Search based Motion Planning Algorithm to intercept a moving target

Sanchit Gupta
Department of Electrical and
Computer Engineering
University of California San Diego
sag006@ucsd.edu

***Abstract*—This report discusses about the methodology to design and implement a search-based motion planning algorithm for the robot to catch a moving target in any environment comprised of obstacles. The A* algorithm, a modification to Label Correcting algorithm, has been implemented in this project. The results of optimal path taken by the robot to catch the target in all the given environments are presented in this report.**

Keywords—Motion Planning algorithm, Label Correcting algorithm, A* algorithm, Optimal Path

I. INTRODUCTION

Motion planning is an integral function for the autonomous navigation of the robot. It is essential to enable the robot to navigate autonomously from a given location to the target location in the shortest path possible in any environment, while avoiding the obstacles as present in the path. Given a map of an environment, the motion planning algorithms comprise of determining the paths to the target from all possible locations using different control inputs and policies. It is further responsible for choosing the shortest path from all the paths possible on the basis of cumulative costs determined for each of the paths. While motion planning is essential in all kinds of environments, it becomes necessary where the maps are denser and comprised of multiple obstacles. Further, implementing a motion planning algorithm becomes challenging where the planning has to be executed in real-time on the basis of a dynamically changing target.

Dynamic Programming algorithms are popularly used to solve the motion planning problems. These algorithms run backward in time, from the target location to the start position, and determine the paths available from each node to the target location along-with costs associated with each of them. This framework allows the robot to decide the shortest path with the least associated cost. Label Correcting algorithm is one such algorithm in Dynamic Programming which runs forward in time. The A* algorithm is a modification to the Label Correcting Algorithm which computes the shortest path by examining the child nodes of every node, up to the target location. In A* algorithm, the requirement for a node to be an optimal node is further strengthened using heuristics for each node along-side their cost labels.

This project requires us to implement a search-based motion planning algorithm to determine shortest path for the robot from start location to target location in all the given environments. The target position changes dynamically and the robot is required to catch the target following an optimal path while avoiding the obstacles. The environment size varies from a grid of 6 x 4 to a grid of 5000 x 5000. In order to design the motion planning function, the project is formulated as a Deterministic Shortest Path (DSP) problem. The A* algorithm is implemented in forward time to determine the optimal path while planning on the fly to catch the target in shortest path possible. The algorithm computes the next move for the robot within 2 seconds and ensures that the obstacles in the path are avoided.

II. PROBLEM FORMULATION

This section will discuss about the formulation of the shortest path problem and its implementation in terms of mathematical formulas. The following details are provided in the project:

- Different environment maps of obstacles (*denoted as envmap*) are provided. Each map is a 2D grid with size varying from 6 x 4 to 5000 x 5000.
- The intial 2D position of the robot (*robotpos*) and intial 2D position of the target (*targetpos*) are given for each map.
- envmap is a matrix consisting of information about free and occupied spaces. For a free cell, envmap(x,y) = 0 and for an occupied cell, envmap(x,y) = 1.
- The next position of the robot can be any cell amongst the 8 adjacent cells to its current position. This means that the robot can move only one step, either in straight directions (*Up, Down, Left, Right*) or in any of the diagonal directions.
- The robot cannot move to any of the occupied cells or to any cell outside the boundary of the map.
- The target position changes dynamically according to the target planner provided. It tries to maximize the minimal distance achieved by the robot using minimax decision rule.

- The robot planner is supposed to provide the next move within 2 seconds. Within these 2 seconds, the target makes a single move in any of the four straight directions. If the robot planner takes more than 2 seconds, then the target will take N steps for every 2N seconds taken by the robot planner.

The objective of the project is to execute a common robot planner using a search-based motion planning algorithm to catch the dynamically moving target in the shortest possible path.

The given problem is deterministic in nature as there is no randomness defined in the movement of the robot and the target. Thus, the problem can be formulated as a Deterministic Shortest Path (DSP) problem. In order to formulate the same, let's consider the environment maps as a graph with details as shown in equation 1.

$$envmap = \begin{Bmatrix} Graph\ with\ finite\ vertex\ set\ v \\ Edge\ set = \varepsilon \epsilon v\ x\ v \\ Edge\ weights = C = \{c_{ij} \epsilon R \mid (i,j) \epsilon \varepsilon\} \\ start\ node = initial\ robotpos = s \epsilon R^2 \epsilon v \\ End\ node = targetpos = \tau \epsilon R^2 \epsilon v \end{Bmatrix} \quad (1)$$

Here, c_{ij} is the cost of going from vertex i to vertex j and the target position τ changes with time. As the robot can move only one step either in straight direction or in diagonal direction, the vertex i in our problem resembles to robot's position and vertex j resembles to any adjacent node.
So, the DSP problem now translates as follows.

- Path = a sequence $i_{1:q} = (i_1, i_2, \dots., i_q)$ of nodes $i_k \epsilon v$ with $i_1 = s\ and\ i_q = \tau$
- Path length = sum of edge weights along the path = $\sum_{k=1}^{q-1} C_{i_k, i_{k+1}}$
- Objective = find a path with minimum path length from node s to node τ.
- Assumption = no negative cost cycles in the graph.

