Multi-Robot Navigation Using Sequential Decision Making Processes

Alexandre Carlhammar
Aeronautics and Astronautics Engineering
Stanford University
Stanford, CA
acarlham@stanford.edu

Arpit Dwivedi
Aeronautics and Astronautics Engineering
Stanford University
Stanford, CA
dwivedi7@stanford.edu

Rohan Garg
Mechanical Engineering
Stanford University
Stanford, CA
rohang73@stanford.edu

***Abstract*—This project focuses on decision-making for multi-robot navigation, with an emphasis on lunar exploration scenarios. The problem is incrementally addressed, beginning with a fully observable Markov Decision Process (MDP) framework using discrete states and actions. The complexity is progressively increased by transitioning to a Partially Observable Markov Decision Process (POMDP) framework with continuous observation spaces and discrete actions, ultimately culminating in fully continuous state and action spaces. The objective is to design a robust sequential decision-making pipeline that enables robots to reach designated goals while navigating uncertainty. Key elements include developing a simulation environment, implementing Extended Kalman Filter (EKF)-based state estimation, and leveraging advanced planning algorithms such as Monte Carlo Tree Search (MCTS), MCTS with Double Progressive Widening (MCTS-DPW), and Partially Observable Monte Carlo Planning with Double Progressive Widening (POMCP-DPW).**

***Index Terms*—Markov Decision Process, Partially Observable, Monte Carlo Tree Search, Progressive Widening**

I. Introduction

Multi-robot systems have gained significant attention for their potential in diverse applications, including search and rescue missions, warehouse automation, and planetary exploration. These systems offer scalability, robustness, and redundancy, making them suitable for tasks in dynamic and uncertain environments. However, effective coordination among robots requires overcoming several challenges, such as partial observability, uncertainty in sensor measurements, and complex decision-making in high-dimensional state and action spaces.

In this work, we address the problem of multi-robot navigation under partial observability, with a specific focus on scenarios where direct localization methods like GPS are unavailable, such as on the lunar surface. Robots must rely on onboard sensors, such as compass systems and Ultra-Wideband (UWB) radios, for state estimation and inter-robot communication. These sensors provide noisy observations, making the decision-making process inherently uncertain.

To tackle these challenges, we adopt a sequential decision-making framework that evolves from a fully observable Markov Decision Process (MDP) to a Partially Observable Markov Decision Process (POMDP). The latter allows for decision-making in environments where the state is not fully observable, leveraging belief states to represent uncertainty. Our approach incorporates advanced planning algorithms, including Monte Carlo Tree Search (MCTS) and Partially Observable Monte Carlo Planning (POMCP) with Double Progressive Widening (DPW). These methods balance exploration and exploitation, enabling robots to navigate effectively toward their goals while minimizing uncertainty.

This paper presents the following contributions:

- Development of a simulation environment that models realistic lunar navigation scenarios with partial observability and noisy sensor data.
- Implementation of an Extended Kalman Filter (EKF) for robust state estimation, fusing compass and UWB sensor readings.
- Implementation of MCTS and MCTS-DPW for simplified approach to the proposed problem.
- Design of a novel algorithm based on POMCP-DPW to act as a controller for efficient decision-making under uncertainty in multi-robot collaboration.
- Empirical evaluation of the proposed framework in both simplified MDP settings and complex POMDP scenarios, highlighting its scalability and performance.

The remainder of this paper is structured as follows: Section II provides a detailed problem statement and models the environment, sensors, and dynamics of the robots. Section III discusses related work, highlighting advancements in MDP, POMDP, and multi-robot navigation. Section IV outlines the methodology, including state estimation, planning, and control. Section V presents the results, and Section VI concludes with insights and directions for future work.

II. Problem Statement

The environment consists of n robots in a lunar surface exploration setting where each rover can communicate with every other rover. The state space is assumed to be two-dimensional and continuous. Each robot is equipped with two sensors:

The project repository is `Multi-robot-navigation`

- **Onboard Compass System**: Provides a noisy estimate of the robot's heading angle, where the noise is modeled as Gaussian.
- **Ultra-Wideband (UWB) Ranging Radios**: Provides inter-ranging distances between the robot and all other robots.

The objective of each rover is to reach its designated goal position while maintaining low uncertainty in its position estimates. The dynamics of the ith robot are governed by the unicycle model, with the state represented as:

$$X_i = [x_i, y_i, \psi_i],$$

where x_i and y_i are the robot's Cartesian coordinates, and ψ_i is the heading angle.

Since GNSS services are not available on the Moon yet, each robot does not know its exact global position. Instead, the robots rely on their noisy compass readings for heading and relative Euclidean distances obtained from the UWB sensor for localization.

