Autonomous Indoor Robot Navigation Using A Search with Dynamic Obstacle Avoidance

Naima Siddika Toha
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh
naima.toha@northsouth.edu

Md Saidur Rahman Antu
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh
saidur.antu@northsouth.edu

Syed Sakibul Haque Sami
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh
sakibul.sami@northsouth.edu

Saima Arafin Smrity
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh
saima.smrity@northsouth.edu

Dr. Mohammad Shifat-E-Rabbi
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh
rabbi.mohammad@northsouth.edu

***Abstract*— This project presents the design and implementation of an autonomous indoor robot navigation system in a grid- based environment. The system gives permission a robot to move to the destination is selected by the user and during this time, it will avoid obstacles. The robot uses the A* search algorithm, which provides a way of finding the shortest path in a grid- based environment. The robot's movement is controlled by physics- based motion and the path is visualized using line rendering. In order to improve user experience, a camera controller is added that automatically switches between a top-down and a follow- camera view. After testing in the environment, the robot shows that it can reach successfully to the main destination while avoiding obstacles or blocked areas. This project demonstrates how A* algorithms, obstacle detection while movement, and visualization techniques can be combined to create an indoor robot navigation system.**

***Keywords*— Autonomous navigation, A* algorithm, autonomous robot navigation, Unity, C#, AI.**

I. INTRODUCTION

In today's world, autonomous navigation is challenging in both computer science and robotics. For a robot to be useful, it needs to reach in the main destination properly by avoiding obstacles or blocked things on the way. In real- world systems, this is the most difficult task because it requires complex algorithms, continuous sensor feedback, and real- time adjustment. The same concepts can be explored in the simulation environments, where the focus is on building the logic rather than dealing with the limitations of hardware.

In this project, we built an autonomous indoor robot simulation that can move toward any location which is chosen by the user. The user simply clicks on the ground and the system starts to calculate a safe path by using A* pathfinding algorithm. The robot follows this path step by step and makes sure while moving it avoids obstacles in the environment. Since the environment of the robot is grid- based, so it's each part of the space is divided into small cells that are either blocked with obstacles or walkable. The A* algorithm uses this grid to decide which route is the shortest and safest for moving the robot. A* algorithm helps to find the path in the shortest way. The robot uses A* search algorithm because it finds the shortest

path or moving path between a starting node and destination or goal node.

To make the simulation more realistic and engaging, a camera system was also included. When the robot is idle, the camera stays in a top-down position so that the user of the robot can clearly investigate and can see the whole environment properly. When the robot starts moving, the camera automatically switches to follow view and make the simulation more dynamic. Some additional features were integrated such as a Lidar-inspired raycasting sensor that provides real- time obstacle detection. A stuck detection mechanism enables the robot to make replan its path if the progress is blocked by something. A collision handler controls and manages unexpected contact with obstacles.

The project carries together ideas from Artificial Intelligence (AI), robotics, game development, and computer science. It shows how a classic algorithm like A* can be used to solve navigation problems in a simple but effective way. Though it is a simulation, the concepts applied here are the same as those used in real- world applications such as house cleaner robot, delivery robot, self-driving cars, and drones.

II. Literatures Review

A. Obstacle Detection and Pathfinding for Mobile Robots

Abiyev and Arslan (2017) proposed a robot navigation system that works AI-based vision with A* pathfinding algorithm. According to the authors, a humanoid NAO robot uses a camera to capture images of its environment. They also applied Machine Learning algorithm, Support Vector Machine (SVM) classifier to identify obstacles and free spaces. In their work, once the environment is processed, A* algorithm starts to make the shortest collision free path to the main goal. This combination shows that, how AI methods for perception and heuristic values can work together to make a robot operate autonomously in unknown environments. Their work and result shows that A* algorithm is highly effective when paired with intelligent sensing and making it practical choice for real- world robotic navigation.

B. Formal Basis for the Heuristic Determination of Minimum Cost Paths

Many algorithms have proposed to help agents move efficiently using an environment. The earliest foundation came from Hart, Nilsson, and Raphael (1968), who introduced the A* algorithm. They showed how heuristic value works. This heuristic refers an estimate of how far the goal is. They showed in the A* algorithm, heuristic value uses in regular search process which makes the process both fast and easier. In Dijkstra's algorithm, it explores almost everything but A* focuses only on the most promising destination. This process and correctness are the main reason it has become so widely used in robotics, and game.

