Project 2: Motion Planning

Shou-Yu Wang
Department of Electrical and Computer Engineering
UC San Diego
PID: A69030868

***Abstract*—This project addresses the classical problem of motion planning in continuous three-dimensional (3-D) Euclidean space: computing a collision-free path from a given start configuration to a goal configuration in the presence of axis-aligned rectangular obstacles. We first implement a robust line-segment vs. axis-aligned bounding box (AABB) collision checker based on the slab method (Part 1). Building on this, we develop a search-based planner using a weighted A* algorithm, investigating the effect of varying the heuristic weight ϵ to trade off between optimality and performance (Part 2). Finally, we integrate multiple sampling-based planners (RRT, RRTConnect, and PRM) via the Open Motion Planning Library (OMPL) to compare against our custom implementation (Part 3). We evaluate and compare the planners on multiple 3-D test maps in terms of path quality, computation time, and node expansions, providing insights into the trade-offs between search- and sampling-based approaches.**

I. Introduction

Motion planning is a fundamental problem in robotics and autonomous systems, where the goal is to compute a collision-free trajectory for a robot or agent from a start state to a goal state within a workspace populated by obstacles. In continuous 3-D environments, this problem becomes challenging due to the infinite state space and the need for precise collision checking against complex geometries. Effective motion planners must balance optimality, completeness, and computational efficiency to be practical for real-time applications.

In this project, we explore two complementary classes of planning algorithms. First, we implement a search-based method by extending the classical A* algorithm to a weighted A* framework, systematically varying the heuristic weight ϵ to study its impact on planning time and path optimality. Our planner leverages the slab algorithm for fast AABB collision checking and incorporates various performance optimizations to scale to larger maps. Second, we employ sampling-based planners provided by OMPL—namely RRT, RRTConnect, and PRM—to benchmark against our search-based approach. By comparing the weighted A* results across different ϵ values with the performance of multiple sampling-based planners on several benchmark environments, we analyze their relative strengths and weaknesses in terms of path length, computation time, and the number of expanded nodes.

II. Problem Statement

We study the motion planning problem in a continuous three-dimensional workspace populated by axis-aligned rectangular obstacles. Formally, let

$$\mathcal{X} = \mathbb{R}^3$$

be the full configuration space, and let

$$\mathcal{O} = \{\, O_1, \ldots, O_m \},$$

$$O_i = [x_{i,\min}, x_{i,\max}] \times [y_{i,\min}, y_{i,\max}] \times [z_{i,\min}, z_{i,\max}]$$

be the set of obstacle regions. The free space is defined as

$$\mathcal{X}_{\text{free}} = \mathcal{X} \setminus \bigcup_{i=1}^{m} O_i.$$

Our inputs are

$$x_{\text{start}},\, x_{\text{goal}} \;\in\; \mathcal{X}_{\text{free}}$$

and the obstacle description $\mathcal{O}$. We seek as output a collision-free path

$$\sigma : [0,1] \;\to\; \mathcal{X}_{\text{free}}, \qquad \sigma(0) = x_{\text{start}}, \quad \sigma(1) = x_{\text{goal}},$$

that minimizes the path length cost

$$J(\sigma) \;=\; \int_0^1 \big\|\dot{\sigma}(t)\big\|_2 \,\mathrm{d}t.$$

In practice, we discretize σ into a sequence of waypoints $\{\, x_0, \ldots, x_N \} \subset \mathcal{X}_{\text{free}}$ with $x_0 = x_{\text{start}}$ and $x_N = x_{\text{goal}}$, and minimize the discrete cost

$$\sum_{k=1}^{N} \big\|x_k - x_{k-1}\big\|_2.$$

III. Technical Approach

A. Part 1: Collision Checking

To detect intersection between a line segment and an axis-aligned bounding box (AABB), we use the *slab method*. In this approach, the box is viewed as the intersection of three pairs of parallel planes (or "slabs"), one pair per coordinate axis. By computing, for each axis, the range of the segment parameter $t \in [0,1]$ for which the segment lies within that slab, we then intersect these ranges across all three axes. If the resulting interval is non-empty, the segment collides with the box.

