SEARCH-BASED PLANNING FOR PR2: ANA* VS A* WITH CUSTOM HEURISTICS

Jinjia Guo

Dec 15, 2024

ABSTRACT

Effective path planning is essential for autonomous robot navigation, especially in complex and dynamic environments. This study uses the PR2 robot as an experimental platform to examine and compare the effectiveness of the A* algorithm and the Anytime Navigation Algorithm (ANA*) in an 8-connected grid environment. ANA* is a anytime search algorithm that gradually enhances inefficient paths over time while ensuring convergence to the best answer. This work examines the influence of heuristic design on the efficacy and safety of path planning algorithms by evaluating the performance of A* and ANA* algorithms in conjunction with several heuristic functions (Euclidean heuristic, modified Euclidean heuristic, and particle filter heuristic). Experimental findings indicate that the Euclidean heuristic efficiently identifies the geometrically optimal path at minimal cost, yet it is inflexible to environmental fluctuations; the modified Euclidean heuristic preserves high efficiency and equilibrium at a marginally increased cost; the particle filter heuristic offers a more secure path planning solution in dynamic and intricate situations by integrating sensor noise and environmental uncertainty assessment, albeit with significant computational overhead. The A* algorithm prioritizes quickness in finding initial solutions and is appropriate for situations with significant real-time demands, whereas the ANA* algorithm demonstrates enhanced robustness and adaptability in dynamic settings via a gradual optimization process. The experimental findings underscore the significant contribution of heuristic functions in enhancing the equilibrium between planning efficiency and safety.To learn more about the Project, visit Github codes

1 Introduction

Path planning is an essential function in robotics, allowing robots to traverse surroundings, circumvent obstacles, and quickly arrive at designated locations. This feature is crucial across several applications, such as autonomous mobile robots, industrial automation, human-robot interaction, and search-and-rescue operations. Autonomous delivery robots must maneuver through warehouses including both static and dynamic barriers, and search-and-rescue robots must rapidly identify pathways in time-sensitive situations. Robots like PR2, intended for human interaction, necessitate advanced planning techniques to navigate intricate situations safely and quickly, rendering path planning essential for applications in healthcare, service robotics, and other fields. Effective path planning relies on heuristics that direct search engines to efficiently navigate the solution space and identify optimal or near-optimal paths.

Numerous real-world situations necessitate path-planning techniques that reconcile optimality and efficiency, particularly in dynamic, resource-limited, or expansive settings. Conventional algorithms such as A* are extensively utilized for their optimality assurances, although they can be computationally intensive, constraining their feasibility in real-time or large-scale applications. The Anytime Navigation Algorithm (ANA*) provides a mechanism for swiftly identifying poor paths and progressively enhancing them towards optimality over time. The efficacy of both A* and ANA* is significantly dependent on the selection of the heuristic. This work presents a custom admissible heuristic to enhance the basic Euclidean heuristic, with the objective of increasing the efficiency and adaptability of both A* and ANA* algorithms. This study examines the influence of several heuristics on solution quality and computing efficiency by comparing them across different navigation circumstances, hence aiding the advancement of more efficient path-planning methodologies for robots in intricate environments.

2 Implementation

2.1 A* algorithm

The A* algorithm [1] is a widely used pathfinding and graph traversal algorithm that finds the least-cost path from a starting node to a target node. It combines the principles of Dijkstra's algorithm and Best-First Search by using both actual cost $g(n)$ and heuristic cost $h(n)$. The total cost function for any node n is defined as:$f(n) = g(n) + h(n)$, where ($g(n)$: The actual cost from the start node to node n,($h(n)$: The heuristic estimate of the cost from node n to the goal,($f(n)$: The total estimated cost.

The A* algorithm maintains two sets of nodes: the *open set* and the *closed set*. The open set contains nodes to be explored, while the closed set contains nodes that have already been explored. Here we show the pseudo-code:

A* Search Algorithm

```
Require: Start node $start$, Goal node $goal$
Ensure: Optimal path from $start$ to $goal$ (if exists)
1:  $openSet \leftarrow \{start\}$                          ▷ Priority queue for nodes to explore
2:  $g[start] \leftarrow 0$                                 ▷ Cost from start to current node
3:  $f[start] \leftarrow g[start] + h(start)$               ▷ Estimated total cost via heuristic $h$
4:  $cameFrom \leftarrow$ empty map                         ▷ Tracks the path
5:  while $openSet$ is not empty do
6:      $current \leftarrow$ node in $openSet$ with lowest $f$[node]
7:      if $current = goal$ then
8:          return ReconstructPath($cameFrom, current$)     ▷ Path found
9:      end if
10:     Remove $current$ from $openSet$
11:     for each $neighbor$ of $current$ do
12:         $tentativeGScore \leftarrow g[current] +$ cost($current, neighbor$)
13:         if $neighbor \notin g$ or $tentativeGScore < g[neighbor]$ then
14:             $cameFrom[neighbor] \leftarrow current$
15:             $g[neighbor] \leftarrow tentativeGScore$
16:             $f[neighbor] \leftarrow g[neighbor] + h(neighbor)$
17:             if $neighbor \notin openSet$ then
18:                 Add $neighbor$ to $openSet$
19:             end if
20:         end if
21:     end for
22: end while
23: return Failure                                          ▷ No path found
```

Here we can see, that there are four main processes in A*:Initialization, loop search, expansion of neighbor nodes, and termination condition. After the target node is expanded and the search is completed, the optimal path is obtained by backtracking from the target node through the parent node to the starting point. Here is the pseudo-code of backtracking:

ReconstructPath

```
Require: $cameFrom, current$
Ensure: List of nodes representing the optimal path
    $path \leftarrow$ empty list
```

```
2: while $current$ is in $cameFrom$ do
       Insert $current$ at the start of $path$
4:     $current \leftarrow cameFrom[current]$
   end while
6: Insert $current$ at the start of $path$
   return $path$
```

The A* algorithm possesses distinct advantages. It can ensure optimality. When the heuristic satisfies the criteria, A* can consistently identify the best path. Moreover, it may adjust to diverse application contexts using distinct heuristic mechanisms. In contrast to the Dijkstra algorithm, A* circumvents superfluous node growth. Fig 1 shows the flowchart of A*.[2]

