ECE 276B Project 2: Motion Planning

Jean Gorby Calicdan
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, United States
jcalicda@ucsd.edu

***Abstract*—This project investigates 3-D motion planning in environments composed of rectangular boundaries and axis-aligned obstacle blocks. We first develop a robust collision-checking module based on line-segment–AABB intersection tests to ensure safe trajectory evaluation. Building on this, we implement a search-based planner using a weighted A* algorithm on a discrete 3-D grid, exploiting admissible heuristics to balance solution optimality against computational effort. To complement this, we design a sampling-based planner following the RRT* framework, which provides asymptotic optimality in continuous configuration spaces. We evaluate both planners across a suite of benchmark environments, comparing path cost, node expansions, and runtime under varying heuristic weights and sampling densities. Our results show that weighted A* achieves high-quality paths with minimal expansions, while RRT* offers scalable coverage in complex 3-D domains.**

***Index Terms*—Motion Planning, Collision Checking, Weighted A*, Rapidly-exploring Random Tree (RRT*), Sampling-Based Planning, Heuristic Search, 3-D Grid Environments, Path Optimization.**

I. Introduction

Motion planning is a cornerstone of autonomous robotics, demanding the synthesis of collision-free trajectories that respect both environmental constraints and system dynamics. In many robotic applications, workspaces can be abstracted as collections of axis–aligned rectangular obstacles, where planning efficiency and path quality hinge on fast, reliable collision checking and intelligent exploration strategies. This project develops a unified motion planning framework that integrates a precise collision-checking module with both search-based and sampling-based planners.

We begin by implementing a collision-checking routine based on line-segment–AABB intersection tests to verify straight-line feasibility against rectangular blocks. Building on this, we design a weighted A* planner over a discretized grid representation, leveraging admissible heuristics to balance path optimality against computational effort. To address continuous-space requirements and provide asymptotic optimality guarantees, we also implement an RRT* planner tailored to our obstacle models. We evaluate both planners on a suite of benchmark scenarios, comparing path cost, node expansions, and runtime under varying heuristic weights and sampling densities. Our comparative analysis highlights the trade-offs between heuristic-guided search and randomized tree exploration in practical motion planning.

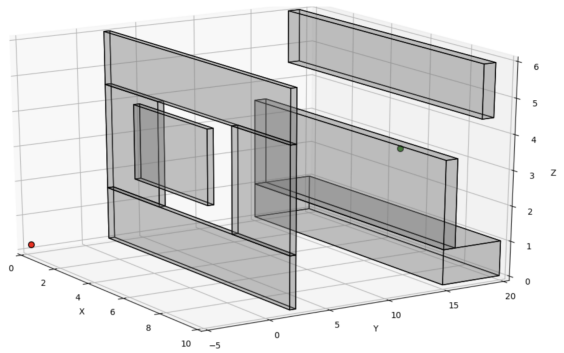

Fig. 1: Overview of the 3-D workspace and obstacle layout.

II. Problem Formulation

We consider a point robot operating in a bounded 3-D workspace with axis-aligned rectangular obstacles. Any point or vector lies in $\mathbb{R}^3$. For each environment we are given:

- **Map boundary:** an axis-aligned bounding box (AABB)

$$B = [x_{\min}, y_{\min}, z_{\min},\ x_{\max}, y_{\max}, z_{\max}] \in \mathbb{R}^6,$$

  defining a rectangular prism that contains the robot.
- **Start and goal:** two points

$$s,\ \tau \in \mathbb{R}^3, \quad s, \tau \in B.$$

- **Obstacles:** M blocks, each encoded as an AABB

$$O_i = [x^i_{\min}, y^i_{\min}, z^i_{\min},\ x^i_{\max}, y^i_{\max}, z^i_{\max}] \in \mathbb{R}^6, \quad i = 1, \ldots, M.$$

Define the collision-free configuration space

$$\mathcal{C}_{\text{free}} = \big\{\, x \in B \;\big|\; x \notin O_i,\ \forall i = 1, \ldots, M \big\}.$$