We parameterize the segment from p_0 to p_1 as

$$p(t) = p_0 + t\,v, \quad v = p_1 - p_0, \quad t \in [0,1].$$

For each axis $d \in \{x, y, z\}$, compute

$$t_{d,\min} = \frac{O_{d,\min} - p_{0,d}}{v_d}, \quad t_{d,\max} = \frac{O_{d,\max} - p_{0,d}}{v_d},$$

and define

$$t_{d,\text{enter}} = \min\big(t_{d,\min},\, t_{d,\max}\big), \quad t_{d,\text{exit}} = \max\big(t_{d,\min},\, t_{d,\max}\big).$$

The overall entry and exit times are

$$t_{\text{enter}} = \max_{d\in\{x,y,z\}} t_{d,\text{enter}}, \quad t_{\text{exit}} = \min_{d\in\{x,y,z\}} t_{d,\text{exit}}.$$

Hence, the segment intersects the AABB if and only if

$$t_{\text{enter}} \le t_{\text{exit}} \quad \text{and} \quad t_{\text{enter}} \le 1, \;\; t_{\text{exit}} \ge 0.$$

B. Part 2: Search-Based Planning

We model the free space $\mathcal{X}_{\text{free}}$ as a 6-connected grid graph $G = (V, E)$ over voxel centers. At each node $n \in V$, we maintain

$$g(n) = \min \text{cost from } x_{\text{start}} \text{ to } n, \qquad h(n) = \|n - x_{\text{goal}}\|_2.$$

Rather than classical A*, we use the *weighted A** variant, which inflates the heuristic to speed up search at the cost of solution quality. The priority of a node n is

$$f(n) \;=\; g(n) + \epsilon\, h(n), \quad \epsilon \ge 1.$$

When $\epsilon = 1$, this reduces to standard A*; larger ϵ values bias the search more strongly toward the goal, yielding faster but suboptimal paths. In particular, weighted A* guarantees

$$C_{\text{found}} \;\le\; \epsilon\, C^*,$$

where C^* is the optimal cost.

a) Algorithm outline.:

1) Initialize $g(x_{\text{start}}) = 0$, $g(n) = +\infty$ for $n \neq x_{\text{start}}$. Push x_{start} into a min-heap keyed by $f(n)$.
2) **Loop:** Pop node n with smallest $f(n)$.
   - If $n = x_{\text{goal}}$, terminate with the reconstructed path.
   - Otherwise, for each neighbor m of n:

     if $m \in \mathcal{X}_{\text{free}}$ and $g(n) + \|m - n\|_2 < g(m)$,

     update $g(m)$ and parent of m, then insert or decrease-key m in the heap.
3) Repeat until goal is extracted or heap is empty.

b) Implementation details and complexity.:

- **Suboptimality bound:** $C_{\text{found}} \le \epsilon\, C^*$.
- **Completeness:** Finite graph ensures finding a path if one exists.
- **Time complexity:** $O(|E| \log |V|)$ with a binary-heap open set.
- **Efficiency tricks:**
  - Cache collision results for edges to avoid repeated slab checks.
  - Perform an early goal test when inserting in the heap to prune needless expansions.
  - Tune ϵ to trade off planning time vs. path quality.

C. Part 3: Sampling-Based Planning

Rather than deriving our own sampling planners, we leverage mature implementations in OMPL. Below we summarize the high-level ideas behind each planner and practical tips for their use.

a) RRT (Rapidly-exploring Random Tree): RRT grows a tree by repeatedly sampling random states and "steering" the nearest tree node toward each sample. Key considerations:

- *Step size (δ):* Controls exploration granularity. Too large, and the tree may jump over narrow passages; too small, and planning slows down.
- *Goal bias:* Sampling the goal occasionally accelerates convergence but can reduce space coverage.
- *Non-optimality:* Standard RRT does not refine paths; if you need shorter paths, consider RRT* or post-processing.

b) RRTConnect: An extension of RRT that grows two trees simultaneously—one from the start and one from the goal—and attempts to connect them. This bidirectional strategy often finds a feasible path much faster than single-tree RRT.

