Autonomous Robots (2019) 43:789–811
https://doi.org/10.1007/s10514-018-9811-9

Real-time distributed non-myopic task selection for heterogeneous robotic teams

Andrew J. Smith[1] · Graeme Best[1] · Javier Yu[2] · Geoffrey A. Hollinger[1]

Received: 13 April 2018 / Accepted: 6 October 2018 / Published online: 7 November 2018

Abstract
In this paper we introduce a novel algorithm for online distributed non-myopic task-selection in heterogeneous robotic teams. Our algorithm uses a temporal probabilistic representation that allows agents to evaluate their actions in the team's joint action space while robots individually search their own action space. We use Monte-Carlo tree search to asymmetrically search through the robot's individual action space while accounting for the probable future actions of their team members using the condensed temporal representation. This allows a distributed team of robots to non-myopically coordinate their actions in real-time. Our developed method can be applied across a wide range of tasks, robot team compositions, and reward functions. To evaluate our coordination method, we implemented it for a series of simulated and fielded hardware trials where we found that our coordination method is able to increase the cumulative team reward by a maximum of 47.2% in the simulated trials versus a distributed auction-based coordination. We also performed several outdoor hardware trials with a team of three quadcopters that increased the maximum cumulative reward by 24.5% versus a distributed auction-based coordination.

Keywords Heterogeneous robotic teams · Non-myopic coordination · Robotic planning · Robotic coordination · Robotic fielded hardware trials

1 Introduction

The capability to coordinate teams of robots across a wide variety of tasks is a critical concern at the core of many robotic applications. As teams of robots take an increasing role in agriculture (Ayanian et al. 2017); assisting the military, police, fire-fighters, and search and rescue (Beck et al. 2016; Petersen et al. 2013); monitoring our environment (Smith et al. 2010); and producing goods (Liu and Chopra 2009), it will become increasingly important to ensure that robots are able to operate not just as individuals but as members of a cohesive team. This problem has been widely studied for nearly 3 decades (Fukuda et al. 1989) and continues to be a promising area of research (Ayanian et al. 2017). This high-level of interest is due to the inherent advantages that multi-robot systems offer; such as: the ability to take distributed actions, being robust to single point failures, and frequently requiring multiple different algorithmically and mechanically simpler individual robots instead of a single robot capable of completing every task (Arkin and Balch 1998).

Distributing planning authority among the team can further extend the advantages of multi-robot systems. For example, distributed teams increase system robustness by removing the dependence upon constant communication to a central controller. This provides two benefits: First, it removes the possibility of a single point failure if the cen-

This is one of the several papers published in *Autonomous Robots* comprising the Special Issue on Foundations of Resilience for Networked Robotic Systems.

This work was supported by NASA Grant NNX14AI10G and Office of Naval Research Grant N00014-17-1-2581.

✉ Andrew J. Smith
smithan7@oregonstate.edu

Graeme Best
bestg@oregonstate.edu

Javier Yu
javieryu@stanford.edu

Geoffrey A. Hollinger
geoff.hollinger@oregonstate.edu

1 Collaborative Robotics and Intelligent Systems (CoRIS) Institute, School of Mechanical Industrial and Manufacturing Engineering, Oregon State University, Corvallis, OR, USA

2 School of Engineering, Stanford University, Stanford, CA, USA

tral controller fails. Second, this frees the team to operate outside of communication range with the central controller and allows them to continue in the event of lost communication with either the central controller or even the entire team. However, these advantages come at the expense of increasing the difficulty of coordinating the team's actions. This is because distributed teams lack a central authority to dictate the actions of each robot and have planning uncertainty between team members that are attempting to coordinate.

Varying team member capabilities (heterogeneity) is another way to increase the capabilities of a team of robots. Heterogeneous teams can be formed to include multiple types of specialized, yet mechanically and algorithmically simpler, robots that individually excel at a small set of tasks instead of a general robot capable of completing every task. However, heterogenous teams may further complicate team coordination as it requires accounting for the abilities of each robot when planning each robot's actions. A solution is to frame the heterogeneous coordination in a way that is agnostic to team member capabilities, allowing the inclusion of wide ranging ability in team members with minimal modification.

A common consideration in robotic planning is whether to perform online or offline planning. Offline planning methods find a complete policy of actions before executing a single action. Online planning methods act and plan in parallel by making decisions in real-time. Online planning generally enables reacting to unpredictable changes in the environment and team composition, while offline planning typically enables searching deeper into the solution space. Consequently, most fielded systems are either offline and centralized, or online and myopic.

In this work we present an online distributed non-myopic method of coordinating teams of heterogeneous multi-robot systems performing general robotic tasks. Our approach uses a novel probabilistic and temporal representation of each robot's potential future actions, named the *probable action timeline*, to communicate their intent to the remainder of the team. A probable action timeline defines a cumulative probability distribution for the belief of the time any robot may claim a task. Each robot re-plans while accounting for the probable actions of their teammates to maximize their contribution to the team's global reward. Each robot searches for a sequence of macro-actions using Monte-Carlo tree search (MCTS) (Browne et al. 2012) to asymmetrically search through the macro-action space and a variant of difference rewards (Tumer and Agogino 2009) to calculate their impact on the team's cumulative reward. Macro-actions include all of the individual actions required to complete a single task; e.g., traveling to the task location and performing the task, with a distribution over the expected completion time. To account for non-stationary rewards in the action space resulting from the actions of team members and changes in the environment we use the Sliding-Window Upper Confidence Bound (SW-UCB) (Garivier and Moulines 2011) search heuristic within MCTS; we refer to this new MCTS variant as SW-UCT. Then, each robot samples their tree and broadcasts the temporal and probabilistic representation of their intended actions to the team to coordinate. This representation allows each robot to coordinate with the other members of their team while remaining agnostic of the team member's capabilities and limitations by focusing specifically on how actions affect the team's cumulative performance when performing the following iterations of search. This approach addresses many of the previously described concerns by allowing for teams of multiple robot types to non-myopically coordinate their efforts in real time.

In summary, we introduce a distributed algorithm that leverages a novel representation of the multi-robot action space utilizing the following components:

1. Robots leverage recent advancements in MCTS and macro-action control to efficiently search for sequences of actions in their individual action space.
2. Actions are evaluated using difference rewards to approximate how actions affect the team's cumulative reward.
3. Sliding window upper confidence bound is used to account for non-stationary rewards resulting from the actions of team members.
4. Each robot broadcasts their intended actions to the rest of the team using a probable action timeline that allows agents to calculate difference rewards.

We performed a series of experiments to evaluate the performance of our algorithm using two simulators, including Gazebo (http://gazebosim.org/), and hardware trials with a team of three quadcopters (source code is available at https://github.com/smithan7). Across these trials the proposed coordination method outperformed a state-of-the-art distributed auction-based method (Gerkey and Mataric 2002) by collecting an additional 30.6–47.2% reward by completing additional or different tasks in the simulated trials and 24.5% in the hardware trials with a team of three quadcopters, Fig. 1. This increase is due to the ability to non-myopically plan the actions of individual robots while accounting for their effect on the team's global reward. In these trials we also found that the proposed method of coordination is robust to communication loss and choice of parameters. This paper summarizes the proposed coordination algorithm, simulated experiments, and both the methods of implementation and results of fielded hardware trials.

The remainder of the paper is organized as follows. Section 2 discusses related literature in multi-robot task assignment and heterogeneous teaming. Section 3 describes the domain of the problem we are addressing. Section 4 describes our coordination algorithm. Section 5 details the experiments used to validate the algorithm and their results.

Fig. 1 The three quadcopters in flight during an experiment. The three safety pilots and ground station are off frame

Section 6 concludes the paper and provides the direction for our intended future work.

2 Related work

The multirobot task assignment problem can be formulated as a decentralized partially observable Markov Decision process with macro-actions (MacDec-POMDPs) (Amato et al. 2016). Here, a macro-action is the set of actions required to complete a higher level goal and includes lower level actions such as moving, manipulating, and observation. In the task assignment problem a team of robots work together under uncertainty to maximize a global team utility function. In heterogeneous teams where only one team member has the ability to complete a specific task, the task assignment problem is simplified significantly. However, when robots have overlapping skill-sets and can complete most tasks with varying levels of competency the problem is complicated. Traditionally such problems are addressed myopically, using markets, auction, and greedy methods (Zavlanos et al. 2008; Luo et al. 2012; Gerkey and Mataric 2002; Liu et al. 2015). The myopic solutions, i.e., methods that plan one task ahead for each robot, are inherently limited by their inability to account for future actions of other robots and the consequences of their actions for their future selves. As our algorithm is non-myopic and plans over sequences of actions it is able to account for the future actions of both their own and their team member's future actions when evaluating potential actions.

There are also non-myopic methods for solving the task assignment problem that instead of allocating single tasks to single robots they allocate sequences or clusters of tasks to robots. Market and auction-based techniques can also be used for non-myopic coordination by allowing agents to bid over multiple tasks and then solving for a tour to complete the selected tasks (Zlot et al. 2002). Convergence guarantees can be provided by decentralized auction methods by additionally performing consensus-based local conflict resolutions (Choi et al. 2009). A related algorithm has been developed for multi-robot exploration that uses a central controller to cluster tasks and then distribute them to each robot, who then solves a traveling salesman problem (Faigl et al. 2012). A related method has been developed for multi-robot routing problems that exploits locality and sparsity of tasks to efficiently plan over local partitions of tasks (Liu and Shell 2012). Our method has two key differences from these non-myopic methods: First, our method uses MCTS to efficiently search each individual robot's action space to non-myopically find sequences of actions for each robot. Second, our robots broadcast a temporal representation of their intended actions that allow robots to coordinate in the joint space while accounting for the broadcast future actions of their team members. This approach enables addressing assignment problems that have time-varying rewards.

Another common approach is to use a centralized controller that dictates the actions of each team member. These centralized approaches may use genetic algorithms (Deng et al. 2013), the Hungarian method (Mills-Tettey et al. 2007), or task swaps (Zheng and Koenig 2009) to coordinate. Related algorithms have been developed for the multiple Traveling Salesman Problem (TSP) Bektas (2006), including variants that require considering temporal constraints and rewards (Desrosiers et al. 1995; Mitrović-Minić et al. 2004; Mathew et al. 2015). Generalizations of the TSP have also been considered for persistent monitoring and information gathering formulations (Lan and Schwager 2016; Best et al. 2018; Yu et al. 2016). TSP variants have primarily been considered in centralized contexts. Centralized systems are limited by the requirement of a constant communication tether with the central robot and the computational burden of exploring the action space of all robots across all tasks collectively. In contrast, we have developed a distributed method where each robot only searches their individual action space but is able to adjust their expected rewards to account for the future actions of their team members. This allows robots to plan their actions over a sequence of tasks without the computation and communication restrictions of centralized methods.

There has been growing interest in robotics of modifying traditionally centralized methods of planning into distributed variants. Recent examples include Dec-MCTS (Best et al. 2018), Dist-Hungarian (Chopra et al. 2017), and Dec-POMDP methods (Omidshafiei et al. 2017). In contrast to Dist-Hungarian, our method addresses probabilistic formulations and time-varying rewards. Dec-POMDP methods are typically considered in offline contexts rather than the online and real-time context considered in this paper. Dec-MCTS addresses probabilistic formulations and is aimed at online contexts, and thus has many similarities to our proposed method. Both algorithms sample the individual action space of each robot and then coordinates with a sparse approximation of the joint action space. However, the main difference in our method is a fundamentally new way for robots to sample and represent the coordinated space and plan over the joint space. Dec-MCTS shares a subset of Monte-Carlo tree branches that are likely to be chosen by the robot and shares the probability of the robot taking an observation from the poses along that branch. Our method more efficiently searches each robot's action space by recursively sampling the Monte-Carlo tree along each branch until a predefined threshold is reached. In our method, time-varying reward functions are explicitly addressed by encoding the temporal information in the messages exchanged between robots. These messages describe, for each task, the probability of task selection and a probability distribution for the completion time. Then, each robot broadcasts this information to their team who update their expected reward of completing tasks by accounting for the expected future actions of team members. Other important differences between Dec-MCTS and the proposed algorithm include different ways of discounting (D-UCB vs SW-UCB), different tree compression, and explicitly addressing heterogeneous teams. Additionally, this paper provides an extensive set of new experiments in simulation for a variety of scenarios, and in hardware with a team of UAVs.

