).

1) Probabilistic Roadmap Construction (PRM): We implement the PRM algorithm as described in [4]. We choose n, the maximum number of nodes, to be a variable defined by the user. This reflects the fact that for more difficult environments the number of nodes needed to find a solution reliably increases. For the more difficult environment there are more closed off regions, with only narrow passages. Thus the chance of randomly sampling a node with a possible connection that finds this opening decreases. For the environments simple, medium and difficult maximum nodes of 1000, 2000 and 5000 nodes achieved reliable results. For δ (in the code denoted with r), the maximum edge length, we set the default value to 1.5 meters, approximately 10% of the map width. However, the code is implemented such that this is a flexible parameter. Increasing δ improves the found path length (all other factors being equal), due to the triangular inequality. This increase in performance comes at the cost of more computation time, as more potential connection need to be checked. This is even more true as in our current implementation we do not limit the number of connections per node (k=n). By not limiting k, the number of connections, we get a better connected graph, at the potential cost of more compute time. This compute time comes from the collision detection of every potential edge. However, a non bound k, removes the need for sorting all nodes based on distance for prioritization, and simplifies the search to a binary one (within search radius δ or not), reducing the computational cost.

2) Sukharev Grid: As an alternative to PRM we also implement a graph construction as a grid. Here we reuse the code from PRM, but instead of sampling positions we take equidistant points on a grid, and use the discretization dx for the search radius δ.

3) Dijkstra and A:* After adding the next goal position to the graph, we utilize A* or the Dijkstra algorithm to find the optimal route on the given graph. The cost function is given by the euclidean distance. The heuristic cost for A* is given by the euclidean distance of the node to the goal position.

C. Collision Detection

For computational efficiency, all the walls and obstacles are reduced to axis-aligned rectangular bounding boxes in the planning phase. The drone itself is approximated as a circle in the 2D plane. Collision checking is performed by verifying whether a given configuration is within the expanded bounding box of any obstacle, where the expansion accounts for the radius of the drone.

This approach provides a conservative approximation of collision detection, as some configurations may be marked as colliding even though the true geometry would allow safe passage. However, configurations classified as collision-free are guaranteed to be safe. This approach ensures fast collision queries, which are critical for sampling-based planners, and safety, as false positives are preferable to false negatives in navigation tasks

D. Local Planner

The global planner produces a discrete sequence of waypoints, $\mathbf{r}_{\text{ref}} = \begin{bmatrix} x_i & y_i & z_i \end{bmatrix}^T$, representing a collision-free path from the start to the goal configuration. Since the waypoints are not equidistant, the reference path is resampled to obtain approximately uniform arc-length spacing, yielding a local reference sequence $\mathbf{r}_{\text{ref}}(k), \ldots, \mathbf{r}_{\text{ref}}(k+N)$ which is updated at every control step and provided to the local MPPI planner. The MPPI planner operates in flat translational space, using the state composed of translational position and velocity, $\mathbf{x}_{\text{MPPI}} = \begin{bmatrix} \mathbf{r} & \dot{\mathbf{r}} \end{bmatrix}^T = \begin{bmatrix} x & y & z & v_x & v_y & v_z \end{bmatrix}^T \in \mathbb{R}^6$. The control input is defined as the translational acceleration, $\mathbf{u} = \begin{bmatrix} \ddot{x} & \ddot{y} & \ddot{z} \end{bmatrix}^T \in \mathbb{R}^3$. The system is modeled as a

[3]The path planning framework was implemented by us.

discrete-time double integrator

$$\mathbf{r}_{k+1} = \mathbf{r}_k + \dot{\mathbf{r}}_k \Delta t, \quad \dot{\mathbf{r}}_{k+1} = \dot{\mathbf{r}}_k + \mathbf{a}_k \Delta t$$

where Δt denotes the discretization time step.

We implement the MPPI algorithm as described in [6]. At each control iteration, the MPPI planner samples a set of stochastic control sequences by perturbing the nominal control input with additive Gaussian noise, $\mathbf{v}_t^{(k)} = \mathbf{u}_t + \boldsymbol{\epsilon}_t^{(k)}, \quad \boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \Sigma)$, where Σ is the control noise covariance matrix and k indexes the rollout. Each rollout is forward-simulated and evaluated using the cost

$$J = \sum_{t=1}^{N} \Big(w_p \|\mathbf{r}_t - \mathbf{r}_{\text{ref},t}\|^2 + w_v \|\dot{\mathbf{r}}_t\|^2 + w_u \|\mathbf{a}_t\|^2 + J_{\text{obs}}(\mathbf{r}_t) \Big)$$