Discrete Shortest-Path Formulation

Exact continuous solutions are intractable, so we discretize $\mathcal{C}_{\text{free}}$ into a graph

$$G = (V, E), \quad V \subset \mathcal{C}_{\text{free}},$$

$$E = \big\{(v_i, v_j) \;\big|\; \text{straight segment } [v_i, v_j] \subset \mathcal{C}_{\text{free}}\big\}.$$

Each edge $(v_i, v_j) \in E$ is assigned cost

$$c_{ij} = \|v_i - v_j\|_2 > 0.$$

Let P be the set of all vertex sequences $\{v_0, v_1, \ldots, v_N\} \subset V$ satisfying

$$v_0 = s, \quad v_N = \tau, \quad (v_n, v_{n+1}) \in E \; \forall n.$$

Then the discrete deterministic shortest-path (DSP) problem is

$$\{v_n^*\} = \arg \min_{\{v_n\} \in P} \sum_{n=0}^{N-1} c_{n,n+1}.$$

Since DSP is NP-hard in general, we employ heuristic search (e.g. A*) and sampling-based planners (e.g. RRT*) to efficiently approximate $\{v_n^*\}$.

III. Technical Approach

A. Collision Checking

We implement collision checking in `functions.py` via two routines: `segment_intersects_aabb()` and `collision_check_path()`. The former tests whether a single line-segment intersects an axis-aligned bounding box (AABB) using the slab method, while the latter applies this test along with an outer-boundary check to an entire piecewise-linear path.

a) Segment–AABB Intersection (Slab Method): Given a segment from p_0 to p_1 and an AABB defined by its minimum corner $b_{\min}$ and maximum corner $b_{\max}$, we parameterize points on the segment as

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

For each axis $i \in \{x, y, z\}$, we compute the entry and exit parameters

$$t_i^{\min} = \frac{b_{\min,i} - p_{0,i}}{d_i}, \quad t_i^{\max} = \frac{b_{\max,i} - p_{0,i}}{d_i},$$

where $d = p_1 - p_0$. We accumulate

$$t_{\text{enter}} = \max_i \min(t_i^{\min}, t_i^{\max}), \quad t_{\text{exit}} = \min_i \max(t_i^{\min}, t_i^{\max}).$$

If $t_{\text{enter}} \le t_{\text{exit}}$, the segment intersects the box; otherwise it does not.

b) Path Collision Checking: To verify a candidate path $\{p_0, \ldots, p_N\}$ against the workspace:

1) Check each waypoint p_k lies inside the outer boundary AABB $(b_{\min}, b_{\max})$.
2) For each segment $[p_i, p_{i+1}]$ and each obstacle AABB $(o_{\min}^j, o_{\max}^j)$, call `segment_intersects_aabb`.

On detecting any out-of-bounds waypoint or intersecting segment, the function reports the offending index and returns Collision, otherwise it returns Free.

Algorithm 1 Collision Checking for Piecewise-Linear Paths

Require: path points $\{p_0, \ldots, p_N\}$, outer boundary AABB $(b_{\min}, b_{\max})$, obstacles $\{(o_{\min}^j, o_{\max}^j)\}_{j=1}^M$
if $\exists\, k$ s.t. $p_k \notin [b_{\min}, b_{\max}]$ **then**
  report out-of-bounds waypoint k and **return** Collision
end if
for $i = 0$ **to** $N-1$ **do**
  $(p_0, p_1) \leftarrow (p_i, p_{i+1})$
  for $j = 1$ **to** M **do**
    if SegmentIntersectAABB$(p_0, p_1, o_{\min}^j, o_{\max}^j)$ **then**
      report collision on segment $i \to i+1$ with obstacle j
      return Collision
    end if
  end for
end for
return Free =0

B. Weighted A*

To scale our planner to large grids while still guiding the search toward the goal, we implement a *weighted A** variant. Like standard A*, this algorithm maintains a frontier `OPEN` and an explored set `CLOSED`, tracking for each node i its best-known cost $g(i)$ from the start and a heuristic estimate $h(i)$ of the remaining cost to the goal. The key difference is that we bias node selection by a weight $\varepsilon \ge 1$, using the priority

$$f(i) \;=\; g(i) \;+\; \varepsilon\, h(i).$$

A larger ε pushes the search more greedily toward the goal, often reducing node expansions at the expense of (controlled) suboptimality.