In the robot planning domain there are many ways of describing when the planning takes place (online vs. offline), when a robot is able to select an action (any-time), and what time constraints are placed on the planning (real-time and run-time). An online algorithm is one that can approach a problem sequentially as information becomes available and provides an immediate response, while an offline algorithm approaches the problem as a whole and waits until it has the complete sequence of information before acting (Karp 1992). Any-time algorithms are those that continue to refine their solution over time but can be stopped prematurely and provide the user with its current best solution (Boddy and Dean 1989). A real-time algorithm where the algorithm is activated or queried at regular intervals by which they must provide a response (Shin and Ramanathan 1994). Online, real-time and any-time algorithms are particularly valuable in robotics applications. MCTS exhibits these desirable properties (Browne et al. 2012), which has motivated a recent surge in popularity for MCTS in the robotics community. MCTS techniques have so far mostly been applied to single-robot contexts, but there has also been recent interest in extensions for multi-robot active perception (Best et al. 2018), exploration (Corah and Michael 2018), marine operations (Best et al. 2018), and patrolling (Kartal et al. 2015). Our presented method, which features MCTS, is also intended to be online and real-time with the ability to query any-time.

An individual robot selecting a task to complete is an example of the multi-arm bandit problem (MAB). Fortunately, the MAB has several solutions with a strong theoretical and experimental support, most prominently the Upper-Confidence Bound search method (UCB) Agrawal (1995). UCB selects the action to search that maximizes a combination of expected reward and uncertainty. However, in situations where the reward distribution of an action may change over time the traditional UCB algorithm is unable to track the transition. This is a concern for a team of robots where the actions, or communicated intended actions, of one robot may affect the actions of the rest of the team. To account for this several non-stationary MAB algorithms have been developed in the literature, including Discounted-UCB (Kocsis and Szepesvári 2006) and Sliding-Window-UCB (Garivier and Moulines 2011). Discounted-UCB reduces the expected reward and increases the uncertainty of prior results. Sliding-Window-UCB acts similarly, but only retains a finite length history of prior actions allowing it to better adapt to abruptly changing rewards. We chose to implement the SW-UCB algorithm to allow robots to search their individual action space and adapt to the non-stationary rewards resulting from their team member's actions. We use the SW-UCB within MCTS as an alternative to the standard UCB (Kocsis and Szepesvári 2006) suitable for multi-robot settings.

Outside the robotics community there has been multiple methods of parallelizing the Monte-Carlo tree search algorithm to allow it to search deeper into the action space by distributing branches of the tree over multiple computing cores (Yoshizoe et al. 2011; Schaefers and Platzner 2015). These methods are distributed and use MCTS, but are fundamentally different from the method presented here. These methods search one tree over multiple distributed computing cores. In contrast, our proposed method searches over multiple trees in a decentralized manner over multiple distributed computing cores, which allows for a team of distributed robots to independently operate. This is because our method has each robot focus their computational effort developing a tree that only searches their individual action space, and thus a robot does not rely heavily on the computation of other robots when making a decision. This allows robots to operate independently in the presence of communication failures.

3 Problem definition

In this paper we address the problem of selecting each individual robot's sequence of actions that will maximize the team's collective reward. To do this, each robot will have to interact with its environment by moving and completing tasks in exchange for reward. This section formally defines the problem by describing the environment, robots, time-varying tasks and rewards, and objective function.

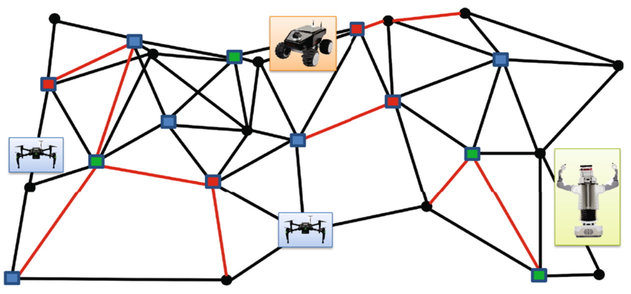

Fig. 2 Example environment graph. Rectangles designate available tasks with the color indicating the task type, black circles are graph nodes without tasks, and lines are graph edges. Black edges are unobstructed to traverse for all robot types while red edges have an additional travel cost for some robot types (Color figure online)

3.1 Environment and robots

The environment the robots operate in is represented as a bi-directional graph, G, consisting of edges and vertices (locations in the environment) as shown in Fig. 2. There is a set of robots, A, where each robot, $a_i \in A$, has an associated type, $type$. Each robot traverses through the environment, which is described as a path through the graph. Each path, P, is a sequence of connected vertices from a fixed starting location. Each edge has a travel cost, $e_{i,type}$, describing the cost to traverse the edge for a robot of type $type$; e.g., a quadcopter may be able to traverse some environments faster than a wheeled vehicle or aquatic vehicle and vice versa. Each edge cost can be either deterministic or described by a probability distribution describing the time required to traverse it. Using these edge costs, $e_{i,type}$, a robot can calculate their expected travel time to traverse path P between two nodes on the graph as the summation of the edge costs that make up the path between the two nodes. Tasks, $\phi_j \in \Phi$, are located on vertices in the graph, but not all vertices are required to have an associated task.

3.2 Tasks and task rewards

A task ϕ_j is described by two components: First, an estimate of the time required to complete it by each agent type, $w_{j,type}$, and, second, the expected reward for completing the task as a function of time, $r_j(t)$. The estimate of the time required to complete a task for each agent type is dependent upon the ability of each robot to complete each task. For example, a robot with a soft gripper may be able to perform a manipulation task faster than another robot with a rigid gripper, i.e., $w_{j,soft} \ll w_{j,rigid}$, while a quadcopter would be unable to complete it, i.e., $w_{j,quad} = \infty$. Then, the expected completion time for task ϕ_j is the sum of the expected travel time and the expected work time,

$$t_{j,type} = t_{P,type} + w_{j,type} \tag{1}$$

The reward function can be any function of time, which may be continuous or non-continuous and may increase or decrease w.r.t. time. Throughout this text, we consider four example reward functions: linearly varying rewards are used for tasks that should be completed in a wide range of times with a preference towards being completed as quickly as possible (linearly decreasing) or as late as possible within the mission budget (linearly increasing),

$$r_j(t) = a_j + b_j; \tag{2}$$

exponentially decreasing rewards that should be completed as quickly as possible with how quickly being determined by the rate of decay provided in the reward function,

$$r_j(t) = a_j e^{-b_j t}; \tag{3}$$

constant rewards that should be completed at the teams convenience,

$$r_j(t) = a_j; \tag{4}$$

and windowed rewards that are paired with another reward type, $f(t)$, for tasks that should only be completed at a specific time,

$$r_j(t) = \begin{cases} f(t), & \text{if } a_j < t < b_j \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

where a_j, b_j, and c_j are constants used to define the reward for task j. Though not done in this work, any function of time, e.g., sinusoidal rewards,

$$r_j(t) = a_j \sin(b_j t + c_j), \tag{6}$$

can be used to define a reward function as shown in Fig. 3.

We emphasize that the four examples shown in Fig. 3 are only a sample of potential reward functions that can be used. The proposed method admits the use of any function, of any arbitrary complexity, as long as when queried for the reward at a specified time it returns a value.

This freedom in reward structure allows for the developed method to be applied to a wide range of potential scenarios. For example, in a search and rescue scenario the area to be searched could be discretized into a minimal set of poses to fully explore the environment. Then, each pose is assigned a linearly or exponentially decreasing reward function that has a magnitude determined by the probability of the lost person being located at that location and the rate of decay set by the assumed danger a person at that location would be in. An additional example would be in a medical treatment center

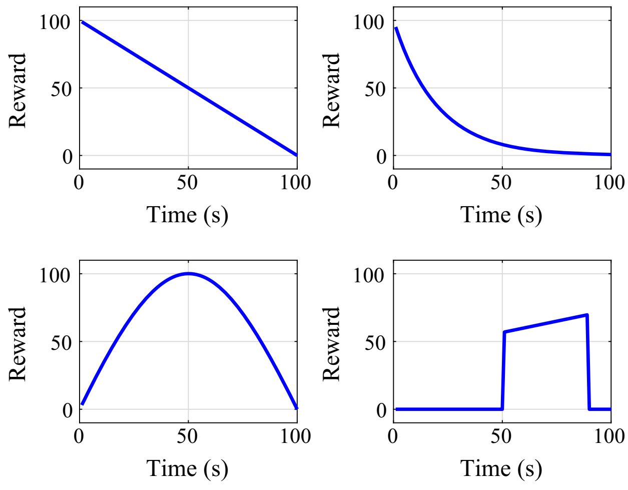

Fig. 3 Examples of linearly decreasing (top-left), exponentially decreasing (top-right), sinusoidal (bottom-left), and linearly increasing window (bottom-right) reward functions

where a team of robots are responsible for delivering food at specific intervals to patients a parabolic or windowed reward structure could be used to encourage delivering the food at the correct times.

The reward offered for completing a task sets the task priority, increasing the offered reward will lead the robots, all other things being equal, to select tasks with larger rewards. Having task rewards that vary with time allow us to describe the urgency and desired scheduling of each task. This combination provides an intuitive approach to setting up coordination problems to represent a wide range of environments and tasks allowing users to express task priorities and allowing the coordination algorithm to match tasks with robots.

3.3 Problem statement

Each robot, $a_i \in A$, is defined by the previously described ability to traverse the graph of the environment, G, and the set of tasks, Φ, that describe the robot's ability to complete them. Robots plan their actions as a sequence of macro-actions, $x_{j,i} \in X_i$. Each macro-action, $x_{j,i}$ consists of both the process of moving to and completing the associated task, ϕ_j. Each robot is assigned either an energy or temporal budget, Ψ, in which to collect the maximum reward, i.e.,

$$X_i^* = \text{argmax}_{X_i \in \Omega}\, R_G(X) \; s.t. \sum_{x_{j,i} \in X_i} e_{j,i} + w_{j,i} \le \Psi, \quad (7)$$

where Ω is the space of all possible macro-action sequences for robot a_i and R_G is the team's cumulative reward,

$$R_G(X) = \sum_{X_i \in X} \sum_{x_{j,i} \in X_i} r_j(t_{j,i}), \quad (8)$$

and $e_{j,i} + w_{j,i}$ represent the sum of the travel and work costs (time or energy) required for robot a_i to complete the task associated with macro-action j and $t_{j,i}$ is the time at which macro-action j is completed by robot a_i. Once a robot completes a task and collects the corresponding reward, no additional reward is awarded for that task. Effectively, each robot is attempting to select a set of tasks to complete at the specific times that will lead to the maximum team reward and while staying under an energy budget.

4 Algorithm description

The main contribution of this paper is a novel algorithm for distributed coordination that solves the planning problem described above. Our method of coordination consists of each agent repeatedly searching the action space for a sequence of tasks to complete and then broadcasting a condensed representation of their intended actions. This representation allows each agent to select actions that will maximize the team's cumulative reward by accounting for the likely future actions of their team members. Throughout this process, each agent is also simultaneously acting in the environment by either moving or completing tasks while continuously re-planning. In this section we begin by providing an overview of our algorithm followed by a detailed description of the main components.

4.1 Algorithm overview

Our method of coordination, illustrated in Fig. 4, repeatedly cycles through a sequence of steps where each robot on a team searches its individual action-space and then communicates its intent with the rest of the team. Each agent begins planning its actions by searching through its action space, Ω, using MCTS for a predetermined number of iterations or computation time. After searching its individual action space, each robot samples its trees to compile a condensed representation of its probable sequence of actions in the form of a timeline, Π^{Σ}. Next, each robot broadcasts its timeline to share its intent with robots on the team in communication range. While planning, each robot may also be simultaneously working on its current task or moving through the environment. Upon completion of a task, robots will broadcast to the team that a task has been completed. On subsequent planning iterations the robot will account for the broadcast timelines of its team members to select actions that maximize the team's cumulative reward. This process is repeated throughout the duration of the mission as each robot continues to extend and update its Monte-Carlo tree, broadcast to team members, and complete tasks. This process is outlined in Algorithm 1.

