Multi-Agent Robotic Navigation in Dynamic Environments: A Comparative Study of Centralized and Decentralized Approaches Using Monte Carlo Tree Search

Sai Jagadeesh Muralikrishnan
Graduate Student
University of Maryland
College Park, Maryland
jagkrish@umd.edu

Varun Lakshmanan
Graduate Student
University of Maryland
College Park, Maryland
varunl11@umd.edu

Nitish Ravisankar Raveendran
Graduate Student
University of Maryland
College Park, Maryland
rrnitish@umd.edu

***Abstract*—This report compares centralized and decentralized algorithms for robot navigation. We explore why robots using these two different approaches perform differently in achieving their goals. By simulating various scenarios, we assessed key performance metrics such as task completion time, communication overhead, adaptability to dynamic environments, and robustness against failures. Our findings show that decentralized algorithms generally outperform centralized ones, offering better speed, scalability, and adaptability. Based on these results, we recommend adopting decentralized methods in robotic systems and suggest future research directions, including developing hybrid strategies that combine the strengths of both approaches.**

***Index Terms*—MCTS, Centralized Algorithms, Decentralized Algorithms, Robot Navigation, ROS2, Efficiency, Scalability, Robustness, Communication Overhead**

I. Introduction

In robotics, the efficient and reliable navigation algorithms are important to the success of autonomous systems. These algorithms play a key role in helping robots understand their environment, making decisions, and carry out tasks to achieve their goals [9]. In this report, we focus on two types of MCTS based navigation algorithms: centralized and decentralized.

Centralized algorithms rely on a single controller that manages all robots by processing global information. The controller collects data from each robot, calculates the best paths or actions, and sends instructions back to guide the team. This approach simplifies coordination and ensures that robots act in sync, but it often suffers from communication delays and scalability issues as the system grows.

In contrast, decentralized algorithms allow each robot to make decisions independently using only local information. This means robots don't depend on a central authority, making the system more robust and scalable. Even if one robot or a communication link fails, the rest of the team can continue to operate. However, this independence can sometimes lead to challenges, like difficulty in coordinating actions across multiple robots or ensuring consistency in their behaviors.

The purpose of this report is to explore why robots using centralized algorithms sometimes perform less efficiently than those using decentralized approaches and what are the trade-offs between them. To investigate this, we simulated various scenarios and evaluated key performance metrics such as task completion time, communication overhead, adaptability to dynamic environments and to failures. By comparing these two approaches, we aim to identify their strengths and weaknesses and provide insights into which is more suitable for specific system requirements like scalability and robustness.

Furthermore, we hope this analysis helps improve robotic navigation systems, ensuring that robots perform efficiently and reliably in a variety of environments.

II. Related Work

Monte Carlo Tree Search (MCTS) has been widely used for decision-making and planning in robot systems. While MCTS has proven effective in various domains and its application within the Robot Operating System (ROS 2) framework, especially for multi-agent navigation tasks but is still limited. This section highlights notable research in this field.

A noteworthy work that uses MCTS for multi-robot exploration in decentralized systems is **Decentralised Monte Carlo Exploration (DMCE)**. The method shows how successful decentralized planning techniques may be, although it is limited to ROS 2 environments because it is based on ROS Noetic and is developed in C++ [1].

The application of MCTS for socially-aware navigation is examined in the research **SCAN: Socially-Aware Navigation Using Monte Carlo Tree Search**. In order to improve navigation in dynamic environments, this work models pedestrian behaviors and using MCTS to forecast future states. However, no simulation-based navigation nor ROS 2 are included in the framework [2].

Similarly, MCTS is included into the "**SoRTS: Learned Tree Search for Long Horizon Social Robot Navigation"** project to enhance motion prediction models that are socially aware. Although this study concentrates on integrating learning-based techniques with MCTS, it misses ROS 2's possibility of simulation- or real-world robot navigation [3].

Dec-MCTS: Decentralized Planning for Multi-Robot Active Perception explores decentralized planning for active perception tasks using MCTS, demonstrating its effectiveness for independent decision-making in multi-agent systems. However its focus is more on perception than navigation [7].

Last but not least, the Monte Carlo Tree Search technique with Python implementation, which served as the inspiration for the original design of our MCTS structure, is thoroughly explained in the **AI Boson tutorial** [5]. Our additions, however, expand this foundational implementation to incorporate multi-agent scenarios and dynamic obstacles adaptations.

III. Methodology

This section describes our methodology for comparing centralized and decentralized robot navigation algorithms. We describe our simulation environment setup, the technologies and tools we employed, and a summary of the algorithms we looked into.