A. Simplified MDP Problem

To simplify the problem, we first consider a fully observable Markov Decision Process (MDP) framework with the following setup:

- **State Space**: The x-y plane is discretized into a 20×20 grid, where each cell has dimensions of 1×1.
  - each robot occupies a single grid
  - the combined state space is written as $S = [x_1, y_1, x_2, y_2 .... x_{n_{robots}}, y_{n_{robots}}]$
  - hence $|S| = [grid\_size \times grid\_size]^{n_{robots}}$, we are not checking for collisions here.
  - in our case it becomes $|S| = 6.4 \times 10^7$, where $grid\_size = 20, n\_robots = 3$
- **Number of Robots**: $n = 3$ robots.
- **Action Space**: Each robot has the following possible actions:

$$A_{robot} = [up, down, right, left, stay]$$

  - the combined action of the system is written as

$$A = [a_1, a_2 .., a_{n_{robots}}]$$

  - hence $|A| = |A_{robot}|^{n_{robots}}$
  - in our case it becomes $|A| = 125$, where $n\_robots = 3, |A_{robot}| = 5$
- **Transition Model**: The transition probabilities for each robot's action are defined as follows:
  - With probability 0.15, the robot remains in its current position.
  - With probability 0.70, the robot chooses a random action from A_{robot}.
  - With probability 0.15, the robot moves to a neighboring cell in a direction not aligned with the chosen action.
- **Reward Model**: The reward for a state s is defined as the negative sum of the ℓ_1-norm between each robot's current position and its target goal:

$$R(s) = -\sum_{i=1}^{n} \|[x_i, y_i] - [x_{i,\text{goal}}, y_{i,\text{goal}}]\|_1.$$

B. Partially Observed Problem with Discrete Actions

In this setting, we consider a unicycle robot model to describe the motion of each robot. The kinematic equations governing the dynamics of the ith robot are given by:

$$\dot{x}_i = v_i \cos(\psi_i), \quad \dot{y}_i = v_i \sin(\psi_i), \quad \dot{\psi}_i = \omega_i,$$

where x_i and y_i are the Cartesian coordinates, ψ_i is the heading angle, v_i is the linear velocity, and ω_i is the angular velocity.

The control input for each robot is represented as $[v, \omega]$, where both v and ω are chosen from a discrete set of values within the range $[-0.5, 0.5]$. Each robot is equipped with two types of onboard sensors:

- **Ultra-Wideband (UWB) Ranging Radios**: These provide inter-robot distance measurements, producing $n(n-1)$ readings for n robots.
- **Onboard Compass System**: This provides noisy estimates of the current heading angles for each robot, producing n readings for n robots.

The sensor readings are processed by an Extended Kalman Filter (EKF) module, which fuses the noisy sensor measurements with predicted dynamics to estimate the current state of each robot. This estimated state is represented as:

$$\hat{X}_i = [\hat{x}_i, \hat{y}_i, \hat{\psi}_i].$$

The EKF output of estimated means and covariance of state values is then fed into a controller module, which utilizes a Partially Observable Monte Carlo Planning framework with Double Progressive Widening (POMCP-DPW) to compute the optimal action for each robot. The transition dynamics in this formulation follow the unicycle model described above, while the observation vector corresponds to the EKF state estimates.

Reward Model: The reward model remains the same as in the fully observable case:

$$R(s) = -\sum_{i=1}^{n} \|[x_i, y_i] - [x_{i,\text{goal}}, y_{i,\text{goal}}]\|_1.$$

III. Related Work

The problem of multi-robot navigation with uncertainty has been widely explored, particularly in the context of POMDPs. The work by Tzikas et al. [3] investigates a multi-robot navigation problem with sensor observations. They extended the standard POMDP framework by introducing belief-based rewards, where the reward depends not just on the robot's

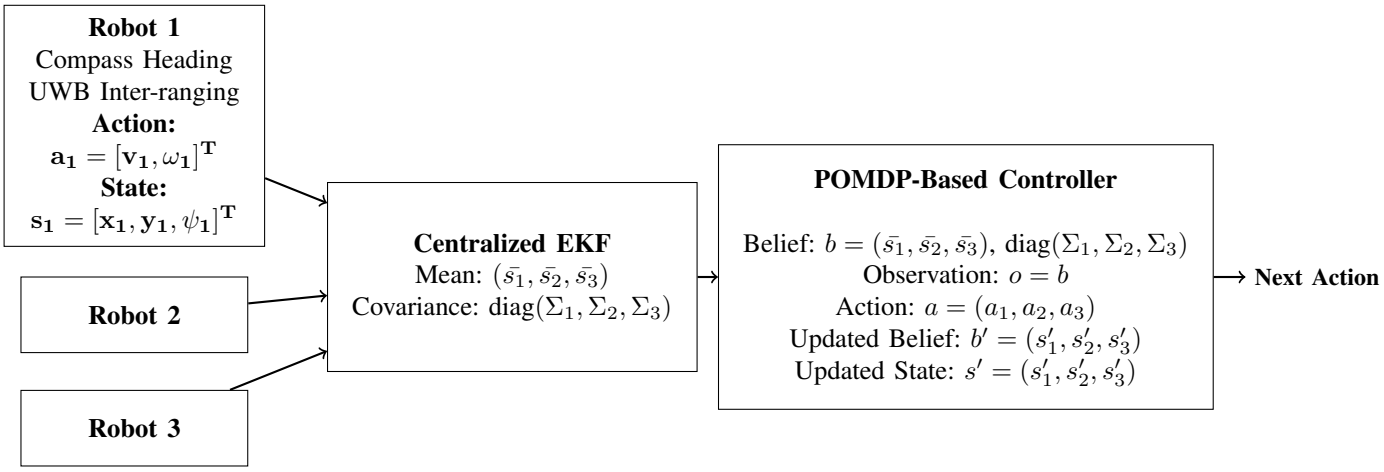

Fig. 1: System architecture illustrating the multi-robot communication system. Robots use onboard sensors to provide measurements, which a centralized EKF processes to estimate state and covariance. The POMDP-based controller then computes the optimal actions for the robots and passes them to the next step.

actions but also on its belief about its state in the environment. The authors proposed a formulation for multi-robot systems in which the individual robots do not have access to the true state of the environment. Instead, they maintain a belief over the possible states of the system, and the reward model is designed to help reduce uncertainty in this belief. The approach is evaluated on a simulation task where robots must collaborate to reach goal locations while accounting for sensor noise and other sources of uncertainty.

A. Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a popular method used to solve decision-making problems in partially observable and stochastic environments, particularly in the context of discrete state and action spaces as shown in Chang et al. [1]. MCTS is a model-free, simulation-based algorithm that incrementally builds a search tree by performing random simulations of the environment. Each simulation generates a sequence of actions and states, and based on the results, the algorithm updates the tree with statistics that guide future search. The key advantage of MCTS in the discrete setting is its ability to balance exploration and exploitation by selecting actions with the best expected outcomes while exploring different possibilities. In discrete state and action space problems, MCTS can be particularly effective due to the relatively smaller search space, which allows for efficient sampling and real-time decision-making. The algorithm has been successfully applied to a range of problems, from board games like Go to multi-agent systems, providing robust solutions in complex environments with large branching factors.

B. MCTS-DPW

The *MCTS-DPW* (Monte Carlo Tree Search with Double Policy Search) algorithm is an extension of the basic MCTS approach, designed to improve performance by combining MCTS with policy search techniques. In the context of discrete state, and action spaces, MCTS-DPW optimizes the policy directly through a sequence of tree searches, rather than relying solely on value iteration or policy evaluation. The key idea is to use MCTS to explore different sequences of actions and their consequences, and then apply direct policy search to refine the policy based on the observed results. This approach allows for better handling of large and complex action spaces, where traditional value-based methods may struggle. By incorporating policy search into the MCTS framework, MCTS-DPW can effectively handle the combinatorial complexity of multi-robot systems, where robots must consider not only their actions but also the interactions with other robots and the environment.

C. POMCP-DPW

The *POMCP-DPW* (Partially Observable Monte Carlo Planning with Double Progressive Widening) algorithm, introduced by Sunberg and Kochenderfer [2], offers an effective approach for solving large-scale POMDP problems. This algorithm extends MCTS by incorporating the technique of Double Progressive Widening (DPW), which addresses the challenge of continuous state, action, and observation spaces. DPW dynamically adjusts the search tree's expansion, initially focusing on the most promising actions and gradually broadening the search as more information becomes available. The key innovation of POMCP-DPW is its use of a belief-based search strategy, where the belief state (a probability distribution over possible states) is updated through Monte Carlo simulations. Actions are selected based on the expected rewards derived from these simulations. This algorithm balances exploration and exploitation more effectively, progressively expanding the search space in a controlled manner. This is particularly advantageous in real-time decision-making scenarios, where the state space is large, continuous and partially observable.

IV. METHODOLOGY

We approached solving the problem first for the simplified case and then partially observed case with discrete action space. The detailed discussion of approach taken is as follows:

A. Approach: MCTS for Simplified MDP Problem

Simplified MDP problem formulated in Section II-A was solved using the Monte Carlo Search Tree with UCB1 exploration. The results for this approach are shown in Section V-A

Algorithm 1 MDP with Monte Carlo Tree Search

Input: π: MDP , time steps : t, number of simulations: m
Output: $final\_position$
Procedure *Controller(initial_position), π, t, m*:
  $\pi.robot\_positions() \leftarrow initial\_position$
  for $i \in 1:t$ **do**
    $s \leftarrow \pi.robot\_positions()$;
    $a \leftarrow$ MCTS(π, s, m)
    π.robot_positions() $\leftarrow \pi.step(s,a)$
  return $\pi.robot\_positions()$
  Input: π: Problem , state : s, iterations : m, depth: d
  Output: a
  Procedure *MCTS(π, s, d, m)*:
    for $i \in 1:m$ **do**
      if $d = 0$ **then**
        return 0
      $s \leftarrow ucb\_explore(\pi, \text{s})$;
      $s', r = \pi.step(s,a) + \pi.reward(s,a)$;
      $q = r + \pi.\gamma \times MCTS(\pi, s, d-1, m)$:
      $\pi.N[(s,a)] += 1$;
      $\pi.Q[(s,a)] = [\pi.Q[(s,a)] * (\pi.N[(s,a)] - 1) + q]/\pi.N[(s,a)]$
    return $\arg\max_{a \in P.A} Q(s,a) + c\sqrt{\frac{\log N(s)}{N(s,a)}}$

B. Approach: POMCP-DPW for POMDP Problem

This subsection will explain our multi-robot navigation method. We will start with mentioning the details of the environment, followed by the details of the EKF state estimation module and then formulate our proposed POMDP-based controller. The proposed architecture for the Partially observable case is shown in Fig 1.