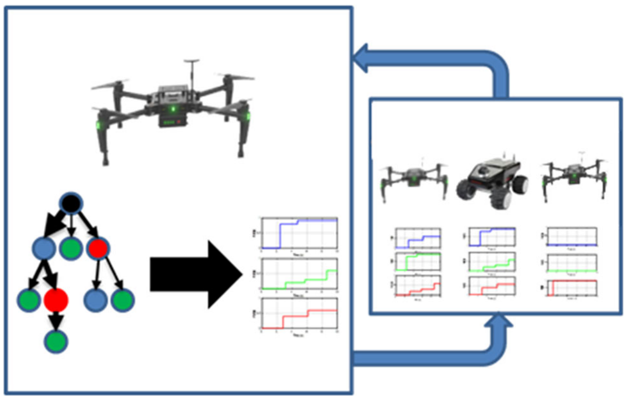

Fig. 4 High level illustration of the coordination procedure. Each robot builds a tree of potential actions using Monte-Carlo tree search which is then used to create a probable action timeline for each task. These probable action timelines are broadcast to the team who are simultaneously doing the same. Each robot then continues to explore the space of potential actions including the effect of the rest of its team

Algorithm 1 Overview of the proposed algorithm for distributed coordination

```
1: procedure COORDINATION OVERVIEW
2:     while true do
3:         if ¬ team initialized then
4:             Broadcast Ready Signal
5:             continue
6:         while computation time budget not exhausted do
7:             tree.SEARCH TREE(G, $\Pi$)                    ▷ Alg. 2
8:         tree.SAMPLE TREE(1.0, $\Pi$, $P_{min}$)           ▷ Alg. 3
9:         BROADCAST TO TEAM($\Pi$)
```

The remainder of this section describes in detail the main components of our algorithm as follows. First, we introduce the probable action timeline that is used to represent each robot's probable future actions in Sect. 4.2 and our method for evaluating how completing a task will affect the team's cumulative reward in Sect. 4.3. Then, in Sect. 4.4 we describe how each robot uses MCTS to search its action space and in Sect. 4.5 we describe how the robots coordinate their actions to maximize the team's cumulative reward.

4.2 Probable action timeline

We introduce the concept of a *probable action timeline* to enable efficiently computing and communicating the probabilities of tasks being completed as a function of time. Each robot maintains one probable action timeline for each task. Each probable action timeline is the cumulative probability, represented as a cumulative distribution function, of any team member completing the associated task as a function of time. Then, using the probable action timeline, each robot can quickly evaluate the probability of another robot completing a task at any time when deciding which task they should complete.

More specifically, the probable action timeline, $\Pi^{\Sigma}_{j,i}$, consists of a series of task claims, $\pi_{j,k} \in \Pi_{j,i}$, that includes the predicted time a task, ϕ_j, is planned to be completed, $t_{j,k}$ (which may be a probability distribution over time); the probability of that action being completed, $P(x_{j,k})$; and the identity of the robot, a_k, planning the action; i.e., $\pi_{j,k} = (t_{j,k}, P(x_{j,k}))$. As the probable action timeline $\Pi^{\Sigma}_{j,i}$ represents the probability of any of robot a_i's team members completing task ϕ_j, it does not include claims by robot a_i. Probable action timelines are initialized for each task with an initial claim with probability of 0 at time 0; i.e., $\pi_{j,\emptyset} = (0, 0)$. As additional probable actions are received from broadcasting team members, they are merged into the probable action timeline as non-mutually exclusive events to represent the cumulative probability of any team member completing task ϕ_j as a function of time using

$$\Pi^{\Sigma}_{j,i}(t) = P_t(x_{j,A} \cup x_{j,B}) \\ = P_t(x_{j,A}) + P_t(x_{j,B}) - P_t(x_{j,A})P_t(x_{j,B}). \tag{9}$$

Here, $P_t(x_{j,A})$ and $P_t(x_{j,B})$ are the probabilities of robots A and B completing task ϕ_j before time t, respectively. $\Pi^{\Sigma}_{j,i}(t)$ is the cumulative probability of either robot A or robot B completing task j before time t. Then, robots can sample $\Pi^{\Sigma}_{j,i}(t)$ to identify the probability of any robot team member completing a task at any time in the future.

An example probable action timeline with task completion times given by normal distributions, defined as $\Pi^{\Sigma}_{j,i} = [(\mathcal{N}(\mu = 20.0\text{s}, \sigma = 1.0\text{s}), 0.1), (\mathcal{N}(\mu = 40.0\text{s}, \sigma = 0.25\text{s}), 0.4), (\mathcal{N}(\mu = 70.0\text{s}, \sigma = 1.5\text{s}), 0.3), (\mathcal{N}(\mu = 80.0\text{s}, \sigma = 0.62\text{s}), 0.6)]$, is shown in Fig. 5. Figure 5 (Top) shows the probability distribution function of each claim reflected by one Gaussian distribution for each claim. Figure 5 (Bottom) is the probable action timeline and it provides the cumulative probability of any agent completing task x_j as a function of time.

Similarly, Fig. 6 (Top) provides discrete claims $\Pi^{\Sigma}_{j,i} = [(20.0\text{s}, 0.1), (40.0\text{s}, 0.4), (70.0\text{s}, 0.3), (80.0\text{s}, 0.6)]$ and (Bottom) the resulting probable action timeline.

An example application of the probable action timeline would be to calculate the expected reward, $R_{E,j}(t_j, \Pi^{\Sigma}_{j,i})$, for completing task j at robot i's expected completion time, $t_{j,i}$, by removing the reward associated with the probability of any other team member completing task j before them, $\Pi^{\Sigma}_{j,i}(t_{j,i})$; i.e.,

$$R_{E,j}(t_j, \Pi^{\Sigma}_{j,i}) = \int_{t_{j,i}}^{\infty} r_j(t) \times (1 - \Pi^{\Sigma}_{j,i}(t))dt \tag{10}$$

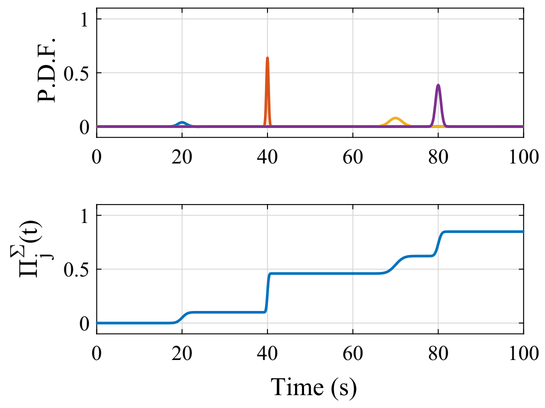

Fig. 5 An example probable action timeline (bottom) with task claims $\Pi^\Sigma_{j,i} = [(\mathcal{N}(\mu = 20.0\text{s}, \sigma = 1.0\text{s}), 0.1), (\mathcal{N}(\mu = 40.0\text{s}, \sigma = 0.25\text{s}), 0.4), (\mathcal{N}(\mu = 70.0\text{s}, \sigma = 1.5\text{s}), 0.3), (\mathcal{N}(\mu = 80.0\text{s}, \sigma = 0.62\text{s}), 0.6)]$ (top)

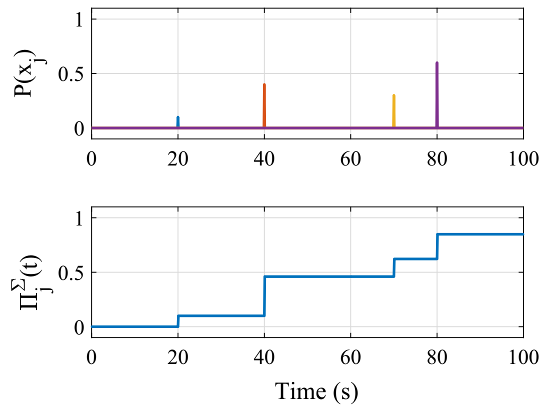

Fig. 6 An example probable action timeline (bottom) with discrete task claims $\Pi^\Sigma_{j,i} = [(20.0\text{s}, 0.1), (40.0\text{s}, 0.4), (70.0\text{s}, 0.3), (80.0\text{s}, 0.6)]$ (top)

which can be approximated by

$$R_{E,j}(t_j, \Pi^\Sigma_{j,i}) = r_j(t_{j,i}) \times (1 - \Pi^\Sigma_{j,i}(t_{j,i})). \tag{11}$$

For a task with a linearly decreasing reward and the discrete claims provided in Fig. 6 the resulting expected reward is provided in Fig. 7. While the expected reward does provide useful information, it can cause robots to 'race' to complete a task as there is no disincentive for completing a task that someone else may complete later. In an extreme example, if robot A is going to complete task j at time 20s with probability $P(x_j) = 1.0$, then robot B will receive a full reward for completing the task at time 19.99 s; even though it may only marginally increase the team reward. While this is an extreme example, it demonstrates the importance of planning in the joint space over the horizon of both robots. To fix this behavior, we introduce the difference reward in the following section that allows agents to account for the future actions of their team members.

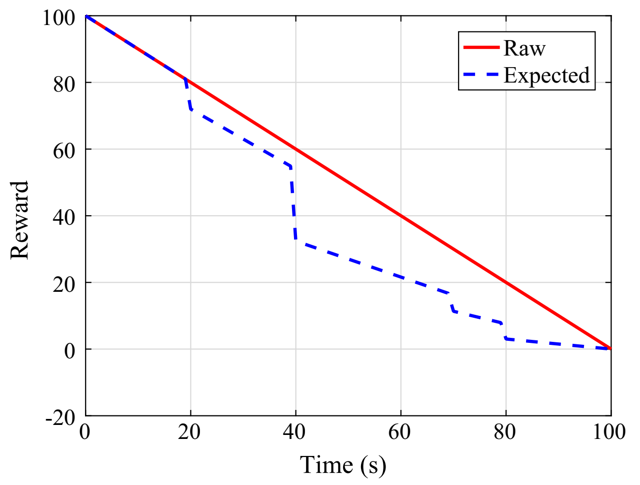

Fig. 7 Raw linearly decreasing reward (Red) and the resulting expected reward accounting for discrete task claims $\Pi^\Sigma_{j,i} = [(20.0\text{s}, 0.1), (40.0\text{s}, 0.4), (70.0\text{s}, 0.3), (80.0\text{s}, 0.6)]$ (Color figure online)

4.3 Difference rewards

As previously described in (7), each robot searches the action space to find a sequence of actions that will result in the largest cumulative team reward. As mentioned in Sect. 4.2 (11), using the expected reward at the time a task is completed can lead to a *race*, where robots select tasks to outcompete each other and not necessarily achieve the largest cumulative team reward. To address this concern, we leverage difference rewards, which were initially developed in the multi-agent systems community (Tumer and Agogino 2009). Our variation of difference rewards is a heuristic that allows each robot to evaluate how an action contributes to the team's cumulative reward and not only its own individual reward.

Before calculating a robot's difference reward, each robot, a_i, must first calculate the reward for completing each task, ϕ_j, at the time it would be completed; which can be either the sum of the distributions defining travel times and work time,

$$t_{j,i} = w_{j,i}(\mu_j, \sigma_j^2) + \sum_{i=1}^{|P_{j,i}|} e_{j,i}(\mu_j, \sigma_j^2), \tag{12}$$

or as an approximation the sum of the expected travel costs and expected work time,

$$\mathbb{E}[t_{j,i}] = \mathbb{E}[w_{j,i}] + \sum_{i=1}^{|P_{j,i}|} \mathbb{E}[e_{j,i}]. \tag{13}$$

Using the predicted completion time, $t_{j,i}$, the predicted reward for completing ϕ_j can be calculated using ϕ_j's reward function,

$$R_{j,i}(t_{j,i}) = \int_{-\infty}^{\infty} r_j(t)\pi_{j,i}(t)dt, \quad (14)$$

and approximated by

$$R_{j,i}(\mathbb{E}[t_{j,i}]) = r_j(\mathbb{E}[t_{j,i}]); \quad (15)$$

where $\pi_{j,i}(t)$ is the distribution of the time robot a_i predicts to complete task ϕ_j. For non-Gaussian distributions and unconventional reward functions the integration will likely require a numerical approximation. Here, $R_{j,i}$ is the predicted reward robot a_i will receive for completing task ϕ_j at the time they are expected to complete it accounting for both the time required to travel to task ϕ_j and the time robot a_i will need to work to complete it. However, $R_{j,i}$ does not account for the actions of team members and how completing task ϕ_j at $t_{j,i}$ will benefit the team's total reward, R_G. Here, we implement our variation of difference rewards to account for the expected actions of the team members. The traditional difference reward for robot a_i is defined as

$$R_{D,a} = R_G(X) - R_G(X - X_i); \quad (16)$$

where R_G is the teams cumulative reward, X is the actions taken by all members of the team, and X_i are the actions taken by robot a_i (Tumer and Agogino 2009). $R_{D,a}$ is used to isolate robot a_i's contribution to the team by removing the contribution made by robot a_i's team members.