Algorithm 2 Weighted A*

Require: start s, goal τ, edge costs $c(i,j) > 0$, heuristic $h(i)$, weight $\varepsilon \ge 1$
`OPEN` $\leftarrow \{s\}$, `CLOSED` $\leftarrow \emptyset$
$g(s) \leftarrow 0$, $g(i) \leftarrow \infty$ for all $i \ne s$
while $\tau \notin$ `CLOSED` **do**
  remove $i \in$ `OPEN` minimizing $f(i) = g(i) + \varepsilon\, h(i)$
  `CLOSED` $\leftarrow$ `CLOSED` $\cup \{i\}$
  for all $j \in \text{Children}(i) \setminus$ `CLOSED` **do**
    if $g(i) + c(i,j) < g(j)$ **then**
      $g(j) \leftarrow g(i) + c(i,j)$
      $\text{parent}(j) \leftarrow i$
      if $j \in$ `OPEN` **then**
        update j's priority to $g(j) + \varepsilon\, h(j)$
      else
        `OPEN` $\leftarrow$ `OPEN` $\cup \{j\}$
      end if
    end if
  end for
end while
return reconstruct path by backtracking from τ =0

a) Properties:

- **(Sub)optimality:** Weighted A* is ε–suboptimal: the returned path cost satisfies $g(\tau) \le \varepsilon\, \text{dist}(s, \tau)$. As $\varepsilon \to 1$, we recover standard A* and optimality.

- **Completeness:** For any finite grid and admissible h, the algorithm will terminate with a solution if one exists (or prove none exists).
- **Time efficiency:** By increasing ε, the search expands far fewer states in practice, steering more directly toward the goal.
- **Memory:** Like A*, we store both `OPEN` and `CLOSED`, so worst-case memory is $O(|V|)$. In many maps, the weighted bias keeps `OPEN` smaller.

C. Rapidly-exploring Random Tree*

We implement a continuous-space planner following the RRT* paradigm. The key idea is to grow a tree of feasible motions by sampling the free space, steering from existing vertices toward samples, and then rewiring nearby vertices to improve path costs. Our implementation in `rrtstar.py` uses the following components:

- **Node representation:** Each node stores its coordinate $x \in \mathbb{R}^d$, a pointer to its parent, and the cost $c(x)$ from the root.
- **Sampling:** At each iteration, draw $x_{\text{rand}} \sim \mathcal{U}(B)$ uniformly in the axis-aligned boundary B.
- **Nearest neighbor:** Find the tree node x_{nearest} minimizing $\|x_{\text{nearest}} - x_{\text{rand}}\|$.
- **Steering:** Move from x_{nearest} toward x_{rand} by at most a fixed step size Δ, yielding a candidate x_{new}.
- **Collision checking:** Verify the segment $[x_{\text{nearest}}, x_{\text{new}}]$ is collision-free via `collision_check_path`.
- **Neighborhood search:** Let n be the current tree size. Compute a radius

$$r_n = \min\Big(r_0\big(\tfrac{\log(n+1)}{n+1}\big)^{1/d},\ \Delta\Big),$$

  and collect all existing nodes within r_n of x_{new}.
- **Choose parent:** Among those neighbors, pick the one that yields minimal cost $c(x_{\text{near}}) + \|x_{\text{near}} - x_{\text{new}}\|$.
- **Rewire:** For each neighbor, if going via x_{new} lowers its cost, reset its parent to x_{new} and update costs.
- **Termination:** Whenever a new node falls within Δ of the goal and the goal-segment is collision-free, connect and extract the final path by backtracking.

a) Properties:

- **Probabilistic completeness:** As iteration count $\to \infty$, the probability of finding a path (if one exists) approaches 1.
- **Asymptotic optimality:** RRT* refines the tree via rewiring so that as samples $\to \infty$, path cost converges almost surely to the optimum.
- **Time efficiency:** Naïve neighbor searches cost $O(n)$ per iteration; overall worst-case is $O(n^2)$, but practical runtimes remain reasonable for moderate $\max\_iter$.
- **Memory:** Stores all tree nodes—worst-case $O(n)$ memory.