which penalizes tracking error, velocity, control effort, and obstacle proximity. Obstacle avoidance is enforced via a high-magnitude penalty, making colliding trajectories extremely unlikely. The rollout weights are computed as

$$w_k = \frac{\exp\left(-\frac{1}{\lambda}(J_k - J_{\min})\right)}{\sum_j \exp\left(-\frac{1}{\lambda}(J_j - J_{\min})\right)}$$

and the optimal control sequence is obtained by $\mathbf{u}_t = \sum_k w_k \mathbf{v}_t^{(k)}$. Following a receding-horizon strategy, only the first control input $\mathbf{u}_0$ is applied at each step.

E. Attitude Control and Execution Model

Yaw is not optimized by MPPI; instead, a path-aligned reference is generated as $\psi_{\text{des}} = \text{atan2}\left(\dot{y}_{\text{ref}}, \dot{x}_{\text{ref}}\right)$. Exploiting quadrotor differential flatness and a linearized hover model, the commanded acceleration is mapped to desired roll and pitch angles:

$$\phi_{\text{des}} = \frac{1}{g}\left(\ddot{x}_{\text{des}} \sin\psi_{\text{des}} - \ddot{y}_{\text{des}} \cos\psi_{\text{des}}\right)$$

$$\theta_{\text{des}} = \frac{1}{g}\left(\ddot{x}_{\text{des}} \cos\psi_{\text{des}} + \ddot{y}_{\text{des}} \sin\psi_{\text{des}}\right)$$

To stabilize the quadrotor, a PD attitude controller is employed. The controller state is defined by the Euler angles and the corresponding angular velocities as, $\mathbf{x}_{PD} = \begin{bmatrix}\phi & \theta & \psi & p & q & r\end{bmatrix}^T \in \mathbb{R}^6$. The control input consists of body torques

$$\mathbf{u} = \begin{bmatrix}\tau_\phi & \tau_\theta & \tau_\psi\end{bmatrix}^T = K_p \begin{bmatrix}\phi_{\text{des}} - \phi \\ \theta_{\text{des}} - \theta \\ \psi_{\text{des}} - \psi\end{bmatrix} + K_d \begin{bmatrix}p_{pref} - p \\ q_{ref} - q \\ r_{ref} - r\end{bmatrix}$$

The translational dynamics are given by $\dot{\mathbf{r}} = \mathbf{v}, \quad \dot{\mathbf{v}} = \mathbf{a}_{\text{des}}$ assuming perfect acceleration tracking by the inner loop. The rotational dynamics follows $\dot{p} = \frac{\tau_\phi}{I_x}, \quad \dot{q} = \frac{\tau_\theta}{I_y}, \quad \dot{r} = \frac{\tau_\psi}{I_z}$ followed by kinematic integration $\dot{\phi} = p, \quad \dot{\theta} = q, \quad \dot{\psi} = r$. Only position and velocity are fed back to MPPI, $\mathbf{x}_{\text{MPPI}} = \begin{bmatrix}\mathbf{r}, \dot{\mathbf{r}}\end{bmatrix}$. Attitude dynamics are handled entirely in the execution layer, ensuring a clean separation between planning and stabilization.

[4]

[4]Statement of code reuse

F. PID control

As a baseline comparison to MPPI, we also integrate a simple PID control to blindly follow the given trajectory. The simulator gym-pybullet-drones [5] already comes with a PID implementation, which computes the control inputs based on the current position and target position. The reference path is the linear interpolation of the list of node positions found by the global planner. We project the drones position onto this reference path and look ahead on the path by 0.1 m, to obtain the target position of the current timestep. In the 2D implementation, we use PD control to minimize the error of the current position and the target position, controlling the acceleration and then propagate the dynamics with Euler discretization.

IV. RESULTS

A. Evaluating the Roadmap Construction:

To enable a fair comparison, the grid discretization dx was chosen as

$$\texttt{dx} = \sqrt{\frac{\texttt{MAP\_HEIGHT} \cdot \texttt{MAP\_WIDTH}}{\texttt{max\_nodes}}}$$

As can be seen in Figure 2, the Sukharev grid is more robust compared to the PRM algorithm. Here we show representatively the evaluation on the environment "medium", but the observations hold for all environments, with the plots for the other environments available in our GitHub repository.
The grid method finds solution with less number of nodes, and is more computationally efficient, as seen in Table IV-B. In our environment, the walls are always parallel to the x or y axis, which suits the grid algorithm, as traversing parallel to the wall means traveling along the edges of the grid.
In contrast to PRM it is also a deterministic algorithm - hence no error bars are displayed. For the PRM algorithm the success rate increases with the number of nodes, as expected. The path length found by A* is consistently 20% shorter when given the graph produced by PRM compared to the grid, and gets within 10% of the optimal path length. The path length of the grid algorithm also plateaus, as for diagonals the grid algorithm produces a stair like pattern, regardless of the resolution of the grid.

