INDIAN INSTITUTE OF
INFORMATION TECHNOLOGY
GUWAHATI

DEPARTMENT OF COMPUTER SCIENCE AND
ENGINEERING

Motion Planning in Multi robotic system

Submitted By:
Om Sharma
Roll No. : 2101134

Project Mentor:
Dr. Nilotpal Chakraborty.

DECLARATION

I hereby declare that the following project, titled **"Motion Planning in Multi Robotic System,"** represents my own work and ideas. Any ideas or words taken from other sources have been appropriately cited and referenced. I attest that I have adhered to all principles of academic honesty and integrity throughout the development of this project.
I further declare that I have not misrepresented, fabricated, or falsified any idea, data, fact, or source in this submission. Additionally, I acknowledge that failure to properly cite sources or obtain necessary permissions may lead to potential legal consequences.

Date: November 20, 2024

Signature: Om Sharma

ACKNOWLEDGEMENT

I express my sincere gratitude to the Indian Institute of Information Technology Guwahati for providing the necessary resources and environment for the completion of this project.
I am deeply thankful to Dr. Nilotpal Chakraborty for his invaluable guidance and support throughout the development of this project. His expertise and encouragement were instrumental in shaping the direction of this work.

Contents

1 Introduction to Multi-Robot Motion Planning

In multi-robot systems, guiding robots to their destinations becomes challenging when multiple robots are moving simultaneously. Path-planning algorithms can efficiently determine the optimal path for a single robot but often struggle in environments that are constantly changing, such as when other robots are moving. This is where motion planning becomes crucial. Motion planning goes beyond simply finding the shortest path; it also tracks the movement of all robots and considers how each robot's actions impact the others, leading to smoother and safer coordination.

To handle dynamic environments, we modify advanced algorithms like A*, which identifies the shortest path by exploring different routes. Additionally, Deep Reinforcement Learning (DRL) enables robots to learn and adapt in real time by rewarding beneficial decisions, such as avoiding obstacles or selecting efficient paths. This helps them improve navigation in unpredictable conditions.

By comparing these algorithms, we can develop a system in which robots not only find the best path but also adapt to real-time changes, such as the movement of other robots. This ensures that robots work together efficiently and safely, even in complex, dynamic environments.

2 Abstract

In distributed multi-robot systems, ensuring collision-free motion planning is a complex challenge, especially in dynamic environments where multiple robots are operating simultaneously. Traditional path-planning algorithms are effective for individual robot navigation but struggle when dealing with the unpredictable movements of other robots in the system. To overcome these limitations, this research focuses on advanced motion planning techniques that ensure safe and efficient robot coordination.

We model the overall motion planning problem in a two-dimensional grid, aiming to minimize both the time and energy taken by the robots to reach their destinations from their respective sources. We enhance standard pathfinding algorithms and apply graph search algorithms like A* along with Deep Reinforcement Learning (DRL) with suitable modifications to adapt to real-time changes. DRL enables robots to learn from their environment and make optimal decisions, improving their ability to avoid collisions and navigate efficiently in uncertain conditions.

By integrating these algorithms and conducting a comparative analysis, we propose a framework that enables robots to dynamically adjust their paths, ensuring collision-free movement and optimal coordination.

Index Terms: A* Algorithm, Deep Reinforcement Learning Algorithm.