IV. Discussion

We evaluate both planners on seven benchmark maps—*single cube*, *maze*, *flappy bird*, *pillars*, *window*, *tower*, and *room*—using the same start/goal and obstacle layouts. For Weighted A* we set $\varepsilon = 1$ (standard A*), and for RRT* we use step size $\Delta = 0.5$, neighbor-radius constant $r_0 = 1.0$, and `max_iter` $= 10^5$. We compare (1) path cost, (2) nodes expanded, and (3) planning time.

Algorithm 3 RRT*

Require: start s, goal τ, boundary B, obstacles $\{O_j\}$, Δ, r_0, max_iter
$\mathcal{T} \leftarrow \{s\}$, set $c(s) = 0$
for $k = 1$ to max_iter **do**
  $x_{\text{rand}} \sim \mathcal{U}(B)$
  $x_{\text{nearest}} \leftarrow \arg\min_{x \in \mathcal{T}} \|x - x_{\text{rand}}\|$
  $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x_{\text{rand}}, \Delta)$
  if $\neg\text{CollisionFree}(x_{\text{nearest}}, x_{\text{new}})$ **then continue**
  end if
  compute $r = \min\big(r_0(\frac{\log|\mathcal{T}|}{|\mathcal{T}|})^{1/d}, \Delta\big)$
  $\mathcal{N} \leftarrow \{\, x \in \mathcal{T} \mid \|x - x_{\text{new}}\| \le r\}$
  $x_{\min} \leftarrow \arg\min_{x \in \mathcal{N}} \big(c(x) + \|x - x_{\text{new}}\|\big)$
  set parent$(x_{\text{new}}) \leftarrow x_{\min}$, $c(x_{\text{new}}) \leftarrow c(x_{\min}) + \|x_{\min} - x_{\text{new}}\|$
  $\mathcal{T} \leftarrow \mathcal{T} \cup \{x_{\text{new}}\}$
  for all $x \in \mathcal{N}$ **do**
    if $c(x_{\text{new}}) + \|x_{\text{new}} - x\| < c(x)$ **then**
      parent$(x) \leftarrow x_{\text{new}}$, update $c(x)$
    end if
  end for
  if $\|x_{\text{new}} - \tau\| \le \Delta$ and $\text{CollisionFree}(x_{\text{new}}, \tau)$ **then**
    connect τ, return path by backtracking
  end if
end for
return failure =0

TABLE I: Planner Performance on Benchmark Maps

Map	Weighted A* ($\varepsilon = 1$)			RRT*		
	Expansions	Time (s)	Length	Expansions	Time (s)	Length
Single cube	23 738	1.06	13.75	24	0.00147	13.35
Maze	94 608	14.69	83.50	8 234	96.89	116.74
Flappy bird	32 566	2.59	38.00	619	0.28	43.19
Pillars	153 624	22.83	45.50	577	0.29	39.46
Window	69 512	7.10	31.75	556	0.23	36.66
Tower	23 698	4.89	42.50	600	0.44	41.39
Room	5 066	0.90	14.50	245	0.13	15.30

A. Path Quality

We compare the actual path lengths produced by each planner (Table 1). Weighted A* always finds the true shortest path in the discretized grid, yielding lengths like 13.75 in the single-cube map and 83.50 in the maze. RRT* returns slightly longer, continuous-space paths—for example 13.35 vs. 13.75 in the single cube, and 116.74 vs. 83.50 in the maze—because its randomized exploration only asymptotically converges to the optimum.