C. Anytime Dynamic A: An anytime, replanning algorithm

Another improvement came from Likhachev et al. (2005), who created Anytime Dynamic A*. This algorithm finds a quick but most probably non-optimal path. Then improves it step by step as time allows. If environment start to change , it updates the solution without starting over again. This tell us how A* can be extended for situations where both speed and adaptability are necessary.

III. Methodology

The system was designed as a combination of eleven scripts that collectively mange camera, sensor, user input, grid generation, robot control, pathfinding, obstacle detection and path visualization which work together.

A. Environment Representation and Robot Workflow

At first, the robot's world is represented as a 2D grid where each cell is free and occupied by

an obstacle. The grid is implemented as a simple 2D which allows quickly update and access. The robot scans its surrounding situations by using sensors like LiDAR and ultrasonic sensors. After that it updates the map in real- world. At the beginning, the robot will get a starting point and a final goal. It will enters a decision -making loop which will continue until it reaches the main destination or encounters an unavoidable blockage. In each loop,

1. Sensor data is read and the map will be updated.
2. A path to the goal is designed by the A* algorithm.
3. The robot follows the planning path by executing movement commands.
4. If any new obstacles appear or blocked things appear, the robot replans the path dynamically.
5. The loop will repeat until the main goal is reached.

This workflow allows and helps the robot to continuously plan, sense, making navigation and decisions, and reliable in a dynamic environment.

B. *A* Path Planning Algorithm*

A* search algorithm is the core algorithm that the robot uses to plan its path. It finds the shortest route from the robot's current position by combining the actual cost from the start [g(n)] and a heuristic value to the goal [h(n)].

The A* evaluation function defined as:

$$f(n) = g(n) + h(n) \qquad (1)$$

Where, f(n)= estimated total cost of the cheapest solution through node n.
g(n): cost from the start node to n
h(n)= heuristic estimate of the cost from n to the goal.

We use Manhattan distance as the heuristic value which works good for grid- based movement. A* algorithm keeps an open list of nodes and closed list of nodes to explore and visit. At each and every step, it allows the node with the lowest f(n) value and evaluate its neighbors. It repeats this process until the main goal is reached and no path exists. This method ensures the robot is finding the shortest path.

C. *Dynamic Obstacle Handling*

The robot is equipped to deal with obstacles or any blocked things in real- world. Whenever new obstacles are detected, the map will update immediately. The robot will replan routing using A*. This gives permission to avoid collisions and help to find alternative routes on the fly. A safety buffer is also given. It maintained around obstacles to account for sensor inaccuracies.

D. *Movement Execution and Control*

Planned paths are converted to step by step movement. Each step in the path corresponds to a grid. Motion commands are used for moving forward, moving diagonally, turning the body. The robot monitors its sensor while moving forward or backward, turning, and where to stop or adjust if unexpected obstacles appear in front of it.

E. *Heuristic and Optimization*

Some strategies improve navigation.

1. Prioritize Node Expansion: If we investigate, neighboring cells closer to the destination and that are explored first to speed up planning.
2. Partial Replanning: Only affected portion of the path is recalculated when obstacles come.
3. Cached Paths: If the environment stays unchanged, then frequently used paths are stored and must be reused.

F. *Input Handling and Control Loop*

The robot follows a continuous loop of sensing something, planning for movement and acting.

1. Robot, update the grid map with new sensor reading.
2. Make plan or replan the path using A* algorithm
3. Execute the next movement.
4. Investigate if the main goal is reached or obstacles is blocking the path.
5. Repeat the processes until the goal is reached without facing any obstacles.

