AI 3603 Artificial Intelligence: Principles and Techniques

By: Kaiyang Xu (523030910085)

HW#: 1

October 17, 2025

I. TASK 1: BASIC A* ALGORITHM

A. Problem Formulation

We consider a 120×120 occupancy grid map where each grid cell represents a $1.0\,\mathrm{m} \times 1.0\,\mathrm{m}$ block. The robot starts from a known start position $s_s = (x_s, y_s)$ and aims to reach a known goal position $s_g = (x_g, y_g)$. Obstacles are encoded by `world_map` $\in \{0,1\}^{120\times120}$ where 1 indicates an occupied cell (non-traversable) and 0 indicates a free cell. The motion model is 4-connected (up, down, left, right) with unit step cost.

B. Method

Let the state space be $\mathcal{S} = \{(x,y) \in \mathbb{Z}^2 \mid 0 \le x, y < 120\} \setminus \mathcal{O}$ where $\mathcal{O}$ is the set of obstacle cells. The neighbor function for 4-connectivity is

$$\mathcal{N}_4(s) = \{(x \pm 1, y),\ (x, y \pm 1)\} \cap \mathcal{S}.$$

We define the path cost $g(s)$ as the number of steps from s_s to state s along the discovered path, and use the Manhattan heuristic as an estimate to the goal:

$$h(s) = |x - x_g| + |y - y_g|.$$

The A* evaluation function is

$$f(s) = g(s) + h(s).$$

We implement the **basic A* algorithm** with the following data structures:

- **open_set**: a priority queue keyed by $f(s)$ implemented via `heapdict`, which supports efficient key updates.
- **closed_set**: a hash set of states that have been expanded and for which the optimal g-value has been determined.
- **parent**: a hash map storing the best predecessor of each discovered state for path reconstruction.
- **g-score**: a hash map storing the current best-known path cost to each discovered state.

The **basic A* algorithm** works as follow:

1. **Initialization:** set $g(s_s) = 0$, compute $f(s_s) = g(s_s) + h(s_s) = h(s_s)$, insert s_s into the `open_set` with priority f.
2. **Main loop:** while the `open_set` is not empty:
   (a) Pop the state s with minimal $f(s)$ from the `open_set`.
   (b) If $s = s_g$, terminate and reconstruct the path by following `parent` backwards from s_g to s_0.
   (c) Insert s into the `closed_set`.
   (d) For each neighbor $s' \in \mathcal{N}_4(s)$ that is in bounds and not an obstacle:
      i. Compute $\tilde{g} = g(s) + 1$.
      ii. If $s' \in$ `closed_set` and $\tilde{g} \ge g(s')$, continue.
      iii. If s' is unseen or $\tilde{g} < g(s')$, then set $g(s') \leftarrow \tilde{g}$, $f(s') \leftarrow g(s') + h(s')$, `parent`$[s'] \leftarrow s$, and insert/update s' in the `open_set` with priority $f(s')$.
3. **Failure:** if the open set becomes empty before reaching s_g, report that no feasible path exists.

C. Result Analysis

Figure 1 shows a representative path produced by **basic A* algorithm** from the given start to the goal on the provided map. The red polyline is the planned path, blue and red crosses denote goal and start, and black dots indicate obstacles.

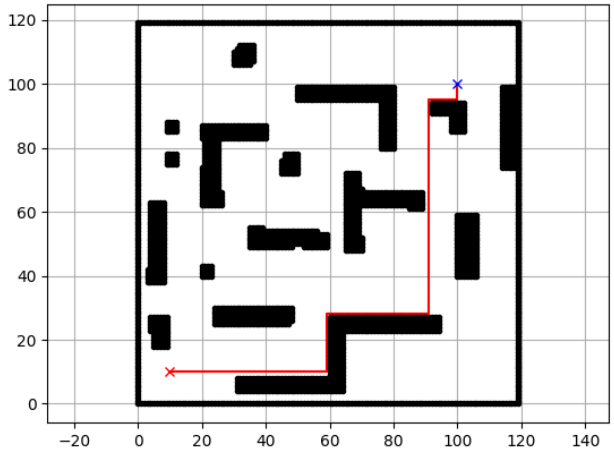


FIG. 1: A* path on the 120×120 map (Task 1).

In terms of **computational time**, the computational time of the **basic A* algorithm** is 58.70 ms.