In this work we take advantage of the probable action timeline, described in Sect. 4.2, to approximate the difference reward and account for the probable actions of each robot when determining how each potential action will benefit the team's global reward, R_G. The probable action timeline, $\Pi^{\Sigma}_{j,i}$, consists of a series of task claims, $\pi_{j,i}$, each containing a predicted task completion time, $t_{j,i}$ (either an expected time or distribution), the probability of that action being selected, $P(x_{j,i})$, the identity of the robot making the claim, a_i. In short, $\pi_{j,i}$ represents the probability of robot a_i completing task ϕ_j at time $t_{j,i}$ and $\Pi^{\Sigma}_{j,i}$ includes the claims of the all of the evaluating robot's team members. Combining these claims results in an estimate of the probability of any team member completing task ϕ_j as a function of time, $\Pi^{\Sigma}_{j,i}(t)$. Using this information we can determine how robot a_i completing task j at time $t_{j,i}$ affects the teams' cumulative reward.

Before calculating the difference reward, begin by assuming that no team members have competing claims. Then, the reward for robot a_i completing task ϕ_j at time $t_{j,i}$ is simply task ϕ_j's reward function at the time robot a_i predicts as shown in (14). However, if team members have competing claims on task ϕ_j with predicted completion times before $t_{j,i}$, we subtract the reward that team members are predicted to collect before robot a_i is able to complete task ϕ_j at time $t_{j,i}$. As the probable action timeline $\Pi^{\Sigma}_{j,i}$ represents the cumulative probability of robot a_i's team members completing task ϕ_j, the reward associated to the probability of a team member completing task ϕ_j before $t_{j,i}$ can be removed leaving the remaining expected reward,

$$R_{E,i}(t_{j,i}, \Pi^{\Sigma}_{j,i}) = \int_{-\infty}^{\infty} R_{j,i}(t) \times (1 - \Pi^{\Sigma}_{j,i}(t))dt, \quad (17)$$

or using $\mathbb{E}[t_{j,i}]$ as an approximation,

$$R_{E,i}(\mathbb{E}[t_{j,i}], \Pi^{\Sigma}_{j,i}) = R_{j,i}(\mathbb{E}[t_{j,i}]) \times (1 - \Pi^{\Sigma}_{j,i}(\mathbb{E}[t_{j,i}])). \quad (18)$$

Note, $R_{E,i}$ only accounts for the actions affecting task ϕ_j robot a_i predicts their team members to take prior to time $t_{j,i}$. In the event that a team member has a competing claim on task ϕ_j after robot a_i's claim at time $t_{j,i}$, robot a_i's reward should be further reduced to account for it. This is to disincentivize robot a_i completing a task that a team member is probably going to complete shortly after them; assuming there are other tasks available. This reduction is done by removing the reward robot a_i's team members would get if robot a_i does *not* complete task ϕ_j at time $t_{j,i}$. In the event of one later competing claim by robot a_k at time $t_{j,k}$ where $t_{j,k} > t_{j,i}$, this penalty would be

$$D_{j,i}(\mathbb{E}[t_{j,i}], \Pi^{\Sigma}_{j,i}) = \int_{t_{j,i}}^{\infty} r(t])(\Pi^{\Sigma}_{j,i}(t))dt; \quad (19)$$

which can be approximated by

$$D_{j,i}(\mathbb{E}[t_{j,i}], \Pi^{\Sigma}_{j,i}) = r(\mathbb{E}[t_{j,k}])(\Pi^{\Sigma}_{j,i}(\mathbb{E}[t_{j,k}])) \quad (20)$$

where $r(t_{j,k})$ is the reward for completing task ϕ_j at time $t_{j,k}$ and $\pi_{j,k}(t_{j,k})$ is robot a_k's claimed probability of completing task ϕ_j. In short, this reduces the reward robot a_i predicts to contribute to the team's reward for completing task ϕ_j by accounting for the predicted future actions of their team member robot a_k if robot a_i does not complete task ϕ_j. This reduction is weighted by both the probability of robot a_k's predicted actions and the reward at the time robot a_k is predicted to take those actions. An example of this is shown in Fig. 8. To account for multiple claims on a single task, the predicted reward must be further discounted by incrementally accounting for the additional probability each team member claim contributes to the team's cumulative probability. For example, in the event of a second competing claim by robot a_m at time $t_{j,m}$, where $t_{j,m} > t_{j,k}$, the difference penalty for robot a_i is found by subtracting both the predicted reward

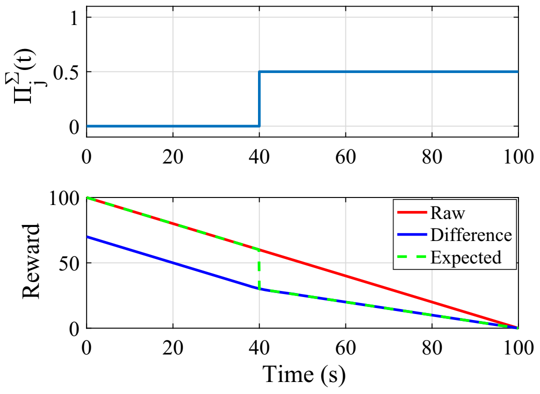

Fig. 8 Comparison of expected rewards and difference rewards (bottom) on a task with a linearly decreasing reward function and one competing claim in the probable action timeline (top)

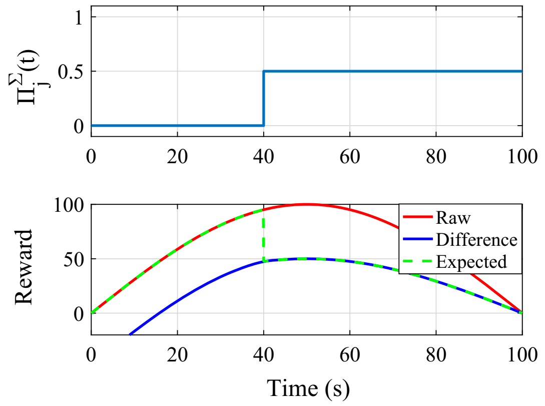

Fig. 9 Comparison of expected and difference rewards (bottom) on a task with a sinusoidal reward function and one competing claim in the probable action timeline (top)

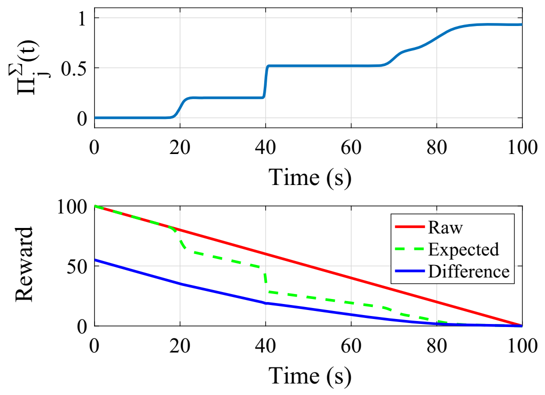

Fig. 10 Comparison of expected and difference rewards (bottom) for a task with a linearly decreasing reward function and multiple claims represented by distributions, from Fig. 5, in the probable action timeline (top). Difference rewards were numerically integrated.

robot a_k contributes to the team and the predicted reward robot a_m contributes in proportion to the amount both of their claims contribute to the team; i.e.,

$$D_{j,i}(\mathbb{E}[t_{j,i}], \Pi_{j,i}^\Sigma) = r(\mathbb{E}[t_{j,k}])\Pi_{j,k}^\Sigma(\mathbb{E}[t_{j,k}]) + r(\mathbb{E}[t_{j,m}])(\Pi_{j,k}^\Sigma(\mathbb{E}[t_{j,m}]) - \Pi_{j,k}^\Sigma(\mathbb{E}[t_{j,k}])). \quad (21)$$

In general, the approximation of the difference penalty is given by

$$D_{j,i}(\mathbb{E}[t_{j,i}], \Pi_{j,i}^\Sigma) = \sum_{\substack{\gamma=0 \\ \mathbb{E}[t(\pi_j^\gamma)]>\mathbb{E}[t_j]}}^{|\Pi_{j,i}^\Sigma|} r_j(\mathbb{E}[t(\pi_j^\gamma)])(\Pi_{j,i}^\Sigma(\mathbb{E}[t(\pi_j^\gamma)]) - \Pi_{j,i}^\Sigma(\mathbb{E}[t(\pi_j^{\gamma-1})])); \quad (22)$$

where γ in π_j^γ indicates the index of the claim in set of claims, $\Pi_{j,i}$, and $\mathbb{E}[t(\pi_j^\gamma)]$ is the expected time task ϕ_j will be completed due to this claim. Next, to address the case where claims are defined probabilistically, we extend (22) to sum the differences at discrete time-steps, δ, where δ is a small step size, i.e.,

$$D_{j,i}(t_{j,i}, \Pi_{j,i}^\Sigma) = \sum_{t=t_{j,i}}^{\infty} r_j(t)(\Pi_{j,i}^\Sigma(t) - \Pi_{j,i}^\Sigma(t-\delta)). \quad (23)$$

Finally, the complete difference reward is given by

$$R_{D,i}(t_{j,i}, \Pi_{j,i}^\Sigma) = R_{E,i}(t_{j,i}, \Pi_{j,i}^\Sigma) - D_{j,i}(t_{j,i}, \Pi_{j,i}^\Sigma). \quad (24)$$

A comparison of the raw task reward, difference reward, and expected reward is shown in Figs. 8, 9, and 11. Figure 8 shows an example probable action timeline with a claimed action, by robot A, $\pi_j = (t = 40\text{s}, 0.5)$, and the raw reward, $r_j(t_j)$; resulting expected reward, $R_{E,j}(t, \Pi_{j,i}^\Sigma)$; and difference reward $R_{D,j}(t, \Pi_{j,i}^\Sigma)$, as a function of time. Notice that the expected reward, $R_{E,j}(t = 39\text{s}, \Pi_{j,i}^\Sigma)$ and raw reward, $r_j(t = 39s)$ are both $61u$. This incentivizes team members of A to *race* robot A for this task by not reducing the reward to account for A's future claim at $t = 40$s, regardless of how it will affect the cumulative team reward. The difference reward compensates for this by accounting for the expected future actions of A, providing $R_{D,j}(t = 39\text{s}, \Pi_{j,i}^\Sigma) = 31.0$.

Figure 9 provides an example expected and difference reward for a task with a sinusoidal reward structure. Notice that the difference reward provides a desirable outcome as shown in Fig. 9 by penalizing robot A's team members with a negative reward for completing the task when it is expected that robot A is likely to complete the task later for a larger reward. Alternatively, a robot using expected reward would attempt to maximize its expected reward by completing the task before robot A at the cost of a reduced team reward.

A final example of the difference reward and expected reward with multiple claims represented by expected arrival times and distributions of arrival times are shown in Figs. 10 and 11, respectively. Notice that in addition to accounting

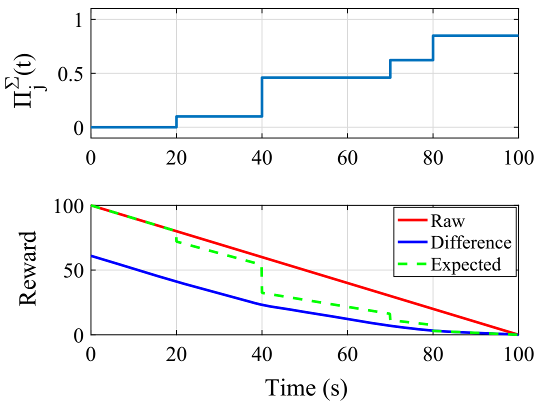

Fig. 11 Comparison of expected and difference rewards (bottom) for a task with a linearly decreasing reward function and multiple discrete claims, from Fig. 6, in the probable action timeline (top)

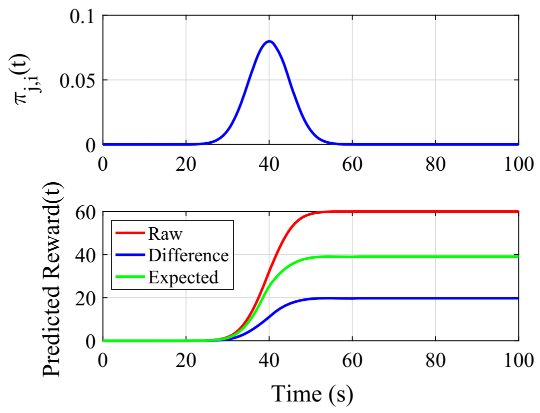

Fig. 12 The predicted reward robot a_i will have with a predicted completion time of $\pi_{j,i} = \mathcal{N}(\mu = 40.0\text{s}, \sigma = 1.0\text{s})$, 1.0 and the probable action timeline and task reward function provided in Fig. 10.

for the future claims, the difference reward is noticeably smoother which may make it easier for some optimization methods.