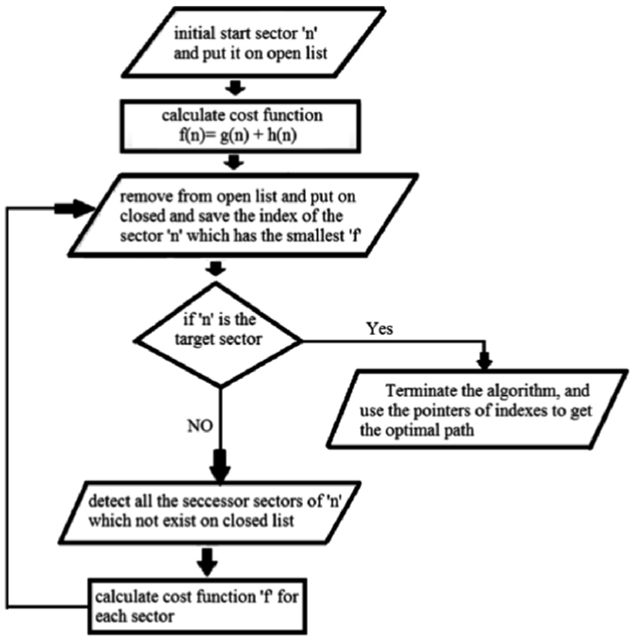
initial start sector 'n' and put it on open list
calculate cost function f(n)= g(n) + h(n)
remove from open list and put on closed and save the index of the sector 'n' which has the smallest 'f'
if 'n' is the target sector
Yes
Terminate the algorithm, and use the pointers of indexes to get the optimal path
NO
detect all the seccessor sectors of 'n' which not exist on closed list
calculate cost function 'f' for each sector

Figure 1: Flow chart of A* algorithm

2.2 ANA* algorithm

Nonetheless, A* is highly reliant on the heuristic function. An improper heuristic design may diminish search efficiency. It is inappropriate for dynamic situations.

The ANA*[3] algorithm was proposed to solve the problem that A* needs to expand a large number of nodes to ensure that the global optimal solution is found. ANA* can return suboptimal solutions in a limited time and gradually improve them. It initially uses a large suboptimality coefficient ϵ, allowing a "good enough" path to be found quickly. Over time, ϵ is gradually reduced and the solution is optimized, eventually converging to the optimal solution.

ANA* uses the evaluation function: $f(n) = g(n) + \epsilon \cdot h(n)$ to control the expansion of nodes. When ϵ is large, the heuristic weight ($\epsilon \cdot h(n)$) is higher, and nodes with lower heuristic values are expanded first. Each time a new solution is found, the current optimal path is updated, and ϵ is adjusted to guide the search for a better path.

The rigidity problem of A* finding the optimal solution in one go is solved through ANA*'s step-by-step optimization mechanism. Even if the environment changes, the solution can still be quickly updated and approach the optimal solution, making ANA* suitable for scenarios with changing obstacles or dynamic targets.

Through the following pseudo-code, we can better understand the algorithm flow of ANA*

```
ANA* Algorithm
Require: Start node $start$, Goal node $goal$, Initial $\epsilon > 1$
Ensure: Suboptimal path from $start$ to $goal$ (refined over time)
    $openSet \leftarrow \{start\}$                    ▷ Priority queue sorted by $f(x) = g(x) + \epsilon \cdot h(x)$
    $g[start] \leftarrow 0$                           ▷ Cost from start to current node
 3: $f[start] \leftarrow g[start] + \epsilon \cdot h(start)$
    $bestPathCost \leftarrow \infty$                  ▷ Tracks the cost of the best path found
    $cameFrom \leftarrow$ empty map                   ▷ Tracks the path
 6: while $openSet$ is not empty do
        $current \leftarrow$ node in $openSet$ with lowest $f$[node]
        if $g[current] + h(current) \geq bestPathCost$ then
 9:         break                                     ▷ No better path is possible
        end if
        Remove $current$ from $openSet$
12:     if $current = goal$ then
            $bestPathCost \leftarrow \min(bestPathCost, g[current])$
                                                      ▷ Keep searching for better paths
15:     end if
        for each $neighbor$ of $current$ do
            $tentativeGScore \leftarrow g[current] +$ cost$(current, neighbor)$
18:         if $neighbor \notin g$ or $tentativeGScore < g[neighbor]$ then
                $cameFrom[neighbor] \leftarrow current$
                $g[neighbor] \leftarrow tentativeGScore$
21:             $f[neighbor] \leftarrow g[neighbor] + \epsilon \cdot h(neighbor)$
                if $neighbor \notin openSet$ then
                    Add $neighbor$ to $openSet$
24:             end if
            end if
        end for
27:     $\epsilon \leftarrow \epsilon \cdot \delta$   ▷ Reduce $\epsilon$ dynamically (e.g., $\delta < 1$)
    end while
    if $bestPathCost < \infty$ then
30:     return ReconstructPath$(cameFrom, goal)$      ▷ Path found
    else
        return Failure                                ▷ No path found
33: end if
```

Also, we need to reconstruct the path:

```
ReconstructPath
Require: $cameFrom$, $current$
Ensure: List of nodes representing the optimal path
    $path \leftarrow$ empty list
    while $current$ is in $cameFrom$ do
        Insert $current$ at the start of $path$
 4:     $current \leftarrow cameFrom[current]$
    end while
    Insert $current$ at the start of $path$
    return $path$
```

2.3 Custom heuristic function

Although heuristic functions are essential in the ANA* and A* algorithms, traditional heuristic methods (such as Euclidean distance) only consider geometric distances and cannot fully represent the uncertainty of the robot state or the dynamic characteristics of the environment. This simplification may cause the algorithm to generate a large number of nodes to make up for the lack of heuristic information, thus affecting search efficiency and performance. In this paper, firstly, we introduced the rotation angle in the heuristic function to avoid the robot from taking meaningless turns. Secondly, we developed a customized heuristic function with integrated particle filtering, which improved the flexibility of the heuristic function for robot operation settings by merging the state estimation results of the perception and localization modules. Particle filtering can better provide a position reference for the robot by introducing uncertainty. This combination enables the heuristic function to dynamically represent the actual cost between the current state of the robot and the target point.

Our bespoke heuristic function integrates the efficiency of particle filtering, while also delivering enhanced accuracy and robustness for ANA* and A* by amalgamating the outputs of this filter with the heuristic function. This architecture may dynamically modify the projected path cost based on sensor observations and action models, thereby substantially decreasing invalid node expansion and enhancing search efficiency. The experimental section will evaluate the efficacy of conventional Euclidean heuristics and bespoke heuristics in ANA* and A* to assess the impact of the heuristic function that integrates particle filtering on enhancing algorithm efficiency and solution quality. This enhancement is highly significant for the real-time navigation of robots in intricate situations.