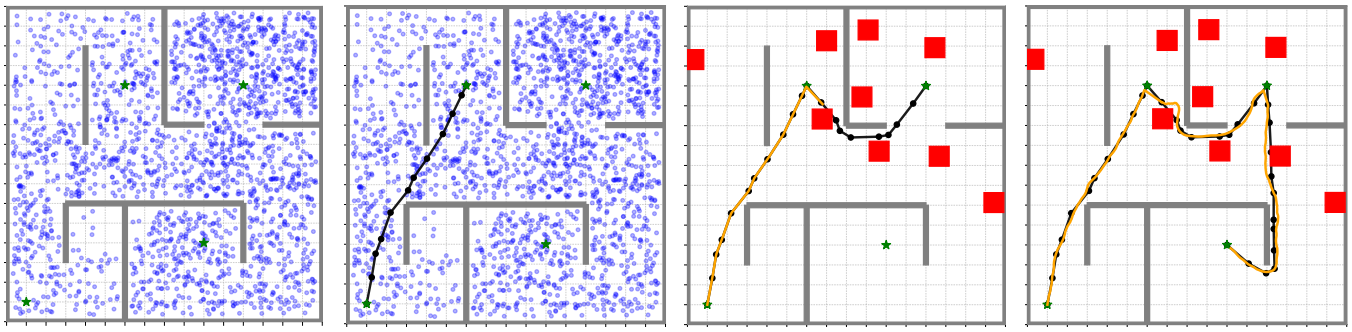
B. Computational Performance

	Easy (1000 nodes)	Medium (2000 nodes)	Difficult (5000 nodes)
PRM	4.30	18.21	108.18
Grid	0.81	2.99	17.04
A*	0.19	0.48	7.71
Dijkstra	0.03	0.15	0.59

TABLE I
COMPARISON OF COMPUTE TIME: GLOBAL PLANNING

The algorithms' compute time across the three environments, measured in seconds. The experiments were run on the HP ZBook X Gli (Intel Core Ultra 7 255H) on Ubuntu Noble.

Interestingly, the compute time for the Dijkstra algorithm is better compared to A*. Indicating that the additional overhead from the heuristic calculation outweighs the potential



(a) Roadmap (PRM) (b) Graph Search (A*) (c) PID (d) MPPI

Fig. 1. Multi Task Execution

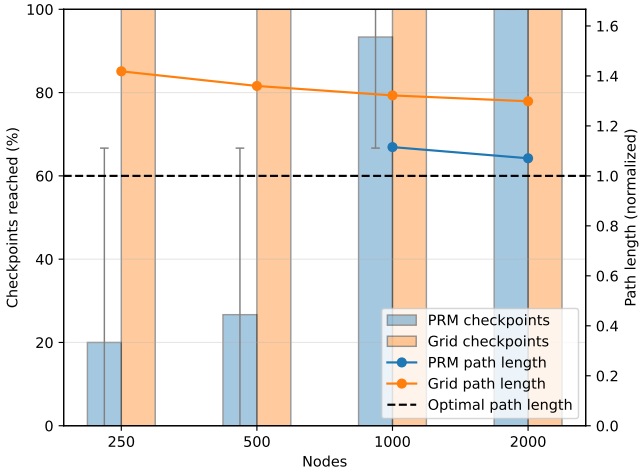


Fig. 2. Comparison of the performance of graph creation with the Probabilistic Road Map (PRM) algorithm and Sukharev Grid for the medium environment. The bars show the average, and the error bars indicate the minimum and maximum, of reached checkpoints across five runs as a percentage. The line-plot layered on top shows the average path length found by A* given the graph produced by PRM or the Grid (only considering paths which reach all checkpoints). The found path length is normalized by the optimal path length.