A. *Tools Used*

We used a variety of hardware and software tools to model and evaluate the performance of centralized and decentralized navigation algorithms:

- **Operating System:** We used **Ubuntu 22.04** to works seamlessly with the Robot Operating System (ROS).
- **Operating System for Robots (ROS 2 Humble):** A versatile framework for creating robot software is provided by ROS 2 Humble. It supports both our centralized and decentralized algorithms and facilitates communication between various components of the robotic system.
- **Simulation Environment:** We used **Gazebo** to create and run our robotic scenarios. Gazebo provides a robust physics engine that allowed us to incorporate custom plugins for TurtleBot3 which is essential for setting up dynamic environments with moving obstacles.
- **Programming Languages and Libraries:**
  - **Python**: We used Python for scripting and implementing various components of the algorithms, including task execution and metric evaluation.
  - **C++ (Custom Plugins)**: Custom plugins were developed in C++ to extend the functionality of the Gazebo simulation environment. These plugins were used to define dynamic behaviors such as the motion of obstacles.
  - **Matplotlib**: This library was employed to visualize performance metrics such as task completion time, communication overhead, and adaptability.

B. *Environment Used*

In order to evaluate the navigation algorithms, we created a simulation environment that was with both dynamic and static obstacles, simulating an actual inventory situation. With goal markers placed at four corners, the environment has both static and dynamic obstacles to evaluate the algorithm performance under various circumstances.

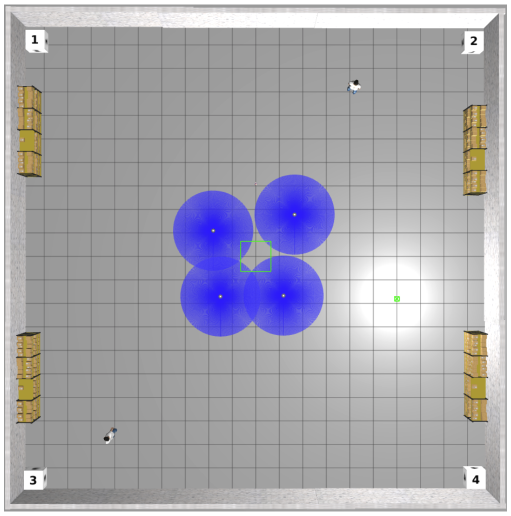

Fig. 1. Simulation Environment in Gazebo: An inventory-like layout with two dynamic obstacles (cylindrical and human) and five static obstacles.

- **Static obstacles:** Five static barriers that were designed to mimic shelves and storage units that are frequently seen in an inventory scenario were included. Robots' pathfinding skills are put to the test as they must maneuver around these obstacles.
- The dynamic obstacle Within the environment, a cylindrical barrier and human like barrier moves back and forth. This dynamic barrier follows a predetermined path, adding unpredictability and putting the algorithm's capacity to adjust to changes in the environment in real time during the test.
- **Robot deployment:** To carry out navigational tasks, we placed several TurtleBot3 Waffle robots across the terrain. The robots had to avoid colliding with both static and dynamic obstacles in order to accomplish predetermined tasks.

C. *Overview of Algorithms*

The base of both the centralized and decentralized approaches is based on the Monte Carlo Tree Search (MCTS) algorithm, as shown in **Algorithm 1**.

In our implementation:

- Rewards are assigned to give reward for goal-reaching, penalize collisions, and account for proximity to obstacles and the goal.
- Rollouts use heuristic based evaluations that considering factors such as distance to the goal and obstacle avoidance.

The base MCTS algorithm 1 is adapted differently for centralized and decentralized systems to accommodate their specific requirements:

1) Centralized Algorithm:

- **State Representation:** A unified global state that representing all robots is used for planning.
- **Action Space:** Joint actions for all robots are evaluated that increases complexity.
- **Tree Structure:** A single tree encapsulates multi-robot decision-making.
- **Reward Function:** Rewards and penalties are team-oriented only focusing on group-level behaviors.

2) Decentralized Algorithm:

- **State Representation:** Each robot considers its state independently.
- **Action Space:** Actions are evaluated on a per-robot basis, reducing complexity.
- **Tree Structure:** Each robot maintains its own MCTS tree, performing parallel decision-making.
- **Reward Function:** Rewards focus on individual performance, emphasizing goal-reaching and obstacle avoidance.

