Motion planning for navigation in 3D space

Arivoli Anbarasu
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, CA 92093
aanbarasu@ucsd.edu

***Abstract*—Identifying a collision-free path for navigation in a dynamic environment is a challenging problem. While methods such as dynamic programming can yield optimal trajectories by exhaustively evaluating all possible configurations of the robot, they are often computationally intensive and impractical for real-time applications, especially in complex and high-dimensional configuration spaces. In this report, we address the problem of motion planning in a 3D Euclidean space containing rectangular obstacles. We present a comparative study of greedy algorithms, search-based planning, and sampling-based planning to evaluate the strengths and limitations of each approach.**

***Index Terms*—A* algorithmn, RRT*, configuration space, collision check.**

I. Introduction

Motion planning involves identifying a safe path considering the obstacles in the environment, the agent's action space and if possible optimize for some goal. Motion planning is inherently challenging due to the high-dimensional nature of configuration spaces, the presence of complex obstacles, and real-time constraints in practical applications such as robotics, autonomous vehicles, and UAVs.

The problem of motion planning can be seen as **Deterministic Shortest Path (DSP)** problem and many algorithmns exist to solve the DSP problem. The algorithmns can be broadly classfied into **search based** and **sampling based**. Search based algorithmn involves searching the shortest path from the start node to end node in a deterministically constructed graph. While, sampling based algorithm involves random sampling of feasible states from start node to the goal node.

In this report, the motion planning for a point robot is done using three approaches namely greedy planning, search - based planning and sampling based planning on seven different challenging environment. A comparison of performance and characteristics of sampling and search based algortihmn is done.

The mathematical formulation of the problem is presented in Section II. Section III details the solution technique and methodology. Finally, Section IV provides a comprehensive analysis of the findings and experimental observations.

II. Problem formulation

In this following subsections, the mathematical formulation of the problem is described.

A. Deterministic Shortest Path Problem

Deterministic Shortest Path problem involves finding sequence of vertices $\mathcal{V}_{opt}$ to travel in a graph made of vertex set $\mathcal{V}$ and egde set $\epsilon$and edge weights $\mathcal{C}$. Figure 1 illustrates an simple deterministic shortest path from start node A to goal node D. The highlighed path in red depicts the shortest path and the optimal path is A, B, C, D.

For the 3D navigation problem, the start node $s = [x_0, y_0, z_0]$ is taken as the starting location of the robot, and the goal node $g = [x_\tau, y_\tau, z_\tau]$ is taken as the goal location. To visualize the problem as a graph, each feasible location $[x_i, y_i, z_i]$ is treated as a node v_i, and the line segment connecting two feasible nodes is considered an edge of the graph, denoted ϵ_{ij}. For the navigation problem, the edge cost c_{ij} is defined as the Euclidean distance along the line segment joining v_i and v_j.

In this work, we use Axis-Aligned Bounding Box (AABB) obstacles. AABBs are among the simplest types of obstacles to represent, as their edges are aligned with the axes of the world coordinate system. This alignment significantly simplifies collision detection with points, line segments, and other AABB objects. An AABB is defined by its minimum corner (or bottom vertex) at $[x_{\min}, y_{\min}, z_{\min}]$ and its maximum corner (or top vertex) at $[x_{\max}, y_{\max}, z_{\max}]$, representing the bounds of the cuboid in 3D space.

The feasible nodes $v_i = [x_i, y_i, z_i]$ in the environment is found by checking whether the coordinate is inside the environment and the coodinate is not inside the obstacle AABB. Let the point be $\mathbf{p} = [x, y, z]$, and let each obstacle $\mathcal{O}_k$ be represented by its lower and upper bounds $[x^k_{\min}, y^k_{\min}, z^k_{\min}]$, $[x^k_{\max}, y^k_{\max}, z^k_{\max}]$. Let δ denote the clearance given to the planner for safety, and the environment boundary be defined by $[x^b_{\min}, y^b_{\min}, z^b_{\min}]$, $[x^b_{\max}, y^b_{\max}, z^b_{\max}]$.

A point $\mathbf{p}$ is considered **invalid** if it lies within any obstacle (with clearance) or lies **outside** the boundary. Equation 1 and Equation 2 depicts the condition in equation form.

