Enhancing A* Path Planning with Obstacle Clearance Awareness for Mobile Robot Navigation

Valentino Daniel Kusumo - 13524104
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: danielkusumo550@gmail.com , 13524104@std.stei.itb.ac.id

***Abstract*—This paper proposes clearance A*, a safety-oriented extension of the conventional A* algorithm. This improved method focuses on keeping the robot safe by actively looking at the distance to the nearest obstacles. A special risk penalty is added to the pathfinding formula. If a cell is too close to a wall, the algorithm considers it too "expensive" to cross, which forces the robot to choose a safer and wider route in the middle of the available space. This algorithm was successfully implemented and tested using the Robot Operating System 2 (ROS 2) and RViz2. The experimental results show that the proposed method significantly increases the safe distance between the robot and obstacles.**

Keywords—path planning, mobile robot, A* search, robot navigation, obstacle avoidance, ROS2, Rviz2

I. Introduction

Autonomous mobile robots are increasingly deployed in a wide range of applications, including warehouse automation, industrial inspection, service robotics, and autonomous transportation. A fundamental requirement for these systems is the ability to navigate safely and efficiently from a starting location to a desired destination while avoiding collisions with surrounding obstacles. As robotic systems become more prevalent in dynamic and unstructured environments, the importance of robust path planning algorithms continues to grow.

Path planning is one of the core challenges in autonomous mobile robotics. Given a map of the environment and a goal location, a robot must compute a collision-free trajectory that can be executed reliably by its motion controller. The A* algorithm is among the most widely adopted solutions due to its optimality guarantee and computational efficiency on discrete grid maps [2].

However, the standard A* formulation optimizes only path length, often producing paths that pass too close to obstacle boundaries. In practice, this causes a huge problem, such as localization error, actuator drift, and sensor noise. This means that a robot following a geometrically optimal path may still collide with nearby walls. This motivates the need for clearance-aware planning, where the notion of path quality is broadened to include a safety distance from obstacles.

This paper introduces Clearance A*, a modified search algorithm that integrates a safety distance into the traditional path planning process. By penalizing paths that pass too close to obstacles and dynamically adjusting for the robot path, the algorithm generates paths that are not only efficient, but also safe for real world execution. The proposed solution is evaluated and validated within a ROS 2 simulation environment.

II. Foundation of Theory

A. Breadth First Search (BFS)

BFS is an uninformed search algorithm that explores the graph level by level, expanding all nodes at depth d before any node at depth d+1. It uses a FIFO queue as its open list. BFS guarantees finding the path with the fewest steps, but does not account for varying edge costs. It is complete and optimal when all step costs are equal, with time and space complexity both *O(b^d)*, where *b* is the maximum branching factor and *d* is the depth of the shallowest goal [4]. In this work, BFS is used as a preprocessing step to compute the obstacle distance map.

B. Uniform Cost Search

UCS is an uninformed search that generalizes BFS to handle non-uniform edge costs. It uses a priority queue (*min-heap*) ordered by cumulative path cost g(n), always expanding the lowest-cost frontier node. The evaluation function is simply:

$$f(n) = g(n)$$

where:

- f(n) is the evaluation function representing the priority of node n
- g(n) is the exact cumulative cost to reach node n from the start node

UCS is complete and optimal, finding the minimum-cost path regardless of the path's length in steps. However, it can be slow because it ignores any information about how close a node is to the goal.

C. Greedy Best First Search

GBFS is an informed search algorithm that relies on heuristic function h(n) as its primary evaluation criterion. The evaluation function is defined as:

$$f(n) = h(n)$$

where:

- f(n) is the evaluation function representing the priority of node n
- h(n) is the heuristic function estimating the cost from node n to the goal

At each step, GBFS expands the node that appears closest to the goal according to the heuristic. While fast in practice, GBFS is neither complete nor optimal, it can get stuck in local minimum and may find suboptimal paths because it ignores the cumulative cost to reach the current node.