Thus, for the given problem and environment maps, the state space X, control space U, motion model p_f and finite time horizon T can be formulated as shown in equation 2. The state space and control space are discreet in our problem.

$$State\ Space\ X = v$$

$$U = \begin{Bmatrix} Up, Down, Left, Right, \\ Diagonal\ moves\ in\ all\ 4\ directions \end{Bmatrix} \epsilon v$$

$$p_f = f((x,y)_t, u_t) = \begin{Bmatrix} (x,y)_t\ if\ node = \tau \\ u_t, \quad otherwise \end{Bmatrix} \quad (2)$$

$$T = |v| - 1$$

$$Discount\ factor\ \gamma = 1\ as\ it\ is\ finite\ time\ horizon$$

According to the given problem, the terminal condition is given in equation 3.

$$Terminal\ condition = |x_t - \tau_x| \leq 1\ and\ |y_t - \tau_y| \leq 1 \quad (3)$$

where, (x_t, y_t) is the position of the robot at time t.

Cost Definition: As each step has a cost associated with it, the edge costs for a move to any straight adjacent node is defined as 1 and the edge costs for a move to any adjacent diagonal node is defined as $\sqrt{2}$. It is elaborated in equation 4.

$$c_{ij} = \begin{Bmatrix} 1\ if\ u_t \epsilon \{(0,1), (0,-1), (-1,0), (1,0)\} \\ \sqrt{2}\ if\ if\ u_t \epsilon \{(1,1), (1,-1), (-1,1), (-1,-1)\} \\ \infty\ if\ vertex\ j\ is\ an\ obstacle\ or\ map\ boundary \end{Bmatrix} \quad (4)$$

According to this, the stage cost $l(x,u)$ and terminal costs $q(x)$ are given in equation 5.

$$l(x,u) = c_{ij}\ for\ nodes\ i\ and\ j\ \epsilon\ v$$
$$q(x) = \begin{Bmatrix} 0\ if\ (x_t, y_t) = \tau \\ \infty\ otherwise \end{Bmatrix} \quad (5)$$

According to the terminal condition defined in equation 3, the robot stops one step away from the target and hence, the stage cost will remain as shown in equation 5 even when the robot catches the target.

In order to solve this shortest path problem, the objective is to minimize the total edge cost between the start node s and terminal node τ.

$$dist(s,\tau) = \min_{i_{1:q}} \sum_{k=1}^{q-1} C_{i_k, i_{k+1}}\ \ where, i_1 = s\ and\ i_q = \tau$$

Using this formulation of the Deterministic Shortest Path problem, the implementation of a search-based motion planning algorithm to solve this problem is discussed in next section of the report.

III. TECHNICAL APPROACH

The technical approach to solve the deterministic shortest path problem as formulated in Section II, in order to intercept a moving target is discussed in detail in this section and is elaborated in different sub-sections below.

A. Formulation of Label Correcting Algorithm

Label Correcting (LC) algorithm is the best suit to solve the DSP problem formulated in section II. The next move for the robot is to be calculated in the minimum time possible, preferably within 2 seconds, thus, LC algorithm suits the requirement as it does not visit every node of the graph. Key ideas for LC algorithm are:

- Label g_i is assigned to each node of graph, which is an estimate of optimal cost from start node s to each visited node $i \epsilon v$
- Adjacent nodes of the robot's current position are considered as child nodes. The labels of the child nodes are corrected as $g_j = g_i + c_{ij}$, where g_i is the label of robot's current node and c_{ij} is the cost of going from node i to node j.
- Next node for the robot to move is chosen as the one with the smallest label amongst all its child nodes.
- The algorithm updates the labels of relevant nodes until:

$$g_i = \min_{j \epsilon Parents(i)} g_j + c_{j,i}$$

- An OPEN list is maintained which is a set of nodes that can potentially be a part of shortest path to target node τ.
- A CLOSED list is maintained which is a set of nodes that have been expanded.

LC algorithm terminates when the OPEN list is empty and the target node has been expanded. After termination:

- The value of label of each node g_i corresponds to the actual path cost of reaching node i from start node s.
- Backtracking all the nodes in CLOSED list from target node τ to start node s gives the required shortest path.

$$i^*_{k+1} = \underset{j \epsilon Parents(i^*_k)}{\operatorname{argmin}} g_j + c_{j,i^*_k} \ until \ i^*_{k+1} = s$$

B. Formulation of A* Algorithm

The A* algorithm is a modification to the LC algorithm which strengthens the requirement of adding a node to OPEN list as follows:

$$\boldsymbol{from}\ g_i + c_{ij} < g_\tau \ in \ LC \ \ \boldsymbol{to}\ \ g_i + c_{ij} + h_j < g_\tau$$

where, g_i is the label of node i, g_τ is the label of target node τ and h_j is the heuristic function.

h_j is the positive lower bound on the optimal cost from any node i to target node τ, such that:

$$0 \leq h_j \leq dist(i, \tau)$$

Inclusion of heuristic function strengthens the requirement of adding a node to the OPEN list and helps in further reducing the number of nodes visited by LC algorithm. Hence, A* algorithm has been finalized and has been implemented in all the environment maps of this project to intercept the moving target.