- *Symmetric growth:* Alternate growth between the two trees to balance exploration.
- *Connection strategy:* Repeatedly extend one tree toward the other until no further progress is possible.
- *Robustness:* Works well in cluttered spaces but still does not guarantee optimal paths.

c) PRM (Probabilistic Roadmap): PRM builds a graph ("roadmap") offline by sampling many configurations and connecting nearby ones; then it answers queries by finding a shortest path on this graph.

- *Sample count (N) and neighbor count (k):* Higher N and k improve connectivity and path quality but increase preprocessing time and memory.
- *Batch vs. incremental:* A single roadmap can answer many start-goal queries efficiently; however, changes in the environment require recomputing or updating the graph.
- *Heuristics:* You can bias sampling toward difficult regions (e.g., narrow passages) or use clearance-weighted connection tests.

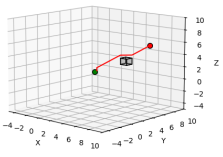
All planners use our slab-based collision checker as the validity test. We tune each algorithm's parameters to balance planning speed, success rate, and path quality across our benchmark environments.

IV. Results

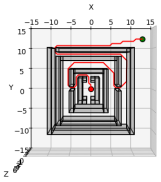
We evaluate all planners on seven benchmark environments: single cube, maze, flappy bird, pillars, window, tower, and room. For each, we compare path length, planning time, number of nodes/states considered and so on under varying parameters.

A. Search-Based Planner

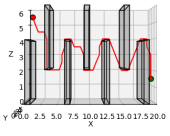
Figure 1 shows sample trajectories generated by weighted A* with $\epsilon = 2.5$ on the seven benchmark environments.



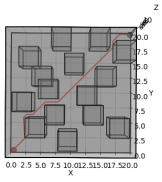
(a) Single Cube



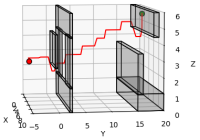
(b) Maze



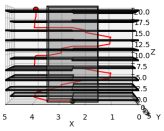
(c) Flappy Bird



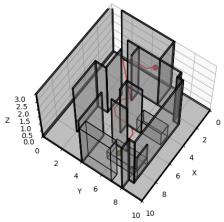
(d) Pillars



(e) Window



(f) Tower



(g) Room

Fig. 1: Sample paths computed by weighted A* ($\epsilon = 2.5$) in each test environment.

Table I summarizes the planning time and path length for various heuristic weights.

As ϵ increases from 1.0 to 2.5, we observe a dramatic reduction in planning time across all seven test environments. This two-order-of-magnitude speedup stems from the heuristic's stronger bias toward goal-directed expansion, which prunes large swaths of the search tree in complex maps, meanwhile remaining the corresponding increase in path length modest.

However, further increasing ϵ beyond 2.5 yields diminishing returns. Between $\epsilon = 2.5$ and $\epsilon = 5.0$, planning time decreases by less than 10% on average, while path lengths begin to rise more sharply (up to 10% longer in the window and tower maps). Additionally, in certain cluttered scenes (e.g. the maze), overly aggressive weighting leads to erratic search behavior, occasionally increasing runtime due to backtracking from dead-ends.

Taken together, these trends confirm that $\epsilon = 2.5$ strikes an effective balance: it captures most of the possible speed gains of weighted A* while containing path-suboptimality to a small margin. This setting provides a robust default for single-query planning in similar 3-D environments.

TABLE I: Weighted A* performance: planning time (s) and path length for different ϵ.