G. Implementation Details

1. Code Structure Overview:
   - The CameraController script manages how the camera will work. When the robot is not moving it stays in a top- down position and when the robot starts to move the camera switch to following view.
   - The ClickToSetDestination script controls mouse input. When the user clicks on the floor, the click location is processed to detect where the robot should go or should not go. The grid is updated to investigate and see the obstacles or blocked areas. Then the pathfinding algorithm like A* is used to calculate the path.
   - The GridManager script creates a virtual grid over the environment. Each and every position represent as a node. These nodes can either blocked by an obstacle or can be walkable. The node class stores information for each grid position, including it is walkable and its pathfinding costs.
   - The PathFinding script is the main part of this project where the A* algorithm is implemented. It calculates the shortest path from the robot's current position to the main destination by using heuristic value. The algorithm uses three values for each node. gCost for the distance from the start, hCost for the estimated didtance, to the target, fCost for which is the sum of the two. The algorithm quickly finds the best route by expanding the node with the lowest fCost.
   - The RobotController, which helps to control the robot's motion. The path is considered as a series of waypoints. The robot moves from one waypoint to the next until the main destination is reached. In this work, the movement uses Unity's physics system to make it realistic project work. A line renderer is used to draw the planned path above the ground.
   - This LidarSensor.cs script implement a simple LiDAR sensor in Unity by casting multiple rays. Each ray checks obstacles within given range using physics theory.
   - The RobotController to recalculate its path.
   - The RobotCollisionHandler dtects collisions between the robot and obstacles in Unity.

2. Environment and Grid Representation

   The robot's environment is represented as a 2D grid environment where each cell shows a discreate space the robot can occupy.

Grid Example Layout (5×5 Grid):

	0	1	2	3	4
0	S			X	
1		X		X	
2					
3	X		X		
4					G

S → Start Position
G→ Goal Position
X→ Obstacles
Blank→ Free Cell

This grid is showing in code as a 2D array (matrix) where each cell has a value corresponding to free space, obstacles, or start or goal.

3. Heuristic value calculation part

```
2 references
int Heuristic(Node a, Node b)
{
    int dx = Mathf.Abs(a.gridX - b.gridX);
    int dy = Mathf.Abs(a.gridY - b.gridY);
    int min = Mathf.Min(dx, dy);
    int max = Mathf.Max(dx, dy);
    return 14 * min + 10 * (max - min);
}
```

A* algorithm uses heuristic h(n) value.

4. Path Calculation code

```
public List<Vector3> FindPath(Vector3 startPos, Vector3 targetPos, bool allowDiagonals = true)

if (grid == null)
{
    Debug.LogError("GridManager not assigned to Pathfinding!");
    return null;
}

// ✅ Always refresh obstacles before computing
grid.ScanForObstacles(grid.unwalkableMask);

Node startNode = grid.NodeFromWorldPoint(startPos);
Node targetNode = grid.NodeFromWorldPoint(targetPos);

if (startNode == null || targetNode == null) return null;
if (!startNode.walkable || !targetNode.walkable) return null;

List<Node> openSet = new List<Node> { startNode };
HashSet<Node> closedSet = new HashSet<Node>();

startNode.gCost = 0;
startNode.hCost = Heuristic(startNode, targetNode);
startNode.parent = null;

while (openSet.Count > 0)
{
    Node current = openSet[0];
    for (int i = 1; i < openSet.Count; i++)
    {
        if (openSet[i].fCost < current.fCost ||
            (openSet[i].fCost == current.fCost && openSet[i].hCost < current.hCost) ||
            (openSet[i].fCost == current.fCost && openSet[i].hCost == current.hCost && Random.value > 0.5f))
        {
            current = openSet[i];
        }
    }

    openSet.Remove(current);
    closedSet.Add(current);

    if (current == targetNode)
    {
        List<Node> nodePath = RetracePath(startNode, targetNode);
        grid.debugPath = nodePath;
        return ToWorldPoints(nodePath);
    }

    foreach (Node neighbour in grid.GetNeighbours(current, allowDiagonals))
    {
        if (!neighbour.walkable || closedSet.Contains(neighbour))
            continue;

        int moveCost = (neighbour.gridX != current.gridX && neighbour.gridY != current.gridY) ? 14 : 10;
        int tentativeG = current.gCost + moveCost;

        if (!openSet.Contains(neighbour) || tentativeG < neighbour.gCost)
        {
            neighbour.gCost = tentativeG;
            neighbour.hCost = Heuristic(neighbour, targetNode);
            neighbour.parent = current;

            if (!openSet.Contains(neighbour))
                openSet.Add(neighbour);
        }
    }
}
```

5. Main Navigation Loop

The robot navigates from start to goal using sensor feedback and path planning.