The other key ideas in A* algorithm remain the same as that of LC algorithm, as described in sub-section A.

C. Initialization of Parameters for A* Algorithm

- OPEN list is initialized as an empty list.
- CLOSED list is initialized as an empty list.
- Labels matrix is initialized as given in equation 6. It is of the same dimensions as that of environment map.

$$g_i(at\ t = 0) = \begin{Bmatrix} \infty \ for \ all \ vertex \ i \ \epsilon \ v \\ 0 \ for \ vertex \ i = s \ (start \ node) \end{Bmatrix} \quad (6)$$

- Heuristics matrix (h_j) is initialized as a zero matrix of the same dimensions as that of the environment map.
- The 8 direction vectors to represent the adjacent nodes of the robot at any position are initialized as shown in equation 7.

$$\underset{vectors}{direction} = \begin{Bmatrix} (0,1) \ represents \ top \ node \\ (0,-1) \ represents \ bottom \ node \\ (-1,0) \ represents \ left \ node \\ (1,0) \ represents \ right \ node \\ (-1,1) \ represents \ top - left \ node \\ (1,1) \ represents \ top - right \ node \\ (1,-1) \ represents \ bottom - right \ node \\ (-1,-1) \ represents \ bottom - left \ node \end{Bmatrix} \quad (7)$$

- Cost (c_{ij}) of moving to any adjacent node is defined as given in equation 4.

$$c_{ij} = \begin{Bmatrix} 1 \ if \ dir \ \epsilon \{(0,1),(0,-1),(-1,0),(1,0)\} \\ \sqrt{2} \ if \ if \ dir \ \epsilon \{(1,1),(1,-1),(-1,1),(-1,-1)\} \\ \infty \ if \ vertex \ j \ is \ an \ obstacle \ or \ map \ boundary \end{Bmatrix}$$

D. Definition of heuristics

The heuristic function (h_j) is an under-estimate of the cost of the shortest path from node $j \epsilon v$ to the target node τ. Thus, the Euclidean distance of every node $j \epsilon v$ to the target node τ works as a good heuristic function. Therefore, the heuristic function for this project is defined as shown in equation 8.

$$h_j = \left|\left|x_\tau - x_j\right|\right|_2 \quad (8)$$

where, $x_\tau \epsilon v$ is the 2D co-ordinate of the target position τ and $x_j \epsilon v$ is the 2D co-ordinate of any node j.

- This heuristic function is admissible as $h_j \leq actual - dist(j, \tau) \ for \ all \ j \epsilon v$.
- This heuristic function is consistent as $h_\tau = 0$ and $h_j \leq c_{ji} + h_i \ for \ all \ j \neq \tau \ and \ i \epsilon \ Children(j)$.
- Thus, the heuristic function defined in equation 8 satisfies the triangle inequality.

In order to decrease computation and make the algorithm faster, the heuristics is only calculated for the child nodes of

the robot's position, rather than calculating for all nodes in the environment map.

E. Function to obtain child nodes of any node

A function 'getChildNodes()' is written, which returns the child nodes of the current position of the robot at any time. The function performs the following steps:

- It calculates the nodes adjacent to the position of the robot in all 8 directions using the direction vectors defined in equation 7.
- It checks if the node calculated is within the boundaries of the environment map. For any node i, $children(i)\ should\ belong\ to\ v$. The child node is discarded if it is outside the boundaries of the map.
- It further checks if any obstacle is present at the child node calculated. For any child node j, if $envmap(j) = 1$, then the node is discarded.
- The valid child nodes are returned to the main robot planner function.

F. Process to update labels of child nodes

Once the child nodes for the robot's position are calculated and returned to the main robot planner function, they are sent to the 'updateLabel()' function for their labels (g_j) to be updated. The labels are updated as follows:

$$for\ j \epsilon\ children\ (i)\ and\ j\ \notin \text{CLOSED list, then}$$
$$check\ if\ g_j > g_i + c_{ij}\ and\ if\ it\ is\ true, then$$
$$g_j = g_i + c_{ij}$$

Here, g_j = Original label of $j \epsilon\ children\ (i)$ – before the update, g_i = label of parent node or the robot's position and c_{ij} is as per defined in equation 4, basis the direction of child node in either straight direction or diagonal direction.

After the labels of the child nodes are updated, they are added to the OPEN list, if they are already not present in it.

G. Process to determine the next node to expand

To determine the new position for the robot, a function 'nodeToExpand()' is written. It performs the following functions:

- For every node $i \epsilon\ OPEN\ list$, the value of $f_i = g_i + h_i$ is compared and the node with the minimum f_i value is chosen.
- If there are multiple nodes with minimum f_i value, then their g_i value is compared. The node with the minimum g_i value is chosen.
- In-case, if the nodes have same g_i value as well, then any of such node is chosen randomly.

The node chosen through this process becomes the next node to expand. This chosen node is removed from the OPEN list and added to the CLOSED list.