In terms of **safety**, The algorithm never inserts obstacle cells into `open_set` (explicit check against `world_map`$[x][y] = 1$), hence every returned path is collision-free with respect to the provided occupancy grid. However, due to frequent turns and close distance to the obstacles, the path planned by the **basic A* algorithm** is still not safe enough, which will be refined later in **Task 2**.

In terms of **optimality**, using the Manhattan heuristic on a 4-connected, unit-cost grid is both *admissible* and *consistent*, ensuring that the first time the goal is popped from `open_set`, the path has minimal number of grid steps. We can also verify its optimality since the grid steps of our path is 180, which equals to $|x_s - x_g| + |y_s - y_g|$.

II. TASK 2: IMPROVED A* ALGORITHM

A. Problem Formulation

We consider a 120×120 occupancy grid map where each grid cell represents a $1.0\,\mathrm{m} \times 1.0\,\mathrm{m}$ block. The robot starts from a known start position $s_s = (x_s, y_s)$ and aims to reach a known goal position $s_g = (x_g, y_g)$. Obstacles are encoded by `world_map` $\in \{0,1\}^{120\times120}$ where 1 indicates an occupied cell (non-traversable) and 0 indicates a free cell. The motion model is 8-connected (up, down, left, right, upper left, upper right, bottom left, bottom right) with step cost as follow:

$$cost_{\text{step}}(s \to s') = \begin{cases} 1, & \text{axis-aligned} \\ \sqrt{2}, & \text{diagonal} \end{cases}.$$

B. Method

The neighbor function for 8-connectivity is

$$\mathcal{N}_8(s) = \{(x \pm 1, y),\ (x, y \pm 1),\ (x \pm 1, y \pm 1)\} \cap \mathcal{S}.$$

We define the path cost $g(s)$ by augmenting the step cost with two penalties:

$$g(s') \leftarrow g(s) + \underbrace{\text{cost}_{\text{step}}(s \to s')}_{\text{cost}_{\text{step}}} + \underbrace{\frac{w_{\text{obs}}}{d(s')+1}}_{\text{cost}_{\text{safety}}} + \underbrace{w_{\text{turn}}\,\mathbf{1}[\,\text{dir}(s \to s') \neq \text{dir}(\text{parent}(s) \to s)\,]}_{\text{cost}_{\text{turn}}},$$

where $d(s')$ is the distance from s' to the nearest obstacle; w_{obs} and w_{turn} are non-negative weights; and $\mathbf{1}[\cdot]$ is the indicator function.

The heuristic is the octile distance

$$h(s) = (\sqrt{2}-1)\min\{|x - x_g|, |y - y_g|\} + \max\{|x - x_g|, |y - y_g|\}.$$

The A* evaluation function is

$$f(s) = g(s) + h(s).$$

We implement the **improved A* algorithm** with the following data structures:

- **open_set**: a priority queue keyed by $f(s)$ implemented via `heapdict`, which supports efficient key updates.
- **closed_set**: a hash set of states that have been expanded and for which the optimal g-value has been determined.
- **parent**: a hash map storing the best predecessor of each discovered state for path reconstruction.
- **g-score**: a hash map storing the current best-known path cost to each discovered state.
- **last_dir**: a hash map storing the incoming motion direction for each state.
- **dist_map**: a map giving the distance from any free cell to the nearest obstacle.
- **safety_buffer**: states with $d(s) \le$ `safety_buffer` are treated as non-traversable.

The **improved A* algorithm** works as follow:

1. **Initialization:** compute the **dist_map** on the free-space mask; set $g(s_s) = 0$ and `last_dir`$[s_s]$ = None; compute $f(s_s) = h(s_s)$ and insert s_s into `open_set` keyed by f.