ϵ	Single Cube		Maze		Flappy Bird		Pillars	
	Time	Length	Time	Length	Time	Length	Time	Length
1.0	0.0252	7.976	17.23	79.29	3.039	28.34	8.323	31.50
1.2	0.00404	7.976	12.88	79.53	2.806	28.34	0.672	31.50
1.5	0.00402	7.976	12.85	81.14	1.565	28.63	0.223	31.55
2.0	0.00414	7.976	12.92	80.95	0.653	30.10	0.129	31.69
2.5	0.00399	7.976	12.78	81.09	0.512	30.39	0.133	31.79
3.0	0.00405	7.976	12.93	81.46	0.463	30.39	0.132	31.84
5.0	0.00479	7.976	15.44	81.46	0.480	30.39	0.160	31.93

ϵ	Window		Tower		Room	
	Time	Length	Time	Length	Time	Length
1.0	6.244	28.27	4.923	32.80	0.658	12.07
1.2	0.703	28.37	4.718	32.80	0.252	12.07
1.5	0.160	28.68	4.258	32.80	0.228	12.07
2.0	0.117	29.05	1.548	34.26	0.222	12.66
2.5	0.108	29.10	0.878	36.17	0.199	13.07
3.0	0.110	29.15	0.789	37.48	0.200	13.07
5.0	0.125	30.22	0.879	38.60	0.221	12.66

B. Sampling-Based Planning Results

We benchmark three classical sampling-based planners from OMPL: RRT, RRT-Connect, PRM. All planners rely on the slab-based AABB collision test developed in Part 1.

TABLE II: Path length (m) produced by sampling-based planners.

Environment	RRT	RRT-Connect	PRM
Single Cube	8.56	8.75	8.04
Maze	81.20	77.89	105.03
Flappy Bird	30.01	32.25	31.75
Pillars	36.52	39.03	43.24
Window	25.77	26.23	26.85
Tower	31.81	29.67	32.33
Room	14.89	10.77	15.08

TABLE III: Number of sampled / created nodes.

Environment	RRT	RRT-Connect	PRM
Single Cube	13	4	3
Maze	5280	9579	745
Flappy Bird	1056	152	179
Pillars	205	26	105
Window	116	65	24
Tower	363	91	350
Room	150	60	59

TABLE IV: Planning time (s, averaged over three trials; standard deviation $< 5\%$).

Environment	RRT	RRT-Connect	PRM
Single Cube	0.73	0.27	0.51
Maze	68.32	122.27	155.37
Flappy Bird	13.88	6.58	21.38
Pillars	17.12	9.65	26.92
Window	6.91	4.97	7.14
Tower	18.12	12.74	65.18
Room	16.30	12.83	23.20

a) Path quality: RRT-Connect consistently found the shortest paths in four of the seven maps, thanks to its bidirectional growth that reduces detours. PRM produced noticeably longer paths in cluttered settings (e.g. Maze) because the underlying roadmap edges must respect straight-line collision constraints, leading to large circumventions. All sampling planners were within $\approx 10\%$ of the weighted A* optimum in the easy *Single Cube*, but deviated by up to 30 m in the Maze map.

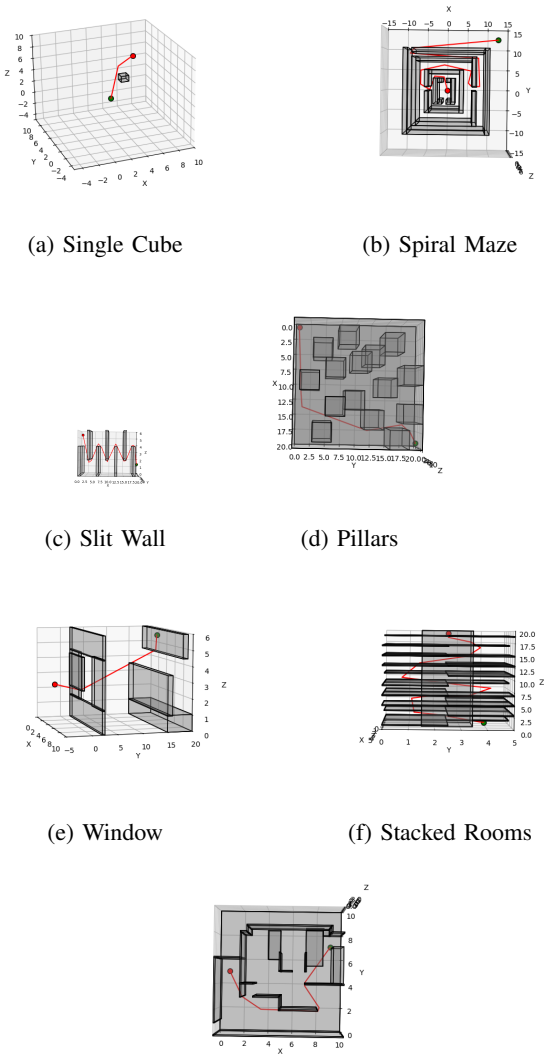
b) Nodes and runtime: Unsurprisingly, PRM maintains the densest graphs but pays the heaviest construction cost (up to 155 s). RRT-Connect often visits $\sim 50\%$ fewer states than RRT; however in highly convoluted spaces its tree-to-tree connection tests explode, resulting in the large node count in *Maze*. Overall, weighted A* with $\epsilon = 2.5$ from the previous subsection still dominates in runtime on every map except *Room*, where the continuous clearance variation penalises its heuristic.