H. Strategy to intercept moving target

- The A* algorithm in one iteration runs till the target node τ does not enter the CLOSED list and is not expanded. The loop terminates after this condition is met and it backtracks from target node τ to the start node s to determine the shortest path.
- The new node for the robot to move is chosen from this shortest path determined. The second node after the start node s from this path becomes the new position for the robot to move.
- In the next call to the A* algorithm, it checks if the target position has changed from the previous call. If yes, then the whole A* algorithm runs again to find the shortest path from new robot position to the new target position. OPEN list, CLOSED list, Labels matrix and Heuristics matrix are re-initialized. Accordingly, the next position for the robot to move is determined.
- This whole process is executed within 2 seconds to ensure the target has not moved far away from the robot.
- The process continues till the terminal condition defined in equation 3 is met.

The pseudo code shown in Figure 1 explains a single iteration of A* algorithm implemented.

I. Additional conditions for Large Maps

- For maps smaller than 500 x 500 in size, the A* algorithm takes than 2 seconds to determine the shortest path.
- For maps larger than 500 x 500, the A* algorithm takes much more than 2 seconds, which can cause the robot to move much farther away from the robot. In such cases:
    - The A* algorithm is terminated once certain number of nodes, say 20,000, are expanded. The shortest route till ε distance from the start node s is determined from these expanded nodes.
    - The nodes from this ε – route is used for another 'n' number of iterations, say 50.
    - This helps the robot to reach closer to the target, after which the A* algorithm takes less than 2 seconds to determine the optimal shortest path.

Algorithm 2 Weighted A* Algorithm

OPEN $\leftarrow \{s\}$, CLOSED $\leftarrow \{\}$, $\epsilon \geq 1$
$g_s = 0$, $g_i = \infty$ for all $i \in \mathcal{V} \setminus \{s\}$
while $\tau \notin$ CLOSED **do** ▷ τ not expanded yet
  Remove i with smallest $f_i := g_i + \epsilon h_i$ from OPEN ▷ means $g_i + \epsilon h_i < g_\tau$
  Insert i into CLOSED
  for $j \in$ Children(i) and $j \notin$ CLOSED **do**
    if $g_j > (g_i + c_{ij})$ **then**
      $g_j \leftarrow (g_i + c_{ij})$
      Parent(j) $\leftarrow i$
      if $j \in$ OPEN **then**
        Update priority of j
      else
        OPEN $\leftarrow$ OPEN $\cup\{j\}$

expand state i:
○ try to decrease g_j using path from s to i

Figure 1: Algorithm implemented for A*

IV. Results

This section discusses about the results as obtained for all the 11 environment maps. For all these environments, the details about the following parameters are tabulated in Table 1:

- Time taken by the A* algorithm to find the target position in the very first iteration
- Number of moves made by the robot to catch the target
- Number of nodes expanded by the A* algorithm in the very first iteration

Table 1: Results of A* Algorithm on different environment maps

Map	Map Size	Time to find target in 1st iteration	Number of moves to catch the target	Number of nodes expanded in 1st iteration
0	6 x 4	0.016 sec	5	10
1	1825 x 2332	1.187 sec	1288	901
2	8 x 10	0.016 sec	13	46
3	473 x 436	18.71 sec	499	10496
4	6 x 7	0.01 sec	9	18
5	42 x 70	0.59 sec	107	1421
6	35 x 37	0.36 sec	41	860
7	5000 x 5000	Did not work completely		
1b	1825 x 2332	-	5891	-
3b	473 x 436	108.2 sec	837	28018
3c	473 x 436	525.58 sec	1363	61300

Important points observed during these tests are listed below:

- The time taken to find the target node decreases in further iterations of A*, as the robot moves closer to the target node. Hence, the time taken in the very first iteration is reported here, since it is the maximum time for any iteration.
- Similarly, the number of nodes expanded to find the target node also decreases in the further A* iterations. As it would be maximum in the first iteration, thus they are reported here.

The plots between the labels g_i and the nodes i after 1st iteration of A* algorithm for two environment maps, Map 5 and 6 are shown in figure 2 and 3 respectively.