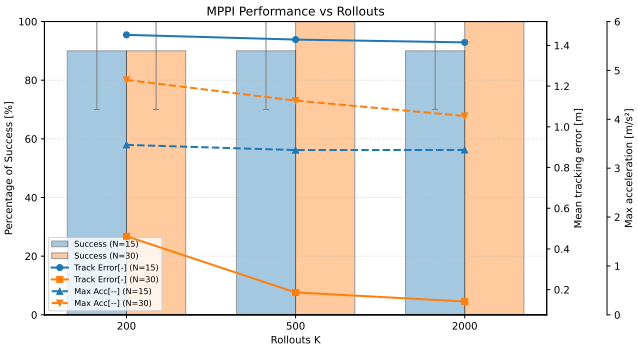


Fig. 3. MPPI performance as a function of the rollout count K in the difficult environment for two horizon lengths, $N = 15$ (blue) and $N = 30$ (orange), averaged over five random seeds. Bars indicate the percentage of successful runs (collision-free checkpoint reached), while solid lines show the mean tracking error between the global and local paths and dashed lines the maximum acceleration (control effort). Error bars denote the standard deviation.

benefits. We use the euclidean distance for the heuristic cost, however the optimal path has to circumnavigate long walls, such that the resulting path diverges from the straight connecting line, making A* with euclidean distance less effective. The bulk of the computation time is spent on the graph creation. This validates our approach of utilizing a multi query method, meaning we only have to compute the graph once. In the application, the graph can be computed upfront and with remote computation.

C. Local Planner

As depicted in Figure 1, the MPPI planner is often able to avoid random obstacles, placed in its reference trajectory, while the naive PD controller fails.

Figure 3 illustrates the performance of MPPI as a function of the rollout count K for two horizon lengths, $N = 15$ and $N = 30$, in a difficult environment. Overall, performance improves with increasing rollouts and horizon length. Higher rollout counts enable broader sampling of the control space, while longer horizons allow MPPI to better anticipate future obstacle interactions and plan ahead. Importantly, rollouts and horizon length are tightly coupled parameters. Increasing the horizon without a corresponding increase in rollouts expands the dimensionality of the control sequence space, reducing sampling density and potentially degrading performance due to increased stochasticity. This behavior reflects the fundamental trade-off between exploration and exploitation in sampling-based optimal control. For the considered system with three control inputs (translational accelerations), increasing the horizon from N to $2N$ leads to a combinatorial growth in the number of feasible control sequences. Consequently, maintaining or improving performance requires a superlinear increase in the number of rollouts relative to horizon length. This relationship is reflected in the success rate trends, which improve consistently when rollouts and horizon length are increased in a balanced manner. Furthermore, higher rollout counts result in reduced maximum acceleration, indicating that the controller is able to identify smoother and less aggressive control policies. Finally, the mean tracking error between global and locally executed trajectories decreases with increasing rollouts and horizon

Horizon N = 15	R = 200	R = 500	R = 2000
Total Comp Time (s)	29.0±12.5	74.7±27.2	299.9±111.7
Freq Capability (Hz)	27.8±7.6	10.4±1.6	2.6±0.3
Simulation Time (s)	55.4±16.0	55.1±15.9	55.0±15.8

Horizon N = 30	R = 200	R = 500	R = 2000
Total Comp Time (s)	55.8±19.5	117.5±15.3	502.9±9.5
Freq Capability (Hz)	13.8±2.1	5.4±0.6	1.2±0.0
Simulation Time (s)	55.1±15.8	63.0±0.2	62.7±0.1

TABLE II

MPPI'S COMPUTE AND SIMULATION TIME FOR DIFFERENT ROLLOUT COUNTS AND HORIZON LENGTHS.

Comparison of compute time for MPPI in the difficult environment. Total compute time, step compute time, and simulation time are shown for varying horizon lengths N and rollout counts K. Results are averaged over five random seeds, reporting mean and standard deviation. Experiments were run on an HP ZBook X G11 (Intel Core Ultra 7 255H) with Ubuntu Noble.

length. However, tracking error must be interpreted in context: successful obstacle avoidance often requires deliberate deviation from the global trajectory, whereas unsuccessful runs may exhibit artificially low tracking error by closely following the global path until collision or premature termination.

As shown in Table II, total compute time increases with both parameters, while the achievable step frequency decreases accordingly. Since frequency capability directly determines how fast MPPI can update control commands, it is a critical factor for real-time quadrotor applications operating in dynamic environments. The largely constant simulation time indicates that performance is primarily limited by the MPPI optimization step rather than simulation overhead.

V. DISCUSSION AND CONCLUSION

A. Roadmap

As discussed in Section IV Results, for the roadmap construction the choice of algorithm depends on the exact application. The Probabilistic Roadmap algorithm constructs a graph with a better solution, but is computationally more expensive and sometimes does not find a solution at all. In our current implementation, when a new node is to be added, we loop through all existing nodes to check if the distance between the two is smaller than r and the connection is collision free. One way to significantly reduce the computational complexity is to embed the search space into a grid and use this to limit the number of nodes to check if they can be connected. A second potential improvement, would be to take advantage of the current design with k=n. Since all nodes can therefore be checked independently, this allows for parallelization of this operation, as opposed to looping through all nodes sequentially on the CPU.