2.3.1 Heuristic Function with angle difference

The heuristic function estimates the cost from the current node to the target node within a search algorithm. The objective is to direct the search process and enhance the algorithm's efficiency. In the A* algorithm, the heuristic function h(n) and the actual cost g(n) collectively form the total cost function $f(n) = g(n) + h(n)$. The node with the minimal$f(n)$ is prioritized for exploration. A more precise heuristic function enhances the algorithm's efficiency.

The A* method frequently employs the Euclidean distance as a heuristic function, particularly effective for shortest path searches in continuous space. In the context of mobile robot challenges, the heuristic function must account for both the Euclidean distance in the two-dimensional plane and the angular difference $\Delta\theta$, hence enhancing its applicability to directional path planning issues.

Consequently, we provide the initial enhanced heuristic algorithm, which incorporates the minimum square of the angle difference to signify the directional cost. The equation is expressed as:

$$\Delta\theta = |\text{wrap\_to\_pi}(\text{node}[2] - \text{goal}[2])|$$

$$h(n) = \sqrt{(\text{node}[0] - \text{goal}[0])^2 + (\text{node}[1] - \text{goal}[1])^2 + \min(\Delta\theta, 2\pi - \Delta\theta)^2}$$

The heuristic function incorporates the angle difference alongside the conventional Euclidean distance, enhancing its applicability for circumstances necessitating directional planning. This function is admissible, meaning the cost predicted by the formula will not surpass the actual cost, as the positional distance and directional disparity represent the minimum costs that must be navigated throughout the actual movement procedure. The angle difference $\Delta\theta$ employs the minimum value $min(\Delta\theta, 2\pi - \Delta\theta)$ to guarantee that it is the smallest angle necessary for effective rotation.

Furthermore, consistency is also upheld. For any two contiguous nodes n1 and n2, the equation holds: $h(n_1) \leq c(n_1, n_2) + h(n_2)$, where $c(n_1, n_2)$ denotes the actual cost from n_1 to n_2. This condition guarantees that the A* algorithm will not backtrack throughout the search process, hence enhancing efficiency. Utilizing solely Euclidean distance as the heuristic function in path planning may neglect directional restrictions, resulting in significant directional adjustments (elevated turning costs) for the robot when nearing the destination point, and perhaps leading to erroneous planning due to directional inaccuracies. The method addresses this issue by incorporating directional costs, enabling the projected route to achieve distance optimization while minimizing supplementary directional adjustments.

Introducing the variation in direction estimation may lead to an exaggerated significance of the angle difference, perhaps resulting in the heuristic function underestimating accuracy and hence increasing the number of investigated nodes.

2.3.2 Heuristic Function with Particle Filter

The enhanced Euclidean heuristic function referenced above considers both positional and directional costs, enabling the robot to plan not just towards the target location but also to orient itself in the target direction, so offering a more accurate estimate for real motion planning. Nonetheless, it presupposes that the geometric distance and angular disparity between nodes remain constant, neglecting the noise and uncertainty inherent in the robot's actual motion, including sensor inaccuracies and deviations in the dynamic model. Moreover, in intricate cases, such as those with several obstructions between pathways, reliance on pure geometry and angular heuristics may fail to effectively direct the search, leading to diminished algorithmic efficiency.

In order to solve the above limitations, an improved heuristic function based on particle filtering is designed. The core is to use the state distribution of the particle filter to dynamically adjust the heuristic value. Particle filtering has unique advantages in dealing with uncertainty and dynamic systems. The core idea is to describe the state distribution through a set of weighted particles, which dynamically reflect the uncertainty of the system. The new algorithm dynamically adjusts the distribution of particles according to the cost of position and direction, reflecting the system's dynamic understanding of the path cost. The heuristic function guided by the particle filter can focus on high-confidence areas faster during the search process and reduce the computational cost of expanding invalid nodes.

First, let's introduce the algorithm flow of particle filtering. Below we show the pseudo-code:

Particle Filter Algorithm

```
Require: Initial pose $initial\_pose$, Number of particles $N$, Motion noise $\sigma_{motion}$, Sensor noise $\sigma_{sensor}$
Ensure: Estimated pose $\hat{pose}$
 1: Initialize $N$ particles with $initial\_pose$ and weights $w = \frac{1}{N}$
 2: $\hat{pose} \leftarrow initial\_pose$
 3: $previous\_odom \leftarrow initial\_pose$
 4: while robot is navigating do
 5:     $odom\_pose \leftarrow \text{OdometryModel}(control\_input, previous\_odom)$
 6:     Action Model: Update particle states:
 7:     for each particle $p$ do
 8:         Compute motion deltas $\Delta x, \Delta y, \Delta\theta$
 9:         Add noise to $\Delta x, \Delta y, \Delta\theta$ and update $p.x, p.y, p.\theta$
10:     end for
11:     $previous\_odom \leftarrow odom\_pose$
12:     $sensor\_reading \leftarrow \text{SensorModel}(robot)$
13:     Update Weights:
14:     for each particle $p$ do
15:         $p.weight \leftarrow \text{MultivariateGaussian}(p, sensor\_reading, \sigma_{sensor})$
16:     end for
17:     Normalize particle weights
18:     Resampling: Generate new particles based on weights
19:     Pose Estimation: Compute $\hat{pose}$ as weighted average:
20:     $\hat{x} \leftarrow \sum_{i=1}^{N} w_i \cdot p_i.x$
21:     $\hat{y} \leftarrow \sum_{i=1}^{N} w_i \cdot p_i.y$
22:     $\hat{\theta} \leftarrow \arctan 2\left(\sum_{i=1}^{N} w_i \cdot \sin(p_i.\theta), \sum_{i=1}^{N} w_i \cdot \cos(p_i.\theta)\right)$
23:     $\hat{pose} \leftarrow (\hat{x}, \hat{y}, \hat{\theta})$
24:     Visualization: Update particle and path plots
25: end while
26: return $\hat{pose}$
```

The key functions of the particle filter algorithm are the action model, low variance sampling, and pose estimation, the algorithm logic of these functions is shown below:

Action Model

```
Require: Odometry pose $odom\_pose$, Previous pose $previous\_odom$, Motion noise $\sigma_{motion}$
```

```
Ensure: Updated particle states
 1: Compute motion deltas: $\Delta x, \Delta y, \Delta\theta$
 2: for each particle $p$ do
 3:     Sample noise for translation and rotation
 4:     Update $p.x \leftarrow p.x + \Delta x + noise$
 5:     Update $p.y \leftarrow p.y + \Delta y + noise$
 6:     Update $p.\theta \leftarrow p.\theta + \Delta\theta + noise$
 7: end for
```

Low Variance Resampling

```
Require: Particles $P$, Number of particles $N$
Ensure: Resampled particles
 1: Compute cumulative weights
 2: $r \leftarrow \text{Uniform}(0, \frac{1}{N})$
 3: $index \leftarrow 0$
 4: for $i = 1$ to $N$ do
 5:     $U \leftarrow r + (i-1) \cdot \frac{1}{N}$
 6:     while $U > cumulative\_weights[index]$ do
 7:         $index \leftarrow index + 1$
 8:     end while
 9:     Copy particle $P[index]$ into new set
10: end for
```

Pose Estimation

```
Require: Particles $P$, Weights $W$
Ensure: Estimated pose $\hat{pose}$
 1: Compute weighted averages: $\hat{x} \leftarrow \sum_{i=1}^{N} w_i \cdot p_i.x, \quad \hat{y} \leftarrow \sum_{i=1}^{N} w_i \cdot p_i.y$
        $\hat{\theta} \leftarrow \arctan 2\left(\sum_{i=1}^{N} w_i \cdot \sin(p_i.\theta), \sum_{i=1}^{N} w_i \cdot \cos(p_i.\theta)\right)$
 2: $\hat{pose} \leftarrow (\hat{x}, \hat{y}, \hat{\theta})$
```

Based on the above particle filter function, we combine it with the heuristic function and follow the steps below:

1. The particle filter class PF initializes a collection of particles, each particle represents the possible position and orientation of the robot. Each particle is assigned an initial weight (uniformly distributed), and the initial position is the initial state of the robot.
2. Update the position and posture of each particle based on the odometer data, and add random noise to simulate the uncertainty of motion.
3. Use sensor data to calculate the weight of each particle, and the weight represents the degree of match between the particle state and the sensor observation.
4. Resample according to the particle weight, retain high-weight particles, discard low-weight particles, and avoid degradation.
5. Calculate the current estimated position and direction of the robot through particle-weighted averaging.
6. Loop steps 2-5.

We will now examine two critical components of the method individually: the calculation of linear velocity v and angular velocity ω. The particle filter's motion model necessitates these two input control variables to compute the robot's posture alteration at each instant. Secondly, we must elucidate how the particle filter characterizes the uncertainty of the robot's present location via dynamic distribution, consequently enhancing the distance estimate of the heuristic function.

Compute Control Input : The control inputs v (linear velocity) and ω (angular velocity) are calculated as follows:

First, we get the initial position of the robot $[x_r, y_r, \theta_r]$ where θ_r is the current orientation angle of the robot and the coordinates of the target point $[x_g, y_g]$

Second, we calculate the distance error, the Euclidean distance from the robot's current position to the target position, and the angle error, the angle deviation of the target point relative to the robot's current orientation. distance_to_target $= \sqrt{(x_g - x_r)^2 + (y_g - y_r)^2}$ and, angle_to_target $= \arctan 2(y_g - y_r, x_g - x_r) - \theta_r$. In the second equation, the angle is wrapped to $[-\pi, \pi]$ to ensure uniqueness.

Third, we use some gain coefficients k_1 and k_2 to control the gain coefficients of speed and angular velocity.$v = k_1 \cdot$ distance_to_target,and $\omega = k_2 \cdot$ angle_to_target

These equations define a simple proportional controller that moves the robot toward the target while adjusting its orientation.

In summary, we have the following:

```
Require: Current pose of the robot $[x_r, y_r, \theta_r]$, Target position $[x_g, y_g]$, Gains $k_1$, $k_2$
Ensure: Linear velocity $v$, Angular velocity $w$
  Input: $[x_r, y_r, \theta_r]$, $[x_g, y_g]$, $k_1$, $k_2$
  Output: $v$, $w$
  Compute distance to target: $d \leftarrow \sqrt{(x_g - x_r)^2 + (y_g - y_r)^2}$
  Compute angle to target: $\alpha \leftarrow \arctan 2(y_g - y_r, x_g - x_r) - \theta_r$
  Wrap $\alpha$ to $[-\pi, \pi]$: $\alpha \leftarrow (\alpha + \pi) \bmod (2\pi) - \pi$
  Compute linear velocity: $v \leftarrow k_1 \cdot d$
  Compute angular velocity: $w \leftarrow k_2 \cdot \alpha$
  return $v$, $w$
```

Particle filter combined with heuristic function: The particle filter articulates the uncertainty of the robot's present location via dynamic distribution, enhances the distance estimation of the heuristic function, mitigates noise interference by employing the weighted average of position and direction, and subsequently incorporates the statistical attributes of the particle set (such as variance) to modify the heuristic cost.

Algorithm Explanation

First, we define the input parameters and initialization

- **Inputs:**
  - `node`: Current node (robot's state as $[x, y, \theta]$).
  - `goal`: Target node (goal state as $[x, y, \theta]$).
  - `particles`: Number of particles used for filtering (default: 100).
  - `robots`, `base_joints`: Robot and joint information for sensor data simulation.
  - `odometry_fn`, `sensor_fn`: Functions to simulate odometry and sensor models.
- **Initialization:**
  - The particle states are initialized as a copy of the current node, repeated `particles` times.
  - Control input (`controls`) is computed based on the current node and goal.

Next, we need to update the Motion and Sensor data. For each particle, we take the following steps:

1. Apply the motion model (`odometry_fn`) to compute the new pose, adding motion noise.
2. Simulate sensor readings (`sensor_fn`) based on the noisy pose, introducing measurement noise.