Further, if robot a_i evaluates their probable action timeline, from Fig. 10, for task ϕ_j with an expected completion time provided by $\pi_{j,i} = \mathcal{N}(\mu = 40.0\text{s}, \sigma = 1.0\text{s})$, 1.0, Fig. 12(Top), then their predicted reward is shown in Fig. 12(Bottom). Unsurprisingly, using difference rewards results in task ϕ_j providing the robot a_i the smallest predicted reward. This is because there are four other robots indicating their intent to complete task ϕ_j and the difference reward removes robot a_i's team members' contributions leaving.

4.4 Searching the action space

Using difference rewards to identify its individual contribution, each robot uses MCTS to optimize a sequence of macro-actions. MCTS traditionally uses a search heuristic called Upper Confidence Bound for Trees (UCT) (Kocsis and Szepesvári 2006) to asymmetrically search the action space and construct a tree of possible action sequences. UCT balances the benefits of exploring the action space by adding new branches of action sequences onto the tree and exploiting existing high-reward branches by extending these sequences. UCT does this by searching the branch beginning with the action that maximizes the heuristic

$$x^* = \text{argmax}_{x_j \in \Omega}\, \bar{R}^{\Sigma}_{D,j}(t_{j,i}, \Pi^{\Sigma}_{j,i}) + c(j), \tag{25}$$

where

$$\bar{R}^{\Sigma}_{D,j}(t_{j,i}, \Pi^{\Sigma}_{j,i}) = \frac{\sum_{s=1}^{p} R^{\Sigma}_{D,j}(t_{j,i}, \Pi^{\Sigma}_{j,i}) \mathbb{1}_{x_s=j}}{\sum_{s=1}^{p} \mathbb{1}_{x_s=j}} \tag{26}$$

is the expected reward of completing the action sequence beginning with $x_{j,i}$ at time $t_{j,i}$, and

$$c(j) = \beta \sqrt{\frac{\epsilon \log N_p}{\sum_{s=1}^{p} \mathbb{1}_{x_s=j}}} \tag{27}$$

is a measure of uncertainty that encourages exploring the action sequence branching from t_j when it has not been recently searched. Here,

$$\mathbb{1}_{a=b} = \begin{cases} 1, & \text{if } a = b \\ 0, & \text{otherwise} \end{cases} \tag{28}$$

and β is a heuristic weight that determines the emphasis on exploring the action space against exploiting previously identified beneficial actions.

Traditional UCT is likely to encounter problems in the multi-robot domain as it can fail in the event of non-stationary rewards; e.g., in this domain a task may initially provide a large reward until another robot broadcasts its intent to complete it causing its reward to significantly reduce. To account for this, we implement a modified Upper Confidence Bound (UCB) search method called Sliding Window-UCB (SW-UCB) (Garivier and Moulines 2011). SW-UCB was chosen because it has a temporally discounted reward structure that allows it to track both incremental and non-continuously changing reward structures. Unlike traditional UCT, SW-UCT only accounts for historical results for the past κ searches and discounts the historical expected rewards within that window by γ^s, where s indicates how long ago that action was searched. This allows it to 'forget' searches that occurred far in the past while favoring the most recent information. SW-UCT also incentivizes searching actions that have not been evaluated recently to update the reward of actions with new information. SW-UCT's policy on search iteration p is to select the action

$$x^* = \text{argmax}_{x_j \in \Omega}\, \bar{R}^{\Sigma}_{D,j}(x_j, \Pi^{\Sigma}_{j,i}) + c(\kappa, j), \tag{29}$$

where

$$\bar{R}^{\Sigma}_{D,j}(t_{j,i}, \Pi^{\Sigma}_{j,i}) = \frac{\sum_{s=p-\kappa+1}^{p} \gamma^{p-s} \bar{R}^{\Sigma}_{D,j}(t_{j,i}, \Pi^{\Sigma}_{j,i}) \mathbb{1}_{t_s=j}}{\sum_{s=p-\kappa+1}^{p} \gamma^{p-s} \mathbb{1}_{t_s=j}} \tag{30}$$

is the expected reward of completing the action sequence beginning with x_j at time $t_{j,i}$, γ is a weighting variable that determines how heavily past results should be weighted compared to more recent ones, κ is the length of the history (i.e., window) used to evaluate actions, and

$$c(\kappa, j) = \beta \sqrt{\frac{\epsilon \log \mathbf{min}(p, \kappa)}{\sum_{s=p-\kappa+1}^{p} \gamma^{p-s} \mathbb{1}_{a_s=j}}} \tag{31}$$

is a padding function to encourage exploring the action sequence branching from t_j when it has not been searched recently. In (31) β is a search heuristic used to weigh between exploiting actions with historically high rewards and exploring under-searched actions. In To plan over sequences of actions, the expected reward uses the node's normalized cumulative down-branch reward, $R^{\Sigma}_{D,j}$; i.e. the summation of reward along the optimal branch from the evaluated node to the leaf of the tree. $R^{\Sigma}_{D,j}$ is normalized:

$$R^{\Sigma}_{D,j} = \frac{R^{\Sigma}_{D,j} - R^{\Sigma}_{D,-}}{R^{\Sigma}_{D,+} - R^{\Sigma}_{D,-}}, \tag{32}$$

where $R^{\Sigma}_{D,-}$, $R^{\Sigma}_{D,+}$ are the sibling nodes with minimum and maximum, respectively, branch rewards.

The strength of SW-UCT is that as the robots coordinate and its team members' plans change, the value of some actions, completing certain tasks, will change over time. SW-UCT will adjust by increasingly discounting older searches in favor of more recent search results when calculating expected rewards, with the severity of the discount determined by γ. This behavior allows the robots to escape an action that was previously an optimal solution that has been affected by the actions of a team member and similarly re-evaluate actions that were unfavorable but not might offer a large reward.

As the Monte-Carlo tree is grown and new nodes (macro-actions of completing tasks) are added to the tree, a simulation is completed to estimate the reward of selecting that branch; as shown in Algorithm 2 line 7. During the MCTS simulation, we use a greedy policy to select tasks that will lead to the largest increase of the team's cumulative reward at the time the task is expected to be completed. This provides a reasonable and computationally efficient estimate of the probable actions the robot will take after completing the queried node and its corresponding rewards. Following the expansion of the node, the tree needs to be updated via back-propagation of the new branch reward, Algorithm 2 lines 13–16. Here, the branch reward is the summation of the node's expected reward and the maximum branch reward of all of its child nodes.

Algorithm 2 Searching the Action Space

```
1: procedure SEARCH TREE(G, $\Pi$)
2:   if ¬ Children exist then
3:       ▷ Make child nodes from available tasks and set
     their initial reward from their expected completion time.
4:     $Children$ = MAKE CHILDREN
5:     for $child \in Children$ do
6:       $child.branch\_reward = child$.ROLLOUT
7:     $child\_reward = 0.0$
8:   ▷ Use SW-UCT to select child to search.
9:   $child^* =$ SW- UCT($Children$)
10:  ▷ Search the selected child.
11:  $child^*$.SEARCH TREE($G, \Pi$)
12:  ▷ Update the reward for my best child.
13:  $child\_reward = \mathbf{max}(child^*.branch\_reward,$
     $child\_reward)$
14:  ▷ Update my expected branch reward.
15:  $branch\_reward = child\_reward + reward$
```

4.5 Coordinated planning

Now that each robot has searched its individual action space and developed a sequence of actions that will maximize its individual reward, the next step is to coordinate with its team members to reduce conflicting plans. To do this each robot must communicate some information with its team members indicating what they are likely to do and how it will affect the team's cumulative reward. We do this using the previously described probable action timeline and difference rewards. This allows each robot to re-evaluate its sequence of actions while accounting for the probable actions of its team members and change its plans accordingly.

To coordinate their actions, after each robot searches the action space and expands its tree, they periodically sample the developed tree and broadcast a condensed version of its likely actions to its team members. Robots sample their tree to assemble their probable action time line; which describes the tasks they will probably complete, how likely they are to complete them, and the predicted time the task will be completed. The sampling process recursively proceeds down the Monte-Carlo tree evaluating the probability of selecting each action from each node. Initially, each action is given a probability of

$$P(x_j) = \frac{r_j}{\sum_{i=1}^{|C|} r_i}. \tag{33}$$

On subsequent iterations, the value is updated by

$$P(x_j)' = P(x_j) + \alpha(\mathbb{1}_{1=max} - P(x_j)), \tag{34}$$

Algorithm 3 Constructing the probable action timeline

```
1: procedure SAMPLE TREE($P_{par}$, $\Pi$, $P_{min}$)
2:     if $P_{self} < P_{min}$ then return
3:     for child in Children do
4:         $P_{child} = P_{self}\frac{r_{child}-r_{min}}{r_{max}-r_{min}}$
5:         $\Pi$.ADD CLAIM($P_{child}$, $t_{child,i}$)
6:         $child$.SAMPLE TREE($P_{child}$, $\Pi$, $P_{min}$)
```

where $\mathbb{1}_{1=max}$ is 1 when action x_j is the action with the largest branch reward and 0 otherwise and α is a gradient update rate that adjusts the broadcast probability. This procedure increases the broadcast probability of the branch with the largest reward and decreases all others in proportion to α and the current broadcast probability of the branch. Following each update, probabilities are normalized to ensure the cumulative probability of the children of each node is 1.

Then, to account for the depth in the tree and the probability of selecting the sequence of tasks that lead to the current node in the tree,

$$P(x_j) = P(x_j)P_{parent}, \tag{35}$$

where P_{parent} is the probability of the parent node being selected, line 4 in Algorithm 3. To begin the sampling process, the root node is assigned a probability of 1. Then the tree is sampled recursively, as described above and in Algorithm 3 line 6, until a lower bound of probability is reached and the sampling terminates along that branch, line 2. When all branches have reached the lower threshold and the recursive sampling terminates, the probable action timeline for all claimed tasks is broadcast to the team.

When a robot receives a broadcast from a team member they update a probable action timeline representing the rest of the team's cumulative claims. First, they remove all existing non-identical claims made by the broadcasting agent. Then, all of the unique broadcast claims are added to the team's probable action timeline as non-mutually exclusive events using (9).

Team members use the probable action timeline to calculate the expected difference reward for completing the task on the next iteration of searching the action space and expanding its tree. Difference rewards of each task are updated non-uniformly, focusing on the most beneficial branches of the tree as selected by SW-UCT. As updating the difference rewards do not require additional simulations or estimates of travel costs it is relatively computationally inexpensive. The repetitive process of broadcasting and then using discounted rewards to plan and re-evaluate action sequences provides team members a way to account for the actions of team members and coordinate their actions in a way to minimize overlap while maximizing the collected reward.

In long-horizon planning problem with discrete tasks the cardinality of the search space grows combinatorially with the number of macro-actions and number of robots. To approach this problem, our developed method distributes the computation across the team by having each robot only plan their own actions. By only searching their own actions the number of combinations to be searched by each robot is significantly decreased and it increases team robustness as it allows each robot the ability to plan and execute their own tasks independently in the presence of communication failures. However, distributing the computation across the team also results in a communication cost that scales linearly with the number of robots. Each robot then uses MCTS and the SW-UCB heuristic to focus their computational effort on the highest value portions of their action space to further mitigate the computational burden resulting from the large action space and long-planning horizon.

5 Experiments

To evaluate the developed method of coordination, four series of experiments were conducted ranging from low-fidelity simulation to fielded hardware trials with unmanned aerial vehicles. These trials were designed to evaluate the performance of the algorithm across a wide range of randomized scenarios and in a fielded hardware implementation. The first series, presented in Sect. 5.1 analyzes the convergence of the coordination with three stationary agents coordinating to complete 30 tasks. Next, we use a low-fidelity simulator in Sect. 5.2 to validate the algorithm over a wide number of team compositions and environments. Then in Sect. 5.3 the algorithm is implemented on a team of quadcopters using the Robot Operating System (ROS) (http://www.ros.org/) and Gazebo (http://gazebosim.org/). Finally in Sect. 5.4 a team of three DJI Matrice M100 quadcopters are used to evaluate the algorithm in hardware trials. The trials and their intent are summarized:

1. Simulated trials exploring robustness to changes in parameters and communication ability.
2. Low-fidelity simulations with large teams over many trials to show statistical significance across a wide range of team compositions and environments.
3. High-fidelity simulations in Gazebo with varying team capabilities and environments.
4. Hardware trials to demonstrate feasibility on existing hardware and confirm the findings from simulated trials.