B. Node Expansions and Time

Weighted A* expands orders of magnitude more nodes in cluttered layouts (e.g. 154 k in *pillars*) and incurs greater

runtime overall. RRT* typically uses only a few hundred expansions per map, yielding very fast planning in open spaces but requiring many iterations to refine paths in highly constrained maps like *maze*.

C. Effect of Parameters

a) Weighted A step size:* To guarantee success on every map, we found that a finer grid "step size" of 0.2 (instead of 0.5) was required so that narrow corridors and thin obstacles were represented accurately. It was found that certain step sizes would essentially never allow for the goal node to be discovered due to the value given to the node. This may be an issue with implementation but a quick parameter tuning was able to fix this. Reducing the step size increases the number of grid vertices and edges, which in turn raises the node-expansion count and overall runtime, but ensures completeness on all seven benchmarks.

b) RRT iterations:* Because RRT* is only probabilistically complete, maps with long, winding passages (e.g. the maze) demanded a very high iteration budget to approach failure probability 0. We tuned `max_iter` upward (to 10^5 samples) so that even the most constrained layouts were connected with high probability. Lowering `max_iter` speeds up planning but risks missing narrow passages altogether.

D. Visual Comparisons

Overall, Weighted A* excels when optimality and completeness are required, while RRT* scales more gracefully in open or moderately cluttered spaces but takes longer to approach optimality in narrow passages. The results for all environments can be found in the next section.

V. Results

A. Weighted A*

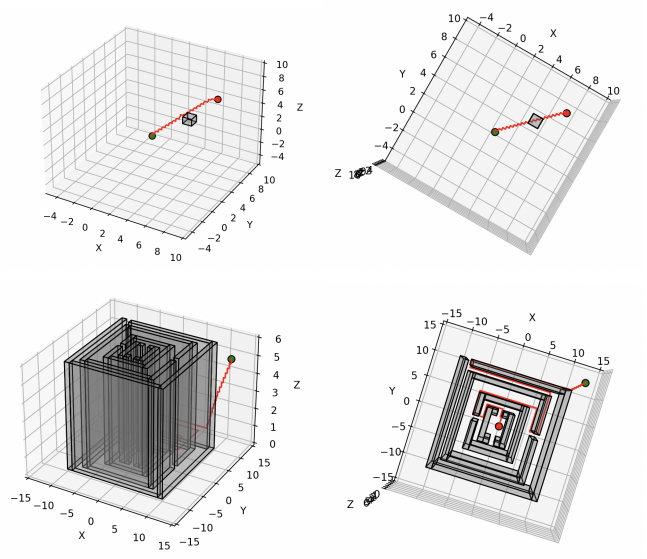

Fig. 2: Weighted A* on Single Cube (top) and Maze (bottom). Left: default view. Right: alternate view.

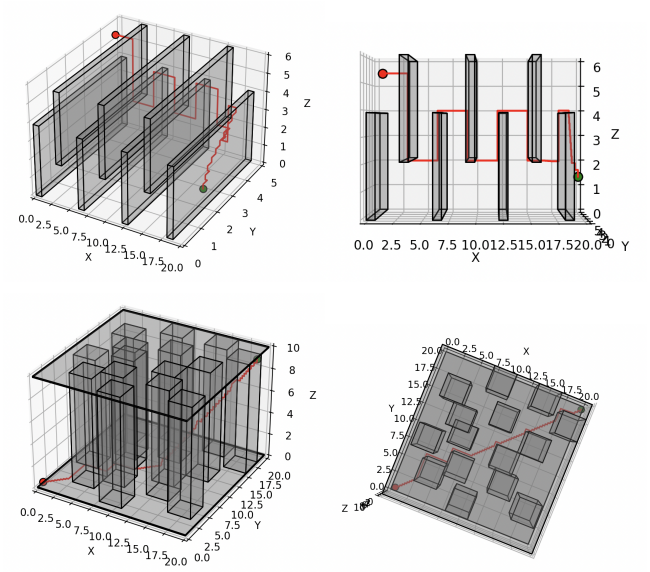

Fig. 3: Weighted A* on Flappy Bird (top) and Pillars (bottom).

