Robot Motion Planning by Non-prehensile Obstruction Handling in Cluttered Environments

Pratik Yuvraj Yawalkar
ai23mtech11006@iith.ac.in
Indian Institute of Technology, Hyderabad
Hyderabad, Telangana, India

Rekha Raja
rekha.raja@ai.iith.ac.in
Indian Institute of Technology, Hyderabad
Hyderabad, Telangana, India
Radboud University
Nijmegen, Gelderland, Netherlands

Abstract

Navigating cluttered environments presents a significant challenge for robots, due to collisions, computational complexity, and varying obstacles shapes and sizes. Traditional path-planning approaches typically assume that obstacles are static, limiting their effectiveness. In this paper, we introduce a novel heuristic path planning technique that enhances robot to actively interact with its environment. This method enables efficient path clearing movable obstacles by non-prehensile manipulation, while avoiding immovable ones and reducing computational cost. The proposed method utilizes a heuristic function that optimally balances two key objectives: (1) minimizing the robot footprint in the search space to efficiently navigate around obstacles and (2) employing a local motion planner to intelligently reposition movable obstacles. This dual optimization minimizes both the travel distance and the exerted energy required to manipulate obstacles. The result is a feasible, optimized path that traditional algorithms, such as A* and RRT, cannot achieve in similar settings. In particular, even in scenarios where traditional algorithms produce feasible paths, our approach demonstrates lower path costs, which significantly improves robotic navigation in densely cluttered environments.

CCS Concepts

• **Computing methodologies** → **Planning for deterministic actions**; **Robotic planning**; *Motion path planning*.

Keywords

Heuristic Path Planning, Dynamic Obstacle Manipulation, Cluttered Environment Navigation, Movable Obstacle Interaction.

ACM Reference Format:
Pratik Yuvraj Yawalkar and Rekha Raja. 2025. Robot Motion Planning by Non-prehensile Obstruction Handling in Cluttered Environments. In *Proceedings of Advances in Robotics (AIR).* ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

AIR, IIT Jodhpur, India

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Robots operating in cluttered, dynamic environments are critical to a wide range of applications such as rescue mission, warehousing, agriculture, construction, healthcare, etc. In these domains, robots often need to navigate through crowded spaces filled with obstacles of varying shapes, sizes, where some are movable and some are not. For instance, in healthcare settings, service robots may need to maneuver through hallways to patient rooms, which often contain movable furniture and equipment. Successfully navigating such environments requires a level of adaptability and strategic interaction that conventional path-planning methods struggle to achieve. Traditional path-planning algorithms such as A* and Rapidly-exploring Random Tree (RRT) treat obstacles as static objects, thereby limiting their ability to produce feasible paths in environments where objects may be movable. This static treatment of obstacles not only restricts the robot's potential pathways but also leads to inefficient routes, increased computational complexity, and, ultimately, the risk of failure in highly congested spaces. The challenges emphasize the need for an innovative approach to path planning that enables the robot to actively alter its environment.

Path Planning is a non-deterministic polynomial-time, NP-Hard problem [2], used to solve issues across fields from basic route planning to advanced action sequence selection for varied environments. Our method departs from traditional approaches by allowing the robot to dynamically manipulate movable obstacles, pushing or pulling them aside to create a navigable pathway. This approach not only expands the robot's feasible options for navigation but also reduces computational load by minimizing the required search space in dense environments. The core contribution of this work is a heuristic function that balances two primary objectives. First, it minimizes the robot's footprint within the search space, enabling it to navigate around obstacles with greater precision. Second, it integrates a local move planner that intelligently selects and relocates movable obstacles based on their mass, ensuring efficient use of energy when clearing the path. This dual optimization significantly reduces both travel distance and exerted energy, enhancing the robot's mobility and adaptability in highly constrained spaces.

2 Background

Efficient navigation in cluttered environments poses a significant challenge in mobile robotics, particularly when both stationary and movable obstacles are present. Traditional path-planning algorithms, such as A* [5] and Rapidly-exploring Random Tree (RRT) [8], are widely used in static environments because of their effectiveness in finding collision-free paths. However, these methods

generally treat obstacles as fixed objects, limiting their effectiveness in real-world, semi-structured environments, where movable obstacles could be leveraged to create more efficient paths. This static assumption leads to limited path feasibility and adaptability in complex, cluttered spaces, where obstacles could potentially be relocated to create new pathways [14]. Dynamic obstacle handling has been explored to improve real-time adaptability in environments where obstacles can appear or move unpredictably. Probabilistic roadmaps [7] and dynamic variants of RRT, such as Anytime RRT [4], enable on-the-fly path updates when new obstacles are detected. These approaches generally view obstacles as either stationary or unpredictably moving, rather than as objects that could be manipulated to enhance navigation. Approaches like D* and its variants [11–13] incrementally update paths when encountering obstacles, yet still assume that obstacles are non-manipulable.