1) Environment: The environment simulates a multi-robot system consisting of $n = 3$ robots, where each robot operates on a unicycle model with states (x, y, θ) representing its position and orientation. The position and orientation of robots is initialized at positions [[0, 1, 0], [0.2, 1, 0], [0.3, 1, 0]].$Robotsevolveoverdiscretetimesteps(\Delta t = 0.1$ seconds) based on velocity v and angular velocity w, with dynamics incorporating Gaussian noise ($\sigma_{\text{dyn}} = 0.05$) to model real-world uncertainties. The values of velocity v and angular velocity w are calculated by the POMDP controller and fed into the environment to update the position of the robots. The environment provides two types of observations: pairwise distances between robots, derived using noisy UWB sensors with a standard deviation of 0.01, and orientation measurements corrupted by compass noise with a standard deviation of 0.05. Observations are structured as vector $\boldsymbol{o} \in \mathcal{R}^{n^2}$ combining the inter-robot distances and their noisy compass readings. This setup enables testing POMDP-based decision-making processes in scenarios with stochastic sensors and motion models.

Algorithm 2 Extended Kalman Filter (EKF)

Input: Control actions at time step $t : \in \mathcal{R}^{2n}$
Estimated robots positions at time step $t-1$: $\hat{\mathbf{s}}_{t-1|t-1} : \in \mathcal{R}^{3n}$
Inter-ranging measurements at time step t: $\in \mathcal{R}^{n(n-1)}$
Compass measurements at time step t:$\in \mathcal{R}^{n}$
while *true* **do**
  Predict step:
  $\mathbf{a}_t \leftarrow$ control actions at time step t
  $\hat{\mathbf{s}}_{t|t-1} \leftarrow \hat{\mathbf{s}}_{t-1|t-1} + B\mathbf{a}_t$
  $\mathbf{P}_{t|t-1} \leftarrow F_t \mathbf{P}_{t-1|t-1} F_t^\top + \mathbf{Q}$

  Update step:
  $\mathbf{z}_t \in \mathcal{R}^{n^2} \leftarrow$ inter-ranging measurements, compass measurements at time step t
  $\tilde{\mathbf{y}}_t \leftarrow \mathbf{z}_t - h(\hat{\mathbf{s}}_{t|t-1})$
  $\mathbf{K}_t \leftarrow \mathbf{P}_{t|t-1} H_t^\top (R + H_t \mathbf{P}_{t|t-1} H_t^\top)^{-1}$
  $\hat{\mathbf{s}}_{t|t} \leftarrow \hat{\mathbf{s}}_{t|t-1} + \mathbf{K}_t \tilde{\mathbf{y}}_t$
  $\mathbf{P}_{t|t} \leftarrow (\mathbf{I} - \mathbf{K}_t H_t)\mathbf{P}_{t|t-1}(\mathbf{I} - \mathbf{K}_t H_t)^\top + \mathbf{K}_t R \mathbf{K}_t^\top$

2) State Estimation: After the control inputs are fed into the environment to update the robot states. We utilize the sensor readings $\boldsymbol{o}$ and the EKF algorithm to estimate the current states of the robots. The algorithm 2 explains the implementation in detail. The Extended Kalman Filter (EKF) is initialized with the following parameter values: the number of robots ($n = 3$) and standard deviations for Ultra-Wideband (UWB) sensor noise ($\sigma_{\text{uwb}} = 0.01$) and compass noise ($\sigma_{\text{compass}} = 0.05$) are inherited from the environment. The UWB covariance ($\mathbf{R}$) is structured such that the first $n(n-1)$ diagonal elements correspond to inter-ranging measurements with variance σ^2_{uwb}, and the last n diagonal elements represent compass heading measurements with variance $\sigma^2_{\text{compass}}$. The process noise covariance matrix ($\mathbf{Q}$) is set to $0.01 \cdot \sigma^2_{\text{dyn}} \cdot \mathbf{I}_{3n}$, and the initial state covariance matrix ($\mathbf{P}_{t-1}$) is initialized as $0.0001 \cdot \mathbf{I}_{3n}$, where $\mathbf{I}_{3n}$ is a $3n \times 3n$ identity matrix. The initial state vector ($\mathbf{s}_0$) is taken as the ground truth state provided by the environment.

Algorithm 3 Common Procedures : POMDP

Input: Belief b, and a maximum depth $d_{\max}$
Output: Optimal action sequence
Procedure *PLAN(b)*:
  for $i \in 1 : n_s$ **do**
    $s \leftarrow$ sample from b
    SIMULATE(s, b, $d_{\max}$)
  return $\arg\max_a Q(ba)$
Procedure *ActionProgWiden(h)*:
  if $|C(h)| \leq B_a$ **then**
    $a \leftarrow$ NEXTACTION(h) $C(h) \leftarrow C(h) \cup \{a\}$
  return $\arg\max_{a \in C(h)} Q(ha) + c\sqrt{\frac{\log N(h)}{N(ha)}}$

Algorithm 4 POMCP-DPW

Input: Current state s, history h, depth d
if $d = 0$ **then**
  return 0
$a \leftarrow$ `ActionProgWiden(`h`)`
if $|C(ha)| \leq B_o$ **then**
  $s', o, r \leftarrow G(s, a)$
  $C(ha) \leftarrow C(ha) \cup \{o\}$
  $M(hao) \leftarrow M(hao) + 1$
  Append s' to $B(hao)$
  if $M(hao) = 1$ **then**
    $total \leftarrow r + \gamma$ `Rollout(`$s', hao, d-1$`)`
  else
    $total \leftarrow r + \gamma$ `Simulate(`$s', hao, d-1$`)`
else
  $o \leftarrow$ select $o \in C(ha)$ w.p. $\frac{M(hao)}{\sum_o M(hao)}$
  $s' \leftarrow$ select $s' \in B(hao)$ w.p. $\frac{1}{|B(hao)|}$
  $r \leftarrow R(s, a, s')$
  $total \leftarrow r + \gamma$ `Simulate(`$s', hao, d-1$`)`
$N(h) \leftarrow N(h) + 1$
$N(ha) \leftarrow N(ha) + 1$
$Q(ha) \leftarrow Q(ha) + \frac{total - Q(ha)}{N(ha)}$
return $total$