Now we taking about the cost estimation. For each noisy particle:
1. Compute the Euclidean distance to the goal in the x-y plane: $\text{Position Cost} = \sqrt{(x_{\text{goal}} - x_{\text{particle}})^2 + (y_{\text{goal}} - y_{\text{particle}})^2}$
2. Compute the orientation cost as the absolute angular difference: $\text{Orientation Cost} = |\theta_{\text{goal}} - \theta_{\text{particle}}|$
3. Combine the position and orientation costs: $\text{Cost} = \text{Position Cost} + \text{Orientation Cost}$

The heuristic value is the mean cost over all particles:$h_{\text{PF}}(n) = \frac{1}{N}\sum_{i=1}^{N}\text{Cost}(i)$, The average expense of all particles, representing the comprehensive projected cost from the current node n to the target throughout the complete particle ensemble. We utilize particles to denote uncertainty and model the probability distribution of the robot's present state. Each particle signifies a potential posture, and the average of the complete particle set amalgamates the prices over all conceivable positions. By averaging, we mitigate the fluctuations in the costs of individual particles that may arise from excessive or insufficient sensor noise or action noise, hence enhancing the stability of the heuristic value.

Combining particle filtering with heuristic functions provides significant advantages by enabling the heuristic to adapt to real-world uncertainties through the incorporation of noise from motion and sensor models. This approach improves cost estimation by considering both position and orientation costs, making it more accurate than traditional Euclidean-based heuristics. Additionally, by simulating noisy particles, the heuristic reflects the robot's true navigation cost, avoiding overly optimistic estimates and reducing path planning errors. The averaging of particle costs further enhances robustness, ensuring that individual particle anomalies caused by high noise do not compromise the overall

3 Results

3.1 Experimental Setup

3.1.1 Virtual Environment

The entire algorithm is implemented on Ubuntu 24.04, Python version is 3.12. PyBullet is an open-source framework for simulating physics, controlling robotics, and conducting research in reinforcement learning. It offers an effective physics engine that facilitates rigid body dynamics, collision detection, flexible body simulation, and constraint system modeling. PyBullet is very adept at robotic simulation tasks, capable of loading prevalent robot models (including URDF and SDF formats), executing motion control, path planning, and sensor simulation. The versatile API and visualization features enable users to rapidly construct intricate simulation environments and effortlessly interface with deep learning frameworks and reinforcement learning algorithms, offering an effective testing platform for the development and validation of robotic algorithms.In this experiment, we use pybullet to create a robot environment and verify the algorithm.

3.1.2 Algorithm testing environment

The experimental environment "env1"(see Fig 2) consists of a rectangular ground and wall, five symmetrically distributed tables as obstacles, and a PR2 robot. The ground and walls form a rectangular area of 8.2×4.2, which is used to limit the robot's range of motion. The five tables are distributed in the center and four positions of the environment, one of which is located at $[1.0, 0.0, 0.74]$ (central table), two are located at the top of the environment ($[-1.2, 0.6, 0.74]$ and $[1.0, 0.7, 0.74]$), and the other two are located at the bottom ($[-1.0, -0.6, 0.74]$ and $[1.0, -0.8, 0.74]$). Each table consists of a rectangular tabletop and four cylindrical legs.

The PR2 robot is initially positioned at $[x = -3.4, y = -1.4, z = 0.05]$ in the lower left corner of the environment, facing the positive x-axis direction, and the initial posture is set to the operation-ready state through the joint angles. The robot's goal is to move from the starting point to the specified end position in the environment while avoiding obstacles such as tables and walls.

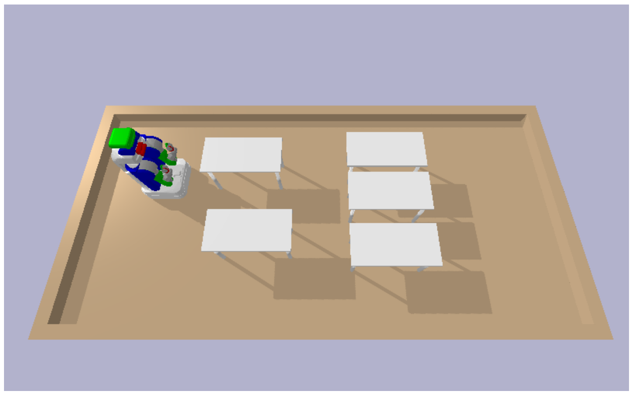

Figure 2: The experimental environment 1

The experimental environment "env2"(see Fig 3) consists of a rectangular floor and walls, two intermediate walls, six tables, and a PR2 robot. The floor and walls form an enclosed area of approximately 8.2×4.2, which is used to limit the robot's range of motion. In the environment, two intermediate walls are located at $[x = 0.8, y = -0.565, z = 1.0]$ and $[x = 0.8, y = 1.65, z = 1.0]$, with dimensions of $[0.1, 1.235, 1.0]$ and $[0.1, 0.15, 1.0]$, respectively, and are used as obstacles to separate areas.

Six tables are placed in different locations: `ikeatable1` is located at $[x = -2.3, y = -0.3, z = 0.74]$, `ikeatable2` is located at $[x = -2.3, y = 0.3, z = 0.74]$, `ikeatable3` is located at $[x = -1.1, y = -0.3, z = 0.74]$, `ikeatable4` is located at $[x = -1.1, y = 0.3, z = 0.74]$, `ikeatable5` is located at $[x = 3.5, y = -1.2, z = 0.74]$, and `ikeatable6` is located at $[x = 3.5, y = 1.2, z = 0.74]$. Each table consists of a rectangular tabletop and four cylindrical legs. The size of the tabletop is $[0.3, 0.6, 0.015]$, and the height of the legs is 0.71.

The robot in the experiment is PR2, whose initial position is $[x = -3.4, y = -1.4, z = 0.05]$, facing the positive x-axis. The task goal is to start from the starting point, cross complex obstacles, and navigate to the specified end position.

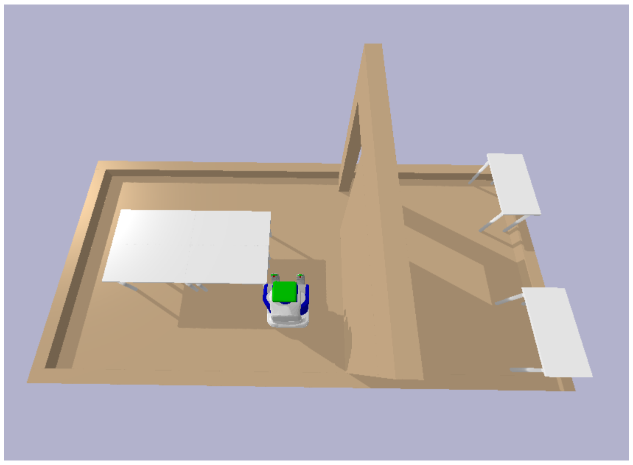

Figure 3: The experimental environment 2

3.2 Experimental Result

The Fig4 and Fig5 show the trajectories chosen by different algorithms in different paths.The algorithms used by the trackball on the path image are from bottom to top: A_EU, ANA_EU, A_EU_MOD, ANA_EU_MOD, A_PF,and ANA_PF.