Lopp Steps-

a. Sensor Update

Read current sensor data.

Mark obstacles in the grid in real-world.

b. Path planning

Use A* algorithm.

6. Dynamic Obstacle Handling

Dynamic Replanning: If a new obstacle is detected on the path then

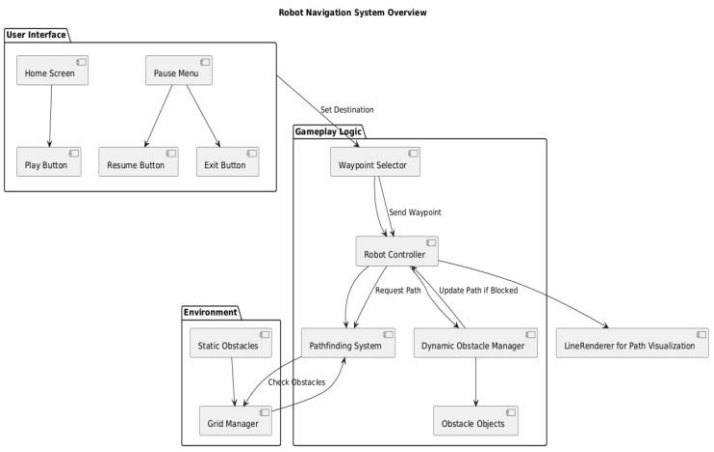
a. stop the current movement.
b. Recalculate the path and updated grid.
c. Resume navigation.

Loop Continuation:

The robot repeats sensing→ planning → movement until the goal is reached.

7. Error Handling and Safety
   a. Collision Avoidance: Stop if an obstacles appears .
   b. Sensor Faults : If sensors fail, then stop robot and retry sensor new update immediately.
   c. Safety limits: Ensure robot never exists the predefined grid area.

8. System Architecture Overview



We proposed a system architecture and implemented. We worked on this and completed work according to this structure. In this structure, we proposed user interface where home screen, play button, pause menu, resume and exit button are shown. In environment, static obstacles and grid manager are given. Waypoint selector, robot controller , dynamic obstacle manager, line renderer, obstacle objects, request path, pathfinding system are given . This is how our robot will work using A*.

9. Algorithms

Algorithm 1 A* Pathfinding Algorithm

```
Initialize OpenSet with S, ClosedSet ← ∅
g(S) ← 0
f(S) ← g(S) + h(S)
while OpenSet is not empty do
    Select node n from OpenSet with minimal f(n)
    if n = G then
        return ReconstructPath(n)
    end if
    Remove n from OpenSet
    Add n to ClosedSet
    for all neighbors m of n do
        if m ∈ ClosedSet then
            continue
        end if
        tentative_g ← g(n) + dist(n, m)
        if m ∉ OpenSet or tentative_g < g(m) then
            parent(m) ← n
            g(m) ← tentative_g
            f(m) ← g(m) + h(m)
            if m ∉ OpenSet then
                Add m to OpenSet
            end if
        end if
    end for
end while
return Failure
```

Algorithm 1: A*

Here, we have shown how A* algorithm works to find the shortest path. The robot will use this algorithm to find the best way to reach at the destination without classing with any obstacles.

Algorithm 2: A* in Unity

Algorithm 1 A* Pathfinding in Unity (with diagonals and obstacle checking)

```
function FindPath(startPos, targetPos, allowDiagonals)
    if GridManager is null then return failure
    Call ScanForObstacles()
    startNode ← NodeFromWorldPoint(startPos)
    targetNode ← NodeFromWorldPoint(targetPos)
    if startNode or targetNode is null or not walkable then
        return failure
    end if
    Initialize openSet ← {startNode}, closedSet ← ∅
    startNode.g ← 0
    startNode.h ← Heuristic(startNode, targetNode)
    while openSet is not empty do
        current ← node in openSet with lowest f = g + h
        if current = targetNode then
            return ReconstructPath(startNode, targetNode)
        end if
        Remove current from openSet
        Add current to closedSet
        for all neighbor ∈ GetNeighbours(current, allowDiagonals) do
            if not walkable or neighbor ∈ closedSet then
                continue
            end if
            moveCost ← 10 or 14 based on diagonal
            tentativeG ← current.g + moveCost
            if neighbor ∉ openSet or tentativeG < neighbor.g then
                neighbor.g ← tentativeG
                neighbor.h ← Heuristic(neighbor, targetNode)
                neighbor.parent ← current
                if neighbor ∉ openSet then
                    Add neighbor to openSet
                end if
            end if
        end for
    end while
    return failure
end function
```

Another algorithm is where we presented A* pathfinding in Unity with diagonals and obstacles checking. This is the core algorithm of our indoor robot project.