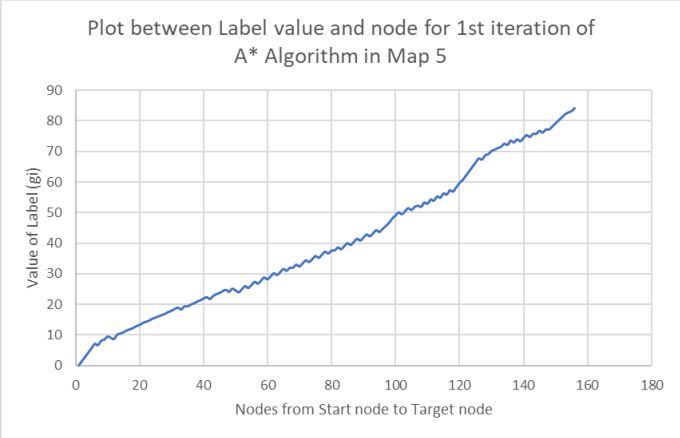


Figure 2: Plot between the label value g_i and nodes for Map 5

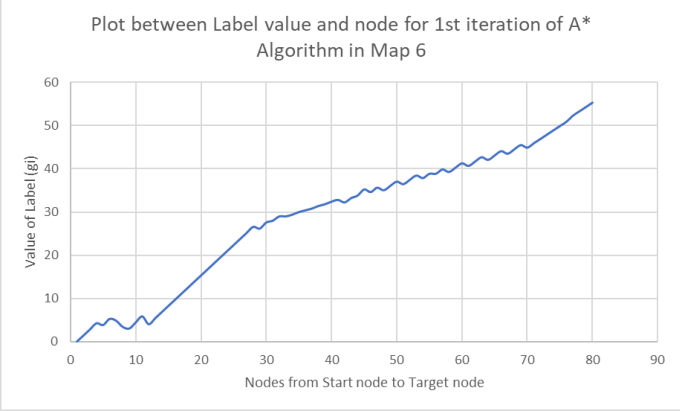
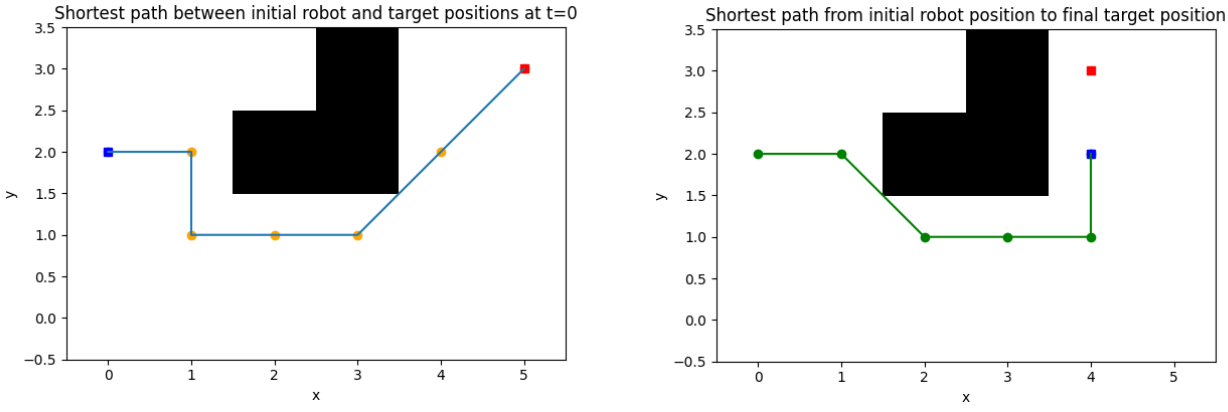


Figure 3: Plot between the label value g_i and nodes for Map 6

Figures 2 and 3 show that the values of labels g_i increase monotonically for the shortest path between start node s and target node τ found by the A* algorithm.

The shortest path between the initial robot and target positions and the final path taken by the robot to catch the target for all the environment maps are shown in figures 4 to 13.

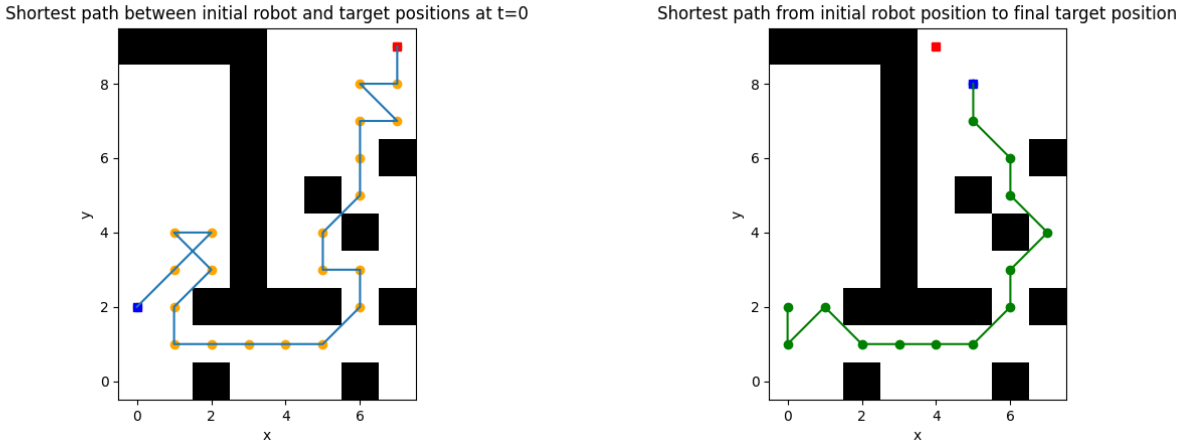


(a) (b)

Figure 4: (a) Shortest path between initial robot and target positions at t = 0 as found by A* algorithm for Map 0, (b) Shortest path taken by the robot to catch the target as found by A* algorithm for Map 0