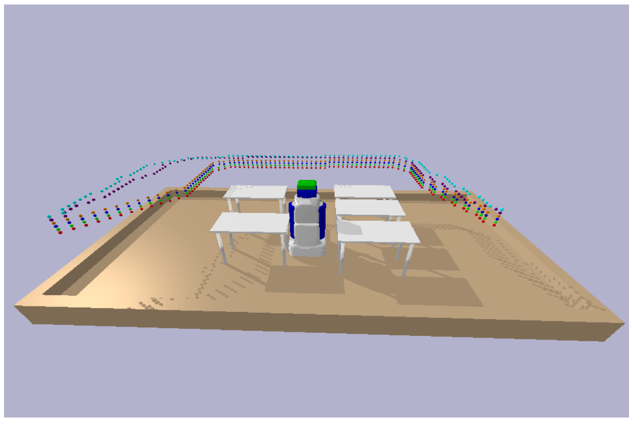
Figure 4: Path For Env1

Table 1: Performance of Different Algorithms in Env 1

Algorithm	Cost	First Solution Time (s)	Total Planning Time (s)
`A_EU`	9.7355	5.0637	5.0637
`ANA_EU`	9.7355	5.0700	15.8895
`A_EU_MOD`	11.2298	4.6546	4.6546
`ANA_EU_MOD`	11.2298	4.6800	19.8557
`A_PF`	46.8444	95.1120	95.1120
`ANA_PF`	48.3434	100.5000	209.6651

From the table, the A* algorithms require just the identification of the optimal solution to conclude, resulting in the initial solution time being equivalent to the entire planning time. Conversely, the ANA* algorithms persist beyond the discovery of a suboptimal solution. Optimization typically results in the entire planning duration being significantly greater than the initial solution time. The formulation of the heuristic function profoundly influences the algorithm's efficacy. The table illustrates the trade-off between cost and efficiency of various heuristics. The particle filter heuristic (PF) requires more time but demonstrates superior adaptability to dynamic situations and uncertainty.

The Euclidean heuristic (EU) yields the minimal path cost of 9.7355, suggesting its superior applicability for geometrically optimal path design contexts. The modified Euclidean heuristic (EU_MOD) considerably reduces the initial solution time with a marginal increase in cost. The initial solution time of A__MOD is 4.6546 seconds, the quickest among all approaches, indicating its superior efficacy in accurately calculating genuine costs. Despite the particle filter heuristic (PF) exhibiting suboptimal performance regarding cost and initial solution time, its outcomes may demonstrate greater robustness in intricate scenarios, taking into account the attributes of sensor noise and dynamic surroundings. Furthermore, in contrast to A*, ANA* offers a more adaptable path planning solution via incremental optimization; nonetheless, this considerably extends the computation time. Particularly when integrated with the particle filter heuristic, the overall planning duration of (ANA_PF) reaches 209.6651 seconds.

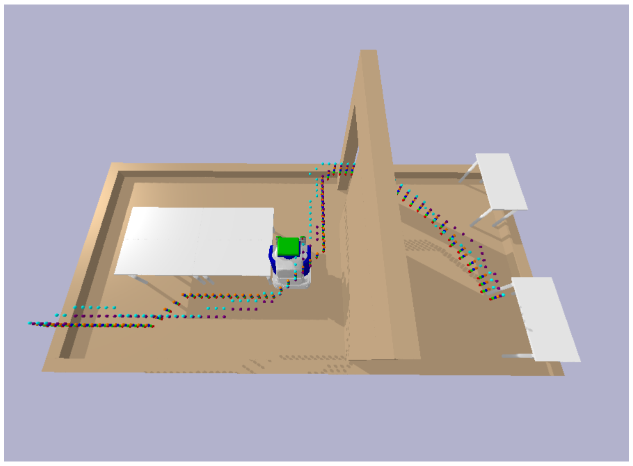

Figure 5: Path For Env2

Table 2: Performance of Different Algorithms with a Time Limit of 450.0s

Algorithm	**Cost**	**First Solution Time (s)**	**Total Planning Time (s)**
`A_EU`	8.9598	4.4822	4.4822
`ANA_EU`	8.9598	4.4300	32.8281
`A_EU_MOD`	10.3955	4.4645	4.4645
`ANA_EU_MOD`	10.3955	4.6000	37.9219
`A_PF`	41.6165	97.5260	97.5260
`ANA_PF`	42.0745	100.1100	329.5866

This table compares the performance of different algorithms and heuristic functions from the three dimensions of path planning cost (Cost), first solution time (First Solution Time) and total planning time (Total Planning Time). The cost is used to measure the quality of the path, the first solution time reflects the efficiency of the algorithm in finding the first feasible path, and the total planning time reflects the optimization and calculation overhead of the algorithm.

A* algorithms (such as A_EU, A_EU_MOD, A_PF) will terminate after finding the optimal solution for the first time, so the first solution time is equal to the total planning time. In contrast, ANA* algorithms (such as ANA_EU, ANA_EU_MOD, ANA_PF) will terminate after finding the suboptimal solution and continue optimizing the path, leading to significantly higher total planning times than the first solution times.

Different designs of heuristic functions affect the performance of the algorithm. The Euclidean heuristic (EU) primarily focuses on the geometric optimal path. The modified Euclidean heuristic (EU_MOD) incorporates adaptations to the actual environment. Finally, the particle filter heuristic (PF) emphasizes handling the uncertainty of dynamic environments.

From the results, it can be observed that the A_EU and ANA_EU algorithms, which use the Euclidean heuristic (EU), perform best in terms of path cost (both achieving 8.9598). Their first solution times are 4.4822 seconds and 4.4300 seconds, respectively, demonstrating their ability to quickly find the geometric minimum optimal path. This performance is due to the Euclidean heuristic's direct targeting of geometric distances, which ignores the impact of environmental dynamics.

The modified Euclidean heuristic (EU_MOD) maintains a similarly short first solution time (A_EU_MOD and ANA_EU_MOD are 4.4645 seconds and 4.6000 seconds, respectively) but incurs slightly higher costs (both 10.3955). This indicates that the heuristic offers advantages in estimating costs more realistically for practical scenarios.

In contrast, the particle filter heuristic (PF) performs poorly in both cost and time metrics. A_PF and ANA_PF extend the first solution time to 97.5260 seconds and 100.1100 seconds, respectively, with a total planning time reaching 329.5866 seconds (ANA_PF). This poor performance stems from the particle filter's need to consider sensor noise and environmental dynamics. Although the costs (41.6165 and 42.0745) are higher, this heuristic provides a more conservative and robust path-planning solution.

Additionally, compared to A*, ANA* improves flexibility by gradually optimizing the path. However, this flexibility comes at the cost of increased computational overhead, particularly under complex heuristics like ANA_PF, where the optimization process imposes significant time costs.

3.3 Discuss

According to the above research, various contexts are appropriate for distinct mixes of algorithms and heuristics. In static or low-noise conditions, A_EU or ANA_EU are favored, enabling planning to be executed with little expense and in a reduced timeframe. In dynamic situations or conditions of significant uncertainty, the particle filter heuristic (A_PF or ANA_PF) can offer a more secure and dependable trajectory, albeit at a greater computing expense. For time-sensitive tasks with moderate path cost constraints, the modified Euclidean heuristic (EU_MOD) in conjunction with A* or ANA* is an effective option.

Furthermore, we analyze the performance of the path cost over time, mainly through the Raw Cost, Smoothed Cost, and the best solution (green mark) finally found to analyze the algorithm performance.

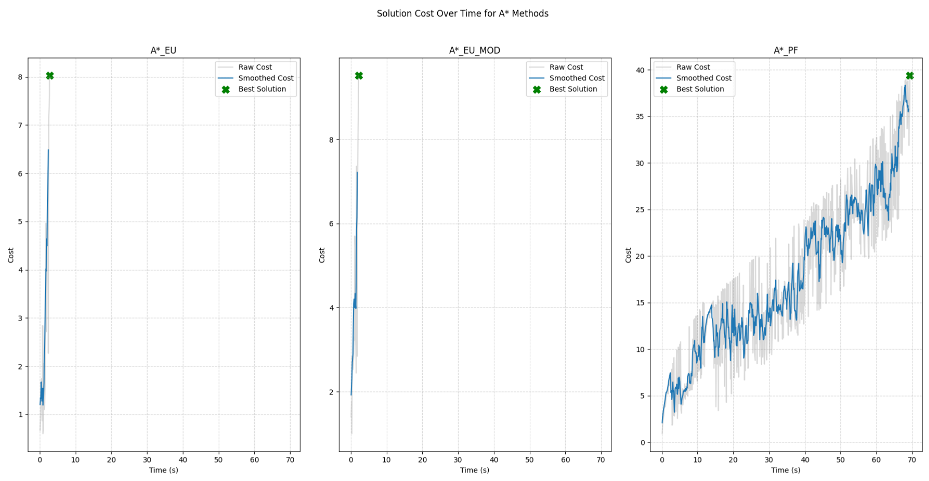

Figure 6: A star For env1

From Fig6, the Euclidean heuristic (EU) is directly based on geometric distance and can quickly find the shortest path, so it is stable. The modified Euclidean heuristic (EU_MOD) modifies the cost estimate moderately, and although it slightly increases the path cost, it can still find the solution in a short time.

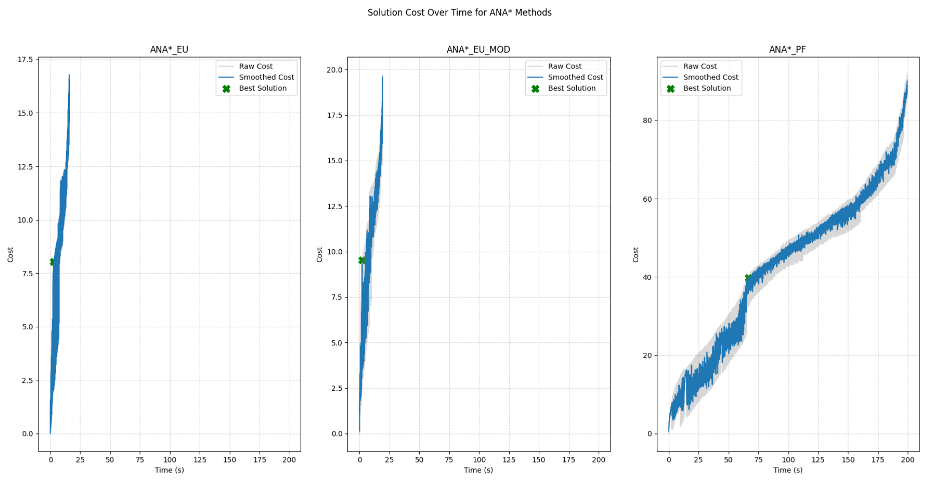

Figure 7: ANA star For env1

From Fig7, the particle filter heuristic (PF) incorporates sensor noise and environmental uncertainty into the estimate, resulting in a more conservative path cost that prioritizes safety over geometric optimality. Due to the introduction of noise, the heuristic estimate is not accurate enough during the search process, causing the cost to slowly increase over time.

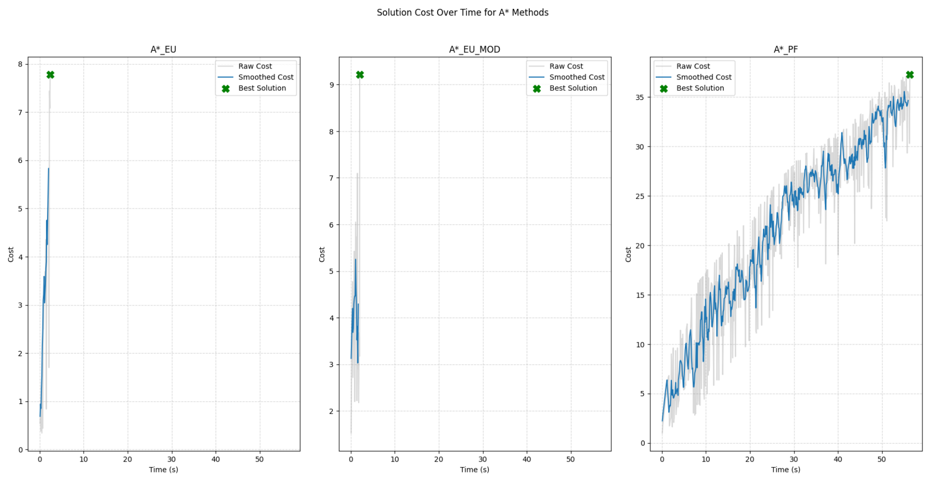

Figure 8: A star For env2

From Fig8, ANA*'s step-by-step optimization mechanism enables the algorithm to continue to improve the path quality after it has found a solution for the first time. Although the initial cost is high, it gradually converges to the optimal solution over time. Therefore, over a long period of time, ANA* can find path solutions that are comparable to or even better than A*.