3 Background

3.1 Introduction

The rapid advancement of robotics has led to an increasing demand for autonomous multi-robot systems in a variety of industries, including logistics, manufacturing, disaster management, and intelligent transportation systems. These systems rely on multiple robots navigating complex spaces to achieve specific objectives. However, ensuring efficient and collision free movement becomes a significant challenge, especially when robots operate independently and simultaneously. Traditional path-planning algorithms like Dijkstra's Algorithm are capable of finding the shortest or cost-effective path for single-robot scenarios. However, in the presence of multiple robots, the environment becomes dynamic where each robot's movement can interfere with others. With multiple robots sharing the same space, they can create moving obstacles for one another, adding a layer of complexity to an already dynamic environment where the location of obstacles is constantly changing. This adds a layer of complexity, requiring the system to balance individual efficiency with collision prevention. Thus, directly applying traditional algorithms to multi-robot scenarios without considering these interactions and dynamic obstacles can lead to inefficiencies, deadlocks, or even collisions. Collision-free motion planning in multi-robot systems is a complex and multifaceted problem that requires a more sophisticated approach than simply identifying the shortest path from a source to a destination. Unlike traditional single robot path planning, which focuses primarily on finding an optimal route in a static environment, multi-robot systems operate in dynamic environments where multiple robots constantly move and interact. This dynamic interaction introduces significant challenges, as one robot's actions can directly influence others' movements, leading to potential collisions. The unpredictable nature of these environments requires robots to make continuous real-time adaptations to their planned trajectories. As new information becomes available such as the sudden appearance of an obstacle or changes in the positions of other robots— robots must update their plan an

3.2 Problem Formulation

We consider the entire motion planning on a two-dimensional surface, by modeling the area as a 2D grid, which is a 2D array of cells. Let the grid be represented as $G = (n \times m)$, where n is the number of rows and m is the number of columns. The system contains

a set of robots. Let $x^r_{i,j}(t) \in \{0,1\}$ be a binary variable that represents if the robot $r \in R$ occupies the cell (i,j) $(= 1)$ at a given time step, or not $(= 0)$. Each robot has a designated source cell (S) from where it must start, and ends at its specified destination cell (D), as described below:

$$x^r_S(t=0) = 1 \quad \forall r \in R$$

$$x^r_D(t=T_r) = 1 \quad \forall r \in R$$

where T_r is the time when robot r reaches D.

Let $y^r_{i,j,d}(t) \in \{0,1\}$ represent the movement of robot r from the cell (i,j). At any point in time, the degree of freedom of any robot can be at most 4, i.e., we restrict the movement of robot r in one of the four directions only from (i,j): up, down, left, right.

However, in our problem, we enforce the condition that a robot can move to a new cell (i',j') only if that cell is unblocked. A blocked cell is defined as a cell that is either currently occupied by a robot or adjacent to an occupied cell. If a robot occupies a cell, that cell and all its neighboring cells (up, down, left, and right) are considered blocked, meaning no other robot can enter these cells to prevent potential collisions. Thus, the blocked status of a cell ensures that only one robot can move into or occupy a given area at any time, maintaining a collision-free environment within the grid.

We define a variable $b_{i,j}(t) \in \{0,1\}$ to indicate whether the cell (i,j) is blocked at time t. Hence, we can mathematically write the blocking condition as follows:

$$b_{i,j}(t) = \sum x^r_{i,j}(t) + x^r_{i+1,j}(t) + x^r_{i-1,j}(t) + x^r_{i,j+1}(t) + x^r_{i,j-1}(t) \quad \forall r \in R$$

With the movement constraint defined above, we define $y^r_{i,j,d}(t)$ as follows:

$$y^r_{i,j,d}(t) \leq 1 - b_{i',j'}(t)$$

Thus, the movement from the cell (i,j) to the cell (i',j') is possible at time t only when $b_{i',j'}(t) = 0$.

Below, we define the specific constraints for the 4 movements from the cell (i,j) by expanding equation (4) as follows:

- Moving up from (i,j) to $(i-1,j)$: $y^r_{i,j,up}(t) \leq 1 - b_{i-1,j}(t)$
- Moving down from (i,j) to $(i+1,j)$: $y^r_{i,j,down}(t) \leq 1 - b_{i+1,j}(t)$
- Moving left from (i,j) to $(i,j-1)$: $y^r_{i,j,left}(t) \leq 1 - b_{i,j-1}(t)$
- Moving right from (i,j) to $(i,j+1)$: $y^r_{i,j,right}(t) \leq 1 - b_{i,j+1}(t)$

Thus, the following constraints must be fulfilled to ensure a collision-free system:

- No two robots can occupy the same cell simultaneously. Mathematically, this can be expressed as:

$$\sum_{r \in R} x_{r,i,j}(t) \leq 1 \quad \forall (i,j) \in G, \forall t$$

  where $x_{r,i,j}(t) \in \{0,1\}$ indicates if robot r occupies the cell (i,j) at time t. The sum over all robots r in the set R should be less than or equal to 1 to ensure that no two robots occupy the same cell.