To overcome traditional limitations in navigation, recent work has explored interactive navigation, where robots not only avoid but also manipulate obstacles to clear a path. RL-based methods like HRL4IN [10] enable robots to learn adaptive manipulation policies for cluttered environments. Xia et al. [15] extended this by integrating motion generation into RL, allowing dynamic interaction with the environment. Cheong et al. [3] used deep Q-networks to select objects for sequential rearrangement, while Haustein et al. [6] proposed a kinodynamic planner for smooth transitions between stable states during rearrangement. These approaches improve path feasibility through object manipulation but often require precise control and are computationally intensive, limiting scalability. Heuristic-based methods address this by improving computational efficiency, for instance, by embedding heuristic functions into A* and RRT to guide the search toward more promising paths [1]. In NAMO-like methods, heuristics that prioritize lighter or more easily movable obstacles, as presented by Levihn et al. [9], can further reduce computational demands. However, these methods typically assume predefined object properties, such as weight or mobility, which may not always be available or accurate in real-world applications.

Building upon these advancements, our approach introduces a novel heuristic-based path-planning method that enables robots to navigate cluttered spaces through dynamic interaction with movable obstacles. Unlike traditional static planning algorithms, our method leverages heuristics to optimize both the robot's footprint and the relocation of obstacles, considering object properties like mass to minimize energy expenditure. By combining path efficiency with active manipulation, our method offers a robust solution for navigating highly constrained environments, showing improved path feasibility and computational efficiency over traditional and state-of-the-art approaches.

3 Problem Formulation

We consider a pathfinding problem in a three-dimensional (3D) grid environment, denoted by $\mathcal{G} = (V, E)$, where V is the set of cells (or voxels) in the grid, and E represents the edges connecting adjacent cells. Each cell in the grid is indexed by coordinates (i, j, k), where $i, j, k \in \mathbb{Z}^+$ represent discrete positions along the three dimensions of the grid. The environment allows for robot navigation and includes both static and movable obstacles.

3.1 Environment Representation

The 3D environment $\mathcal{G}$ is represented as a collection of cells $V = \{v_{i,j,k} \mid i, j, k \in \mathbb{Z}^+\}$. The robot's initial position is at cell $S \in V$, and the goal is to reach a target cell $G \in V$. Cells in $\mathcal{G}$ are classified into the following categories:

$$V = \mathcal{F} \cup O_{\text{static}} \cup O_{\text{movable}},$$

where, $\mathcal{F}$ represents free cells that the robot can traverse, O_{static} denotes cells occupied by static obstacles that cannot be moved, and robot must avoid, and O_{movable} represents cells containing movable obstacles.

In this 3D grid environment, the robot can navigate to any of the 26 neighboring cells surrounding its current position, enabling movement along the horizontal, vertical, and depth axes, as well as diagonally in each dimension. The set of possible neighbors for a cell $v_{i,j,k} \in V$ is defined as:

$$\begin{aligned}\mathcal{N}(v_{i,j,k}) = \{&v_{i',j',k'} \in V \mid \ |i - i'| \leq 1, \ |j - j'| \leq 1, \\ &|k - k'| \leq 1 \text{ and } (i, j, k) \neq (i', j', k')\}.\end{aligned}$$

The objective is to find a collision-free path π from S to G, with the ability to relocate movable obstacles in O_{movable} if they block the path.

3.2 Obstacle Constraints and Movable Obstacle Properties

Each movable obstacle $o \in O_{\text{movable}}$ is associated with a weight $w(o) \in \mathbb{R}_+$, which represents the force actions (push or pull) required to relocate the obstacle. The robot has a maximum pushing/pulling capacity, denoted by $w_{\max}$. For a cell $v \in V$, a movable obstacle o can only be relocated if $w(o) \leq w_{\max}$, ensuring that the robot does not attempt to move obstacles beyond its capacity:

$$w(o) \leq w_{\max}, \quad \forall o \in O_{\text{movable}}.$$

Thus, the feasibility of relocating an obstacle depends on the robot's capacity relative to the weight of the obstacle. This constraint limits the path planning algorithm to interactions with obstacles within the robot's handling capabilities, enforcing realistic and practical constraints for robot navigation in 3D environments.

3.3 Path Constraints

Goal Reachability: The algorithm should ensure that a path exists from the start cell S to the goal cell G, if exist at least one path. The path π is valid if: $\exists\ \pi$ such that $S \rightarrow G$ with either $\pi \subseteq \mathcal{F}$ or $\pi \subseteq \mathcal{F} \cup O_{\text{movable}}$.

Collision Avoidance and Obstacle Relocation: The path π must avoid static obstacles and should only interact with movable obstacles when necessary.

3.4 Objective Function

The objective is to find an optimal path $\pi = \{S = v_{0,0,0}, v_{1,1,1}, \ldots, v_{n,n,n} = G\}$ that minimizes the total cost $C(\pi)$, considering both traversal costs and the costs of moving obstacles. Let, $c_{\text{traverse}}(v_{i,j,k}, v_{i+1,j+1,k+1})$ represent the cost of traversing from cell $v_{i,j,k}$ to cell $v_{i+1,j+1,k+1}$, and $c_{\text{move}}(o_{i+1,j+1,k+1}, o')$ represent the cost of relocating a movable obstacle o at the cell i+1,j+1,k+1 to a neighboring cell o', that is the new location of movable obstacles.