3) Controller: For sequential decision-making in this dynamic environment, we implemented a POMCP-DPW based planner as explained in Algorithm 5. This algorithm is explained in detail by Sunberg and Kochenderfer [2]. At each time step, the EKF estimates the current state and covariance using sensor readings and the previous state, providing a belief over each robots' positions. The closest observation node to the sensor reading is identified to maintain consistency, and the tree is updated to reflect the new root based on this observation value. These estimates are passed to the POMDP planner, explained in Algorithm 3 & 4, to determine the optimal next action while updating the belief tree's node (history tree) with new observations.

However, we have modified the implementation proposed by Sunberg and Kochenderfer [2]. The modified algorithm is explained in Algorithm 5. The selected action is then executed in the environment, and its corresponding node is added to the tree as a child of the current observation node. The tree is pruned to select the branch having best action. This process repeats iteratively until the robots reach close enough to the destination. This high-level loop combines EKF's accuracy with POMDP's strategic planning for robust real-time decision-making.

$G(s, a)$ is a generative model that outputs the robot's next state if it's currently at state s and took action a. The *ROLLOUT* policy is a simple direct-to-go planner from that state and taking actions up to depth d.

Algorithm 5 EKF and POMDP-based Action Selection

Input: Sensor readings, Previous control actions, previous state estimate
Output: Runs till last time step
while $i < n_{steps}$ **do**
  $\mathbf{s}_{t-1} \leftarrow$ EKF state estimate
  Env_readings $\leftarrow$ Env.get_states()
  $\mathbf{z}_t \leftarrow$ Env.get_observations()
  if $i \geq 1$ **then**
    Find closest observation node $\mathbf{o}_{\text{closest}}$ to $\mathbf{z}_t$
    Update observation of $\mathbf{o}_{\text{closest}}$ with $\mathbf{z}_t$
    Detach and make $\mathbf{o}_{\text{closest}}$ root of history tree
  else
    Add initial observation node as child of action node
    Set as root of history tree
  Env.step(a_{prev})
  $\mathbf{s}_{\text{EKF}}, \mathbf{P} \leftarrow$ EKF.estimate($\mathbf{z}_t, \mathbf{s}_{t-1}, a_{\text{prev}}$)
  Pass $\mathbf{s}_{\text{EKF}}$ and $\mathbf{P}$ to POMDP planner
  a_{next}, updated_tree $\leftarrow$ PLAN($\mathbf{s}_{\text{EKF}}, \mathbf{P}$, history_tree)
  Add a_{next} as child of current observation node
  Prune history tree to focus on subtree of a_{next}
  Reset action node and $a_{\text{prev}} \leftarrow a_{\text{next}}$
  $i \leftarrow i + 1$

V. Results

A. Simplified MDP Problem

As outlined in Section II-A, we consider a scenario with $n_{\text{robots}} = 3$, where the initial positions of the robots are depicted in Figure 2. In this figure, the position marked as R_i represents the initial position of robot i, and G_i indicates its assigned goal.

1) MCTS: With the parameters set as $n_{\text{iterations}} = 100$, $d = 10$, $m = 100$, and $c = 6$, the final positions of the robots are illustrated in Figure 3. To evaluate the performance, we compare the MCTS policy against a direct-to-goal policy and

a random policy. Each policy is tested under identical initial and goal positions. We conduct 20 rollouts and compute the average reward, plotting the results along with the 95% confidence interval over iterations or time steps (Figure 4). Initially, the MCTS policy demonstrates behavior similar to the direct-to-goal policy. However, at later time steps, the MCTS policy does not converge precisely to the goal but rather stabilizes near the target positions. The confidence interval of the reward for the MCTS policy exhibits greater variability, indicating stochastic behavior. In contrast, the direct-to-goal policy shows minimal stochasticity, with minor variations due to the inherent randomness in the environment's transition function. These observations highlight the trade-offs in performance and behavior between the tested policies.

2) MCTS:DPW: In MCTS-DPW, we introduce action widening, similar to the procedure **ActionProgWiden()** described in Algorithm 3, to enhance our base MCTS tree. However, for MCTS implementation we have utilized variable widening bound defined by $k_a(N(s))^{\alpha_a}$ instead of fixed bound B_a. Further details about this integration are provided in Section III-B. State widening results are not presented here, as they showed minimal performance improvements during experimentation.

To evaluate the advantages of Double Progressive Widening (DPW), we conducted experiments by varying the tree depth (**d**) and the number of simulations (**m**), while keeping the exploration constant (**c**) and the number of environment time steps identical.