- If a robot occupies a cell, the neighboring cells (up, down, left, right) are blocked and cannot be occupied by other robots. This is mathematically represented as:

$$b_{i,j}(t) = \sum_{r \in R} x_{r,i,j}(t) + x_{r,i+1,j}(t) + x_{r,i-1,j}(t) + x_{r,i,j+1}(t) + x_{r,i,j-1}(t)$$

  where $b_{i,j}(t) \in \{0,1\}$ indicates whether the cell (i,j) is blocked at time t. If any robot occupies a cell, the neighboring cells are also blocked to prevent other robots from entering them.

4 Algorithm Implemented

4.1 A*(A-Star) Algorithm:

The A* algorithm is employed to enable robots to efficiently find their shortest paths in the grid while avoiding collisions. The steps involved in the algorithm are as follows:

1. **Grid Setup:** A 16x16 grid is created, with all cells initially unblocked (represented by '1'). Each robot has a starting position and a destination. The grid can be modified to block cells when a robot occupies a position to prevent other robots from moving through.
2. **Blocking/Unblocking Cells:** Whenever a robot moves, its current position and adjacent cells are marked as blocked for the other robot. This prevents both robots from colliding. These blocked cells are cleared (unblocked) when it's the other robot's turn to move.
3. **A* Pathfinding:**
   - Each robot uses the A* algorithm to find the shortest path from its current position to its destination.
   - The Manhattan distance is used as the heuristic (h), which calculates the straight-line distance in grid-based movements, considering only vertical and horizontal moves. The formula for Manhattan distance is:

     $$h = |x_1 - x_2| + |y_1 - y_2|$$

     where (x_1, y_1) is the current position and (x_2, y_2) is the destination.
   - The total cost (f) is computed as the sum of:

     $$f = g + h$$

     where:
     - g is the actual cost to reach the current cell (number of steps taken).
     - h is the estimated cost to reach the destination (Manhattan distance).
4. **Turn-based Movement:**

- The robots take turns moving. Each robot searches for the shortest path using A* and tries to move towards its destination.
- If the next cell on the path is blocked by the other robot, the moving robot waits for the other to move.
- If a robot reaches its destination, it stops.

5. **End Condition:**
   - The algorithm continues until both robots either reach their destinations or the maximum number of steps is reached.
   - If both robots successfully reach their destinations, the average number of steps taken (cost) is calculated. If either robot fails to reach the destination, the algorithm reports a failure.

4.2 Deep Reinforcement Learning (DRL):

In this multi-robot Q-learning algorithm, the implementation incorporates blocking and unblocking cells to manage collisions and ensure smooth navigation through the grid. The following is an integrated view of how the DRL algorithm works:

1. **Hyperparameters and Algorithm Setup:** The algorithm utilizes Q-learning with parameters such as:
   - **Gamma (discount factor):** Balances immediate and future rewards, helping robots weigh the benefits of immediate movement versus potentially more rewarding paths.
   - **Alpha (learning rate):** Controls how much new information overrides old Q-values, allowing robots to adapt their paths over episodes.
   - **Epsilon (exploration factor):** Drives the balance between exploration and exploitation, with epsilon gradually decaying to encourage optimal pathfinding as the learning progresses.
   - **Max_Steps:** A parameter that defines the maximum number of steps each robot is allowed to take per episode before the episode terminates. This is set to prevent infinite loops and ensures that the algorithm does not continue indefinitely in situations where the robots may struggle to reach their destinations.
2. **Robot Actions and Intended Positions:** Each robot selects actions (move up, down, left, right, or stay) based on exploration or exploitation strategies. These actions map to "intended positions," indicating each robot's planned move. This allows for a tentative plan before actual movements are made, enabling the blocking mechanism to take effect.