2. **Main loop:** while `open_set` is not empty:
   (a) Pop the state s with minimal $f(s)$ from `open_set`.
   (b) If $s = s_g$, terminate and reconstruct the path by following `parent` backwards from s_g to s_s.
   (c) Insert s into `closed_set`.
   (d) For each neighbor $s' \in \mathcal{N}_8(s)$ that is in bounds and not an obstacle:
      i. If $d(s') \le$ `safety_buffer`, skip s' (treated as non-traversable).
      ii. Let $\text{cost}_{\text{step}} = 1$ for axis-aligned moves and $\sqrt{2}$ for diagonals.
      iii. Let $\text{cost}_{\text{safety}} = \frac{w_{\text{obs}}}{d(s')+1}$.
      iv. Let $\text{cost}_{\text{turn}} = w_{\text{turn}}$ if the direction from s to s' differs from `last_dir`$[s]$, else 0.
      v. Compute $\tilde{g} = g(s) + \text{cost}_{\text{step}} + \text{cost}_{\text{safety}} + \text{cost}_{\text{turn}}$.
      vi. If $s' \in$ `closed_set` and $\tilde{g} \ge g(s')$, continue.
      vii. If s' is unseen or $\tilde{g} < g(s')$, set $g(s') \leftarrow \tilde{g}$, $f(s') \leftarrow g(s') + h(s')$, `parent`$[s'] \leftarrow s$, `last_dir`$[s'] \leftarrow \text{dir}(s \to s')$, and insert/update s' in `open_set` with priority $f(s')$.
3. **Failure:** if `open_set` becomes empty before reaching s_g, report that no feasible path exists.

C. Result Analysis

Figure 2 shows a representative path produced by **improved A* algorithm** from the given start to the goal on the provided map. The red polyline is the planned path, blue and red crosses denote goal and start, and black dots indicate obstacles. We set $w_{\text{obs}} = 3$, $w_{\text{turn}} = 0.5$ and `safety_buffer` $= 3.0$.

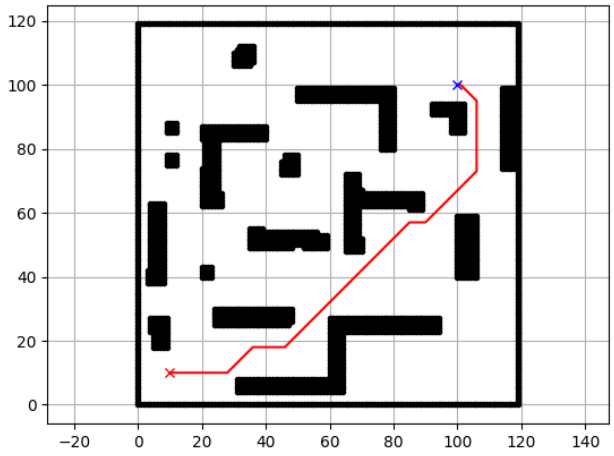


FIG. 2: A* path on the 120×120 map (Task 2).

In terms of **computational time**, the computational time of the **improved A* algorithm** is 184.21 ms, which is about three times of the **basic A* algorithm**.

In terms of **safety**, with the introduction of w_{obs} (close-to-obstacle penalty) and w_{turn} (turning penalty), the **improved A* algorithm** encourages paths with fewer direction changes and larger clearance from obstacles. In our run, the final path contains only **7 turns**. Beyond the soft penalty, we also enforce a *hard* safety buffer by setting `safety_buffer` $=$ **3.0** cells, i.e., any state with $d(s) \le 3.0$ is treated as non-traversable. Compared to the **basic A* algorithm**, this strengthens safety by both discouraging near-obstacle traversal and forbidding it within the buffer.

In terms of **optimality**, using the octile heuristic on a 8-connected grid is both *admissible* and *consistent*, and penalties are added only to g, not to h. Therefore, the first time the goal is popped from `open_set`, the

path is optimal with respect to the augmented cost($\text{cost}_{\text{step}}$ + $\text{cost}_{\text{safety}}$ + $\text{cost}_{\text{turn}}$). Thus, both **basic A* algorithm** and **improved A* algorithm** are optimal.

III. TASK 3: PATH PLANNING FOR SELF-DRIVING

A. Problem Formulation

We consider a 120×120 occupancy grid map where each grid cell represents a $1.0\,\mathrm{m} \times 1.0\,\mathrm{m}$ block. The robot state is continuous and non-holonomic: $s = (x, y, \theta)$, where $\theta \in [0, 2\pi)$ is the heading. Obstacles are encoded by `world_map` $\in \{0,1\}^{120\times120}$ where 1 indicates an occupied cell (non-traversable) and 0 indicates a free cell.