In Figure 5, we consider a smaller tree depth with fewer simulations to populate the MCTS tree. Under these conditions, we observe a widening performance gap between the direct policy and MCTS. This can be attributed to the tree not exploring the state-action space thoroughly. As anticipated, MCTS-DPW outperforms the standard MCTS since, at each state node, it initially explores a subset of the $|A| = 125$ possible actions. As the node is visited more frequently, the algorithm incrementally explores more actions. For MCTS-DPW, the values chosen for parameters $k_a = 40$ and $\alpha_a = 0.5$. These results highlight the potential advantages of MCTS-DPW in continuous state and action spaces.

In a subsequent experiment, we increase the tree depth and the number of simulations. As expected, this allows the MCTS tree to explore the space more comprehensively by evaluating all actions at each node. Consequently, the algorithm is better able to identify the optimal action at each time step. The results of this run are illustrated in Figure 6.

B. POMDP Problem

We execute Algorithm 5 for 1000 time steps. Figures 7 and 8 illustrate the trajectories of all three robots simulated under the proposed framework for 2 different settings.

The controller employs several key parameters to balance exploration and exploitation. The discount factor $\gamma = 0.95$ encourages long-term rewards, while $B_a = 65$ and $B_o = 50$ regulate progressive widening for actions and observations. The exploration constant $c_{\text{exploration}} = 1$ governs the tradeoff between exploration and exploitation. These parameters guide the planning process, with a rollout depth of $d = 12$ and $n_s = 550$ simulations for optimal action selection. These values remain constant across the two test cases. The target goal for all 3 robots is set to $(7, 7)$ in the 2 test cases.

In the first test case, the robots are initialized at three distinct positions on the surface: $[[0, 1, 0], [0.2, 1, 0], [0.3, 1, 0]]$. The results show that the robots rapidly converge to the target position, with their trajectories stabilizing after 400 to 800 iterations. During this process, the EKF consistently provides highly accurate state estimations.

In the second test case, the robots start at the same initial positions, but each robot is assigned a different target position: $[[7, 5], [5, 7], [2, 7]]$. Over the course of 1000 time steps, the robots are observed to converge to regions near their respective target positions. However, their convergence is slower and less precise compared to the case where all robots are given the same target. This suggests that when the robots share a common target (case 1), their goals are more closely aligned, allowing them to focus more on the secondary objective of reducing state uncertainty, while the primary objective of reaching the target is achieved more efficiently.

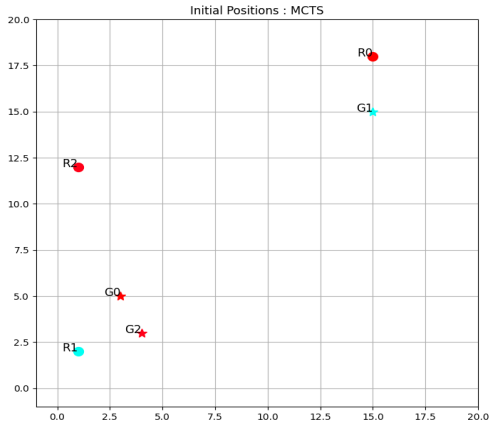

Fig. 2: Initial Position of the Robots in Simplified Problem

VI. Conclusions

This paper presents a robust framework for multi-robot navigation in partially observable environments, focusing on lunar exploration scenarios where GPS-like services are unavailable. By systematically transitioning from a fully