B. Rapidly-exploring Random Tree*

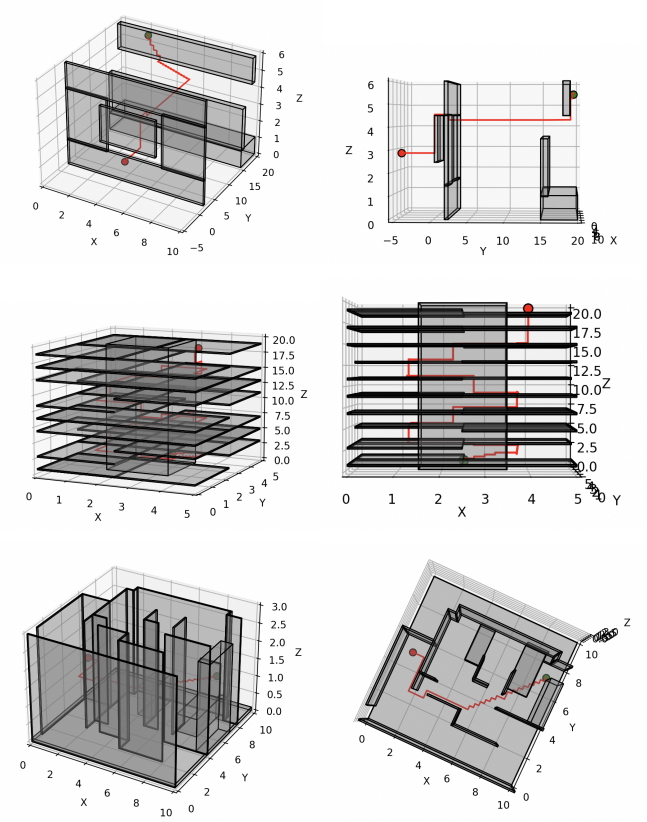

Fig. 4: Weighted A* on Window (top), Tower (middle), and Room (bottom).

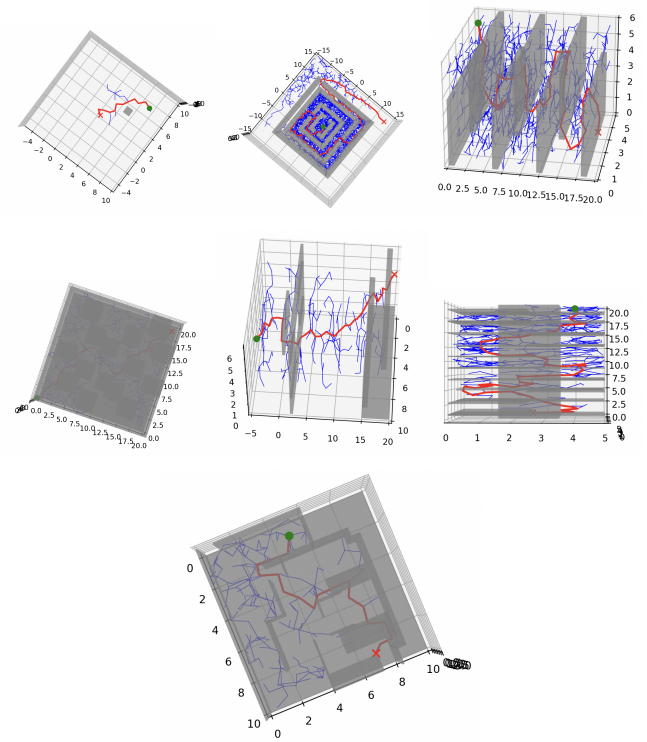

Fig. 5: RRT* on all seven maps. Top row: Single Cube, Maze, Flappy Bird. Middle row: Pillars, Window, Tower. Bottom: Room.