5.1 Robustness to environmental and parameter variations

To evaluate the convergence of the coordination plans and robustness to environmental and algorithm parameter variations of the developed algorithm, a series of offline trials were conducted. In these trials three robots coordinate to form a plan to complete 30 tasks placed randomly in the environment, but they do not either move or complete a task; i.e., the robots only plan and coordinate their intended future actions without taking them. This allowed us to evaluate how varying algorithm parameters affected the convergence and performance of the coordinated planning. For these trials we allowed each robot to search its action space 5000 times. As the action-space of an individual robot selecting 10 tasks of 30 consists of $2.7 \times 10^{32} = 30! - 20!$ possibilities, we are only sampling a small portion of the action space. Periodically while searching its action space each robot will sample its Monte-Carlo tree and potentially broadcast its intent to the remainder of the team. While these trials are intended to be representative of general use, it may be helpful for reference that the embedded processor used in the accompanying hardware trials, Sect. 5.4, is capable of performing $\mu = 928.92 (\sigma = 87.16, N = 100)$ searches on a graph with 30 tasks in 0.1s. Unless mentioned otherwise, the default parameter values used in this subsection are: gradient ascent rate $\alpha = 0.99$, exploration factor $\beta = \sqrt{2}$, historical weighting factor $\gamma = 0.999$, the dropout rate is $P_{do} = 0.05$, and the broadcast period is 250.

In these trials the parameters we investigated are the sampling frequency, α, γ, and β as well as how communication dropouts affect the convergence. Each of these are investigated individually below in their own sub-subsection.

5.1.1 Convergence example

The offline simulator allows us to glimpse at how the coordination converges from each robot's individual perspective. Figure 13 shows the reward a single robot expects to collect and how it changes through out the coordination phase. Notice that an individual robot initially has a higher reward as they actively seek out the largest individual reward. Then, as they gain information about what each of its team members is doing the difference rewards of some of its potential actions are changed and its expected reward may be reduced. The robot then replans with the most recent coordination information to maximize its collected difference reward leading to an increase in its expected reward. Simultaneously, the cumulative team reward increases as the team of agents coordinate their actions as shown in Fig. 13.

5.1.2 Broadcast frequency

As this algorithm involves multiple robots periodically communicating with the members of their team it is important to investigate how the frequency of communication affects the team's ability to coordinate. To evaluate this, we investigated a variety of communication rates over 20 different randomly generated environments; environments were held

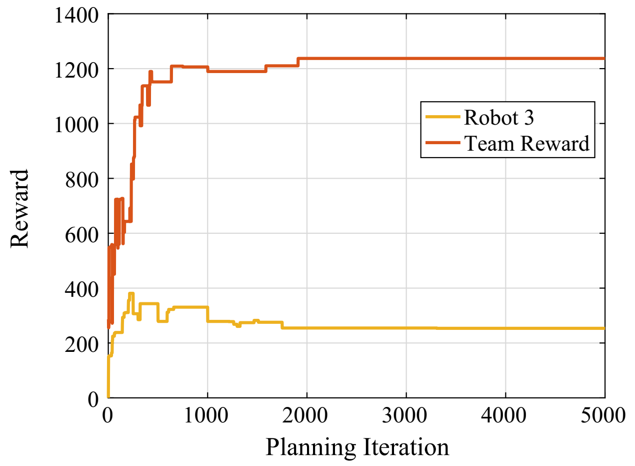

Fig. 13 The captured reward of both a single robot and the complete team throughout the planning process. Note that each single robot continuously attempts to maximize its collected reward. However, every 250 planning iterations they receive broadcasts from their team members causing some of the tasks in the environment have their expected reward changed. Consequently, the robot may seek out alternative tasks that offer a lower raw reward but a larger difference reward

constant across each communication rate. Figure 14 provides the results of the experiment for agents broadcasting their intent with varying numbers of planning iterations between broadcasts. Notice that early in the process of coordinating a higher broadcast frequency performs better, which intuitively makes sense as each robot is able to account for the actions of their team members earlier into the process. However, you will notice that late in the exploration less frequent updates and broadcasts outperforms the higher broadcast frequencies. This is because when a robot receives the plans of another robot they do not immediately update every node in their expanded Monte-Carlo Tree (MCT). Instead, they continue to search the space including the updated information as it affects the actions they are searching. This has two consequences for a high-frequency update rate: First, the robot's broadcast actions may not be searched and updated with the latest information causing their broadcast actions to not reflect the current state of the team's plans. Second, the robot does not have time to adequately search the space and exploit the broadcast information.

5.1.3 Communication dropout

One common concern in multi-robot coordination problems is the effect communication failures will have on the coordinated planning process. To evaluate this we implemented a communication period of 250 planning iterations and hindered the coordination with probabilistic communication drop outs; e.g., communication fails with probability P_{do}. We found that even with high levels of dropout ($P_{do} \leq 0.75$) the robots plan still reliably converges as shown in Fig. 15.

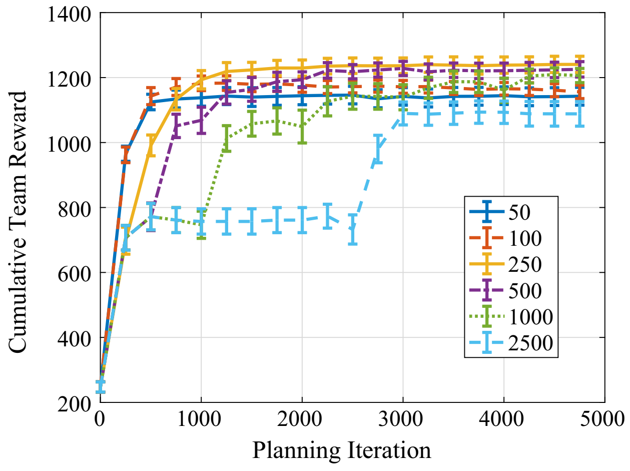

Fig. 14 The captured cumulative team reward with varying broadcast periods. Each line represents the number of planning iterations between broadcasts. Notice that at both extremes, too frequent and too sparse communication, reduces the team performance

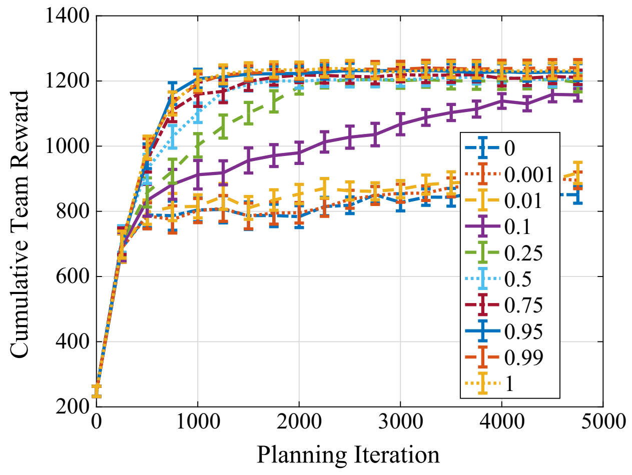

Fig. 16 The effect of α on the convergence of the algorithm; α is the learning rate that determines how quickly a robot modifies the strength of their claims as explained in (34). Probabilities are updated and broadcast every 250 planning iterations

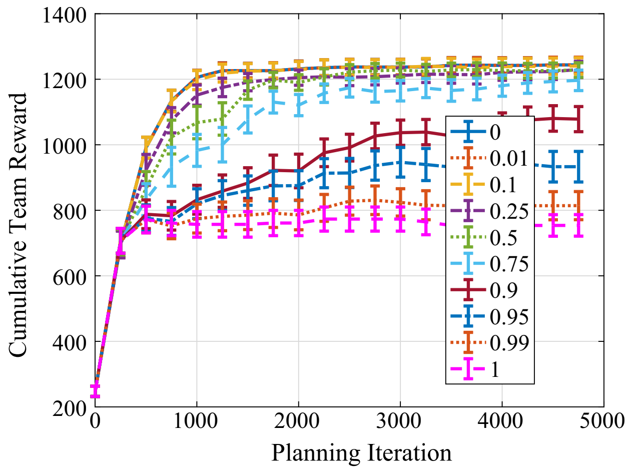

Fig. 15 Coordination performance with varying communication dropout rates. In these examples the agents attempted to communicate every 250 planning iterations and failed with P_{do}. Notice that with $P_{do} \leq 0.5$ the coordination converges at a similar rate as trials without any failures

Notice that for agents with $P_{do} = 1.0$ the cumulative team reward converges to a team reward significantly below that of teams with higher levels of communication. This demonstrates the ability of the algorithm to gradually improve the performance through iterations of searching and broadcasting.

5.1.4 Varying gradient ascent rate

These offline experiments also allowed us to evaluate how the coordination parameters effect the robot's ability to coordinate their planning. We investigated how changing the gradient ascent rate, α, affects the coordination convergence as shown in Fig. 16. We found that $\alpha \approx 0.99$ yielded the best results with a sampling period of 250 planning iterations. This implies that agents should make strong but not absolute claims about their intended actions. This results in agents who are changing their claim to strongly disincentivize other agents from claiming that task without removing all reward after a single iteration of coordination.

5.1.5 Varying SW-UCT heuristic

Another parameter of concern is the SW-UCT heuristic weight β. Traditionally β is $\sqrt{2}$ (Chang et al. 2005). However, for this work the role of exploitation is slightly modified. In this work when an action is searched it is also updated with the most recent plans collected from the team and so it is important that the process of exploring the action space and updating the best actions is balanced to ensure that the robot is acting upon the most recent coordination information. We compared multiple values of β and found that $\sqrt{2} \leq \beta \leq 2\sqrt{2}$ provided the best results, as seen in Fig. 17. β values below this range did not explore enough of the action space to find high reward action sequences. Alternatively, values of β above this range did not emphasize updating and exploiting high reward action sequences which resulted on planning and broadcasting on outdated information.

5.1.6 Varying weight of historical values

The next parameter we evaluated was the effect of γ which determines the weighting of historical results in the SW-UCB search heuristic as shown in (30) and (31). As γ increases from $0 \rightarrow 1$ the length of SW-UCT's 'memory' increases and historical search results are more heavily weighted. We

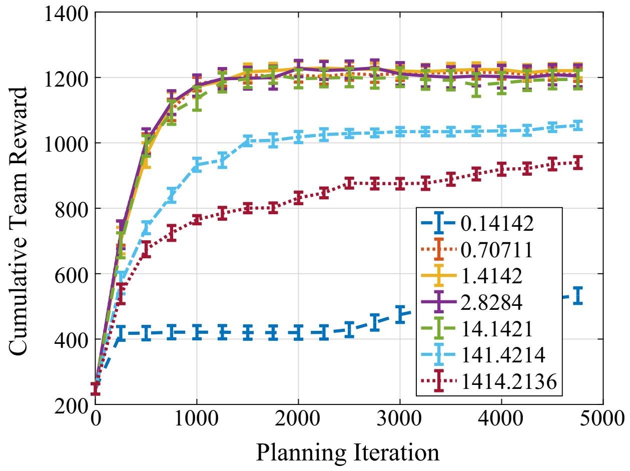

Fig. 17 β is used to weigh between exploration and exploitation when selecting which branch of the Monte-Carlo tree to search. Notice that values near the traditional value of $\sqrt{2}$ perform well while values significantly larger or smaller do not perform as well

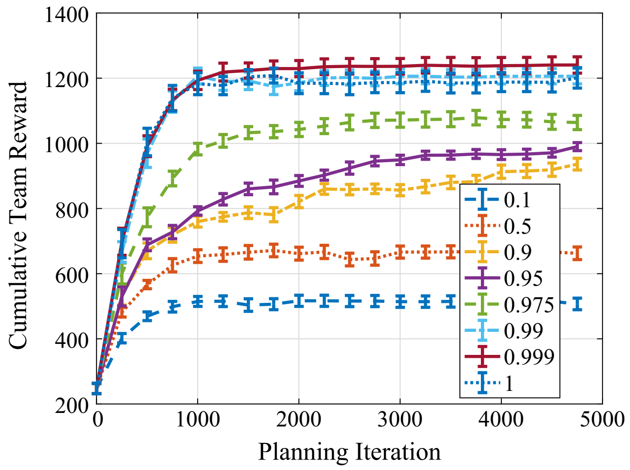

Fig. 18 The parameter γ is used to determine the weight of past searches in the SW-UCT algorithm with γ

noticed in our trials, Fig. 18, that having a $\gamma \lessapprox 1.0$ balances the ability to discount searches that occurred a long time ago and also retain useful information about the action space. In the following trials we enforce a maximum memory length of 5000 by assigning this value to κ in (30) and (31).