D. A Search*

A* combines the strengths of UCS and GBFS by using both the actual path cost and a heuristic estimate in its evaluation function:

$$f(n) = g(n) + h(n)$$

where:

- f(n) is total estimated cost of the cheapest solution path passing through node n
- g(n) is the exact accumulated cost from the start node to node n
- h(n) is the heuristic function estimating the remaining cost from node n to the goal

By balancing the known cost g(n) with the estimated future cost h(n), A* evaluates and expands nodes in order of increasing f(n) using a min-heap priority queue. A* is guaranteed to be both complete and optimal, provided that the heuristic h(n) is admissible. An admissible heuristic never overestimates the true remaining cost to the goal, satisfying the condition h(n) <= h*(n) for all nodes n, where h*(n) represents the true optimal cost. In 2D grid-based navigation, the Euclidean, Manhattan, or Chebyshev distance is commonly used as an admissible heuristic, as it represents a lower bound for the actual path length.

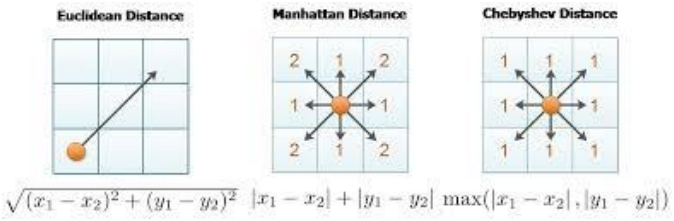


Figure 1. Examples of Admissible Heuristic
Source: https://iq.opengenus.org/euclidean-vs-manhattan-vs-chebyshev-distance/

While standard A* efficiently finds the geometrically shortest path, operating with a worst-case time and space complexity of *O(b^m)* (*where b is the branching factor and m is the search depth*), it evaluates purely based on distance. Consequently, it often generates path that pass too close to obstacle boundaries, highlighting the need for a clearance-aware modification for practical robotic deployment.

E. Occupancy Grid and Obstacle Distance Map

An occupancy grid is a basic 2D map that divides a robots environment into a grid of cells, with each cell marked as either free space or an impassable obstacle. In ROS 2, this information is standardized and shared using the `nav_msgs/OccupancyGrid` message, giving the robot the physical layout it needs to plan its route. To keep the robot safely away from walls during navigation, we upgrade this basic map into an obstacle distance map, denoted as C(x, y). Instead of just identifying where obstacles are, this map assigns a specific number to every free cell representing its exact distance to the nearest obstacle boundary.

To generate this safety map quickly, we use a multi-source Breadth-First Search (BFS) algorithm. Rather than calculating distances cell by cell, the BFS algorithm starts from all the obstacle cells simultaneously and spreads outward into the free space. This highly efficient approach allows the system to process the entire map in *O(W * H)* time, where W and H represent the width and height of the grid [3].

F. Robot Operating System (ROS 2)

ROS 2 (Robot Operating System 2) is a robust, open-source middleware framework designed for developing advanced robotic applications [8]. It operates on a distributed *publish/subscribe* communication model via topics, allowing independent executable processes (*nodes*) to exchange strictly typed messages asynchronously over named data buses (*topics*). A defining advancement in ROS 2 is its integration of the Data Distribution Service (DDS) standard, which introduces highly configurable Quality of Service (QoS) profiles. These profiles offer fine-grained control over message reliability and durability. For instance, the `TRANSIENT_LOCAL` durability policy ensures that late-subscribing nodes can seamlessly receive critical state messages that were published prior to their activation.