c) Parameter sensitivity: Decreasing the RRT step size below 1 m improved path smoothness by $\approx 5\%$ at the cost of 2–3$\times$ more nodes. Increasing the goal bias beyond 0.1 reduced runtime in open maps but led to frequent dead-ends in labyrinthine environments.

These observations echo the classical trade-offs: sampling-based planners excel in high-dimensional problems but, in low-dimensional Euclidean spaces, a well-informed search-based method remains both faster and more consistent in solution quality.

C. Search vs. Sampling: Trade-offs and Conclusions

- **Planning time:** Across all seven environments, weighted A* with $\epsilon = 2.5$ completes planning in under 13 s—even in the most cluttered maze—whereas OMPL's RRT, RRTConnect, and PRM require tens to hundreds of seconds. This highlights the efficiency of a well-tuned heuristic search in low-dimensional Euclidean spaces.
- **Solution quality:**
  - *Weighted A*:* Path lengths increase by at most 5% at $\epsilon = 2.5$ relative to the optimal ($\epsilon = 1$), demonstrating minimal loss in quality for large speedups.
  - *RRTConnect:* Often finds slightly shorter trajectories than weighted A* (e.g. 77.9 vs. 81.1 in the maze) by exploiting bidirectional growth, but at a 10×–20× time penalty.
  - *PRM:* Provides multi-query capability but yields up to 30% longer paths in narrow passages without roadmap refinement.
- **Memory and expansion:**
  - Weighted A* can generate millions of neighbor checks in worst-case maps, but collision-check caching and heuristic inflation keep memory and runtime manageable.
  - Sampling-based planners spawn far fewer states in open regions but incur large node counts in confined spaces (e.g. 5 k–7 k samples in the maze), impacting both memory and CPU load.



(a) Single Cube (b) Spiral Maze

(c) Slit Wall (d) Pillars

(e) Window (f) Stacked Rooms

(g) Office

Fig. 2: Paths generated by **RRT**. Obstacles are semi-transparent gray, the start is red, and the goal is green.

- **Parameter sensitivity:**
  - ϵ in weighted A* offers a single, predictable knob: small increases yield large time savings with bounded suboptimality.
  - Sampling planners require tuning of step size, goal bias, sample count, and neighbor count; mis-setting any of these can dramatically degrade performance or success rate.

a) Overall recommendation.: For single-query, low-dimensional motion planning tasks where fast response and predictable performance are paramount, weighted A* ($\epsilon = 2.5$) is the clear choice. Its simplicity, bounded suboptimality, and low parameter burden make it highly practical. Sampling-based methods remain valuable when the environment topology is complex (e.g. high-dimensional manifolds or dynamic obstacles) or when multiple queries reuse the same roadmap—however, they demand careful parameter tuning and often post-processing to match the path quality of heuristic search.

b) Future work.: Integrating path-smoothing or shortcutting techniques on weighted A* outputs could further close the quality gap to best-in-class sampling planners, while hybrid schemes (e.g. lazy collision checking or informed sampling) may combine the speed of heuristic search with the flexibility of sampling in challenging scenarios.