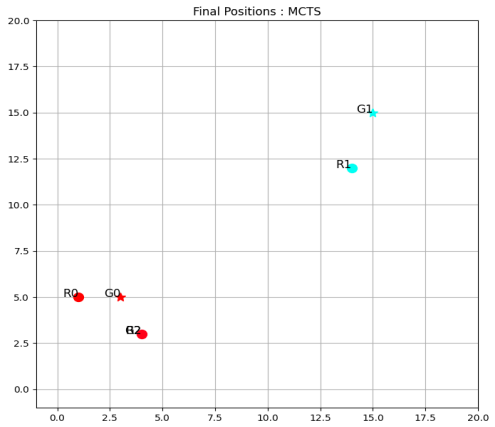

Fig. 3: Final Position of the Robots from MCTS

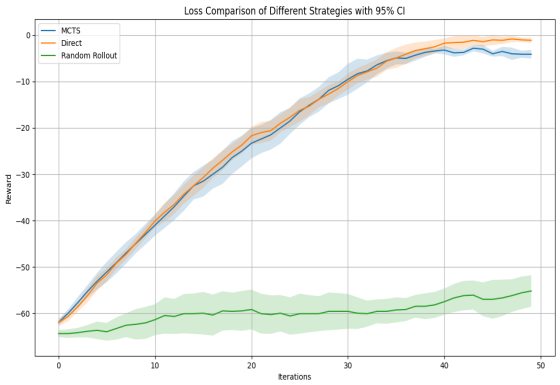

Fig. 4: 95% Confidence Interval of Trajectory from multiple runs of MCTS

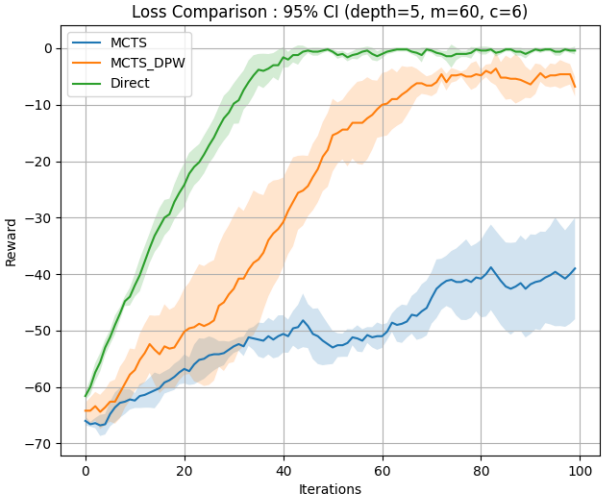

Fig. 5: 95% Confidence Interval with MCTS-DPW : smaller depth and simulations

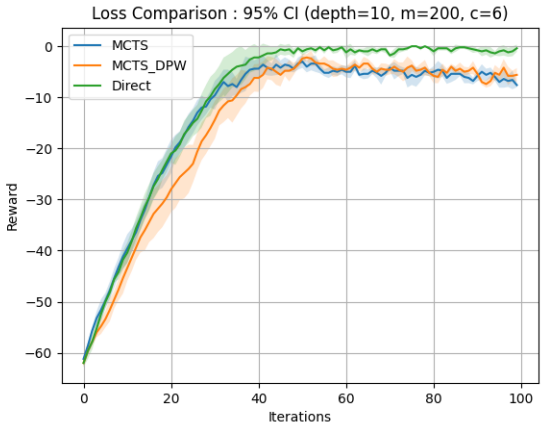

Fig. 6: 95% Confidence Interval with MCTS-DPW : larger depth and simulations

observable MDP framework to a partially observable POMDP framework, we address challenges associated with uncertainty in state estimation and action planning.

The results demonstrate that integrating Extended Kalman Filters (EKFs) with POMCP-DPW provides accurate state estimates while enabling efficient decision-making under uncertainty. The POMDP-based controller proved effective in guiding robots toward their goals, even in dynamic and noisy environments. The progressive widening approach further enhances computational efficiency, allowing the framework to scale to larger state and action spaces.

Our experiments validated the framework's ability to handle complex scenarios, including collaborative decision-making among robots with noisy inter-robot communications. The comparative analysis of policies highlighted the advantages of the proposed approach over baseline methods, showing significant improvements in convergence speed and goal attainment under uncertainty.

These findings demonstrate the potential of this framework for real-world applications, including space exploration and other environments where robust, decentralized navigation is critical.

VII. Future Works

Several steps can be taken to enhance the overall realism of the simulation. Analyzing results under these more realistic conditions could provide new valuable insights.

First, the environment can be made more complex by randomly distributing obstacles of varying sizes for example to simulate rocks. Additionally, terrain features such as gradients, hills, and crevices could be modeled with a random distributions or derived from a true terrain map. The reward function would need to adapt to this enhanced environment,

penalizing robots for getting stuck or colliding with obstacles.

Second, realistic communication between robots can be modeled by limiting the effective communication range. Instead of each robot being able to observe the relative positions of all $n1$ other robots, they would only detect robots within a defined radius. This constraint could encourage the policy to consider an additional objective: maintaining proximity to other robots to facilitate collaboration.

CONTRIBUTION

A. Alexandre Carlhammar

Problem Formulation, Engineering Solution, Code Debugging, Writing Report, Literature Survey

B. Arpit Dwivedi

Problem Formulation, Engineering Solution, Code Debugging, Writing Report, Literature Survey

C. Rohan Garg

Problem Formulation, Engineering Solution, Code Debugging, Writing Report, Literature Survey

REFERENCES

[1] Hyeong Soo Chang, Michael C Fu, Jiaqiao Hu, and Steven I Marcus. A survey of some simulation-based algorithms for markov decision processes. 2007.

[2] Zachary Sunberg and Mykel Kochenderfer. Online algorithms for pomdps with continuous state, action, and observation spaces. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 28, pages 259–263, 2018.

[3] Alexandros E Tzikas, Derek Knowles, Grace X Gao, and Mykel J Kochenderfer. Multirobot navigation using partially observable markov decision processes with belief-based rewards. *Journal of Aerospace Information Systems*, 20(8):437–446, 2023.