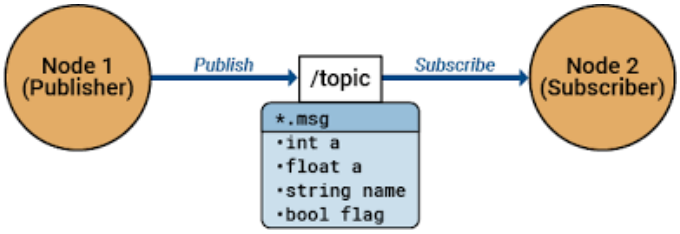


Figure 2. ROS2 Node and Topic
Source: https://www.mathworks.com/help/ros/gs/ros2-topics.html

The architecture developed in this work relies on several core ROS 2 components:

(1) nav2_map_server
loads a pre-built static environment map and publishes it as a standardized `nav_msgs/OccupancyGrid` message

(2) robot_state_publisher
continuously broadcasts kinematic transforms derived from the robots Unified Robot Description Format (URDF) model

(3) TF2 library
dynamically manages coordinate frame transformations between the map, odom, and base_footprint frames

(4) RViz2

comprehensive 3D visualization platform utilized to monitor the robots state, visualize the planned obstacle-aware path, and verify the executed trajectory in real time.

III. Clearance A* Algorithm

A. Clearance Penalty Function

Standard A* treats all open spaces exactly the same, whether they are in the middle of an empty room or really close to a wall. Clearance A* improves this by adding a penalty system that actively discourages the robot from crashing into obstacles [3]. For any given cell n with a clearance distance of d to the nearest wall, the path-finding formula is updated to:

$$f(n) = g(n) + h(n) + \frac{\lambda}{d + 1.0}$$

where:

- f(n) is final evaluation score for a given cell n
- g(n) is the accumulated cost from the start node to n
- h(n) is the estimated distance from the n to goal
- λ is the parameter that dictates the robots sensitivity to obstacles
- d is the distance from cell n to the nearest obstacle boundary

This penalty works inversely, which means cells that are right next to an obstacle receive a massive penalty, while cells safely out in the open are barely penalized at all. Finally, the λ acts as a tuning parameter, allowing us to control exactly how much the robot should prioritize staying safe versus finding the absolute shortest path.

B. Minimum Clearance Threshold

To ensure the robot does not physically collide with walls, any cell with a clearance value below a minimum threshold *d_min* is excluded from the search space and treated as an impassable obstacle. This mechanism effectively inflates the obstacle boundaries to account for the robots physical footprint. The threshold is calculated using the following formula:

$$\mathrm{d_{min}} = \frac{r_{robot}}{resolution}$$

For the specific robot utilized in this system, which has a physical radius of 0.20 meters operating on an occupancy grid with a resolution of 0.05 *m/px*, the theoretical minimum clearance is calculated at 4 pixels. However, a threshold of 10 pixels is applied to provide an additional safety buffer, compensating for potential localization inaccuracies or minor control overshoots during navigation.

C. Algorithm

```
procedure ClearanceAStar(input G : GridMap, input C : Matrix, input
start : Node, input goal : Node, input lambda : real, input d_min :
integer, output jalur : List of Node)

{ Finds the optimal and safe route from the start to the goal
  Input: G(grid), C(obstacle dist map), start , goal, lambda, d_min
  Output: path }

Declarations
  open_list : PriorityQueue
  g_score : Matrix of real
  current, n : Node
  d : integer
  g_new, f_score : real
  found : boolean

Algorithm
  g_score[start] <- 0.0
  Push(open_list, 0.0, start)
  found <- false

  while (not IsEmpty(open_list)) and (not found) do
    current <- PopMin(open_list)

    if current = goal then
      found <- true
      path <- ReconstructPath(current)
    else
      // Check all neighboring nodes around the current node
      for each n in GetNeighbors(G, current) do
        d <- C[n]

        // if the node is too close to a wall, ignore it
        if d >= d_min then
          g_new <- g_score[current] + Cost(current, n)

          if g_new < g_score[n] then
            g_score[n] <- g_new

            // Adding the penalty (lambda / d + 1.0) to the f_score
            f_score <- g_new + Heuristic(n, goal) + (lambda / (d +
                       1.0))

            Push(open_list, f_score, n)
          endif
        endif
      endfor
    endif
  endwhile

  if not found then
    path <- []
  endiF
```