5.2 Low-fidelity simulations with variety in environments and team compositions

A simulator was developed to test our method's ability to coordinate a heterogeneous team of robots completing heterogeneous task types. The environment of the simulator is a PRM style graph, as described in Sect. 3, where a random subset of PRM nodes contain tasks or random types. For each trial the same experiment, identical map, task, and robot configurations, was completed once by a distributed auction-based planner and then our developed algorithm introduced in Sect. 4. Then, on the following trial a new graph, team of robots, and set of tasks are randomly generated. Throughout the trials, the environment is held fixed at a simulated $1000 \times 1000\,\text{m}^2$ area for all experiments with 100 vertices randomly placed. Edges are created by connecting the Euclidean nearest five vertices.

Tasks are randomly generated at a portion of the nodes depending upon the number of robots in the current experiment. Generated tasks are assigned one of four random types that have varying reward amounts, reward function types, and the amount of work that each robot type must perform to complete it. Task rewards are varied across all tasks and not held constant for each task type. Generated robots belong to one of four types with uniformly randomly sampled travel speeds of 5.0–25.0 ms and ability to perform of work on each task type in each time-step; varied uniformly randomly, $w_i = 1{-}100$, with some probability that an robot type will not be able to work on a given task type, i.e., $P(w_i = 0.0|a, t) = 0.1$. Similarly, a random subset of edges have obstacles that either slow or prevent some robot types traveling along them with $P(slow|a) = 0.25$ and $P(block|a) = 0.05$.

Robots operate in simulated real time where they are allowed one time-step, of duration 0.1 s, to search through their action space, sample the tree and broadcast their current plan to the team. Each robot attempts to maximize their expected reward and collectively work to optimize their teams cumulative reward collected. Robots use A* (Hart et al. 1968) to identify the minimum cost path to each potential task. When making claims, each robot calculates the time they expect to complete a task by summing the expected time to traverse each edge along the path and the time required to complete the selected task upon arrival.

An auction-based planner is used as a comparative baseline (Gerkey and Mataric 2002) due to its ability to adapt to the developed problem setup. A key strength of our developed method is the ability to account for the time at which a task is completed when calculating the reward function. Unfortunately, including this flexibility in the problem setup made direct comparison to other existing non-myopic methods infeasible. In the auction-based planner, robots select tasks that will result in the largest reward at the time the task is completed. Robots then publicly bid on the task with their expected travel cost and the robot with the lowest bid receives the task until it is either completed or they select another task. The results of the auction are enforced by removing all potential reward for the bid task from the losing robots; equivalent to a claim with probability of 1.0 at the expected time of completion.

We evaluated three different team sizes for 100 trials each and the collected reward is reported in Table 1. The parameters used for SW-UCT were $\beta = \frac{\sqrt{2}}{2}$, $\gamma = 0.999$, $\epsilon = 0.5$,

Table 1 Summary of the low-fidelity simulation results for team sizes and the corresponding probability of each of the 100 tasks being active during the trials

Number of robots	3 robots $P_{active} = 0.25$	5 robots $P_{active} = 0.5$	7 robots $P_{active} = 0.75$
Auction	$\mu = 3674.46$	$\mu = 8348.65$	$\mu = 18{,}721.27$
Reward	$\sigma = 14.74$	$\sigma = 25.22$	$\sigma = 1146.63$
Proposed algorithm	$\mu = 5406.94$	$\mu = 11{,}650.86$	$\mu = 24{,}445.67$
Reward	$\sigma = 17.73$	$\sigma = 22.55$	$\sigma = 556.47$
Mean utility gained	47.15%	39.55%	30.6%

For each set of trials new robot abilities, travel speeds, graph structure, and task types were randomly generated. The presented result is the mean cumulative reward collected by the team at the end of each test. Mean Utility Gained $= 100.0\% \times \frac{\mu_D - \mu_A}{\mu_A}$

and $\kappa = 5000$. These values were adapted from Garivier and Moulines (2011) and empirically tuned. The coordination update parameter was $\alpha = 0.9$. These experiments were perform on a Laptop with a Core-i7 processor with 16 GB of RAM. During the alloted planning time, 0.1 s, robots were able to perform $\mu = 308.2(\sigma = 61.6, N = 900)$ searches of the MCT and each iteration of sampling the MCT required $\mu = 3.8 \times 10^{-4}$ s$(\sigma = 4.9 \times 10^{-4}$ s, $N = 900)$.

The distributed auction baseline myopically selects task with the largest expected reward at the time of completion while accounting for competing claims of other robots, i.e.,

$$R_A(k_j, A) = r_j(k_j) \times (1 - P(x_j)). \tag{36}$$

Then, robots claim the task with their expected time of completion with $P(x_j) = 1.0$, effectively removing all reward after the time of their claim. In this way robots attempt to take the action that will result in the largest reward and resolve claims by the robot who will arrive at the task first.

The developed coordination algorithm outperformed the auction-based coordination algorithm across the range of team sizes, robot, task, and environment configurations tested by 30.6–47.2% as shown in Table 1. This shows that our proposed algorithm is able to coordinate the robots in a more efficient manner that allows them to collect a larger amount of reward. Throughout the simulations conducted it was a common trend that the auction coordination algorithm would initially outperform the proposed method in the beginning of each simulation, as demonstrated in Figs. 19 and 20. This is because the auction coordinator myopically searches for the largest individual reward available, that it will arrive at first, at the potential cost of future rewards. Alternatively, our proposed method plans over sequences of rewards in the joint space allowing it to avoid both resource scarcity and congestion.

In addition to coordinating which robot should complete each task, our proposed method also determines when each robot should complete each task to maximize their difference reward and the team's cumulative collected reward. This difference allows our proposed method to outperform the

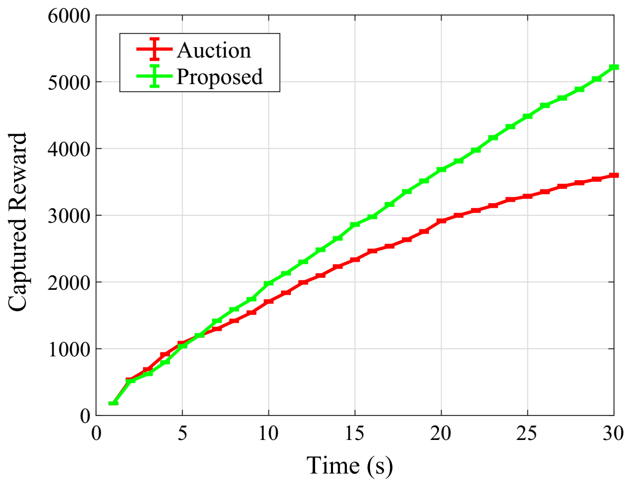

Fig. 19 A comparison of the mean reward collected by the proposed method and auction baseline for three robot trials. The error bar provides the standard errors of the mean

auction coordinator implemented as a baseline. While the proposed method did outperform the baseline, as the number of robots increases the performance gain of the proposed method decreases. This is because the tasks become oversubscribed as the number of robots available to complete the tasks increases.

5.3 ROS Gazebo experiments in varied environments

Additional simulated trials were performed using ROS and Gazebo to evaluate the performance of the algorithm in varied environments in a higher-fidelity simulator. These trials used the Hector quadcopter (Meyer et al. 2012) to simulate the motion of the quadcopter, Gazebo to simulate the physics of the quadcopter and provide absolute localization information, the rtabmap package (Labbé and Michaud 2014) and a depth camera to sense and navigate the environment. In these trials maps were randomly generated with obstacles which were spawned in Gazebo as red cylinders (Figs. 21, 22). The test environment was 100 m × 100 m.

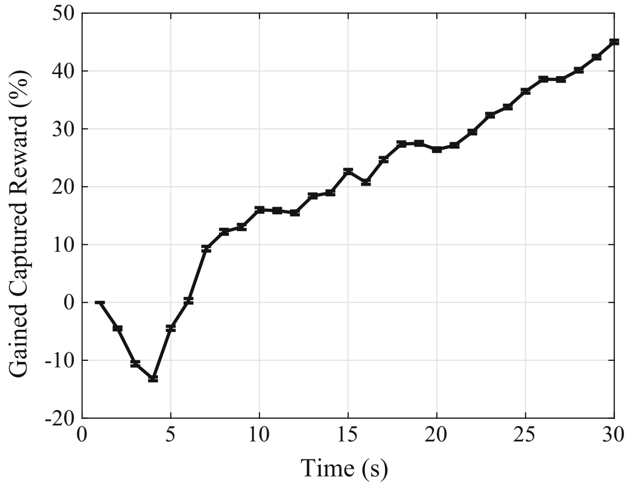

Fig. 20 The gain in collected reward from the proposed method over the auction baseline. Notice, that the auction robot initially leads the proposed algorithm. This is because the auction baseline is myopic and sacrifices later reward for immediate returns. The proposed method alternatively sacrifices immediate returns for cumulatively higher returns collected over the duration of the experiment

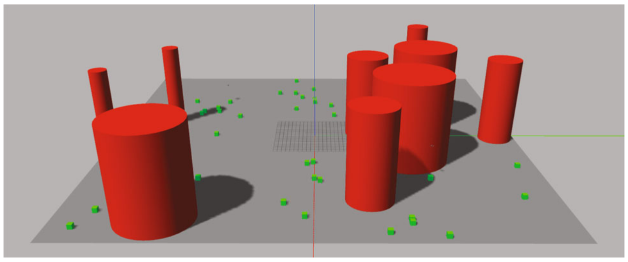

Fig. 21 The Gazebo test environment. The red cylinders are randomly placed obstacles in the environment and the green cubes are the ground locations of markers (Color figure online)

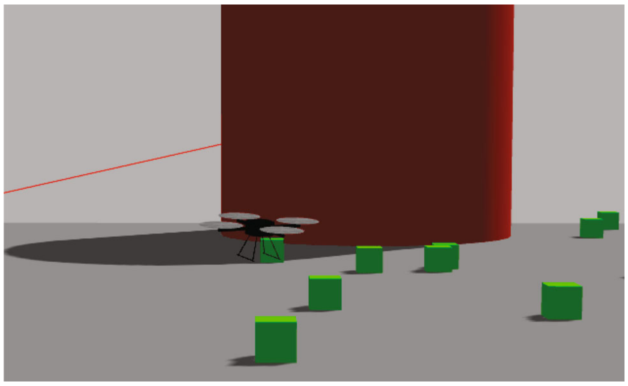

Fig. 22 The Gazebo test environment close up of the quadcopter moving through the environment. The red cylinder is an obstacle and the green cubes mark the locations of tasks (Color figure online)

The results from these trials are shown below in Table 2. Notice that there is a decrease in performance from the low-fidelity simulator. This decrease is assumed to come from two sources: First, in the low fidelity simulator the robot's arrival time is perfectly predictable allowing the robots to accurately forecast their actions. In the Gazebo trial this accuracy is decreased as the quadcopter must be controlled and avoid the obstacles on apriori unknown paths. Second, in the low fidelity simulator each quadcopter is allowed a full planning cycle to plan their actions without additional processes competing for processing power. In the gazebo simulator each robot is simultaneously being simulated in Gazebo, searching the macro-action space, and controlling the motion of the quadcopter. The extra computational burden of simulating multiple quadcopters reduced the computational resources available for searching the action space.

Table 2 Summary of the gazebo simulation results for team sizes and the corresponding probability of each of the 100 tasks being active during the trials

Number of robots	3 robots $P_{active} = 0.25$	5 robots $P_{active} = 0.5$
Auction reward	$\mu = 3425.59$	$\mu = 9419.72$
	$\sigma = 895.26$	$\sigma = 1857.31$
Proposed algorithm reward	$\mu = 4707.13$	$\mu = 12{,}182.99$
	$\sigma = 1447.68$	$\sigma = 1120.43$
Mean utility gained	37.41%	29.33%

For each set of trials new robot abilities, travel speeds, graph structure, and task types were randomly generated. The presented result is the mean cumulative reward collected by the team at the end of each test

5.4 Hardware experiments

In addition to the simulated experiments, hardware trials were conducted to provide a real world comparison between the proposed method and a distributed-auction-based coordination using DJI M100 quadcopters (https://www.dji.com/matrice100) and to demonstrate the feasibility of the proposed approach on existing hardware. In these trials a random graph was created over an open field and each quadcopter was provided a random agent type with random capabilities, similar to Sect. 5.2. Then, each quadcopter moved about the graph attempting to complete their tasks. The following sections begins by describing the hardware system used in these and then describes the experimental results.