The decentralized algorithm includes heuristic-driven rollouts, prioritizing actions that minimize distance to the goal and avoid obstacles effectively. This allows for faster and more adaptive decision-making in dynamic environments. **Heuristics are not implemented for centralized approach to mitigate computational overload.**

Parameter Explanations:

- S: The initial state of the robot including position and orientation.
- C: Exploration constant in the UCT formula for balancing exploration and exploitation.
- N: The total number of MCTS iterations to perform.
- D: The maximum depth for rollout simulations.
- (G, T, O, D_c): Robot-specific parameters where G is the goal position, T the target threshold, O the obstacle information, and D_c the collision distance threshold.
- Q_i: The total accumulated reward for node i.
- N_i: The number of times node i has been visited.
- N_p: The number of visits to the parent node during selection.
- A_{best}: The action selected as optimal after MCTS iterations.

Algorithm 1 Base Monte Carlo Tree Search (MCTS) Used for Robot Navigation

```
1:  Input: Initial robot state $S$, exploration constant $C$, number of iterations $N$, rollout depth $D$, robot-specific parameters $(G, T, O, D_c)$, sensor data
2:  Output: Best action $A_{\text{best}}$ for each robot
3:  Initialize root node $R$ with state $S$.
4:  Set $R$.visits $= 0$ and $R$.total_reward $= 0$.
5:  Subscribe to sensor data (LaserScan and Odometry) for obstacle positions and robot states.
6:  Initialize metrics for performance tracking.
7:  for each robot $r$ in the environment do
8:      Initialize MCTS for robot $r$ with its local state $S_r$.
9:      for $i = 1$ to $N$ (MCTS iterations) do
10:         Selection:
11:         Traverse the tree using the UCT formula:
                $$UCT = \frac{Q_i}{N_i} + C \cdot \sqrt{\frac{\log(N_p)}{N_i}}$$
12:         Select the child node $C_{\text{best}}$ with the highest UCT value.
13:         Expansion:
14:         if selected node is not terminal then
15:             Select an unvisited action $A$.
16:             Simulate $A$ to compute the next state $S'$.
17:             Add $S'$ as a new child node.
18:         end if
19:         Simulation (Rollout):
20:         Simulate random or heuristic-based actions up to depth $D$ or until $S'$ is terminal.
21:         Compute reward $R$ for the terminal state:
                $$R = \begin{cases} +1000, \text{if goal is reached} \\ -1000, \text{if collision occurs} \\ -(\text{distance to goal}) - (\text{obstacle penalty}), \text{otherwise.} \end{cases}$$
22:         Backpropagation:
23:         Propagate $R$ up the tree:
                $$Q_i = Q_i + R, \quad N_i = N_i + 1$$
24:         Update visit counts and total rewards for all nodes in the path from leaf to root.
25:     end for
26:     Action Selection:
27:     Select the action $A_{\text{best}}$ with the highest visit count in the tree.
28:     Execute $A_{\text{best}}$ for robot $r$.
29: end for
30: Termination:
31: if all robots reach their goals, a collision occurs, or maximum steps are exceeded then
32:     Stop robots and log metrics.
33: end if
34: Output: Best actions $A_{\text{best}}$ for all robots.
```

IV. Experiments and Results

We explain the experiments carried out during the project and the outcomes in this section. Specific features of the centralized and decentralized Monte Carlo Tree Search (MCTS) algorithms were tested in each experiment. There is also an explanation of the difficulties encountered and the changes done to enhance performance. The trials provide an organized approach to guarantee that both algorithms were examined in increasingly more complex situations.

A. Single Robot Testing

We started by implementing the decentralized algorithm into practice and testing it in an empty environment on a single TurtleBot3 Waffle. A goal point was provided to the robot, and its capacity to effectively navigate the surroundings and reach at its goal was assessed. Initially, we encountered issues with the reward function. While the robot could plan paths near the goal, it often failed to reach the goal itself. To address this, we experimented with multiple reward functions, eventually finding one that reliably rewarded the robot to reach the goal. Simultaneously, the centralized approach was framed and tested with two robots to observe message sharing and synchronization. Using tools like `rqt_graph`, we verified message exchanges between the robots and the central node. The centralized approach successfully achieved the goal state but took significantly more time to plan due to the overhead of coordinating two robots in a centralized manner.

B. Multi-Robot Testing in Empty World

We moved to an instance with four TurtleBot3 robots in an empty environment after successfully applying the decentralized algorithm to a single robot and getting reliable results. A shared goal was given to the robots, and their performance was assessed. We saw robot collisions even though the decentralized approach enables the robots to reach their shared goal. The robots' inability to avoid one another as moving barriers caused this problem.