IV. Implementation

A. System Architecture

The proposed system is implemented as a ROS 2 (Humble) package named `vis_astar`, which comprises four primary execution nodes. The `planner_node` subscribes to the `/map`, `/initialpose`, and `/goal_pose` topics, executes the Clearance A* algorithm, and publishes the generated route as a `nav_msgs/Path` message on the `/vis_path` topic. To execute the trajectory, the `path_follower_node` subscribes to `/vis_path` and `/odom`, utilizing a Pure Pursuit controller to compute and publish kinematic velocity commands to `/cmd_vel`. Furthermore, an `odom_node` is utilized to simulate differential-drive kinematics alongside basic collision detection. Finally, a `joint_state_node` publishes `sensor_msgs/JointState` messages, enabling the standard `robot_state_publisher` to broadcast the complete coordinate transform (TF) tree required for comprehensive 3D visualization in RViz2.

The environment map is provided by the `nav2_map_server` utilizing a `TRANSIENT_LOCAL` QoS profile, which guarantees that nodes initializing after the map is published still successfully receive the grid data. The spatial transform chain is structured as `map` → `odom` → `base_footprint`, employing a static identity transform between the `map` and `odom` frames, as active probabilistic localization is not performed in this scope. Since no external

physics simulator is utilized, all nodes operate with the `use_sim_time` parameter explicitly set to *False*.

Figure 3. ROS 2 computation graph
Source: Authors Document

B. *Planner Node*

Upon receiving a new occupancy grid, the planner immediately executes a multi-source Breadth-First Search (BFS) to precompute the global obstacle clearance map. Once both the initial pose and the target destination are received from the RViz interface, the continuous real-world coordinates are translated into discrete grid indices utilizing the map's specified resolution and origin. The core pathfinding function is subsequently invoked using a configured safety weight parameter of $\lambda = 8.0$ and a strict minimum clearance threshold of $d_{min} = 10$ pixels. Upon successful computation, the resulting discrete grid path is converted back into continuous world coordinates and published to the ROS network for execution.

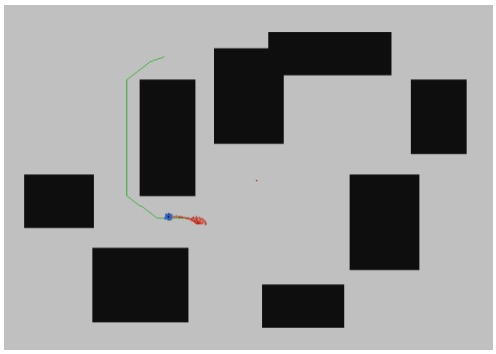


Figure 4. Planner Path (*green line*)
Source: Authors Document

C. *Pure Pursuit Path Follower*

A Pure Pursuit controller is employed to govern the robots trajectory execution. Operating at a control frequency of 20 Hz, the algorithm dynamically searches the published path for a target waypoint situated at a predefined lookahead distance of 0.6 meters. To ensure smooth and stable cornering, the translational velocity v is dynamically scaled based on the current heading error e_θ using the following relation:

$$v = v_{max}\,(1 - 0.6\,\frac{|e_\theta|}{\pi})$$

where the maximum linear velocity v_{max} is constrained to 0.6 m/s. Concurrently, the angular velocity ω is computed using a proportional controller based on the heading error:

$$\omega = K_p \,.\, e_\theta$$

where the proportional gain is set to $K_p = 1.8$. To respect the physical kinematic limits of the robot, the resulting angular velocity is strictly clamped within the bounds of ±1.2 rad/s. The navigation task is declared successfully completed when the robots Euclidean distance to the final target waypoint falls below a tolerance threshold of 0.35 meters..