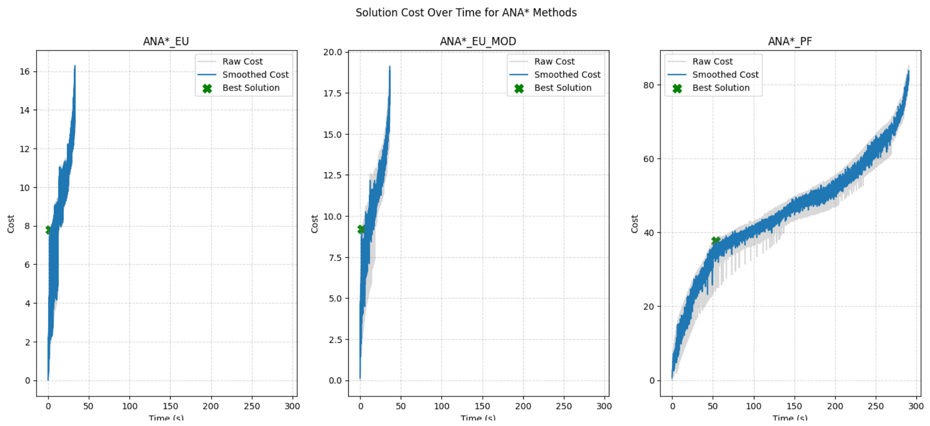

Figure 9: ANA star For env2

From Fig9, the particle filter heuristic introduces dynamic noise and environmental uncertainty, which makes the heuristic estimation less accurate and the path quality lower. In addition, the step-by-step optimization mechanism of ANA* takes a long time under the PF heuristic, mainly because the search space is disturbed by noise, which makes it difficult to quickly converge the path while continuously optimizing.

In both experiments, A_EU and ANA_EU of the Euclid heuristic (EU) always showed the lowest path cost, for example, the cost in the second experiment was 8.9598, and the first solution time was short (all less than 5 seconds). This is because the Euclid heuristic takes the geometric shortest path as the goal, which can provide clear direction guidance for the search algorithm. However, the idealized assumption of EU ignores the dynamic changes or noise effects in the actual environment, so it is more suitable for static or low-uncertainty scenarios. The modified Euclid heuristic (EU_MOD) has a slightly higher cost (such as 10.3955 in the second experiment), but still maintains a high computational efficiency. The first solution time is similar to that of EU, indicating that it is closer to the real cost by adjusting the estimate, and is suitable for tasks that need to balance efficiency and certain environmental changes. The particle filter heuristic (PF) shows the highest path cost and the longest solution time, but this is mainly because it takes into account sensor noise and environmental dynamic changes, making it more robust and secure in complex and high-uncertainty scenarios.

From an algorithmic perspective, the core difference between A* and ANA* is whether or not to perform step-by-step optimization. A* aims at high efficiency and stops immediately after finding the optimal solution for the first time, so its first solution time is the same as the total planning time. ANA* allows the path to continue to be optimized after the suboptimal solution is first found, so that it can adapt to the dynamic environment and eventually converge to the global optimal solution. For example, in combination with the PF heuristic, the total planning time of ANA_PF is 329.5866 seconds, which is much higher than _PF's 97.5260 seconds, and the cost is slightly higher, but its step-by-step optimization process makes it more suitable for dynamic environments or scenarios with high security requirements. Although the step-by-step optimization of ANA* brings greater computational overhead, it provides important guarantees in terms of robustness and path security.

Combining the results of the two experiments, the role of the heuristic algorithm is reflected in providing guidance for estimating costs for path planning. EU provides direct guidance of geometric optimization, but lacks consideration of dynamic changes in the environment; EU_MOD achieves a better balance between cost and efficiency; PF provides a safer but more costly path solution by introducing dynamic uncertainty estimates of sensors and the environment. A* performs well in efficiency and is suitable for tasks with high real-time requirements, while ANA* has stronger adaptability in scenarios with large environmental dynamics and high requirements for path safety. Overall, the combination of different algorithms and heuristics needs to find the best balance between efficiency and safety according to task requirements.

4 Conclusion

The experimental findings confirmed the substantial influence of several heuristic functions on the path planning algorithm. A_EU and ANA_EU of the Euclidean heuristic (EU) attain the minimal path cost, exhibit brief initial solution times, and demonstrate great computing efficiency, making them appropriate for static or low-noise situations; the modified Euclidean heuristic (EU_MOD) enhances these attributes further. The assessment of the actual cost preserves a brief initial solution time despite a minor rise in cost, illustrating a commendable equilibrium between efficiency and ecological adaptation. Despite the extended initial solution time and overall planning duration of the particle filter heuristic (PF) A_PF and ANA_PF, it delivers a more resilient solution by dynamically estimating environmental uncertainty and sensor noise, making it appropriate for intricate dynamics. scene.

The A* algorithm seeks to efficiently identify the ideal answer from an algorithmic standpoint. The initial solution time coincides with the overall planning duration and is appropriate for time-critical jobs; conversely, ANA* enhances flexibility and resilience through incremental path optimization, although The computational duration escalates substantially, particularly with intricate algorithms. In conclusion, the choice of various algorithms and heuristic functions must strike a balance among path cost, planning efficiency, and environmental adaptability based on task requirements. The findings of this research serve as a significant reference for the optimal selection of path planning algorithms across various application contexts.

References

[1] Hart, Peter E and Nilsson, Nils J and Raphael, Bertram. A formal basis for the heuristic determination of minimum cost paths In *IEEE transactions on Systems Science and Cybernetics*, pages 100–107. IEEE, 1968.

[2] Xuan, Doan Thanh and Nam, Le Giang and Viet, Dang Thai and Thang, Vu Toan. A-star algorithm for robot path planning based on digital twin. In *Regional Conference in Mechanical Manufacturing Engineering*, pages 83–90. Springer, 2021.

[3] Jur P. van den Berg and Rajat Shah and Arthur Huang and Ken Goldberg Anytime Nonparametric A In *AAAI Conference on Artificial Intelligence*, 2011.