We changed the reward function to take dynamic obstacle avoidance into consideration in order to solve issue. The robots' dynamic navigation and collision avoidance skills were enhanced by this update. On the other hand, because of the computing burden of controlling several robots simultaneously, the centralized method required significantly more time to plan and navigate. This was particularly noticeable when static barriers were added because it was difficult for the centralized controller to calculate effective routes for every robot at once. To see the video of the single robot's testing in an empty environment, https://youtu.be/hkWiSsE46tU

C. Formation Testing

The robustness of the decentralized algorithm was then evaluated by instructing the robots to navigate toward a target in different formations. Line, square, and triangular layouts were among the shapes used to test the algorithm's capacity to plan routes in grid-like areas with little exploration. Four robots were used to test each formation, and both shared and independent goals were set. We carried out increasingly difficult experiments after attaining satisfactory performance in every arrangement. To see the video results of testing multiple robots in an empty world while they move toward their adjacent goal to form a square like pattern, https://youtu.be/L3AC8AWpGdQ

D. Dynamic Obstacle Testing

In this experiment, we placed a human-like and moving cylindrical obstacle in the surrounding area. A human-like obstacle and the cylinder moved in a to-and-fro manner along the x and y axes. Our objective was to compare the effectiveness of centralized versus decentralized approaches in the presence of dynamic obstacles.

Initially, the simulation used 200 iterations per planning step, which led to significant computation delays. To address this we reduced the number of iterations per planning step to 100, which improved performance of our local machine but slightly degraded the searching of the dynamic obstacles and other robots. Additionally, we decreased the linear and angular speeds of the robots to ensure smoother navigation but finally we ended up in 200 iterations per planning step to showcase better path planning. These adjustments enabled the decentralized algorithm to navigate around the moving obstacle without any collisions.

For the centralized approach, we tested the same dynamic obstacle scenario with two robots. The centralized node successfully navigated the robots around the moving obstacle but required much more time due to the computational cost of managing dynamic obstacle avoidance for multiple robots in parallel.To view the video results of multi-robot navigating to goal using centralized and decentralized approach, https://youtu.be/OaaVQO0hlwI

V. Analysis of Results

This section provides a combined analysis of two key performance metrics: *Distance to Goal vs Time* and *Time vs MCTS Calls*, as shown in Figures 2, 3, 4, and 5. These metrics assess the computing cost of MCTS planning phases as well as the algorithms' ability to guide robots to their goals.

1) Distance to Goal vs Time: The *Distance to Goal vs Time* graphs in Figures 2 and 3 track how efficiently the robots reduce their distance to the goal in less time.

Centralized Algorithm:

- **Delayed initial progress:** Robots take longer to start that keeping their distance to the goal still or slightly reduced due to the computational overhead of joint planning. This delay grows with the number of robots.
- **Smooth trajectories:** Once the robots start moving, their paths are smooth and coordinated which shows the benefits of global synchronization. However, this comes at the trade-off of slower overall progress.
- **Waiting behavior near the goal:** Robots wait for all others to complete their navigation before fully stopping. For example, Robot 4, despite being near its goal early, does not exit until Robot 2 and Robot 1 complete their navigation tasks, increasing the overall task time.

Decentralized Algorithm:

- **Faster initial progress:** Robots independently plan and execute their paths that leads to quicker initial progress toward their goals.
- **Dynamic obstacle handling:** Robot 4 takes more time due to delays caused by dynamic obstacles. Replanning steps and MCTS iterations (we set to 200 per planning step) add occasional bottlenecks.
- **Independent trajectories:** Because every robot has a unique trajectory, scalability is improved. This indepen-

dence, however, may occasionally result in inefficiencies or poor global coordination.

2) Time vs MCTS Calls: The *Time vs MCTS Calls* graphs in Figures 4 and 5 measure the time taken for planning relative to the number of MCTS calls, which is one MCTS call equals on planning step with 200 iterations.

Centralized Algorithm:

- **Linear growth in time:** The time taken grows linearly with MCTS calls, as the centralized algorithm synchronizes actions for all robots at every planning step.
- **High computational overhead:** Each MCTS call processes a joint action space for all robots that leads to significant delays. This effect gets worse with more robots or dynamic obstacles.
- **Cumulative delays:** Robots tends to wait for others to complete their planning steps further increasing the total time required to reach the goal.