Both two algorithms have highlighted here.

TABLE I
COMPARISON BETWEEN FOUR ALGORITHMS

Algorithm	Guarantees Shortest Path	Efficiency	Suitable for Large Ma
BFS	Yes	Low	No
DFS	No	High	No
Dijkstra	Yes	Medium	Limited
A*	Yes	High	Yes

We compared between four algorithms. If we analyze BFS, DFS, A* and Djkstra, we can see how well A* woks to reach at the goal with shortest time and path. A* also have high efficiency more than others.

10. Programming Language

The project was built by using C# programming language.

11. Robot Setup

A rigidbody was included to the robot to allow a physics- based movement. Movement was managed step by step toward waypoints.

12. Grid and Path Visualization

A line renderer was figured to display paths in green colour. The floor was divided into a grid using the MeshRenderer and MeshFilter.

.

13. User Input and Target Selection

Unity's new Input System was used for detection of clicks. From the mouse position a raycast determined the point on the floor that the user clicked.

14. Pathfinding and Navigation

A pathfinding algorithm calculated a path from the robot's current position to the main destination and avoiding unwalkable nodes. After reaching the destination, the line Renderer was cleared to remove the path which was beeing displayed.

V. RESULT AND DISCUSSION

The project demonstrated the effectiveness work of the A* algorithm in a grid- based environment. We also used C# programming language to do this work. The robot was able to avoid obstacles without need any kind of advanced sensors. We were able to apply our AI knowledge here. The integration of multiple scripts worked perfectly. However, some challenges were observed while doing the work. The path finding performance decreases if the grid becomes very large because more nodes must be checked. The robot's movement is limited from straight lines to waypoints. Obstacles were detected using simple sphere checks. As a result, it may not manage complex shapes.

VI. CONCLUSION

Our project successfully implemented robot navigation system using the A* algorithm. The robot can move toward main destination selected by users. While working the robot can detect obstacles and avoid those to go to the main path. A* algorithm helps the robot to go the destination within the shortest path. While there are few limitations, the project demonstrates the core principles of autonomous navigation in a simulated environment successfully. It also foundation for future work on more advanced robotics, game development, and AI navigation techniques.

REFERENCES

[1] Abiyev, R. H., & Arslan, M. (2017). Obstacle detection and pathfinding for mobile robots. *Neural Computing and Applications, 28*(12), 4037–4048. https://doi.org/10.1007/s00521-016-2290-7

[2] Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic

determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, *4*(2), 100–107. https://doi.org/10.1109/TSSC.1968.300136

[3] Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., & Thrun, S. (2005). *Anytime Dynamic A*: An anytime, replanning algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling* (Vol. 15, No. 1, pp. 262–271). AAAI Press. https://doi.org/10.1609/icaps.v15i1.13068

[4] Unity Technologies. (2023). *Unity User Manual: Rigidbody component*. Unity. https://docs.unity3d.com/Manual/class-Rigidbody.html

[5] Lau, B., & Stentz, A. (2013). Efficient grid-based spatial representations for robot navigation. *Robotics and Autonomous Systems*, 61(11), 1259–1276. https://doi.org/10.1016/j.robot.2013.06.003

[6] Connell, D., & La, H. M. (2017). Dynamic path planning and replanning for mobile robots using RRT*. *arXiv*. https://arxiv.org/abs/1704.04585

[7] Prasanth, S. (2025, April 11). *A* algorithm vs Unity NavMesh: Choosing the right pathfinding for your game*. Medium. https://sivakumar-prasanth.medium.com/a-algorithm-vs-unity-navmesh-choosing-the-right-pathfinding-for-your-game-202a5c776385

[8] Unity Technologies. (n.d.). *Getting started with AI pathfinding*. Unity Learn. https://learn.unity.com/project/beginner-ai-pathfinding