3. **Blocking and Unblocking of Cells:** A key feature of this setup is the blocking mechanism:
   - **Blocking cells:** To avoid collisions, intended positions are checked against a set of "blocked cells," which includes other robots' intended moves and their current positions. If a robot's next move collides with another's intended move, the cell is marked blocked.
   - **Unblocking and adjacent blocking:** After moving, robots block not only their occupied cell but also adjacent cells, simulating a temporary buffer zone to prevent immediate encroachment. This ensures that robots avoid paths that could lead to close interactions, fostering a smoother, cooperative movement pattern.
   - **Robots unable to move due to blocked cells:** Robots unable to move due to blocked cells stay in place, marking their current position as blocked for that step.
4. **Learning Dynamics with Blocking:** Through the reward system, each robot is encouraged to move efficiently:
   - Positive rewards for reducing distance to goals discourage lingering, while penalties for stalling or moving away incentivize finding and maintaining efficient paths.
   - This reward structure, coupled with Q-learning's iterative updates, helps each robot adapt its pathfinding to avoid blocking-heavy paths over time.

   As a result, the algorithm gradually teaches robots to coordinate better, reducing unnecessary blocking situations and improving collective efficiency.
5. **Adaptive Convergence:** Over episodes, epsilon decay reduces random movements, and each robot's Q-values encode the most efficient, least-blocked paths. The adaptive convergence minimizes congestion, making it effective even for large grids.
6. **Hyperparameter Tuning:** Below is the image of the hyperparameters that have been tuned for each size of grid to achieve optimal and variable results.

	DRL Hyperparameters			
	Gamma	Alpha	Epsilon Decay	Max steps
Size Of Grid				
4*4	0.95	0.1	0.1	2000
8*8	0.95	0.1	0.1	2000
16*16	0.95	0.1	0.1	5000
32*32	0.9	0.2	0.995	5000
64*64	0.9	0.2	0.995	10000
128*128	0.9	0.2	0.995	20000
256*256	0.9	0.2	0.995	5000

Figure 4.1: Hyperparameters Tuned for Q-learning Algorithm

5 Results

In this section, we compare two algorithms: A* (A-Star) and DRL (Deep Reinforcement Learning), based on their performance in terms of execution time and average cost for varying grid sizes. The comparison is shown below.

The algorithms have been run on Google Colab, with the following specifications:

- **CPU:** The default CPU is an Intel Xeon CPU with 2 virtual CPUs (vCPUs). You can upgrade to a higher CPU configuration, such as up to 96 vCPUs.
- **RAM:** The default RAM is 13 GB.

5.1 Comparison of Execution Time and Average Cost

Below is a graphical representation comparing the execution time and average cost for the A* and DRL algorithms on varying grid sizes.

	Algorithms			
	A* Star		DRl Algorithm	
Grid size	Exe. Time(S)	Avg. Cost	Exe.Time(S)	Avg. Cost
4*4	0.0059	6	1.076322317	6
8*8	0.1036	14	2.002675056	14
16*16	0.4807	30	7.581232786	30
32*32	3.0483	62	1.128479242	62
64*64	15.4763	126	3.442433596	126
128*128	84.7217	254	73.49922943	254
256*256	607.4264	510	1612.102803	510

Figure 5.1: Comparison of A* and DRL algorithms based on Execution Time and Average Cost for varying grid sizes

This comparison reveals how the execution time and average cost of both algorithms vary as the grid size increases. The A* algorithm performs faster with smaller costs for smaller grid sizes, while DRL shows a more gradual increase in both execution time and cost, highlighting the more complex learning and decision-making process of DRL.