Decentralized Algorithm:

- **Independent planning:** Each robot performs MCTS independently that leads to variations in the number of MCTS calls and planning times among the robots.
- **Faster overall completion:** In this case Robot 1 and Robot 2 complete their planning quickly, while Robot 4 experiences delays due to dynamic obstacles and re-planning.
- **Non-linear behavior:** Time vs MCTS Calls does not grow uniformly as each robot adapts to its local environment that reduces unnecessary planning steps in simpler scenarios capared to senarios nears dynamic obstacles.

3) Complexity comparison: centralized vs decentralized: The computational complexity of centralized and decentralized MCTS approaches differs a lot because of how they handle decision-making and planning.

- **Centralized complexity:** In the centralized approach, the complexity is roughly $O(M \cdot A^N)$, where M is the number of MCTS iterations, A is the number of possible actions per robot, and N is the number of robots. This means that as the number of robots increases, the computation grows exponentially because all possible joint actions for the robots need to be considered. This makes centralized MCTS harder to scale when dealing with many robots.
- **Decentralized complexity:** For the decentralized approach, each robot runs its own MCTS independently. The complexity for one robot is $O(M \cdot A)$, and for N robots, it becomes $O(N \cdot M \cdot A)$. This linear growth with N makes decentralized MCTS much more scalable, even for larger teams of robots.

As the number of robots increases, centralized MCTS rapidly becomes computationally costly, mostly due to the exponential growth in the joint action space. However, because each robot operates independently, decentralized MCTS scales more effectively; however, this comes at the cost of less coordinated team behavior.

4) Comparison: centralized vs decentralized: The combined analysis of these metrics highlights the following key differences:

- **Efficiency:** The decentralized algorithm shows faster progress toward goals (Figure 3) and shorter overall planning times (Figure 5) due to distributed decision-making.
- **Scalability:** Decentralized planning works better with the increased number of robots, as each robot independently handles its own MCTS planning. The centralized approach struggles with scalability due to the combined increase of the action space.
- **Coordination:** The centralized algorithm achieves smoother and more synchronized paths, but robots wait for other robots to complete their tasks which adds delays. The decentralized algorithm avoids such dependencies, though it may lack precise global coordination.
- **Adaptability:** The decentralized system is better suited for dynamic environments, as robots can replan locally without relying on a central controller.

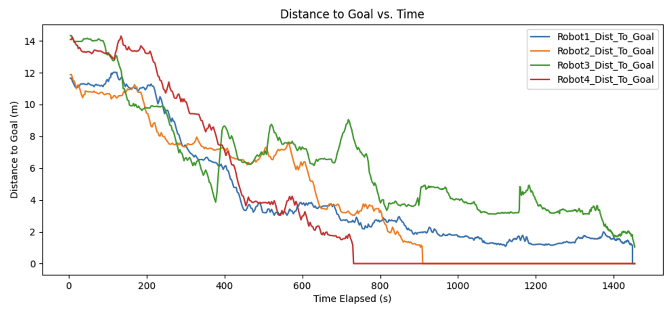

Fig. 2. Centralized MCTS: Distance to Goal vs Time

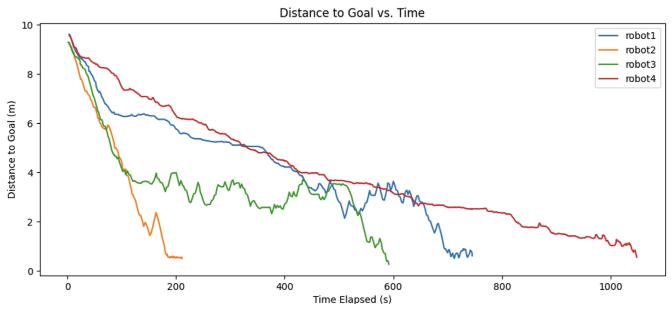

Fig. 3. Decentralized MCTS: Distance to Goal vs Time

VI. Individual Contributions

Nitish Ravisankar Raveendran

Nitish concentrated on using ROS 2 Humble to implement and analyze the centralized MCTS node. He worked to **modularize the codebase** and made sure that the centralized logic was successfully worked. The framework is adaptable for additional decentralized experimentation because of its flexibility, which makes it simple to modify the system to test various logics.