B. Transferability

In this report we show our findings in the 2D space, as it is easiest to visualize. However, our algorithmic implementations are in 3D, we simply set z to 1, to enable the visualization in 2D. In order to demonstrate that our solutions are applicable in the real world, we make use of the simulator gym-pybullet-drones [5]. There we can test our solution in a more realistic environment. See our repository for more details.

C. Theoretical Properties and Limitations

From a theoretical perspective, the Probabilistic Roadmap planner is known to be probabilistically complete. This implies that, if a collision-free path exists, the probability of finding a solution approaches one as the number of sampled nodes tends to infinity. However, the method is not optimal in its basic form, since the shortest path in the roadmap is only an approximation of the true optimal path in the continuous configuration space. Optimal variants such as PRM* could be employed to guarantee asymptotic optimality.

Regarding computational complexity, the roadmap construction phase dominates the overall runtime. In the current implementation, node insertion requires checking all existing nodes, leading to a worst-case time complexity of $\mathcal{O}(n^2)$ for n samples. This limits scalability in high-dimensional spaces or dense environments. Query time, on the other hand, is comparatively low, as it reduces to a graph search problem.

While MPPI demonstrates strong performance in complex environments, it is inherently a sampling-based approximation and does not guarantee global optimality, particularly for long horizons or high-dimensional control spaces. As a result, performance is sensitive to the choice of rollout count and horizon length.

In addition, computational cost scales rapidly with both parameters, which can limit real-time applicability on CPU-based systems; practical implementations therefore often rely on large rollout counts executed in parallel on GPUs. Finally, MPPI assumes accurate system dynamics and bounded stochastic disturbances, and model mismatch, unmodeled dynamics, or highly non-Gaussian noise may lead to degraded or suboptimal control performance.

D. Practical Performance

In practice, the planner performed reliably in simple and moderately cluttered environments. Nevertheless, in narrow passages or highly constrained regions, the roadmap often lacked sufficient connectivity, leading to failed queries,specially with lower number of nodes.

Compared to alternative planners such as Rapidly-exploring Random Trees (RRT), the roadmap-based approach is more suitable for multi-query scenarios, but less efficient for single-shot planning problems. An RRT-based method could potentially find feasible paths faster in complex environments, although at the cost of solution quality and reusability.

E. Conclusion

Overall, the roadmap-based planning approach provides a flexible and conceptually simple framework for motion planning in both simulated and real-world environments. While the current implementation demonstrates the feasibility of the method, further improvements in efficiency, optimality, and adaptability are required for deployment in more complex and dynamic scenarios.

REFERENCES

[1] M. B. Sushma, B. Mashhoodi, W. Tan, K. Liujiang, and Q. Xu, *Spatial drone path planning: A systematic review of parameters and algorithms*, Journal of Transport Geography, vol. 125, p. 104209, May 2025.

[2] F. Li, S. Zlatanovac, M. Koopman, X. Bai, and A. Diakité, *Universal path planning for an indoor drone*, Automation in Construction, 2018.

[3] M. R. Banik, L. Ngeljaratan, and M. A. Moustafa, *Performance evaluation of path planning algorithms for autonomous UAV deployment using two urban scenarios*, Canadian Science Publishing, September 2025.

[4] J. Alonso-Mora, *Lecture 9: Global Path Planning*, Course slides, CourseCode: RO47005 Planning and Decision-making, Delft University of Technology, 2025, slide 27. [Online]. Available: `https://brightspace.tudelft.nl/d2l/le/content/775411/viewContent/4637132/View` (accessed Nov. 25, 2025).

[5] J. Panerati, H. Zheng, S. Zhou, J. Xu, A. Prorok, and A. P. Schoellig, *Learning to Fly—a Gym Environment with PyBullet Physics for Reinforcement Learning of Multi-agent Quadcopter Control*, in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2021, pp. 7512–7519, doi: 10.1109/IROS51168.2021.9635857.

[6] J. Alonso-Mora, *Lecture 7: Sampling-base MPC (MPPI)*, Course slides, CourseCode: RO47005 Planning and Decision-making, Delft University of Technology, 2025, slides 11-15. [Online]. Available: `https://brightspace.tudelft.nl/d2l/le/content/775411/viewContent/4479492/View`