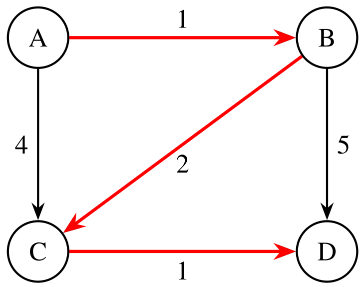


Fig. 1: Illustration of Deterministic Shortest Path Problem.

$$
\begin{aligned}
x^k_{\min} - \delta \le x \le x^k_{\max} + \delta, \\
y^k_{\min} - \delta \le y \le y^k_{\max} + \delta, \\
z^k_{\min} - \delta \le z \le z^k_{\max} + \delta
\end{aligned}
\tag{1}
$$

$$
\begin{aligned}
x < x^b_{\min} \quad \text{or} \quad x > x^b_{\max} \quad \text{or} \\
y < y^b_{\min} \quad \text{or} \quad y > y^b_{\max} \quad \text{or} \\
z < z^b_{\min} \quad \text{or} \quad z > z^b_{\max}
\end{aligned}
\tag{2}
$$

III. Technical Approach

Solving the deterministic shortest path problem defined in section II to obtain optimal trajectory is difficult. In the following subsections we will look into sample based motion planning algorithmn, sampling based motion planning algorithmn and collision algorithmn developed for verification of path obtained.

A. Collision detection algorithmn

Once a path is generated using a motion planning algorithm, it is essential to verify whether the path is collision-free before execution. To this end, we implement a path validation step that checks for intersections between each path segment and a set of static obstacles represented using Axis-Aligned Bounding Boxes (AABBs).

The path, represented as a sequence of points $\{p_0, p_1, \ldots, p_n\}$, is broken into individual line segments (p_i, p_{i+1}).

For each segment, we compute the shortest Euclidean distance to every AABB. If the distance between any segment and any box is exactly zero, a collision is detected, indicating that the path is invalid. If all segments clear all obstacles, the path is deemed valid.

Algorithmn **??** formally outlines the procedure used to detect collisions between a planned path and a set of AABB obstacles.

B. Weighted A*

Weighted A* is a variant of the standard A* search algorithm that prioritizes faster convergence over guaranteed optimality. The standard A* algorithm selects nodes for expansion based on the heuristic distance from start through goal through a node. **??** depics the function used.

Algorithm 1 Path–Obstacle Collision Checking Using AABBs

Require: A path as an ordered list of points $P = \{p_0, p_1, \ldots, p_n\}$, and a set of axis-aligned bounding box (AABB) obstacles $\mathcal{B} = \{B_1, B_2, \ldots, B_m\}$
Ensure: Boolean flag indicating whether the path collides with any obstacle

for each obstacle $B_j \in \mathcal{B}$ **do**
  Compute center c_j and size s_j of the box from its min and max corners
end for
for each segment $(p_i, p_{i+1}) \in P$ **do**
  for each box (c_j, s_j) **do**
    Compute the shortest distance d_{ij} between the segment and box B_j
    if $d_{ij} = 0$ **then**
      return True {Collision detected}
    end if
  end for
end for
return False {No collisions detected}

$$f_i = g_i + \epsilon h_i \quad \epsilon \ge 1 \tag{3}$$

where g_i is the cost to reach node i from the start, and h_i is the heuristic estimate from i to the goal. In Weighted A*, the heuristic is scaled by a factor $\epsilon \ge 1$. This inflation of the heuristic makes the algorithm more greedy, favoring nodes that appear closer to the goal based on the heuristic. As a result, the search often expands fewer nodes and terminates more quickly, at the cost of potentially suboptimal solutions. The A* algorithmn to get optimal path is given in Algorithmn 2

Based on the heuristic to be used, the algorithm is made generalized by allowing nodes to be reopened. The Euclidean Heuristic is admissible and consistent. Thus with $\epsilon = 1$, the planner gives optimal path.

C. RRT*

RRT* is the optimal version of RRT involving **rewiring** - improving the path compared to RRT. The RRT (Rapidly-Exploring Random Tree) algorithm involves randomly sampling nodes from the space and extending the *search tree* by **steering** toward each sampled point from the **nearest** node already in the tree. The tree grows rapidly into unexplored areas of the space. The expansion typically continues until a node reaches sufficiently close to the goal.

In RRT*, an additional **rewiring** step is introduced. After a new node is added to the tree, it checks for nearby nodes within a certain radius to see if connecting through the new node would result in a lower-cost path (from the start). This allows the algorithm to optimize paths over time, eventually converging toward the optimal solution as the number of samples increases [1]. OMPL library [2] is used to implement

Algorithm 2 A* Search Algorithm

Require: Graph G, start node s, goal node g, heuristic $h(n)$
Ensure: Path from s to g, or failure

Initialize open set OPEN $\leftarrow \{s\}$
Initialize closed set CLOSED $\leftarrow \emptyset$
$g(s) \leftarrow 0$
$f(s) \leftarrow g(s) + h(s)$
Store parent$(s) \leftarrow$ None
while OPEN $\neq \emptyset$ **do**
    $n \leftarrow \arg\min_{x \in \text{OPEN}} f(x)$
    if $n = g$ **then**
        return reconstructed path from s to g
    end if
    Remove n from OPEN, add to CLOSED
    for all neighbors m of n **do**
        Tentative cost $t \leftarrow g(n) + \text{cost}(n, m)$
        if $m \notin$ OPEN and $m \notin$ CLOSED **then**
            Add m to OPEN
        end if
        if $t < g(m)$ (or $g(m)$ undefined) **then**
            $g(m) \leftarrow t$
            $f(m) \leftarrow g(m) + h(m)$
            parent$(m) \leftarrow n$
            if $m \in$ CLOSED **then**
                Remove m from CLOSED, add back to OPEN
            end if
        end if
    end for
end while
return failure

TABLE I: Path length obtained from motion planning

Environment	Greedy	Weighted A*	RRT*
Cube	8.00	7.99	7.89
Maze	1000.00	76.23	75.19
Flappy bird	1000.00	31.34	29.49
Pillars	1000.00	31.06	28.75
Window	1000.00	27.78	25.89
Tower	1000.00	29.78	29.79
Room	1000.00	11.72	10.92

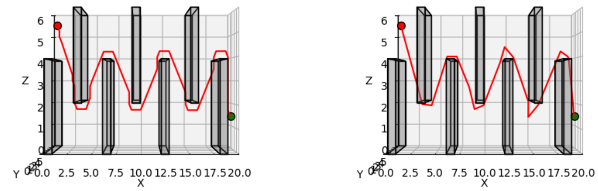
algorithmn in this work. Algorithmn **??** depicts the RRT* used in OMPL.

IV. Result

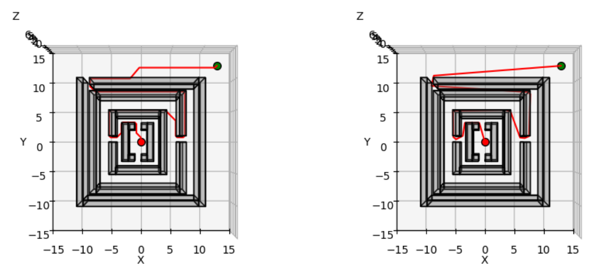
The DSP problem is solved for seven different environment namely *Cube*, *Maze*, *Flappy bird*, *Pillars*, *Window*, *Tower*, *Room* and the results for each is shown below.

A. Comparison of different planner

Table I summarizes the optimal path length obtained across different environments. All plots from different views are avaiable in the code submission. The trajectory generated by A* and RRT* for different environment is illustrated in Figure 2, 3, 4, 5.

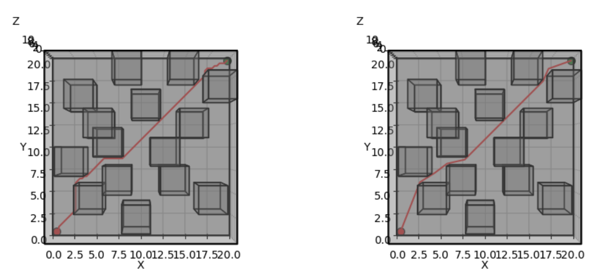


a) Weighted A* a) RRT*

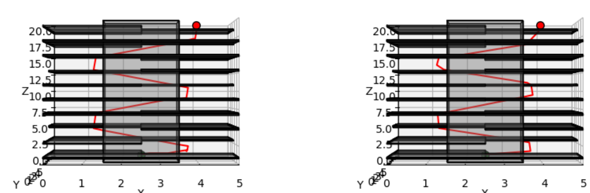


b) Weighted A* b) RRT*

Fig. 2: Comparison of trajectory in a) Flappy bird b) Maze.



c) Weighted A* c) RRT*



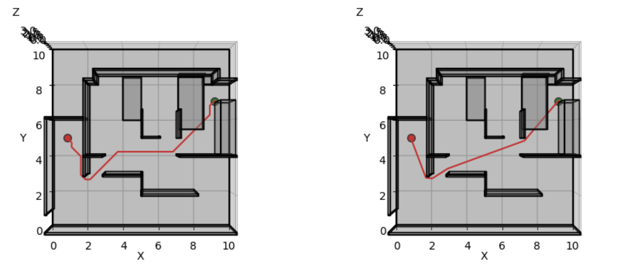
d) Weighted A* d) RRT*

Fig. 3: Comparison of trajectory in c) Pillars d) Tower.

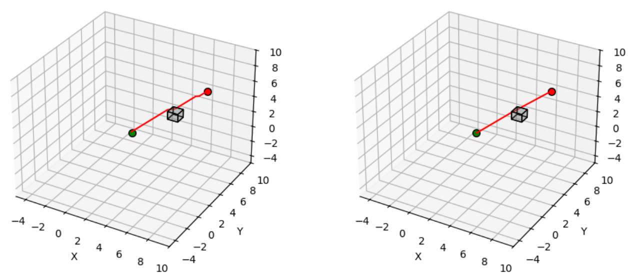
Algorithm 3 RRT* (Rapidly-exploring Random Tree Star)

Require: Start state x_{init}, goal region X_{goal}, obstacle space X_{obs}, number of iterations N
Ensure: A collision-free path from x_{init} to X_{goal} (asymptotically optimal)

Initialize tree $T \leftarrow \{x_{\text{init}}\}$
for $i = 1$ to N **do**
  Sample a random state $x_{\text{rand}} \in X$
  $x_{\text{nearest}} \leftarrow \text{Nearest}(T, x_{\text{rand}})$
  $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x_{\text{rand}})$
  if ObstacleFree$(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \text{Near}(T, x_{\text{new}}, r)$ {Find nearby nodes within radius r}
    $x_{\text{min}} \leftarrow x_{\text{nearest}}$
    $c_{\text{min}} \leftarrow \text{Cost}(x_{\text{nearest}}) + \text{LineCost}(x_{\text{nearest}}, x_{\text{new}})$
    for all $x_{\text{near}} \in X_{\text{near}}$ **do**
      if ObstacleFree$(x_{\text{near}}, x_{\text{new}})$ **then**
        $c_{\text{near}} \leftarrow \text{Cost}(x_{\text{near}}) + \text{LineCost}(x_{\text{near}}, x_{\text{new}})$
        if $c_{\text{near}} < c_{\text{min}}$ **then**
          $x_{\text{min}} \leftarrow x_{\text{near}}$
          $c_{\text{min}} \leftarrow c_{\text{near}}$
        end if
      end if
    end for
    Add edge $(x_{\text{min}}, x_{\text{new}})$ to T
    for all $x_{\text{near}} \in X_{\text{near}}$ **do**
      if ObstacleFree$(x_{\text{new}}, x_{\text{near}})$ **and** $\text{Cost}(x_{\text{new}}) + \text{LineCost}(x_{\text{new}}, x_{\text{near}}) < \text{Cost}(x_{\text{near}})$ **then**
        Change parent of x_{near} to x_{new}
      end if
    end for
  end if
end for
return Best path from x_{init} to X_{goal}

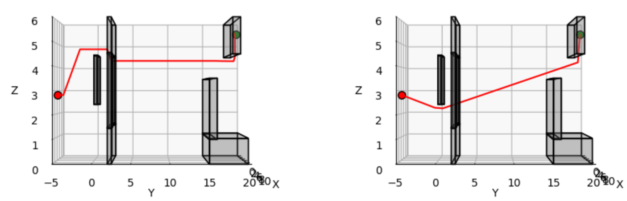


e) Weighted A* e) RRT*



f) Weighted A* f) RRT*

Fig. 4: Comparison of trajectory in e) Room f) Cube.



g) Weighted A* g) RRT*

Fig. 5: Comparison of trajectory in g) Window.

B. A : Heuristic*

For the A* algorithm, Euclidean (default) and Manhattan distance are evaluated for different environment. One observation is that Manhattan heuristic opens less nodes and thus achives goal faster compared to euclidean heuristic. Figure 6 illustrates the frontier of A* for euclidean and manhattan for room and cube environment. In Manhattan, the frontier moves towards the goal, while in euclidean, sometimes the nodes nearer to start opens even when nodes near goal are opened. This can be explained by the fact that the Manhattan heuristic is inadmissible in this case. For example, diagonal movement across a face has a true cost of $\sqrt{2}$, but the Manhattan heuristic estimates it as 2. Similarly, diagonal movement across the cube has a true cost of $\sqrt{3}$, while Manhattan estimates it as 3. Because it overestimates the actual cost, the heuristic is inadmissible, which causes A* to expand fewer nodes and reach the goal faster, though the resulting path may not be optimal. Table II depicts the path length identified by A* algorithm for euclidean and manhattan heuristics.

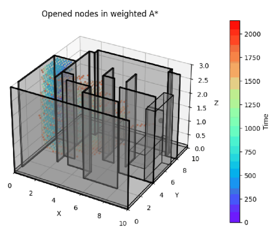
C. RRT : Goal bias*

In our implementation, the goal bias parameter in RRT* (as provided in OMPL) is evaluated to understand its impact on the quality of the resulting paths across various environments. The *goal bias* controls the probability of sampling the goal state directly during the random sampling process. A higher goal bias encourages the planner to attempt connections to the goal more frequently, which can influence convergence speed and path quality.

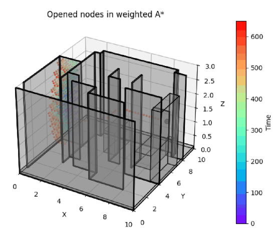
Table III shows the comparison of path lengths for two different goal bias values: 0.05 (default in OMPL) and 0.5 (a more aggressive bias towards the goal).

TABLE II: Path length obtained with different heuristics

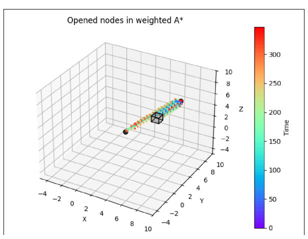
Environment	**Euclidean**	**Manhattan**
Cube	7.9889	7.9889
Maze	76.2315	76.2315
Flappy bird	31.3383	31.3383
Pillars	31.0613	31.4227
Window	27.7809	27.9639
Tower	29.7870	29.8352
Room	11.7175	12.1569



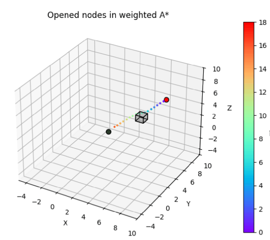
a) Euclidean



a) Manhattan



b) Euclidean



b) Manhattan

Fig. 6: Comparison of opened nodes in weghted A* for different heyristics in a) room b) cube.

TABLE III: Comparison of path length for different goal bias in RRT*

Environment	**0.05**	**0.5**
Cube	7.89	7.94
Maze	75.19	75.36
Flappy bird	29.49	29.08
Pillars	28.75	28.56
Window	25.89	25.81
Tower	29.79	30.12
Room	10.92	11.12

V. Acknowledgment

I would like to express my sincere gratitude to the professor and TA team for their valuable support throughout this project. The discussion on Piazza was particularly helpful in troubleshooting various challenges I encountered, and I truly appreciate the collaborative effort.

References

[1] N. Atanasov, “Sampling-based motion planning (lecture 9: Planning & learning in robotics, ece276b),” Online lecture notes, 2025, eCE276B: Planning & Learning in Robotics, University of California San Diego. Available at: https://natanaso.github.io/ece276b/ref/ECE276B_9_SamplingBasedPlanning.pdf.

[2] I. A. Şucan, M. Moll, and L. E. Kavraki, “The Open Motion Planning Library,” *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, https://ompl.kavrakilab.org.