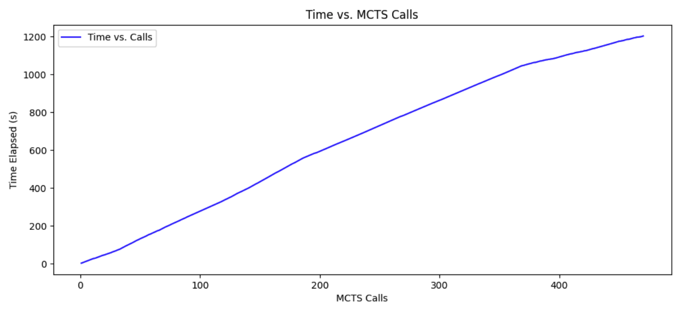

Fig. 4. Centralized MCTS: Time vs MCTS Calls

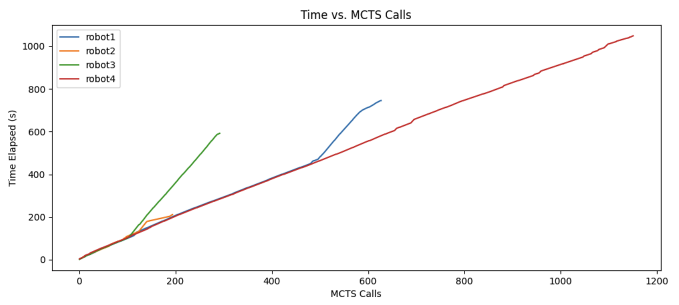

Fig. 5. Decentralized MCTS: Time vs MCTS Calls

Sai Jagadeesh Muralikrishnan

Sai Jagadeesh contributed by **reviewing existing work** in the field. He also **designed multiple custom simulation environments**, including static and dynamic obstacle setups with goal markers using ROS 2 Humble in Gazebo. These environments were carefully designed to mimic real-world applications and enabled robust testing of both centralized and decentralized algorithms under various conditions. Sai also performed detailed **testing and evaluation of the both algorithms by deriving the performance metrics** in these environments and studied those metrics for more refinement in parameter tuning and structure of algorithms.

Varun Lakshmanan

Varun focused on the implementation and **replication of the decentralized algorithm in ROS 2 Humble**. He worked on ensuring that the decentralized system functioned effectively without relying on peer-to-peer communication and **improved logic in moving obstacle avoidance**. His contributions included analyzing the performance of this non-peer-to-peer decentralized approach and comparing it to existing decentralized systems that incorporate peer-to-peer communication, providing valuable insights into its robustness and scalability.

VII. Applications

The centralized and decentralized MCTS algorithms have their strengths, making them suitable for different types of applications. Centralized MCTS works best in scenarios where coordination and global synchronization are critical, like managing robots in a warehouse or conducting search-and-rescue operations where tasks need to be highly organized. On the other hand, decentralized MCTS is a better fit for dynamic environments, such as delivery robots navigating busy city streets or exploration tasks in remote areas like planets or underwater. The decentralized approach handles these situations well because it allows each robot to act independently, adapting to obstacles or changes without needing constant coordination.

VIII. Future Scope

This project sets a better starting point for improving multi-robot navigation using MCTS. Moving forward, we plan to explore the following methods:

- Include parallel computation for the algorithm to work on multiple threads to increase the efficiency during the computation of MCTS iterations per planning step.
- Combining centralized and decentralized approaches to create a hybrid MCTS model that takes advantage of centralized coordination while maintaining the scalability and adaptability of decentralized methods.
- Testing our work in real-world scenarios with physical robots to see how well it handles larger and more complex environments, that makes sure if it's robust and reliable beyond simulations.

IX. Conclusion

This report presented a comparative study of centralized and decentralized MCTS algorithms for multi-robot navigation in dynamic environments. Our experiments demonstrated that while the centralized approach provides smoother trajectories and better global coordination, it suffers from scalability and computational overhead. On the other hand, the decentralized method exhibits greater flexibility, effectiveness, and scalability, which makes it more appropriate for large-scale, dynamic applications.

References

[1] VIS4ROB-lab, "Decentralised Monte Carlo Exploration," GitHub Repository, [Online]. Available: https://github.com/VIS4ROB-lab/dmce. [Accessed: Dec. 2024].

[2] S. Li, A. Gupta, and M. Johnson, "SCAN: Socially-Aware Navigation Using Monte Carlo Tree Search," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2023, pp. 1234–1240.

[3] D. Jain, R. Patel, and K. Tan, "SoRTS: Learned Tree Search for Long Horizon Social Robot Navigation," *arXiv preprint arXiv:2309.13144*, 2023.

[4] Robotis, "TurtleBot 3," [Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/. [Accessed: Dec. 2024].

[5] AI Boson, "Monte Carlo Tree Search (MCTS) Algorithm Tutorial," [Online]. Available: https://ai-boson.github.io/mcts/. [Accessed: Dec. 2024].

[6] T. Dam, G. Chalvatzaki, J. Peters, and J. Pajarinen, "Monte-Carlo Robot Path Planning," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 1234–1240, 2022.