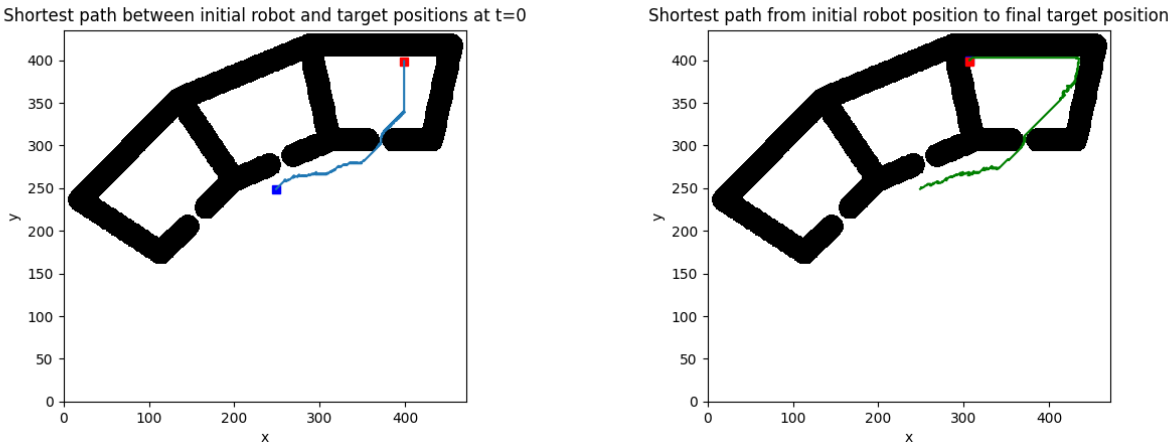
(a) (b)

Figure 5: (a) Shortest path between initial robot and target positions at t = 0 as found by A* algorithm for Map 1, (b) Shortest path taken by the robot to catch the target as found by A* algorithm for Map 1



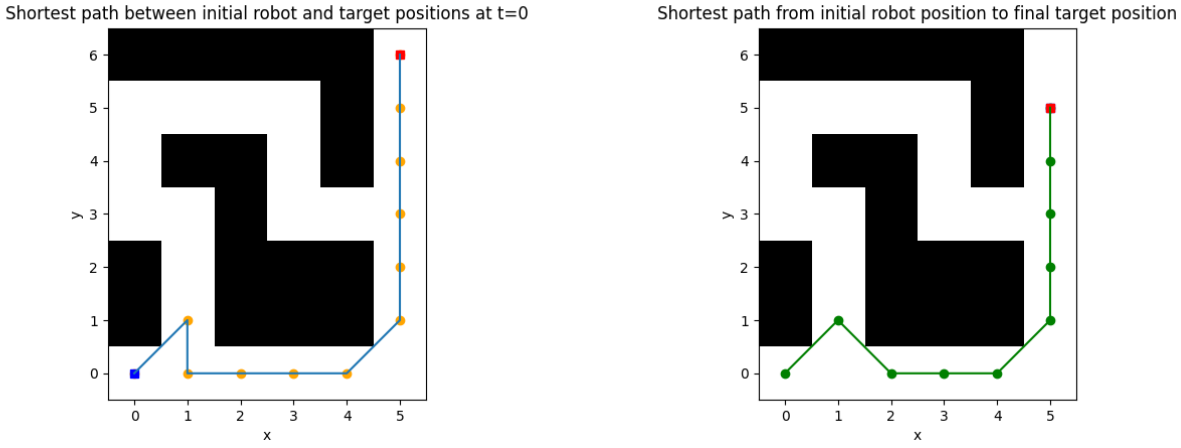
(a) (b)

Figure 6: (a) Shortest path between initial robot and target positions at t = 0 as found by A* algorithm for Map 2, (b) Shortest path taken by the robot to catch the target as found by A* algorithm for Map 2



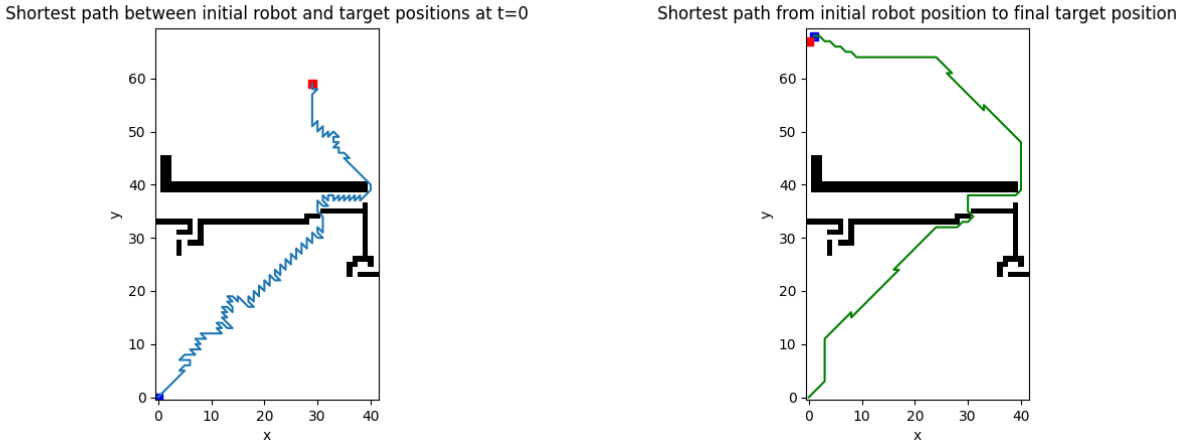
(a) (b)