The total path cost is then given by,

$$C(\pi) = \sum_{(v_{i,j,k}, v_{i+1,j+1,k+1}) \in \pi} \Big(c_{\text{traverse}}(v_{i,j,k}, v_{i+1,j+1,k+1}) + \sum_{o \in O_{\text{movable}}} c_{\text{move}}(o, o') \Big), \quad (1)$$

The overall objective is to find a feasible path π that minimizes the total cost $C(\pi)$ while adhering to the above constraints:

$$\pi^* = \arg\min_{\pi} C(\pi). \quad (2)$$

3.5 Optimal Path Selection

If both a collision-free path and a path requiring obstacle relocation are available, the algorithm should (i) select the collision-free path if its total cost is equal to the path requiring obstacle relocation. (ii) select the path with obstacle relocation only if its total cost is strictly less than the cost of the collision-free path.

4 Proposed Modified A* Algorithm for Navigation with Movable Obstacle Handling

This section introduces the proposed modified A* algorithm for optimal path planning in a 3D grid environment with both stationary and movable obstacles. The algorithm extends traditional A* by adding cost terms for obstacle relocation, enabling it to minimize both traversal and manipulation costs while respecting the robot's handling constraints.

4.1 Initialization

The algorithm begins by initializing an *open list*, represented as a priority queue, to store cells that are pending exploration, starting from the robot's initial cell S. Each cell in the grid environment $\mathcal{G}$ is represented by the coordinates (i, j, k). The cost of the initial cell $C(S)$ is set to zero. A *closed list* is also initialized to track the cells explored, and the total cost of the target cell G is initially set to infinity. The robot's maximum handling capacity for movable obstacles, denoted by $w_{\max}$, is also defined as an input parameter to guide obstacle relocation decisions. In addition to these two lists, there are two more auxiliary lists: the searched list, which stores all the visited or explored cells, and the moved list, which stores the new coordinates $vn_{i,j,k}$ of any movable obstacles that are relocated during the search process. The moved list is particularly useful when an obstacle needs to be relocated while searching the grid and also in feasible path reconstruction.

4.2 Cost Function and Heuristic Design

For each cell transition from cell ($v_{i,j,k}$ to cell $v_{i+1,j+1,k+1}$), the cost function is defined as:

$$C(v_{i,j,k} \to v_{i+1,j+1,k+1}) = c_{\text{traverse}}(v_{i,j,k}, v_{i+1,j+1,k+1}) + \sum_{o \in O_{\text{movable}}} c_{\text{move}}(o, o'), \quad (3)$$

where c_{traverse} represents the cost of traversing from cell $v_{i,j,k}$ to $v_{i+1,j+1,k+1}$, and c_{move} denotes the cost of relocating a movable obstacle $o \in O_{\text{movable}}$ to a neighboring cell $o' \in \mathcal{F}$.

The *heuristic function* $h(v)$, estimating the cost-to-go from the current cell v to the goal G, is chosen as the Euclidean distance plus any environment cost given to current cell:

$$h(v) = \sqrt{(i_G - i)^2 + (j_G - j)^2 + (k_G - k)^2} + C(i, j, k). \quad (4)$$

The *total cost* for each cell v is given by the A* priority function:

$$f(v) = g(v) + h(v), \quad (5)$$

where $g(v)$ represents the accumulated cost from S to v, incorporating traversal and relocation costs up to the current cell. $g(v)$ also, keeps track of both the previously available free spaces and any new free spaces created by the relocation of obstacles, facilitating robot movement.

4.3 Neighbor Exploration and Obstacle Handling

For each cell v dequeued from the *open list*, the algorithm identifies its neighboring cells $\mathcal{N}(v)$. For each neighbor $v' \in \mathcal{N}(v)$:

Static Obstacles: If v' is occupied by a static obstacle, i.e., $v' \in O_{\text{static}}$, the neighbor is skipped and is declared impassable.

Movable Obstacles: If v' contains a movable obstacle $o \in O_{\text{movable}}$, the algorithm checks if the weight of obstacle $w(o)$ is within the robot's handling capacity $w_{\max}$. If $w(o) \leq w_{\max}$, the obstacle is eligible for relocation, and the cost $c_{\text{move}}(o, o')$ for moving o to an adjacent free cell, if available $o' \in \mathcal{N}(o)$ is computed. To represent the relocation cost mathematically, we can define the relocation cost $c_{\text{move}}(o, o')$ based on two primary factors:

(i) *distance* $d(o, o')$ between the current position of the obstacle o and the target position o'. If o is at position (i, j, k) and o' is at (i', j', k'), the distance can be calculated as:

$$d(o, o') = \sqrt{(i' - i)^2 + (j' - j)^2 + (k' - k)^2}$$

(ii) *effort factor* $e(o)$ which depends on the characteristics of the obstacle, such as its weight or difficulty to move. For simplicity, we took the effort factor $e(o)$ as obstacle weight $w(o)$, representing the physical effort required to move it, i.e., $e(o) = w(o)$. Hence the total relocation cost is:

$$c_{\text{move}}(o, o') = \alpha \cdot d(o, o') + \beta \cdot e(o)$$

where, α and β are scaling coefficients that weight the importance of distance and effort, respectively. This equation provides a straightforward way to compute $c_{\text{move}}(o, o')$ and allows flexible weighting of distance and effort factors to suit specific scenarios.