[7] R. Fitch et al., "Dec-MCTS: Decentralized Planning for Multi-Robot Active Perception," *The International Journal of Robotics Research*, vol. 37, no. 13-14, pp. 1610–1626, 2018.

[8] W. Li, Y. Liu, Y. Ma, K. Xu, and J. Qiu, "A Self-Learning Monte Carlo Tree Search Algorithm for Robot Path Planning," *Frontiers in Neurorobotics*, vol. 17, 2023.

[9] M. Świechowski, K. Godlewski, B. Sawicki, and J. Mańdziuk, "Monte Carlo Tree Search: A Review of Recent Modifications and Applications," *Artificial Intelligence Review*, vol. 56, pp. 2497–2562, 2023.

Appendix

This appendix provides a comprehensive overview of the experiments, testing environments, performance metrics, and additional work carried out during this project.

A. *Experimentation Environments*

We conducted experiments in both simulation and real-world scenarios to evaluate the centralized and decentralized MCTS algorithms.

- **Empty World with Shared Goal:** In this scenario, four robots had to navigate an empty space in order to arrive at their shared goal. Testing how well these algorithms manage multi-robot collaboration in the absence of obstacles was the main goal as shown in Fig. 6.
- **Formation Testing:** Robots had to go to adjacent goal spots in order to make a square. This put their capacity to coordinate or independently plan routes while reaching a predetermined arrangement to the test as shown in Fig. 7.
- **Faraway Goal on Varying Terrain:** Goal points were positioned more than 100 meters apart in a difficult, uneven terrain setting for robot testing. The purpose of this experiment was to assess the algorithms' robustness in challenging situations. The decentralized method worked better because it gave each robot the freedom to manage local planning on its own, which sped up task completion. On the other hand, the computational burden of team-wide shared planning caused major delays for the centralized strategy. To view the video of the multi-robot being tested in a remote location on a variety of terrains, https://youtu.be/6PX75gFp8XA

B. *Performance Metrics*

Additional metrics were calculated to evaluate the algorithms in detail:

- **Distance to Goal vs MCTS Calls:** This metric evaluates how efficiently each algorithm plans during navigation as shown in Fig 8 and 9. **Centralized approach:** The centralized graph displays a single graph representing the average distance across all robots as the central node plans for the entire multi-robot system. While the distance to the goal decreases steadily with each MCTS call, this gradual convergence highlights the computational overhead of processing joint actions for all robots.

  Decentralized approach: The decentralized graph, on the other hand, highlights autonomous decision-making by displaying different paths for every robot. Due to their local operations and lack of centralized bottlenecks, robots show faster initial progress toward their goals. However, because local settings and interactions differ, robots perform differently, and some converge more quickly than others. Although this method scales successfully, there are times when it results in uneven development and small inefficiencies due to a lack of global coordination.

All the important graphs used in the main report and additional graphs, including these metrics, are included in this appendix for reference.

C. *PyQt5-Based User Interface*

algorithms' performance. With the help of this UI, users can explore several graphs, including the ones used in this report, and import the CSV file created during trials. For both centralized and decentralized systems, the user interface (UI) offers an easy-to-use method of analyzing parameters like task completion time, distance to target, and collision rate.

D. *Experiment Images and Graphs*

Below are images and graphs documenting the results of our experiments:

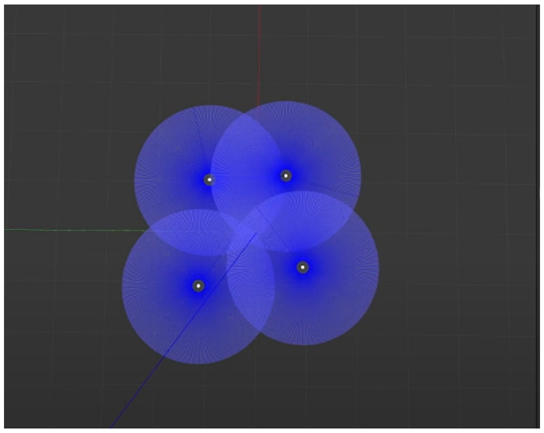
Fig. 6. Empty World Scenario: Four robots navigating to a shared goal.