5.2 Graphical Comparison of Execution Time and Average Cost

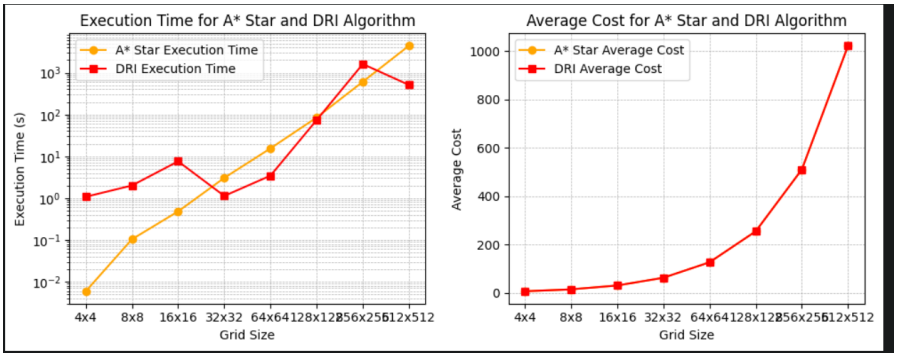

Figure 5.2: Line graph for Execution Time and Average Time vs Grid Size for A* and DRL algorithms

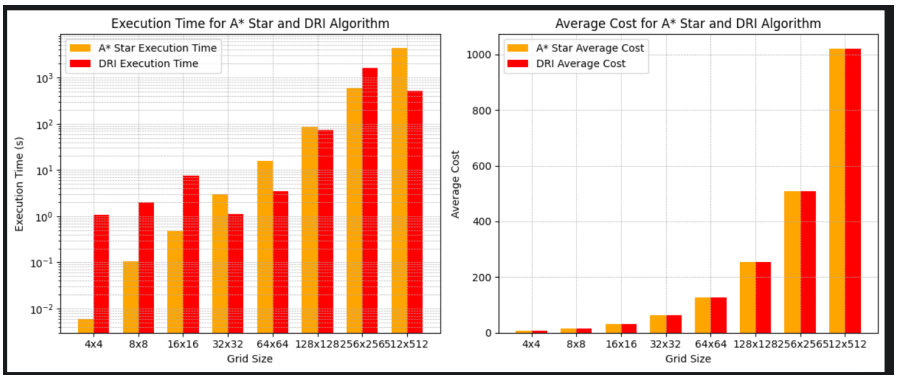

Figure 5.3: Bar Graph for Execution Time and Average Cost vs Grid Size for A* and DRL algorithms

6 Conclusion

Though there is consistency in the cost observed from the results, this occurs due to the current setup of the environment, which does not consist of any static obstacles. However, based on the execution time of each algorithm, several conclusions can be drawn:

1. **Execution Time Comparison at Smaller Grids (4x4 to 32x32):**
    - For smaller grid sizes (4x4, 8x8, and 16x16), the DRL algorithm has a noticeably higher execution time than A*, indicating that it may be more computationally intensive at these scales.
    - As the grid size increases up to 32x32, the difference in execution time narrows, with DRL actually having lower execution time than A* on the 32x32 grid.
2. **Performance in Larger Grids (64x64 and above):**
    - For larger grid sizes (64x64, 128x128, and 256x256), A*'s execution time continues to increase at a slower rate compared to the DRL algorithm.
    - However, the DRL algorithm's execution time jumps significantly at 128x128 and 256x256 grid sizes, indicating it may not scale as well as A* with the current hyperparameters.
3. **Effect of Hyperparameters on DRL Performance:**
    - The drastic increase in execution time at larger grid sizes could suggest that the current settings are not optimized for scalability.
    - Adjusting these hyperparameters (like the learning rate, exploration rate, or neural network architecture) could potentially reduce the DRL algorithm's execution time on larger grids, making it more competitive with A*.
4. **Scalability Observation:**
    - Overall, while DRL may have a competitive edge in execution time at specific mid-range grid sizes (e.g., 32x32), it struggles with scalability for very large grids with the current parameters.
    - If hyperparameters were optimized to handle larger grids, the DRL algorithm might close the performance gap in execution time for larger grids or even outperform A*.

7 References

1. Liu, L., Wang, X., Yang, X., Liu, H., Li, J., & Wang, P. (2023). Path planning techniques for mobile robots: Review and prospect. *Expert Systems with Applications*.
2. Aggarwal, S., & Kumar, N. (2023). Path planning techniques for unmanned aerial vehicles: A review, solutions, and challenges.
3. Rutili de Lima, C., Khan, S. G., Tufail, M., Shah, S. H., & Maximo, M. R. O. A. (2024). Humanoid robot motion planning approaches.