If the calculated total cost $f(v')$ for reaching v' is lower than any previously recorded cost, the cost is updated, and v' is added to the *open list* for further exploration, and if any obstacles are moved during this procedure, then they are added to the moved list for investigations.

4.4 Obstacle Relocation Strategy

When the robot encounters a movable obstacle o in its path, the algorithm ensures a feasible and optimal relocation. The strategy for relocating obstacles is described as follows:

Step 1: Check Immediate Neighboring Cells First. The algorithm first identifies all immediate neighboring cells of o as potential relocation destinations, denoted by $\mathcal{N}(o)$. For each cell $o' \in \mathcal{N}(o)$: (i) check if o' is free (i.e., not occupied by another obstacle), (ii) if free, computes the cost $c_{\text{move}}(o, o')$ for relocating o to o', where this cost may factor in distance and effort, (iii) ensure that $w(o) \leq w_{\max}$; if true, add o' to a list of feasible relocation cells along with their associated costs.

Step 2: Extended Free Space Search Beyond Immediate Neighbors. If no immediate neighboring cell $o' \in \mathcal{N}(o)$ is free, the algorithm extends the search outward up as, (i) if the cell is free, it is designated as a potential relocation target, (ii) if the cell contains another movable obstacle, the weight of that obstacle $w(o'')$ is added to the cumulative weight in that direction, (iii) the search in a given direction stops as soon as the cumulative count of movable obstacles reaches two or the cumulative weight of encountered obstacles surpasses $w_{\max}$, i.e., $\sum_{o'' \in O_{\text{movable}}} w(o'') \leq w_{\max}$. Afterward, the path becomes feasible for sequentially moving all obstacles to clear the way.

Step 3: Selecting Optimal Neighbors for Path Expansion. Once the valid combination is found, the algorithm selects neighbors based on: - Minimizing $f(v)$, the sum of traversal cost $g(v)$ and heuristic $h(v)$ to the goal. - Preferring neighbors where the path cost is minimal, factoring in both the traversal and relocation costs.

If a neighboring cell's cumulative cost $g(v) + h(v)$ is lower than any previously encountered cost, the cell v is added to the open set $\mathcal{OS}$ for path expansion.

Step 4: Updating the Path with Relocated Obstacles. For each relocation, the state of the grid is updated, and relocated obstacles are tracked. The set $\mathcal{M}$ maintains records of obstacle positions after each move to ensure consistency in further calculations and prevent cycles in relocations.

4.5 Pseudo Code of Algorithm

In order to increase path-finding efficiency, modified A* incorporates a modified heuristic function, based on the modification of the A* search method, as shown in *Algorithm 1*. The optimal path π^* and the most recent locations of the obstacles in the environment are output by the *ReconstructPath* function. The present path cost and the existence of barriers are taken into account when the *HeuristicValue* function calculates to an integer cost value. In order to make sure that obstacles don't obstruct the present path, the *FindFreeSpace* function finds available spaces in neighborhood (the free space also depends on the direction of search, in experimentation we started from north-east clockwise) and determines the new locations of obstacle to determine an agent's ideal orientation. Finally, the *FindMovedObstacle* function updates the environment matrix appropriately to enable additional path analysis.

4.6 Working of Algorithm

The algorithm functions analogously to traditional A* algorithm but incorporates a specific variation in its *FindNeighbours* function as stated in *Algorithm 2*, operates with refined mechanics to address particular situations, leading to improved operational efficiency in those conditions. Our algorithm starts searching from top-right