Figure 7: (a) Shortest path between initial robot and target positions at t = 0 as found by A* algorithm for Map 3, (b) Shortest path taken by the robot to catch the target as found by A* algorithm for Map 3



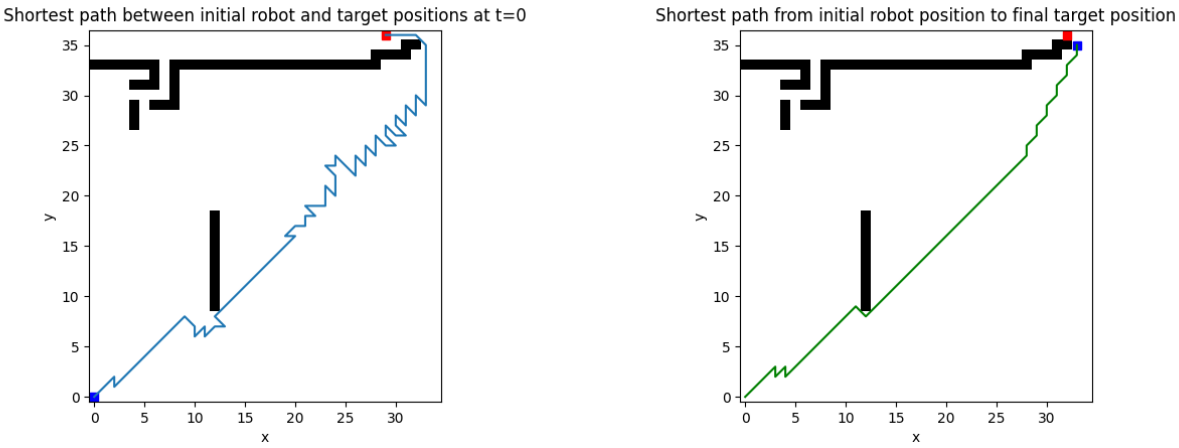
(a) (b)

Figure 8: (a) Shortest path between initial robot and target positions at t = 0 as found by A* algorithm for Map 4, (b) Shortest path taken by the robot to catch the target as found by A* algorithm for Map 4



(a) (b)

Figure 9: (a) Shortest path between initial robot and target positions at t = 0 as found by A* algorithm for Map 5, (b) Shortest path taken by the robot to catch the target as found by A* algorithm for Map 5



(a) (b)

Figure 10: (a) Shortest path between initial robot and target positions at t = 0 as found by A* algorithm for Map 6, (b) Shortest path taken by the robot to catch the target as found by A* algorithm for Map 6