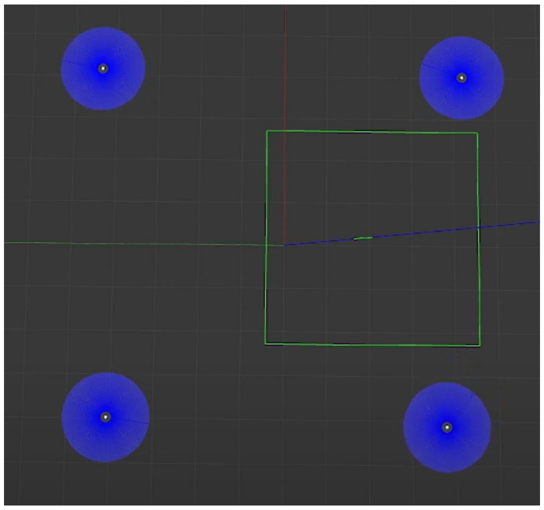
Fig. 7. Formation Testing: Robots forming a square by reaching adjacent goals.

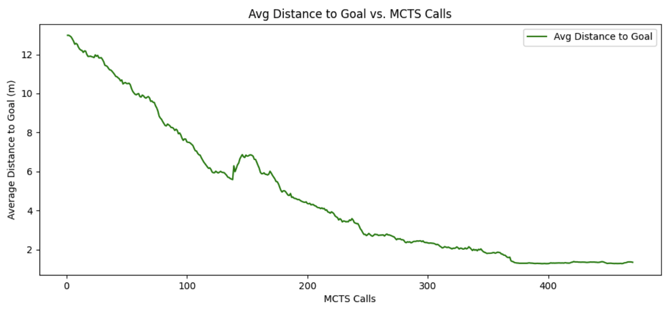

Fig. 8. Avg. Distance to Goal vs MCTS Calls of centralized algorithm.

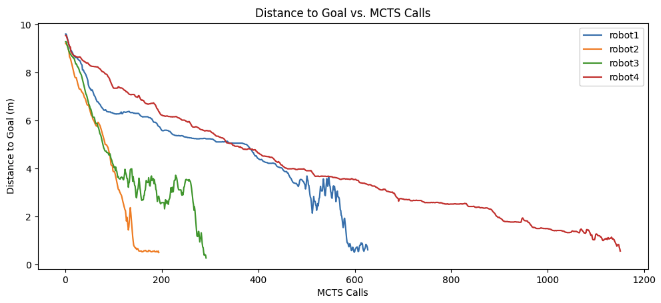

Fig. 9. Distance to Goal vs MCTS Calls decentralized algorithm.

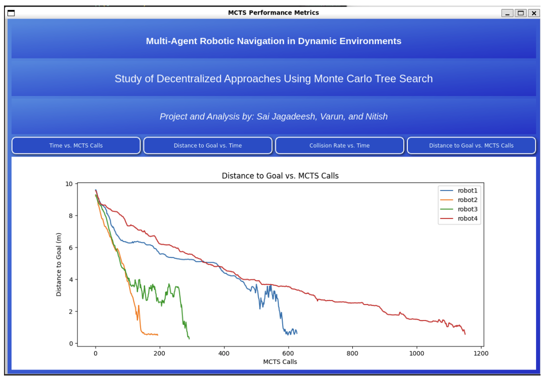
MCTS Performance Metrics
Multi-Agent Robotic Navigation in Dynamic Environments
Study of Decentralized Approaches Using Monte Carlo Tree Search
Project and Analysis by: Sai Jagadeesh, Varun, and Nitish
Time vs. MCTS Calls
Distance to Goal vs. Time
Collision Rate vs. Time
Distance to Goal vs. MCTS Calls
Distance to Goal vs. MCTS Calls
robot1
robot2
robot3
robot4
Distance to Goal (m)
MCTS Calls

Fig. 11. PyQt based UI: Distance to Goal vs MCTS Calls of decentralized algorithm.

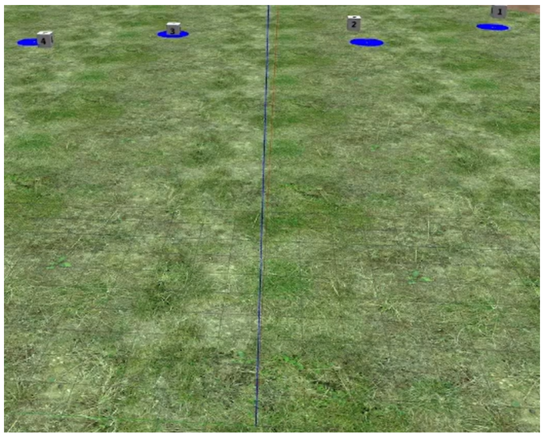

Fig. 10. Faraway goal testing on varying terrain: Robots navigating uneven terrain to reach goal points over 100 meters away.