Algorithm 1 Modified A* Path Finding

Input: S, G, E, O, gw, gh, $w_{\max}$
Output: π^*, $\mathcal{D}$, $\mathcal{M}$

Initialize $\mathcal{OS} \leftarrow S$, Initialize $\mathcal{CF} \leftarrow \emptyset$
$\mathcal{GS} \leftarrow \{S : 0\}$, $\mathcal{FS} \leftarrow \{S : HeuristicValue(S)\}$
$\mathcal{D} \leftarrow \emptyset$, $\mathcal{M} \leftarrow \emptyset$
while $\mathcal{OS} \neq \emptyset$ **do**
  $\mathcal{V} \leftarrow$ lowest $f(v)$ cell from $\mathcal{OS}$
  $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{V}$
  if $\mathcal{V} \equiv G$ **then**
    $\pi^*, \mathcal{M}'' \leftarrow ReconstructPath(\mathcal{CF}, \mathcal{V}, O \cup \mathcal{M}, \mathcal{E})$
    return π^*, $\mathcal{D}$, $\mathcal{M}''$
  end if
  for each $\mathcal{NV}, c, \mathcal{M}'$ in $FindNeighbours(\mathcal{V}, O \cup \mathcal{M}, \mathcal{E}, w\_max, G)$ **do**
    $temp\_g(.) \leftarrow g(\mathcal{V}) + c$
    if $\mathcal{NV} \notin \mathcal{GS}$ **or** $temp\_g(.) < g\_(\mathcal{NV})$ **then**
      $\mathcal{CF}[\mathcal{NV}] \leftarrow \mathcal{V}$
      $\mathcal{GS}\_[\mathcal{NV}] \leftarrow temp\_g(.)$
      $temp\_h\_(.) \leftarrow HeuristicValue(\mathcal{NV}, G, \mathcal{E})$
      $\mathcal{FS}\_[\mathcal{NV}] \leftarrow temp\_g\_(.) + temp\_h\_(.)$
      $\mathcal{OS} \leftarrow \mathcal{OS} \cup \mathcal{FS}\_[\mathcal{NV}]$
      if $\mathcal{M}' \neq \emptyset$ **then**
        for each (o, new_pos) in $\mathcal{M}'$ **do**
          Relocate o to new_pos and update $\mathcal{M}$
        end for
      end if
    end if
  end for
end while

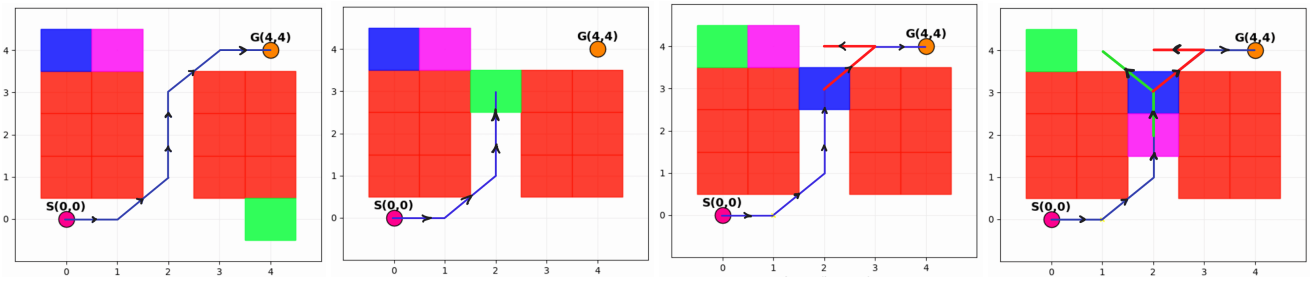
clockwise for the free space, whenever there is free space it relocates robot to that location for further planning. The algorithm provides solutions to special conditions that typical A* routines would overlook including dynamic objects and changing constraints. An illustration in the Figure 1 follows to show multiple special scenarios where the algorithm operates demonstrating its ability to adapt in challenging situations within complex environments. In addition to the aforementioned scenarios, our algorithm also deterministically addresses less common cases such as obstacles located at the goal position and inherently unreachable goals.

- Free Space is Available in Neighborhood
- Free Space is not Available in Neighborhood (Robot's Maximum Capacity Reached)
- Free Space is not Available in Neighborhood (Single Layer Obstacle)
- Free Space is not Available in Neighborhood (Double Layer Obstacle)

4.7 Algorithm Complexity and Optimality

The complexity of the proposed modified A* algorithm depends on the grid-size and the number of movable obstacles.

4.7.1 Proof of complexity. The Modified A* algorithm's complexity includes both time and space factors. Like the standard A*, its time complexity depends on the number of nodes explored, with the



(a) Case 1: No Obstacle (Free) (b) Case 2: Single Obstacle (NM) (c) Case 3: Single Obstacle (M) (d) Case 4: Two Obstacles (M)

Figure 1: The figure illustrate 4 distinguishing cases, showcasing the functionality of Modified A* algorithm, along with final positions of moved obstacles. Obstacles were categorized by color: red for static, green for hard-movable, blue for medium-movable, and purple for soft-movable. Case 1 indicates a free space available to reach the goal, while Case 2 indicates that Robot is not able to move green obstacle as robot's maximum capacity is reached. Case 3 depicts the modifications done on A* to move any neighboring obstacle within the robot's limits. Case 4 shows that two layered obstacles can also be moved simultaneously (within the robot's limit) to different locations.