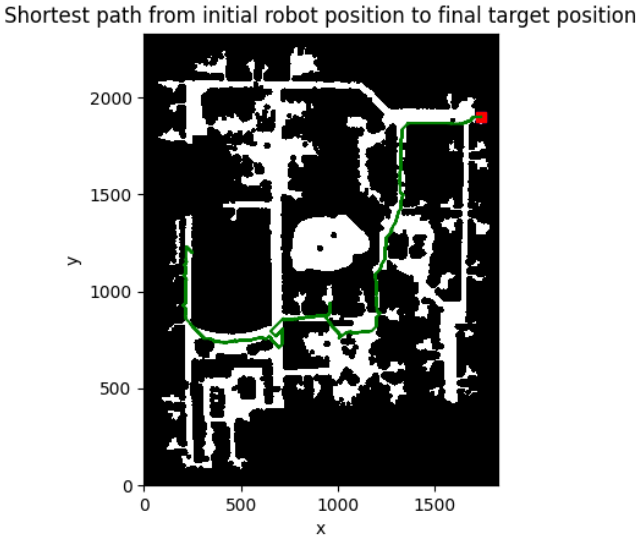
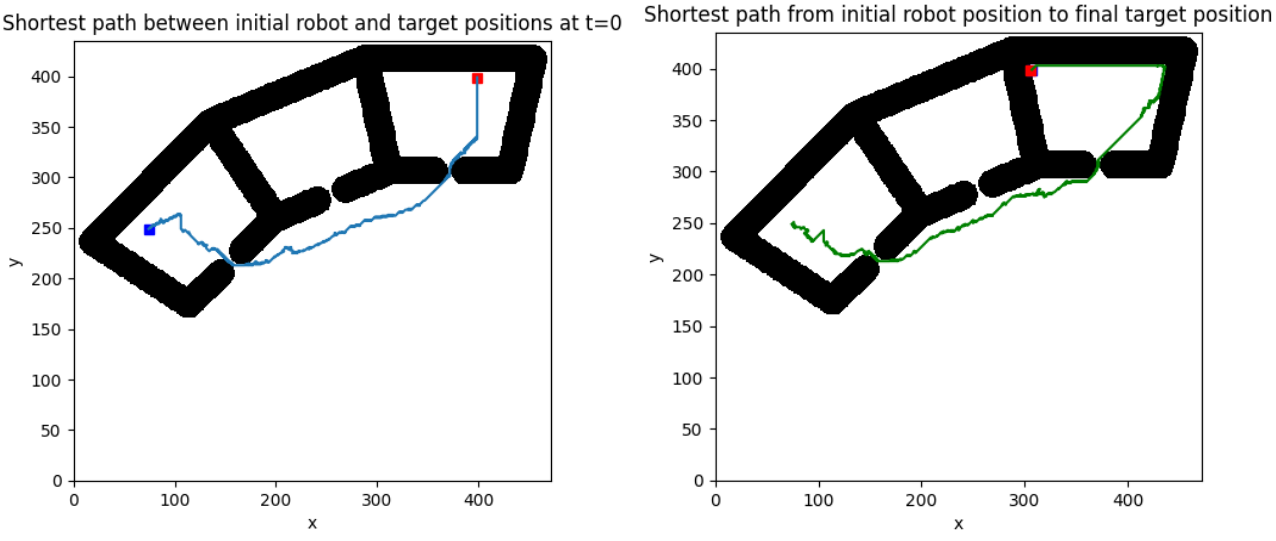
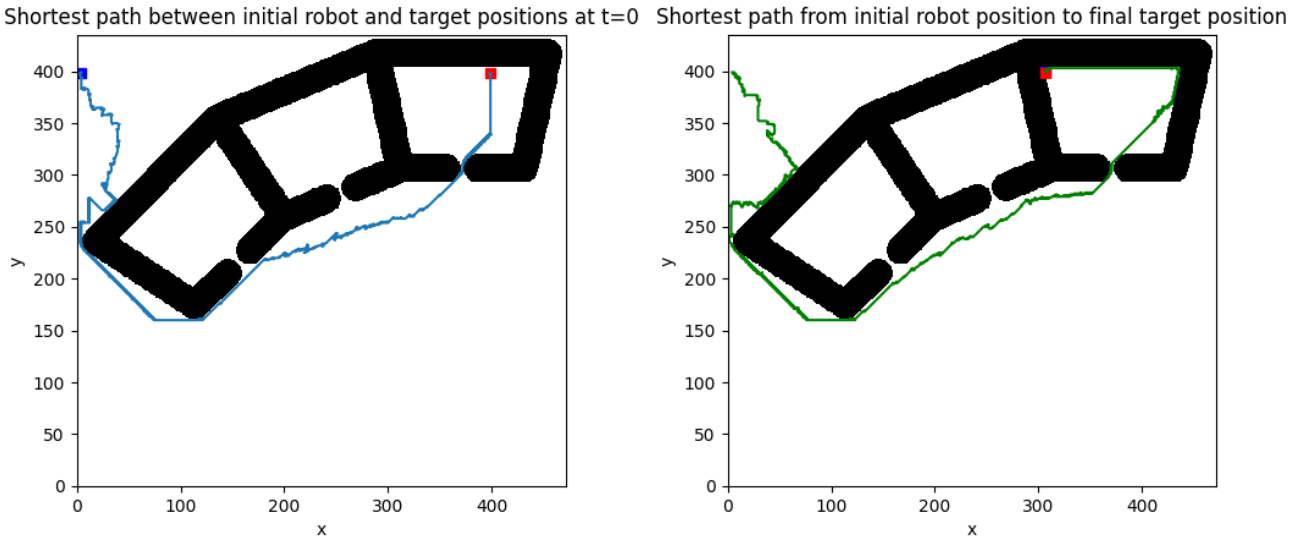


Figure 11: Shortest path taken by the robot to catch the target as found by A* algorithm for Map 1b



(a) (b)

Figure 12: (a) Shortest path between initial robot and target positions at t = 0 as found by A* algorithm for Map 3b, (b) Shortest path taken by the robot to catch the target as found by A* algorithm for Map 3b



(a) (b)

Figure 13: (a) Shortest path between initial robot and target positions at t = 0 as found by A* algorithm for Map 3c, (b) Shortest path taken by the robot to catch the target as found by A* algorithm for Map 3c

Resulting Discussion:

- The A* algorithm worked perfectly and beautifully for all the environment maps. For all the maps, it was able to find the shortest path in all the iterations. For large maps where the algorithm was taking more than 2 seconds, a partial path calculated by A* algorithm was used and for such maps, it is observed that the algorithm corrected the path as it moved closer to the target.
- As map 7 is a huge map of size 5000 x 5000, the algorithm was taking a long time to go through the nodes and find the shortest path. Even the partial path strategy with expansion of only 50,000 nodes did not work for this map. With this strategy, it seemed that the algorithm got stuck in a minima and the robot was oscillating between a few co-ordinates.
- I tried implementation of update of heuristics based on RTAA* algorithm also in the code. However, for all the maps except Map 7, the A* algorithm does not get stuck in a minima and hence, any significant difference between A* and RTAA* could not be observed. Due to time constraints, I could not run it completely in map 7.
- If I had more time, I would have focused on trying the following things:
    1. Implementation of RTAA* algorithm for map 7
    2. Implementation of sampling-based algorithm using OMPL library and compare its results with A* algorithm
    3. Efforts in reducing the time for one iteration of A* algorithm to within 2 seconds for larger maps