5.4.1 Hardware system

The three quadcopters used for these hardware trials were DJI Matrice 100s, each equipped with a ZED RGBD camera, NVIDIA Jetson TX2 GPU, and DijiKey Xbee-Pros, Fig. 23. The ZED camera is mounted on the 'front' of the Matrice, and the orientation of the quadcopter is controlled so that the camera is always facing the direction of forward motion. The Jetson TX2 operates using Ubuntu 16.04 and Robot Operating-System (ROS) Kinetic-Kame, and the ZED ROS

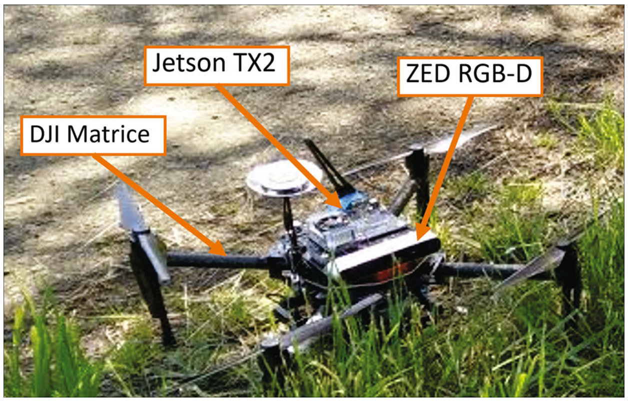

Fig. 23 The DJI Matrice M100 used in the hardware trials. The Jetson TX2 is a NVIDIA GPU that runs Ubuntu 16.04 and performs the onboard computation. The ZED RGBD camera is used to create a 3D map of the environment the quadcopter is operating in

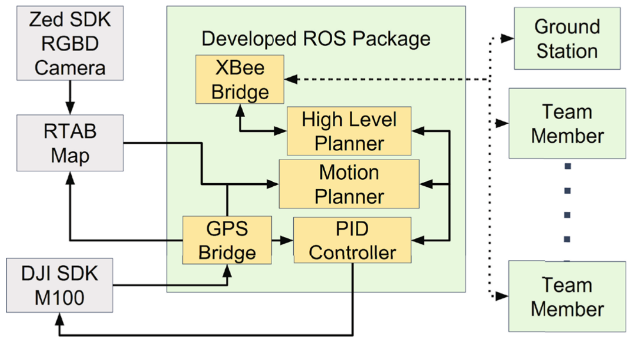

Fig. 24 The ROS nodes used in this paper. Nodes encapsulated by green boxes were developed for this project. Arrows indicate the direction of information flow. Solid arrows indicate ROS messages being passed locally while dashed arrows indicate communication over the XBee Diji-Mesh network (Color figure online)

wrapper is used to access the depth images from the ZED cameras. The DJI On-Board ROS SDK is used to acquire odometry and GPS positional information from the Matrice, and to allow the Jetson TX2 to control the motion of the quadcopter. A 2D Occupancy grid with inflated obstacles is generated from the depth images and Matrice odometry using the RTAB-Map ROS package. The Matrice uses the A* search algorithm on the 2D occupancy grid to plan a path to goal locations while avoiding obstacles. Inter-team communication, between Matrices and a ground station replaying the human search path, is performed over using a mesh network via Diji-Key Xbee Modules and a custom ROS node.

As mentioned, the Jetson TX2 used a series of ROS nodes, outlined in Fig. 24, to conduct each experiment. The coordination and task selection was carried out in the High-Level Planner node that used custom messages to coordinate with other agents through the XBee Bridge node. The XBee Bridge node converts standard ROS messages and custom ROS messages to a string of characters and broadcasts them over the XBee Diji-Mesh network. Then, at 100 Hz the Xbee Bridge reads all messages in the buffer and publishes them to ROS as ROS messages. To coordinate the quadcopters, their local odometry is generated by the GPS bridge node which converts their global position, provided by GPS, to a local shared frame; allowing quadcopters to operate in the same ROS local coordinate frame. The Motion Planner node uses the costmap, seeded by satellite imagery and updated using the ZED RGBD camera and RTABMap ROS Package, to plan a path to the current goal. The PID controller publishes velocity commands to the DJI M100 that attempt to follow the path published by the Motion Planner. Custom nodes created for this work are available at https://github.com/smithan7.

In addition to the three quadcopters, a ground station was used to conduct these experiments. The ground station did not assist in the coordination or inter-team communication but instead was used as a graphical user interface for the operator (shown in Fig. 25), recorded team performance, and simulated the tasks in the environment. To simulate tasks, the ground station received and processed ‘work requests’ from robots; i.e., when a robot is attempting to work on a task it notifies the ground station. After receiving the work request, the ground station determines the amount of work the robot performed and adjusts the remaining amount of work in the task and notifies the team. When the robot has successfully completed the amount of work required to complete the task, the ground station notifies the team that the task has been completed and records the amount of reward awarded for completing the task at the time it was completed. Alternatively, in application each robot could broadcast to the team when they have completed a task so that other robots may prune their Monte-Carlo trees of the completed task. In addition to recording the actions of the quadcopters, the ground station also ensured that quadcopters started at the same location for each experiment and synchronized the quadcopter starts.

The hardware experiment is used to validate the results collected in the Low-Fidelity and Gazebo Simulators, Sects. 5.2 and 5.3, and demonstrates feasibility on hardware subjected to communication and computation restrictions. These experiments were conducted in an unstructured outdoor environment with random tasks distributed across a generated graph covering the environment as shown in Fig. 25.[1] The quadcopters begin a the top center of the map and are released simultaneously by the ground station to begin each experiment. Starting locations were consistent across all trials. To simulate a heterogeneous team, we took

[1] The hardware flight trials conducted for this research were conducted under Oregon State University’s (OSU) Federal Aviation Administration Certificate of Authorization and logged in OSU’s compliance software, Drone Complier.

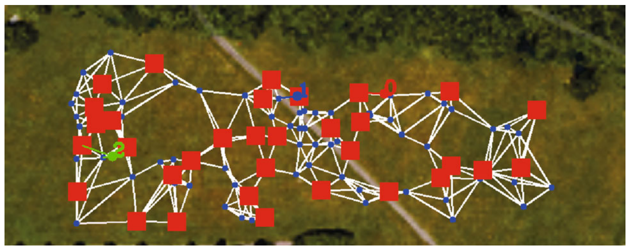

Fig. 25 An overhead view of the field the hardware experiments were conduced in. The field is 200 × 80 m in size. Graph edges are white, vertices are blue, and tasks in the environment are red squares. The quadcopters are identified by number (0, 1, 2) and color (red, blue, green). Each quadcopter has lines connecting them to the vertices of the edge they are occupying (Color figure online)

Table 3 The results of the hardware trials conducted for the proposed method and distributed auction coordination algorithms

Number of robots	3 robots $P_{active} = 0.4$
Auction reward	$\mu = 7603.07$
	$\sigma = 300.79$
Proposed algorithm reward	$\mu = 9462.41$
	$\sigma = 339.59$
Mean utility gained	24.46%

Three trials were conducted for each and the mean collected reward is reported for both methods

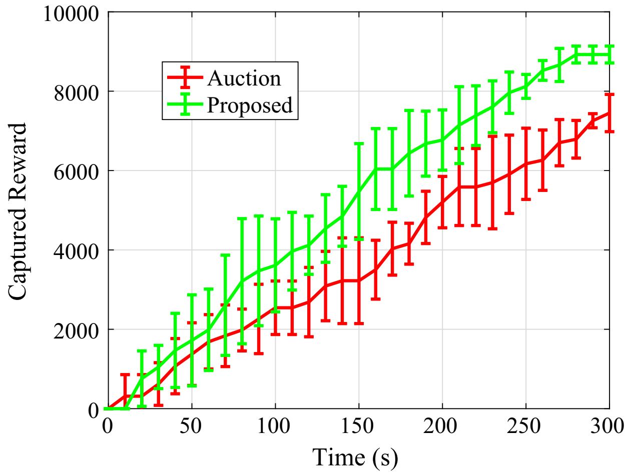

Fig. 26 The mean collected reward from the proposed method and the auction baseline for the six hardware trials (three of each method). Notice, that the proposed method collected reward at a larger rate over the duration of the experiment

advantage of the controllability of the quadcopters to vary each agent's movement speed and time spent to complete each task type.

5.5 Experimental results

In the experiments conducted we successfully performed three trials of both the distributed auction-based coordination and our proposed method. In these trials our proposed method was able to collect an additional 24.46% reward compared to the distributed auction baseline, as shown in Table 3 and Fig. 26. While this is still a significant increase over the distributed auction, the relative difference is less than resulted in the Gazebo simulators. This difference could be the result of differences in the map size or graph used in the hardware trials and Gazebo trials, increasing the number of active tasks on the graph by 15%, the limited battery life of the quadcopters restricting the mission time which may have a larger adverse effect on the proposed method, or a complication from implementation on hardware in the environment.

We had considered evaluating the performance of our algorithm against other non-myopic coordination methods in our experiments. One of our algorithm's primary contributions is the temporal component of the coordination and the resulting flexibility provided in both task reward types and the environment travel costs. Unfortunately, the other non-myopic planning methods we investigated for inclusion in this work do not include this level of flexibility, hindering a direct comparison. In the future it would be interesting to see a direct comparison between our method and adapted versions of other non-myopic coordination methods.

6 Conclusion

In this paper we introduced a novel algorithm for task-selection in distributed heterogeneous teams. The proposed method leverages recent advancements of Monte-Carlo tree search and macro-action control to efficiently search each individual robot's action space while compensating for the non-stationary rewards resulting from team member actions. A novel temporal-action-space representation allows each agent to evaluate how actions will affect the team's cumulative reward while searching their individual actions space and remaining agnostic to team member capabilities. Our proposed method was compared against an auction-based task selection technique in simulated environments with randomly generated tasks, robot types, and environment structures and costs. In every simulated experiment our proposed algorithm was able to collect a larger amount of reward than the distributed auction-based planner.

7 Extensions and future work

Using our developed method as a starting point we have identified multiple avenues for extension:

- It would be interesting to analyze the theoretical convergence properties of the our algorithm by extending the analysis presented in Best et al. (2018) for our new MCTS variant. This may provide additional insight into the parameters identified in Sect. 5.1 and methods to improve the computational efficiency of our algorithm.
- We believe the algorithm would be tolerant to local (non-global) coordination. Robots using our algorithm individually plan and act and only dependent upon team members to coordinate. As the team members that will require the highest level of coordination are likely to be the team members that are also the closest each we believe that the team performance would decline gracefully as the range and ability to communicate decline. This has been demonstrated with a related algorithm, and can be exploited to judiciously select when to communicate (Best et al. 2018); it would be interesting to extend this idea for the problem formulation addressed in this paper.
- In some scenarios it may be desirable to have tasks completed in a specific sequence; e.g., before robot a_0 can operate a piece of machinery it must be powered on by robot a_1. Using the probable action timeline robot a_0 can predict the probability of required tasks being completed by their team members when calculating their expected reward for completing tasks which should allow task sequences to be evaluated. It would be an interesting and valuable extension to investigate this further.
- As the team size and action space grows into large teams (20+ robots) and the task space scales (1000's of tasks) our algorithm, as implemented, will likely not scale well. This is because each robot currently plans over every task and coordinates with every agent in the environment; which may be computationally infeasible as the numbers of tasks and agents grows. However, this problem can be solved by restricting each robots view of the world. First, the tasks available to each robot could be restricted by distance. Second, using an ad-hoc communication network as a template, we would enforce local communication to the robots most likely to interfere with each others actions. Then, each robot can pass along a merged probable action timelines of all upstream robots to represent their cumulative actions to all receiving down-stream robots.

References

Agrawal, R. (1995). Sample mean based index policies by O(log n) regret for the multi-armed bandit problem. *Advances in Applied Probability*, *27*(4), 1054–1078.

Amato, C., Konidaris, G., