Search-based Planning: A* vs ANA*

Srushti Hippargi

***Abstract*—This report compares the ANA* and A* algorithms for solving navigation problems in an 8-connected grid space. ANA* is an anytime algorithm that quickly finds an initial suboptimal path and improves it over time, while A* guarantees the optimal path from the start. Both algorithms were evaluated using the Euclidean, Octile, Manhattn and Chebyshev distance heuristics to understand their performance. The results show that ANA* is significantly faster in finding an initial solution, making it ideal for time-sensitive tasks, while A* requires more time but delivers the optimal path in a single computation. The choice of heuristic was found to greatly influence the speed and quality of the solutions for both algorithms. To run with GUI (only in Ubuntu 22), use `python3 demo.py --use_gui_d`, and to run without GUI (on Ubuntu 24), use `python3 demo.py --no_gui_d`.**

I. Introduction

Pathfinding is an important problem in robotics and AI. It helps robots move from a starting position to a target while avoiding obstacles and minimizing travel cost. The A* algorithm [1] is commonly used for this because it always finds the best (optimal) path when a good heuristic is used. However, A* can take a lot of time in large or complex environments since it focuses on finding the perfect path before stopping. In real-world applications, like robot navigation, sometimes a "good enough" path found quickly is more useful than waiting for the best one.

The ANA* (Anytime Nonparametric A*) algorithm [2] solves this by finding an initial path very quickly and then improving it over time. This makes ANA* ideal for time-sensitive tasks where quick solutions are needed. In this report, we compare A* and ANA* on an 8-connected grid, where robots can move in all directions, including diagonally. We also test different distance heuristics like Octile, Euclidean, and Chebyshev to see how they affect performance. Our goal is to measure how fast each algorithm works, how good the paths are, and how the choice of heuristic makes a difference.

II. Implementation

This section explains the implementation of A* and ANA* algorithms for pathfinding on an 8-connected grid space, incorporating rotational components. Different heuristics were used to guide the search and evaluate performance.

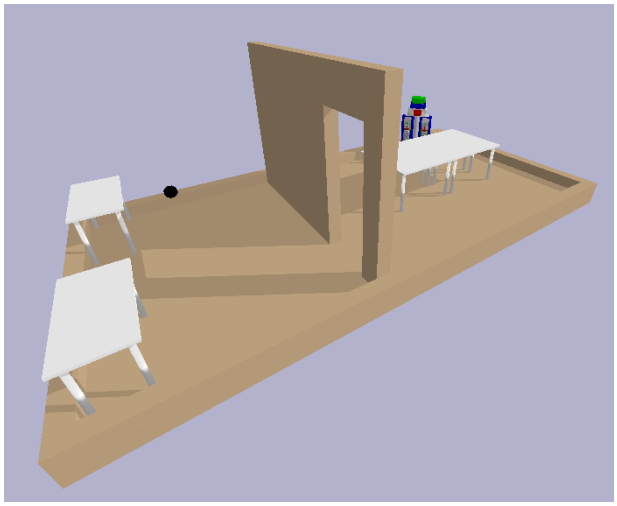

Fig. 1: Scene 1: PR2 doorway (modified)

A. Node Representation

Each grid state is represented as a `Node` with the following attributes:

- x, y, θ: Position and orientation. These represent the node's location in 2D space (x, y) and its heading direction θ, which accounts for rotational motion in realistic scenarios.
- `cost`: Cost to reach the node from the start. This evaluates the cumulative effort to traverse the path.
- `heuristic`: Estimated cost to the goal. The heuristic function provides an informed guess.

B. Heuristic Functions

The following heuristics were implemented to estimate the cost to the goal. Each heuristic [3] has specific use cases and impacts the performance of A* and ANA* in terms of path quality and computational efficiency:

- **Octile Distance**:
  $$h = \max(\Delta x, \Delta y) + (\sqrt{2} - 1) \cdot \min(\Delta x, \Delta y)$$
  The Octile distance is tailored for grid-based environments where diagonal movements are allowed. It balances accuracy and computational efficiency by approximating straight-line motion through diagonal steps.
- **Euclidean Distance**:
  $$h = \sqrt{(\Delta x)^2 + (\Delta y)^2}$$

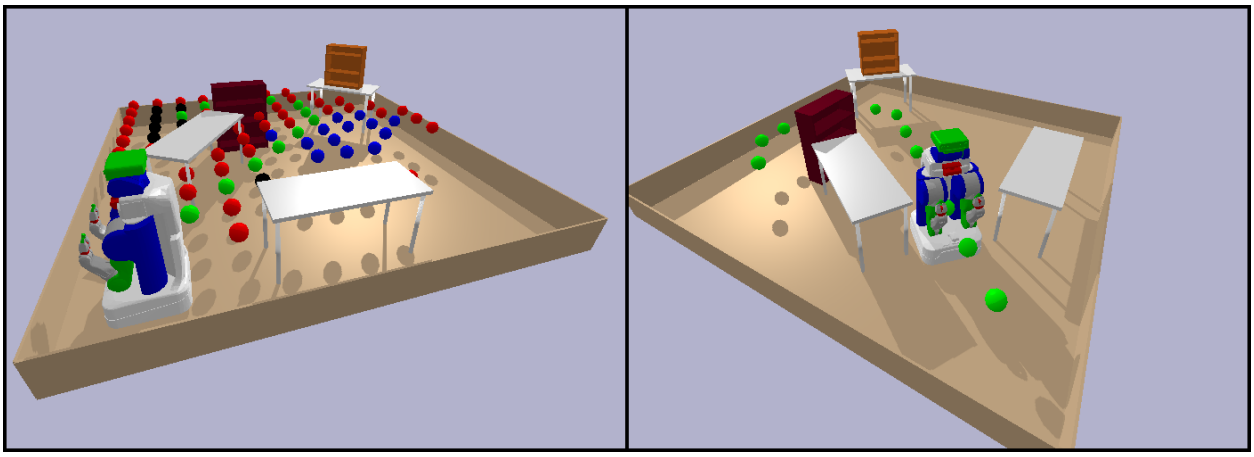

Fig. 2: ANA* vs A* path in the Obstacle scene

Euclidean distance provides the most accurate estimate of straight-line movement, making it ideal when precise cost estimation is critical. However, it is computationally heavier than simpler heuristics.

- **Chebyshev Distance**:
$$h = \max(\Delta x, \Delta y)$$
Chebyshev distance assumes uniform cost for both straight-line and diagonal moves.
- **Manhattan Distance**:
$$h = |\Delta x| + |\Delta y|$$
Manhattan distance is effective in structured grid spaces where only horizontal and vertical moves are allowed, such as urban navigation scenarios.

Here, Δx and Δy represent the differences in position between the current node and the goal. Comparing these heuristics allows us to analyze their trade-offs:

- **Accuracy:** Euclidean and Octile distances provide more precise estimates, giving better path quality.
- **Efficiency:** Manhattan and Chebyshev distances are computationally simpler, reducing overhead for heuristic calculations.
- **Applicability:** Each heuristic aligns with specific assumptions about the environment, such as whether diagonal moves are permitted.

C. Neighbor Generation

Each node generates neighbors by considering both movement and rotation:

- **8-connected moves:** Horizontal, vertical, and diagonal moves are allowed, ensuring flexibility in navigating the grid.
- **Rotation:** $\theta \pm \Delta\theta$, where the rotational increment $\Delta\theta$ enables orientation changes.

The combination of 8-connected moves and rotational updates ensures the algorithms can adapt to more complex motion models, improving the applicability to real-world navigation tasks.

D. Traversal Cost

The traversal cost to move between two nodes (n, m) is defined as:
$$g(n, m) = \sqrt{(\Delta x)^2 + (\Delta y)^2 + (\Delta\theta)^2}$$

Here:

- Δx and Δy are the differences in position.
- $\Delta\theta$ is the smallest angular difference, capturing the cost of changing orientation.

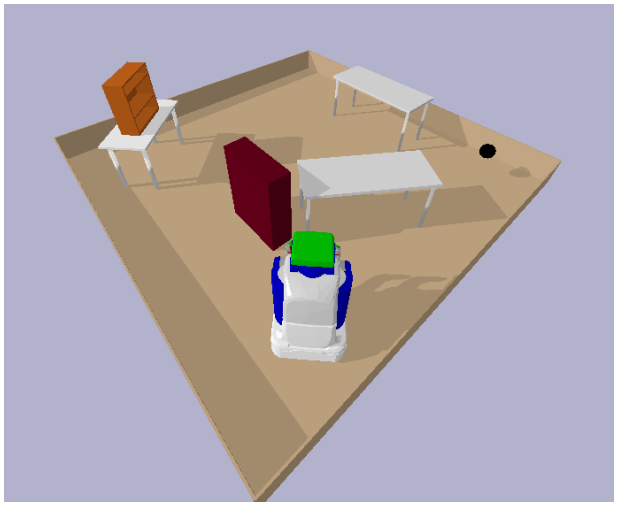

Fig. 3: Scene 2: OBSTACLES

This cost function incorporates both translation and rotation, reflecting the true effort of moving in a grid with orientation changes. Below are the overviews of both algorithms used in this project.

Algorithm 1 A* Algorithm

Input: Start node (s), Goal node (g), Heuristic (h)
Output: Optimal path from s to g or failure
Initialize open list as a priority queue
Initialize closed list as an empty set
s.cost $\leftarrow 0$, s.heuristic $\leftarrow h(s, g)$
Insert s into open list
while open list is not empty **do**
    $current \leftarrow$ node with $\min(f = \text{cost} + \text{heuristic})$ in open list
    if $current = g$ **then**
        return Path by backtracking from $current$
    Move $current$ from open list to closed list
    for all neighbor n of $current$ **do**
        if $n \in$ closed list **then**
            continue
        $g_{\text{tent}} \leftarrow current.\text{cost} + g(current, n)$
        if $n \notin$ open list **or** $g_{\text{tent}} < n.\text{cost}$ **then**
            n.cost $\leftarrow g_{\text{tent}}$
            n.heuristic $\leftarrow h(n, g)$
            n.parent $\leftarrow current$
            Insert or update n in open list
end while
return Failure

Algorithm 2 ANA* Algorithm

Input: Start node (s), Goal node (g), Heuristic (h), Time limit (T)
Output: Path from s to g (improves over time) or failure
Initialize open list as a priority queue
Initialize closed list as an empty set
s.cost $\leftarrow 0$, s.heuristic $\leftarrow h(s, g)$
Insert s into open list
$best\_path \leftarrow \emptyset$
Start timer
while open list is !empty **and** time elapsed $< T$ **do**
    $current \leftarrow$ node with $\min(f = \text{cost} + \text{heuristic})$ in open list
    if $current = g$ **then**
        $best\_path \leftarrow$ backtrack from $current$
        Update cost $current$.cost
    Move $current$ from open list to closed list
    for all neighbor n of $current$ **do**
        if $n \in$ closed list **then**
            continue
        $g_{\text{tent}} \leftarrow current.\text{cost} + g(current, n)$
        if $n \notin$ open list **or** $g_{\text{tent}} < n.\text{cost}$ **then**
            n.cost $\leftarrow g_{\text{tent}}$
            n.heuristic $\leftarrow h(n, g)$
            n.parent $\leftarrow current$
            Insert or update n in open list
end while
return $best\_path$ (may be suboptimal) or Failure

E. Collision Handling

A collision detection function ensures that moves leading to invalid positions (e.g., overlapping obstacles) are excluded during neighbor generation.

III. Results

This section details the experimental setup, the tests conducted to evaluate the A* and ANA* algorithms, and the results obtained. The performance of both algorithms was compared using various heuristics to understand their behavior in terms of solution quality and computational efficiency. A clear cut comparison for the paths is shown in Fig. 2

A. Experimental Setup

The experiments were conducted on an 8-connected grid space with rotational components, simulating the PR2 robot, same as the various assignments in the course.

- **Grid Environment:** 2 test environments were used: the Fig. 1 is from the Homework assignments and the other scene is found online with some obstacles in the open space as shown in Fig. 3
- **Algorithms:** A* and ANA* were implemented with the same heuristic and environment for a fair comparison.
- **Heuristics:** Octile, Euclidean, Chebyshev, and Manhattan heuristics were tested to observe their impact on performance.

B. Experiments

The following experiments were conducted to evaluate the performance of A* and ANA* under different conditions:

1) **Effect of Heuristics on Solution Quality:** Both algorithms were tested using the four heuristics to analyze their impact on path optimality.
2) **Computation Time Comparison:** The time taken to compute the path for each heuristic was recorded for both A* and ANA*.
3) **Anytime Performance of ANA*:** ANA* was evaluated under a time limit to observe how the solution quality improved over time.

C. Summary of Results

The following observations were made:

- Both A* and ANA* always found the optimal path, demonstrating correctness.

- ANA* produced an initial solution faster than A* but took longer overall to refine the path.
- A* consistently completed its execution faster than ANA* in most cases, especially for environments with fewer obstacles.
- The choice of heuristic impacted the computation time, with simpler heuristics (Manhattan and Chebyshev) performing faster but producing paths similar in quality to complex heuristics (Octile and Euclidean).

D. Comparison of Algorithms

The results for both environments and heuristics are summarized in Tables I and II.

TABLE I: Performance Comparison in `obstacles.json` Environment

Algorithm	**Heuristic**	**Initial Solution (s)**	**Total Runtime (s)**
ANA*	Octile	0.8436	6.0562
A*	Octile	-	3.1580
ANA*	Euclidean	1.3207	5.7371
A*	Euclidean	-	1.8161
ANA*	Manhattan	1.0193	8.3298
A*	Manhattan	-	3.7470
ANA*	Chebyshev	0.6362	6.3066
A*	Chebyshev	-	4.2561

TABLE II: Performance Comparison in `pr2doorway.json` Environment

Algorithm	**Heuristic**	**Initial Solution (s)**	**Total Runtime (s)**
ANA*	Octile	2.9887	30.4267
A*	Octile	-	22.7873
ANA*	Euclidean	5.9289	25.3332
A*	Euclidean	-	18.4395
ANA*	Manhattan	4.9356	30.5423
A*	Manhattan	-	32.1113
ANA*	Chebyshev	1.8781	34.0545
A*	Chebyshev	-	33.2839

E. Analysis of Results

a) Initial Solution Time: ANA* produced an initial solution much faster than A* for all heuristics. This behavior demonstrates the anytime nature of ANA*, which quickly finds a feasible path and refines it over time as seen from the Figures. 4 and 5.

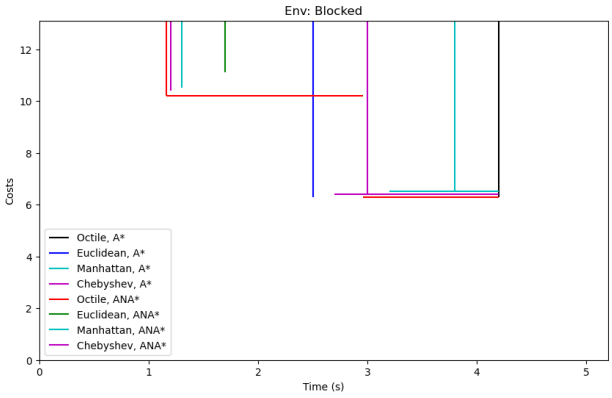

Fig. 4: Performance overtime for Obstacle scene

b) Total Planner Runtime: While ANA* was slower overall, it consistently refined its solutions after finding an initial path. A* performed faster overall as it directly computes the optimal path without intermediate refinements.

c) Heuristic Impact:

- **Octile and Euclidean:** Produced high-quality paths at the cost of increased computation time.
- **Manhattan and Chebyshev:** Faster but slightly less efficient in complex environments. However, the path quality remained comparable.

d) Environment Differences: In the `obstacles.json` environment, both algorithms completed quickly due to fewer obstacles. In contrast, the `pr2doorway.json` environment severe blockage with the path being only the door, leading to longer runtimes. ANA* showed greater variability in runtime as it iteratively improved paths in obstacle-heavy scenarios.

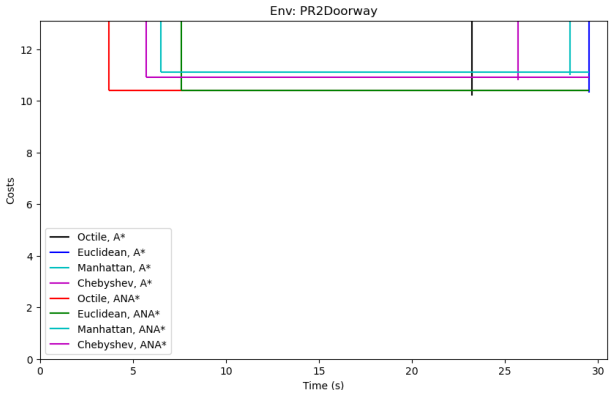

Fig. 5: Performance over time for PR2 doorway

IV. Conclusion

The performance of A* and ANA* algorithms was compared across different heuristics in two environ-

ments: `Blocked` and `PR2Doorway`. The results, as shown in Figures X and Y, reveal the following:

- **Initial Solution Discovery:** In both environments, ANA* consistently finds an initial solution faster than A*, as seen in the sharp drops in cost early in the ANA* curves. This highlights ANA*'s advantage as an anytime algorithm capable of returning feasible solutions quickly.
- **Total Runtime and Path Cost:** A* achieves optimal solutions more efficiently in terms of runtime. The flat horizontal lines in A* plots indicate direct convergence to the optimal path. In contrast, ANA* gradually refines the path cost over time, as evident from its step-wise cost improvements.
- **Heuristic Behavior:** - The **Octile** and **Chebyshev** heuristics perform similarly, as reflected in their comparable path costs and runtime trends.

In conclusion, ANA* excels in applications requiring early feasible solutions with progressive refinements, as observed in its intermediate cost improvements. A* remains more suitable for deterministic, optimal pathfinding with shorter runtimes when computational efficiency is prioritized.

REFERENCES

[1] 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.

[2] E. A. Hansen and R. Zhou, "Anytime heuristic search," *Artificial Intelligence*, vol. 129, no. 1-2, pp. 35–81, 2005.

[3] J. van den Berg, S. Shah, and K. Goldberg, "Ana*: Anytime a* with provable bounds on suboptimality," in *AAAI Conference on Artificial Intelligence*, pp. 1–6, 2011.