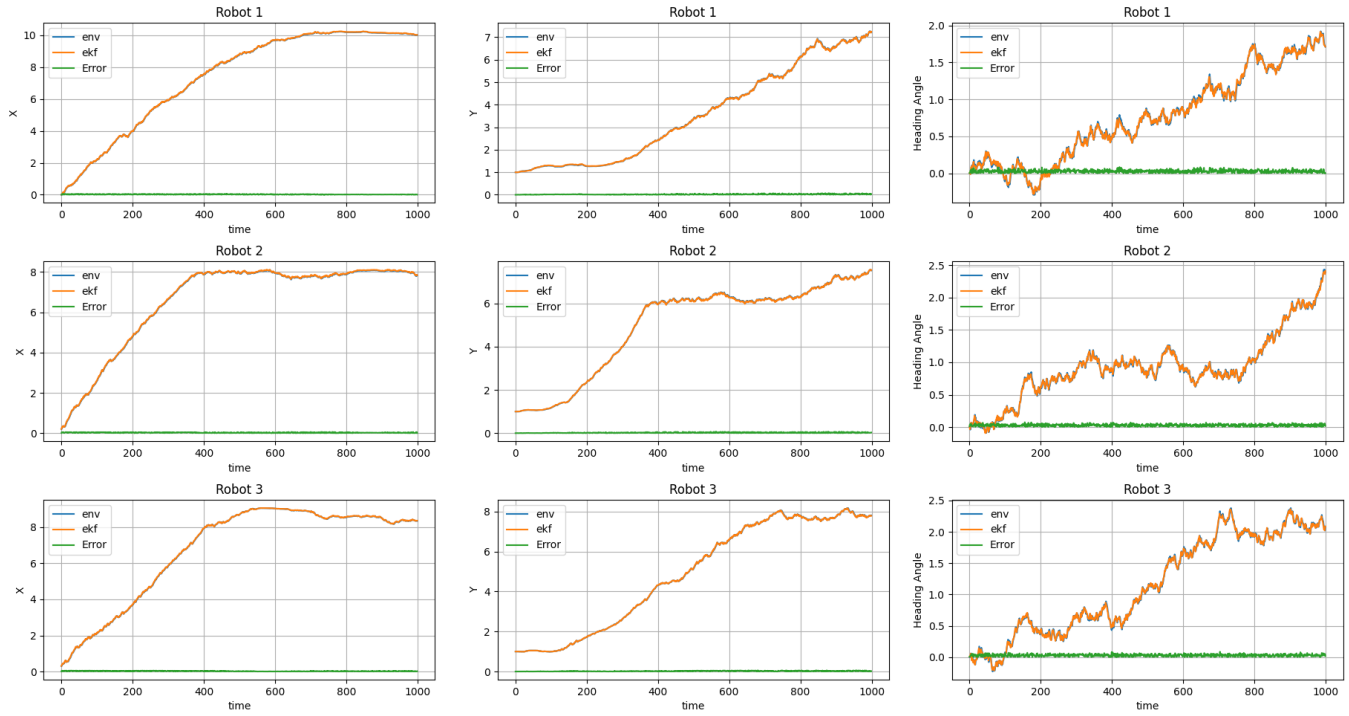

Fig. 7: Robot Trajectories with POMCP-DPW Controller. The initial positions and orientation of robots were set as $[[0, 1, 0], [0.2, 1, 0], [0.3, 1, 0]]$ and the target goal position for all robot was set to $[7, 7]$

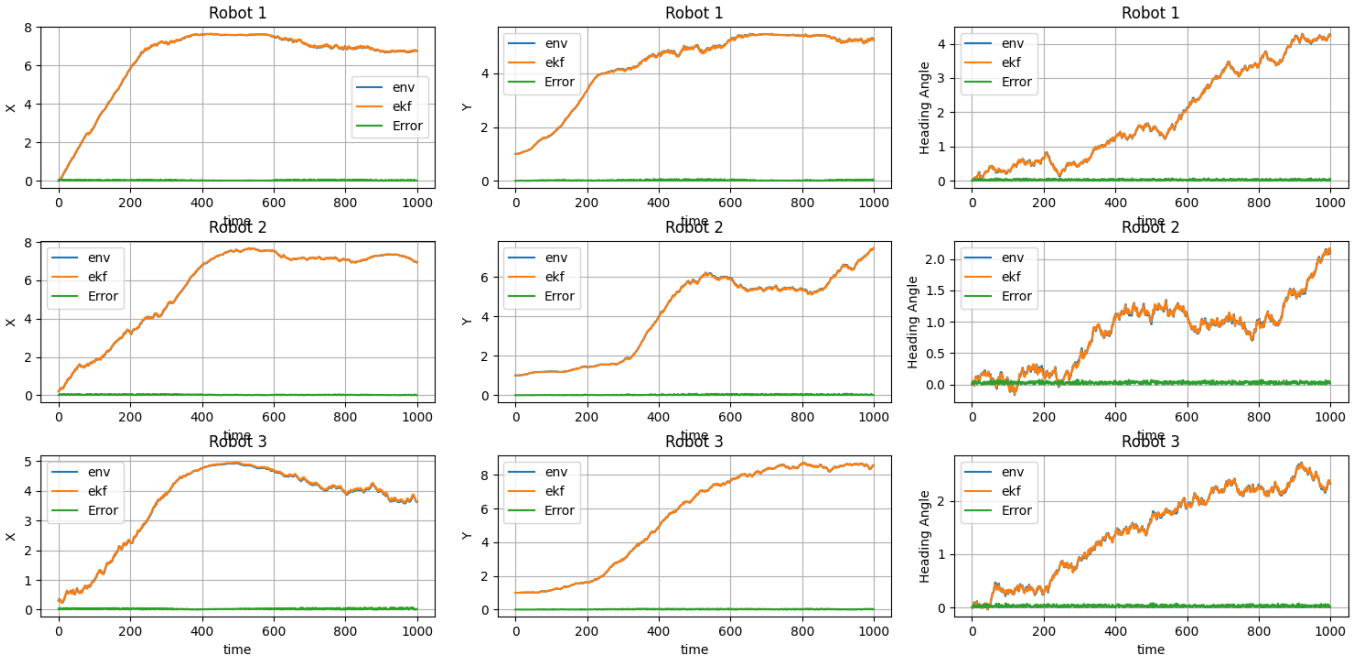

Fig. 8: Robot Trajectories with POMCP-DPW Controller. The initial positions and orientation of robots were set as $[[0, 1, 0], [0.2, 1, 0], [0.3, 1, 0]]$ and the target goal position for all robot was set $[[7, 5], [5, 7], [2, 7]]$