For a real-world car model, we assume a minimum turning radius $R = 5.0$ and fixed step length $\Delta s = 1.0$. We use three motion primitives per expansion: straight, left-arc, and right-arc. Their continuous kinematics are

$$\text{straight:} \quad x' = x + \Delta s \cos\theta, \quad y' = y + \Delta s \sin\theta, \quad \theta' = \theta, \tag{1}$$

$$\text{left-arc:} \quad \Delta\theta = \frac{\Delta s}{R},\ \theta' = \theta + \Delta\theta, \tag{2}$$

$$x' = x + R\big(\sin\theta' - \sin\theta\big), \quad y' = y - R\big(\cos\theta' - \cos\theta\big), \tag{3}$$

$$\text{right-arc:} \quad \Delta\theta = \frac{\Delta s}{R},\ \theta' = \theta - \Delta\theta, \tag{4}$$

$$x' = x + R\big(\sin\theta' - \sin\theta\big), \quad y' = y - R\big(\cos\theta' - \cos\theta\big). \tag{5}$$

For convenience, we will still consider the car as a point mass, disregarding its collision area.

B. Method

Let $d(s)$ denote the Euclidean distance (in cells) from s's position to the nearest obstacle. We enforce a *hard* safety buffer `safety_buffer` > 0: any state with $d(s) \le$ `safety_buffer` is treated as non-traversable. The path cost augments the geometric step length with two soft penalties (close-to-obstacle and turning), yielding

$$g(s') \leftarrow g(s) + \underbrace{\Delta s}_{\text{geometric}} + \underbrace{\frac{w_{\text{obs}}}{d(s')+1}}_{\text{safety penalty}} + \underbrace{w_{\text{turn}}\,\mathbf{1}\big[\,\text{dir}(s \to s') \neq \text{dir}(\text{parent}(s) \to s)\,\big]}_{\text{turning penalty}}, \tag{6}$$

subject to the hard constraint $d(s')$, $d(\text{samples on } s \to s') >$ `safety_buffer`. The indicator is 1 if the incoming direction changes.

The heuristic h is the **Dubins shortest-path length** (inplemented by the library `easydubins`) from the current pose to the goal pose under radius R:

$$h(s) = \begin{cases} L_{\text{Dubins}}\big(s, s_g; R\big), & \text{if Dubins solves} \\ \|(x,y) - (x_g, y_g)\|_2, & \text{otherwise.} \end{cases} \tag{7}$$

The A* evaluation fuction is

$$f(s) = g(s) + h(s).$$

We implement the **Hybrid A* algorithm** with the following data structures:

- **open_set**: priority queue (`heapdict`) keyed by f.
- **closed_set**: hash set of closed keys (i_x, i_y, i_θ).
- **parents**: map $(i_x, i_y, i_\theta) \mapsto (\text{state}, \text{parent\_key})$ for reconstruction.
- **g-score**: a hash map storing the current best-known path cost to each discovered state.

- **state_map**: map from key to continuous state (x, y, θ).
- **last_dir**: map storing the incoming direction vector for turning penalty.
- **dist_map**: a map giving the distance from any free cell to the nearest obstacle.
- **safety_buffer**: states with $d(s) \leq$ `safety_buffer` are treated as non-traversable.

The **Hybrid A* algorithm** works as follow:

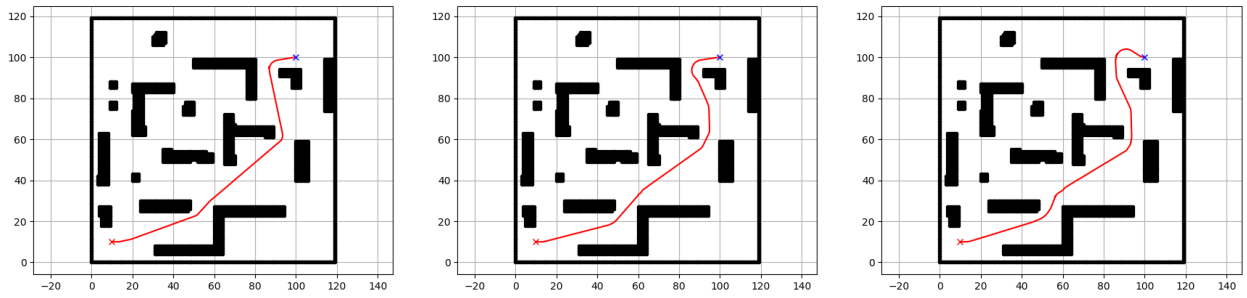
1. **Initialization:** compute the **dist_map** on the free-space mask; set $g(s_s) = 0$ and `last_dir`$[s_s]$ = None; compute $f(s_s) = h(s_s)$ and insert s_s into `open_set` keyed by f.
2. **Main loop:** while `open_set` is not empty:
   (a) Pop the state s with minimal $f(s)$ from `open_set`.
   (b) If s is within goal tolerance, terminate and reconstruct the path by following `parent` backwards from s to s_s.
   (c) Insert the closed key of s into `closed_set`.
   (d) If near goal (distance threshold) or every K expansions, attempt an analytic connection via `easydubins.get_curve`; accept and terminate if the entire curve passes occupancy and hard-clearance checks (all samples satisfy in-bounds, free, and $d >$ `safety_buffer`); then backtrack parents and return the concatenated path.
   (e) For each motion-primitive successor $s' \in \{\text{straight, left-arc, right-arc}\}$ derived from the kinematics:
      i. Linearly sample between (x, y) and (x', y') with N samples; if any sample is out-of-bounds, in obstacle, or violates hard clearance $d \leq$ `safety_buffer`, skip s'.
      ii. Let $\text{cost}_{\text{step}} = \Delta s$, $\text{cost}_{\text{safety}} = \frac{w_{\text{obs}}}{d(s')+1}$, and $\text{cost}_{\text{turn}} = w_{\text{turn}}$ if the incoming direction differs from `last_dir`$[s]$, else 0.
      iii. Compute $\tilde{g} = g(s) + \text{cost}_{\text{step}} + \text{cost}_{\text{safety}} + \text{cost}_{\text{turn}}$.
      iv. If the closed key of s' is in `closed_set` and $\tilde{g} \geq g(s')$, continue.
      v. Set $h(s')$ to the Dubins shortest-path length $L_{\text{Dubins}}(s', s_g; R)$ via `easydubins.dubin_path`; if Dubins fails, fallback to Euclidean distance.
      vi. If s' is unseen or $\tilde{g} < g(s')$, set $g(s') \leftarrow \tilde{g}$, $f(s') \leftarrow g(s') + h(s')$, `parent`$[s'] \leftarrow s$, `last_dir`$[s'] \leftarrow \text{dir}(s \to s')$, update `state_map` for s', and insert/update s' in `open_set` with priority $f(s')$.
3. **Failure:** if `open_set` becomes empty before reaching the goal, report that no feasible path exists.
4. **Post-processing.** We optionally apply *Dubins shortcut smoothing*: repeatedly select indices $i < j$ (with a minimal gap), attempt a Dubins connection between path points p_i and p_j, and replace the middle if (i) collision-free under hard clearance and (ii) reduces total length.

C. Result Analysis

We assume that the heading degree of both start and goal are 0°.

We conduct controlled sweeps for three key parameters: `safety_buffer`, w_{obs}, and w_{turn}. Unless otherwise stated, other settings are fixed to the practical configuration above (minimum turning radius $R = 5.0$, step $\Delta s = 1.0$, theta bins $N_\theta = 72$, analytic period $K = 50$, samples per step $N = 8$). Each row shows three runs placed side-by-side for visual comparison.

Sweep 1: `safety_buffer`. Larger buffers increase clearance but may lengthen paths or block narrow corridors. We compare 3.0, 3.5, and 4.0 cells.

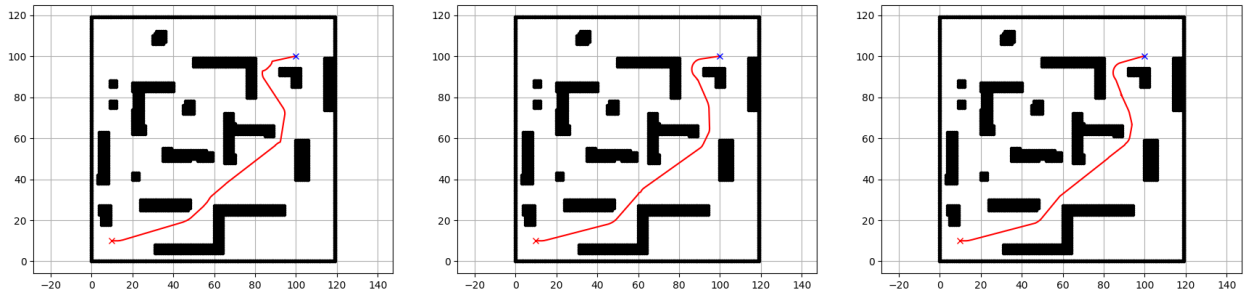


safety_buffer=3.0 safety_buffer=3.5 safety_buffer=4.0

FIG. 3: Effect of safety buffer on path shape and clearance.