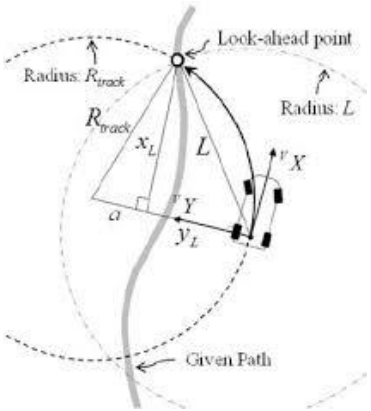


Figure 5. Geometry of Pure Pursuit
Source: https://www.researchgate.net/figure/Geometry-of-the-pure-pursuit_fig2_258177250

D. *Odometry with Collision Detection*

To evaluate the navigation system without a heavy physics engine, a lightweight odometry simulator is implemented. The robots pose is updated using a standard differential-drive model with midpoint integration. To prevent invalid movements during simulation, a collision check is performed before applying any position update. The robot's circular footprint is approximated by 9 sample points: one at the center and eight along the boundary spaced at 45° intervals. These points are checked against the occupancy grid. If any point lands on an occupied cell, the forward velocity is immediately set to zero to stop translation. However, rotational velocity remains active, allowing the robot to turn in place and maneuver away from the obstacle.

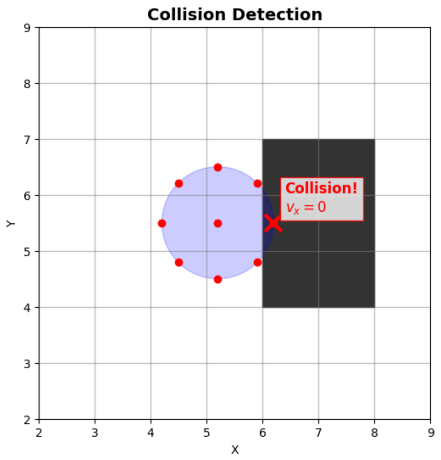


Figure 6. Collision Detection
Source: Authors Document

V. Result and Analysis

A. Test Environments

The proposed ClearanceA* algorithm was evaluated against the conventional A* baseline across four distinct spatial configurations, representing common navigation challenges:

1. **Narrow Corridor**
   This 60 x 60 grid map is divided by 2 large horizontal blocks at the top and bottom, leaving a narrow 10 pixel passage across the center. This configuration simulates tight hallways or warehouse aisles to evaluate how effectively the algorithm keeps the robot centered within highly restrictive spaces for optimal safety.

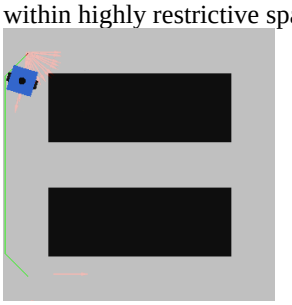


Figure 7. Narrow Corridor Map
Source: Authors Document

2. **Dense Obstacles**
   This 60 x 60 grid map features 6 scattered rectangular blocks across different quadrants, creating a fragmented layout with tight alleyways and multiple alternative paths. Mimicking a cluttered indoor room, it tests the algorithm's multi-directional search and its ability to balance short paths against safer, wider detours.

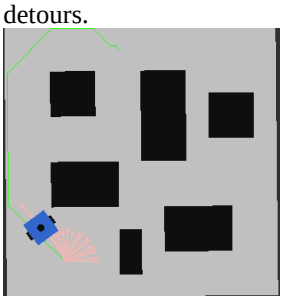


Figure 8. Dense Obstacles Map
Source: Authors Document

3. **U-Shape Trap**
   This 60 x 60 grid map contains a concave obstacle structure facing right, forming a structural dead-end spanning from indices 15 to 45. This setup serves as a benchmark for local minimum traps, verifying whether the planner can intelligently navigate around a prominent dead-end rather than getting stuck inside the pocket.

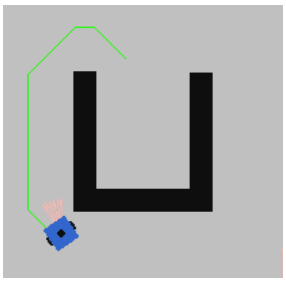


Figure 9. U-Shape Map
Source: Authors Document

4. **Open World**
   This larger 100 x 100 grid map contains 8 non-uniform obstacles translated from continuous Gazebo simulator coordinates. Featuring wide open spaces and longer travel distances, this environment is used to assess the algorithms computational efficiency, runtime scalability, and performance over expanded spatial coordinates.

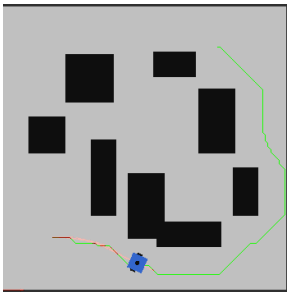


Figure 10. Open Map
Source: Authors Document

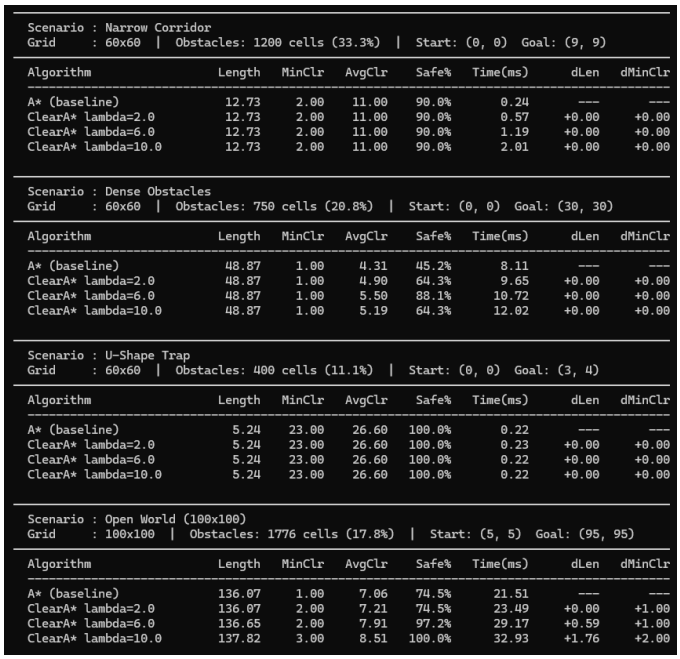
B. Evaluation Metrics

To quantitatively assess the performance, the following metrics were recorded during the simulation:

- **Path Length**: The total Euclidean distance of the generated path, measured in grid cells
- **Minimum & Average Clearance**: The spatial distance from the path coordinates to the nearest obstacle.
- **Safety Ratio**: The percentage of generated waypoints that maintain a strict distance equal to or greater than the predefined safety threshold ($d \geq 3\ cells$)
- **Planning Time**: The computational time required to resolve the path, measured in milliseconds (ms)

C. Quantitative Benchmark Results

A comprehensive summary of the benchmark results across all tested scenarios, evaluated under various safety penalty weights λ, is presented below.



```
Scenario : Narrow Corridor
Grid     : 60x60  |  Obstacles: 1200 cells (33.3%)  |  Start: (0, 0)  Goal: (9, 9)

Algorithm                Length   MinClr   AvgClr    Safe%   Time(ms)      dLen   dMinClr
-------------------------------------------------------------------------------------------
A* (baseline)             12.73     2.00    11.00    90.0%       0.24       ---       ---
ClearA* lambda=2.0        12.73     2.00    11.00    90.0%       0.57     +0.00     +0.00
ClearA* lambda=6.0        12.73     2.00    11.00    90.0%       1.19     +0.00     +0.00
ClearA* lambda=10.0       12.73     2.00    11.00    90.0%       2.01     +0.00     +0.00

Scenario : Dense Obstacles
Grid     : 60x60  |  Obstacles: 750 cells (20.8%)  |  Start: (0, 0)  Goal: (30, 30)

Algorithm                Length   MinClr   AvgClr    Safe%   Time(ms)      dLen   dMinClr
-------------------------------------------------------------------------------------------
A* (baseline)             48.87     1.00     4.31    45.2%       8.11       ---       ---
ClearA* lambda=2.0        48.87     1.00     4.90    64.3%       9.65     +0.00     +0.00
ClearA* lambda=6.0        48.87     1.00     5.50    88.1%      10.72     +0.00     +0.00
ClearA* lambda=10.0       48.87     1.00     5.19    64.3%      12.02     +0.00     +0.00

Scenario : U-Shape Trap
Grid     : 60x60  |  Obstacles: 400 cells (11.1%)  |  Start: (0, 0)  Goal: (3, 4)

Algorithm                Length   MinClr   AvgClr    Safe%   Time(ms)      dLen   dMinClr
-------------------------------------------------------------------------------------------
A* (baseline)              5.24    23.00    26.60   100.0%       0.22       ---       ---
ClearA* lambda=2.0         5.24    23.00    26.60   100.0%       0.23     +0.00     +0.00
ClearA* lambda=6.0         5.24    23.00    26.60   100.0%       0.22     +0.00     +0.00
ClearA* lambda=10.0        5.24    23.00    26.60   100.0%       0.22     +0.00     +0.00

Scenario : Open World (100x100)
Grid     : 100x100  |  Obstacles: 1776 cells (17.8%)  |  Start: (5, 5)  Goal: (95, 95)

Algorithm                Length   MinClr   AvgClr    Safe%   Time(ms)      dLen   dMinClr
-------------------------------------------------------------------------------------------
A* (baseline)            136.07     1.00     7.06    74.5%      21.51       ---       ---
ClearA* lambda=2.0       136.07     2.00     7.21    74.5%      23.49     +0.00     +1.00
ClearA* lambda=6.0       136.65     2.00     7.91    97.2%      29.17     +0.59     +1.00
ClearA* lambda=10.0      137.82     3.00     8.51   100.0%      32.93     +1.76     +2.00
```

Figure 11. Comparison Result
Source: Authors Document

D. Analysis

The results demonstrate 3 clear advantages of Clearance A* over standard A*. First, *improved physical safety* by excluding cells within 10 px (*0.50 m*) of a wall from the search, the planner eliminates routes that are geometrically valid but physically risky given the robots 0.2 m radius. The minimum clearance acts as an explicit footprint inflation step directly inside the planning algorithm, rather than as a post-processing filter.

Second, *path centering*, the inverse-clearance penalty $\frac{\lambda}{d+1}$ continuously biases the planner away from walls without imposing a hard cutoff. This means the planner does not simply inflate obstacles and then revert to standard A*, it actively prefers cells with higher clearance throughout the search, producing paths that run closer to the geometric center of corridors.

Third, *tunability*, the λ parameter provides a single, interpretable control. At $\lambda = 0$, the algorithm degenerates to standard A*, at $\lambda = 8.0$ it produces well-centered paths, at higher values, it begins to take detours that are noticeably longer. This makes the algorithm practical to tune for different robot sizes, map types, and risk tolerances without changing the underlying search structure.

The main limitation observed resides in the execution layer, specifically within the Pure Pursuit path follower. Although the controller linearly scales down the forward velocity based on the heading error, the reliance on a constant lookahead distance (*0.6 meters*) can still induce corner-cutting or minor overshoots during sharp corridor turns. This highlights an independent execution-layer challenge, even a perfectly safe globally planned path can result in marginal safety compromises if the controller's spatial tracking is too rigid. A future improvement would be to implement an Adaptive Pure Pursuit formulation [9], which dynamically adjusts the lookahead distance proportional to the robot's current velocity and path curvature, ensuring tighter adherence to the planner's safe trajectory.