Algorithm 2 FindNeighbours

Input: $\mathcal{V}, O, \mathcal{E}, w\_max, G$
Output: $\mathcal{NV}, c, \mathcal{M}'$

$v_{i,j,k} \leftarrow \mathcal{V}$
$\mathcal{NV} \leftarrow \emptyset$
for each $v_{i+1,j+1,k+1}$ in 26 directions available **do**
  if $v_{i+1,j+1,k+1}$ is valid cell **then**
    $c \leftarrow$ valid cost to move from $v_{i,j,k}$ to $v_{i+1,j+1,k+1}$
    $\mathcal{M}' \leftarrow \emptyset$
    if $v_{i+1,j+1,k+1}$ is in O **then**
      $\mathcal{ES} \leftarrow FindFreeSpace(v_{i+1,j+1,k+1}, O, G)$
      if $\mathcal{ES} \neq \emptyset$ **then**
        $\mathcal{M}' \leftarrow FindMovedObstacles(v_{+1}, \mathcal{ES}, O, \mathcal{E}, w\_max)$
        if $\mathcal{M}' \neq \emptyset$ **then**
          $pc \leftarrow \sum_{i,j,k} \mathcal{E}[v]$ for $v \in \mathcal{M}'$
          if $pc \leq w\_max$ **then**
            $c \leftarrow c + pc$
            $\mathcal{NV} \leftarrow \mathcal{NV} \cup \{v_{i+1,j+1,k+1} : c\}$
          end if
        end if
      end if
    else
      $\mathcal{NV} \leftarrow \mathcal{NV} \cup \{v_{i+1,j+1,k+1} : c\}$
    end if
  end if
end for
return $(\mathcal{NV}, \mathcal{NV}[.], \mathcal{M}'[.])$

worst-case scenario being $O(b^d)$, where b is the branching factor and d the depth of the optimal path. Heuristics guide the search toward the goal, reducing unnecessary node exploration. However, handling movable obstacles adds overhead, increasing time complexity to $O(b^d \cdot m)$, with m representing the number of such obstacles, especially in dense environments. For space, A* uses a priority queue for open nodes and storage for visited ones, also reaching $O(b^d)$ in the worst case. The modified version requires extra space $O(b^d + k)$, where k accounts for tracking movable obstacle states. This added memory usage enables better performance in dynamic and complex settings.

4.7.2 Proof of optimality. The Modified A* algorithm builds on A*'s principles, aiming to find the lowest-cost path from start to goal. Optimality is preserved when the heuristic is admissible, meaning it never overestimates the true cost—achieved, here via Euclidean distance combined with environmental costs. While the algorithm finds the best path given current obstacle positions, it cannot guarantee optimal obstacle relocation. Sub-optimal configurations may lead to higher-cost paths. Thus, although path-finding remains optimal within a fixed setup, the algorithm cannot always ensure a globally optimal solution. Its performance is sensitive to obstacle arrangements strategies, functioning best in favorable conditions.

4.8 Evaluation Metrics

Real-world assessments of algorithm performances require evaluation metrics to measure their operational effectiveness and quantitative performance. The descriptions of the metrics we used are outlined below:

4.8.1 Planning Time (seconds). Total runtime for algorithm planning, including the time spent on the actual path calculation.

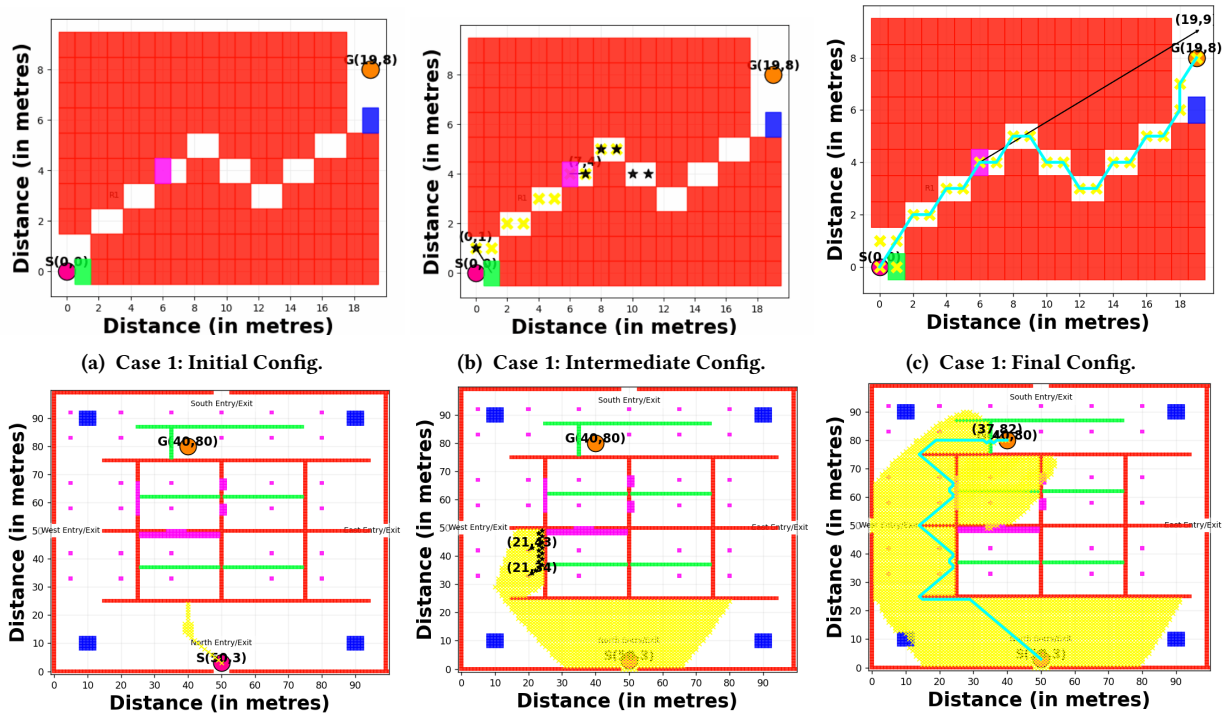
$$T_{\text{planning}} = T_{\text{search}} + T_{\text{reconstruct}} \tag{6}$$

where T_{search} is the estimated time the robot will spend planning or finding path from start to goal along with time needed for adjustments or corrections during movement , and $T_{\text{reconstruct}}$ is the reconstruction time from goal to start location.

4.8.2 Execution Time (seconds). Time taken for the robot to reach from start to goal, using the calculated path.

$$T_{\text{execution}} = T_{\text{motion}} + T_{\text{adjustment}} \tag{7}$$

where T_{motion} is the time the robot spends physically moving from start to goal, and $T_{\text{adjustment}}$ is the additional time needed for adjustments or corrections during movement (such as real-time replanning or obstacle avoidance).



(a) Case 1: Initial Config. (b) Case 1: Intermediate Config. (c) Case 1: Final Config.

(d) Case 2: Initial Config. (e) Case 2: Intermediate Config. (f) Case 2: Final Config.

Figure 2: Two distinct situations are shown, highlighting the robot's start and goal positions, as well as the area explored (in yellow) during the planning process. Case 1 demonstrates that our algorithm effectively identifies a feasible path through a narrow passage, successfully navigating an obstacle by moving it to the passage's exit. In Case 2, the algorithm finds a viable path in a more complex environment, showcasing its robustness and adaptability.

4.8.3 Total Time (seconds). Total runtime for robot to start execution and reach goal (Sum of Planning Time and Execution Time).

$$T_{\text{total}} = T_{\text{planning}} + T_{\text{execution}} \tag{8}$$

4.8.4 Total Distance Traveled (meters. Total distance for robot to travel from start to goal position.

$$D_{\text{total}} = D_{\text{path}} + D_{\text{adjustments}} \tag{9}$$

where D_{path} is the distance covered along the planned path (in meters), and $D_{\text{adjustments}}$ is any additional distance traveled due to adjustments or corrections (such as detours, re-routing, or obstacle avoidance).

5 Results

5.1 Experimental Setup and Environment

To evaluate the Modified A* algorithm, a series of simulations were conducted in various 2D grid environments. The robot is modeled as a point agent with non-prehensile capabilities (push/pull), and a maximum obstacle-handling capacity w_{max} = 6 units. Obstacles are categorized into four types: i. Static (Red): Non-movable, ii. Hard-Movable (Green): Weight = 6 units, iii. Medium-Movable (Blue): Weight = 3 units, iv. Soft-Movable (Purple): Weight = 2 units.

We generated four unique 100 × 100 2D grid maps, each containing different combinations of free space and obstacles. Additional smaller maps were used for illustration purposes. A total of 200 simulation trials were conducted across these environments. Performance was measured in terms of i. planning time, ii. execution time, iii. total path cost, and iv. distance traveled.

5.2 Case Studies of Modified A* Performance

To better understand the algorithm's behavior, two representative scenarios were analyzed (Figure 2).

Case 1 – Navigating a Narrow Passage: The Modified A* algorithm explores possible paths through a constrained narrow spaces by strategically relocating an obstacle at the passage's exit. This demonstrates the algorithm's ability to enable movement where standard planners might fail. The path is refined towards a minimum cost path and eventually finds an sub-optimal route.

Case 2 – Handling Complex Layouts: The algorithm finds available routes between points and eliminates stationary barriers while processing multiple leading solutions. It continuously improves defined path to ensure accurate movement within limited spaces.

5.3 Comparative Performance Evaluation

Table 1 compares the Modified A* against A*, RRT, and RRT* in terms of planning time, execution time, total time, and distance traveled. Despite a slight increase in planning time, the Modified

Table 1: Performance Comparison
Units: Time in seconds, Distance in meters

Evaluation Metrics (Averaged)	A*	Modified A*	RRT	RRT*
Planning Time	0.6490	0.7137	3.4860	6.9586
Execution Time	2.2002	2.3556	1.6293	1.0933
Total Time	2.8492	3.0693	5.1153	8.0519
Distance Traveled	79.721	75.380	84.870	78.006

A* consistently yields shorter paths, demonstrating more efficient navigation by leveraging obstacle manipulation.

The practical path plots in Figure 3, which depicts the key comparison between standard A*,RRT, RRT* and Modified A* algorithms based on a set of performance metrics. Results indicate a shorter actual distance traveled but greater averaged total time as expected in our algorithm, implying better optimization of path.

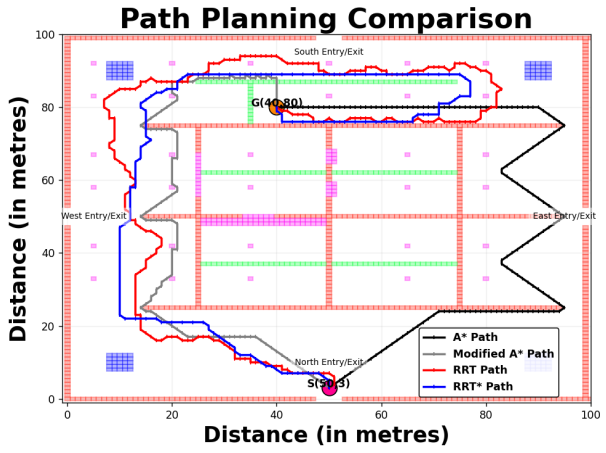


Figure 3: Comparison of A*, Modified A*, RRT, and RRT* algorithms for robot path planning in a cluttered environment, It clearly depicts that our algorithm takes less cost path that other algorithms by moving an obstacle in the path.

6 Conclusion

In this study, we suggested an improved heuristic-based path-planning method that allows robots to actively engage with movable barriers in order to negotiate congested settings. The method strikes an ideal balance between path efficiency and energy expenditure by including barrier management into the A* search framework. The experimental findings show that our approach performs better than conventional A* in terms of path viability, computational effectiveness, and environmental adaptability. These results demonstrate how our method can be used in a variety of real-world scenarios, including search and rescue operations, service robotics, and warehouse automation. Overall, our approach provides a significant improvement over conventional path-planning techniques by expanding feasible navigation possibilities in environments with mixed static and movable obstacles.

7 Future Work

The extension of the suggested method to situations in which impediments are stacked in layers larger than two will be our main goal. Besides this, we will focus on the practical implementation of our algorithm using ROS simulations and mobile robots. Our objective is to avoid needless travel and further lower overall path costs by improving obstacle repositioning techniques by incorporating machine learning techniques.

References

[1] Subhrajit Bhattacharya, Vijay Kumar, and Maxim Likhachev. 2012. Search-based path planning with homotopy class constraints. In *Proceedings of the 2012 AAAI Conference on Artificial Intelligence*. AAAI, 123–129.
[2] B. Chen and G. Quan. 2008. NP-Hard Problems of Learning from Examples. In *2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery*. Jinan, China, 182–186. doi:10.1109/FSKD.2008.406
[3] S. Cheong, B. Y. Cho, J. Lee, et al. 2021. Obstacle rearrangement for robotic manipulation in clutter using a deep Q-network. *Intelligent Service Robotics* 14 (2021), 549–561. doi:10.1007/s11370-021-00377-4
[4] Dave Ferguson and Anthony Stentz. 2006. Anytime RRTs. In *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 5369–5375.
[5] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4, 2 (1968), 100–107.
[6] J. Haustein, J. King, S. Srinivasa, and T. Asfour. 2015. Kinodynamic randomized rearrangement planning via dynamic transitions between statically stable states. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*. 3075–3082.
[7] Lydia E. Kavraki, P. Svestka, Jean-Claude Latombe, and Mark H. Overmars. 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* 12, 4 (1996), 566–580.
[8] Steven M. LaValle. 2001. Rapidly-exploring random trees: A new tool for path planning. In *The Annual Research Report*. 91–100.
[9] Martin Levihn, Rachid Alami, and Mike Stilman. 2013. A hierarchical approach to planning with movable obstacles. In *Proceedings of the 2013 IEEE International Conference on Robotics and Automation*. IEEE, 802–807.
[10] Chengshu Li, Fei Xia, Roberto Martin-Martin, and Silvio Savarese. 2020. Hrl4in: Hierarchical reinforcement learning for interactive navigation with mobile manipulators. In *Conference on Robot Learning*. PMLR, 603–616.
[11] Li-sang Liu, Jia-feng Lin, Jin-xin Yao, Dong-wei He, Ji-shi Zheng, Jing Huang, and Peng Shi. 2021. Path planning for smart car based on Dijkstra algorithm and dynamic window approach. *Wireless Communications and Mobile Computing* 2021, 1 (2021), 8881684.
[12] Jie Qi, Hui Yang, and Haixin Sun. 2020. MOD-RRT*: A sampling-based algorithm for robot path planning in dynamic environment. *IEEE Transactions on Industrial Electronics* 68, 8 (2020), 7244–7251.
[13] Anthony Stentz. 1994. Optimal and efficient path planning for partially-known environments. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, Vol. 4. IEEE, 3310–3317.
[14] Jiankun Wang, Max Q-H Meng, and Oussama Khatib. 2020. EB-RRT: Optimal motion planning for mobile robots. *IEEE Transactions on Automation Science and Engineering* 17, 4 (2020), 2063–2073.
[15] Fei Xia, Chengshu Li, Roberto Martín-Martín, Or Litany, Alexander Toshev, and Silvio Savarese. 2021. Relmogen: Integrating motion generation in reinforcement learning for mobile manipulation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 4583–4590.

Received 24 February 2025; revised 7 May 2025