Observation. As `safety_buffer` grows, minimum clearance increases and the path arches farther from obstacles; overly large values can make narrow passages infeasible.

Sweep 2: $w_{\mathbf{obs}}$. This weight pulls the path away from obstacles via the soft penalty. We compare 1.0, 3.0, and 5.0.

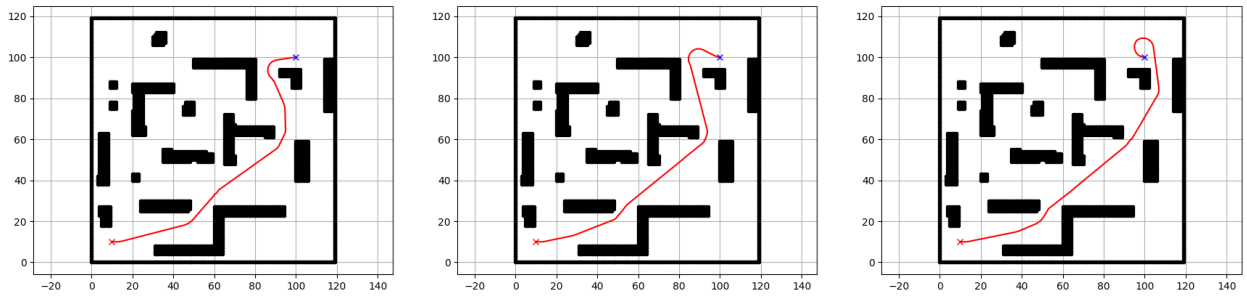


$w_{\text{obs}} = 1.0$ $w_{\text{obs}} = 3.0$ $w_{\text{obs}} = 5.0$

FIG. 4: Effect of obstacle penalty weight on clearance preference.

Observation. Larger w_{obs} promotes wider berth and fewer near-obstacle segments, at the expense of longer, sometimes more circuitous paths. Very high values can overshadow geometry and cause detours.

Sweep 3: $w_{\mathbf{turn}}$. This weight regularizes heading changes to suppress zig-zag. We compare 0.5, 1.5, and 3.0.



$w_{\text{turn}} = 0.5$ $w_{\text{turn}} = 1.5$ $w_{\text{turn}} = 3.0$

FIG. 5: Effect of turning penalty weight on path smoothness.

Observation. Increasing w_{turn} reduces small heading oscillations and produces smoother arcs; too large values may force wide turns and lengthen the route.

Summary. A balanced setting for this map is `safety_buffer` ≈ 3.5, $w_{\text{obs}} \approx 3.0$, $w_{\text{turn}} \approx 0.5$ in Figure 6, consistent with the recommendations in the tuning section. These achieve reliable feasibility and smoothness without excessive detours. Compared with **Basic A* algorithm** and **Improved A* algorithm**, the turns are noticeably smoother, making the planner more suitable for real-world applications.

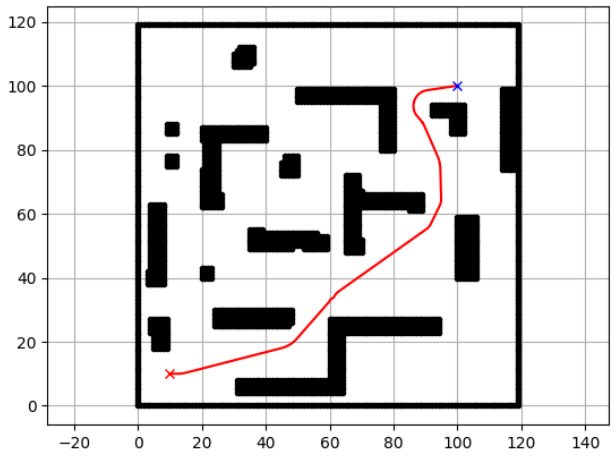


FIG. 6: A* path on the 120×120 map (Task 3).