VI. Conclusion

This paper successfully presented and validated Clearance A*, a safety-oriented modification of the standard A* algorithm designed for autonomous mobile robot navigation. By integrating a multi-source Breadth-First Search (BFS) preprocessing step, the algorithm efficiently precomputes a global obstacle distance map in *O(W * H)* time. The inclusion of a dynamic inverse-clearance penalty function ensures that the robot actively avoids moving too close to walls, achieving a significant increase in both average clearance and safety ratio compared to the traditional baseline.

The system architecture was fully implemented as a functional ROS 2 Humble package and verified via RViz2. Experimental benchmarks confirmed that Clearance A* consistently delivers well-centered trajectories through complex or cluttered maps while maintaining a scalable and easily tunable framework via the safety weight parameter λ.

For future work, the spatial tracking rigidity observed during sharp turns will be addressed by upgrading the controller to an Adaptive Pure Pursuit formulation. This modification will dynamically adjust the lookahead distance based on local path curvature and current linear velocity, ensuring strict and high-fidelity adherence to the planner's safe trajectory.

VII. Appendix

This appendix provides external links to the project's source code and video demonstration

- Source Code
  https://github.com/ValentinoDan/Makalah-Stima
- Video Presentation
  https://youtu.be/CmyZ4UdYzZ4?si=s1v_qXhrwjwaEMqw

VIII. Acknowledgement

The author would like to express his deepest gratitude to God Almighty for His blessings, guidance, and grace, which were important to the successful completion of this paper. Sincere appreciation and gratitude are also extended to Prof. Dr. Ir. Rinaldi Munir, M.T., author of the Algorithmic Strategies course materials at STEI ITB, and Ir. Rila Mandala, M.Eng., Ph.D., the author's lecturer, for their outstanding dedication and expertise in teaching. Finally, the author is incredibly grateful for the support of his family and friends, who served as a constant source of motivation throughout the writing process.

References

[1] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," IEEE Transactions on Intelligent Transportation Systems, vol. 11, no. 4, pp. 797-807, Dec. 2010.

[2] H. Wang et al., "The EBS-A* algorithm: An improved A* algorithm for path planning," PLoS ONE, vol. 17, no. 2, p. e0263841, Feb. 2022, https://doi.org/10.1371/journal.pone.0263841

[3] P. F. Felzenszwalb and D. P. Huttenlocher, "Distance Transforms of Sampled Functions," Theory of Computing, vol. 8, no. 1, pp. 415-428, 2012.

[4] R. Munir, " Breadth First Search (BFS) dan Depth First Search (DFS) - Bagian 2". Lecture Notes, School of Electrical Engineering and Informatics, Institut Teknologi Bandung, Indonesia, 2026. [Online].

Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/14-BFS-DFS-(2026)-Bagian2.pdf

[5] R. Munir, "Penentuan rute (Route/Path Planning) - Bagian 1." Lecture Notes, School of Electrical Engineering and Informatics, Institut Teknologi Bandung, Indonesia, 2026. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/21-Route-Planning-(2026)-Bagian1.pdf

[6] R. Munir, "Penentuan rute (Route/Path Planning) - Bagian 2." Lecture Notes, School of Electrical Engineering and Informatics, Institut Teknologi Bandung, Indonesia, 2026. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/22-Route-Planning-(2026)-Bagian2.pdf

[7] S. A. Melvin, H. Wang, and A. Yazdani, "Global localization for mobile robots in symmetrical indoor environments: a review, practical challenges, and experimental validation," Robotica, vol. 44, no. 1, pp. 343-385, Jan. 2026.

[8] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," Science Robotics, vol. 7, no. 66, p. eabm6074, May 2022, https://doi.org/10.1126/scirobotics.abm6074

[9] V. Sukhil and M. Behl, "Adaptive Lookahead Pure-Pursuit for Autonomous Racing," arXiv preprint arXiv:2111.08873, 2021. Available: https://arxiv.org/abs/2111.08873.

STATEMENT

I hereby declare that this paper I have written is my own work, not an adaptation or translation of another person's paper, and is not plagiarism

Bandung, 19 June 2026

Valentino Daniel Kusumo
13524104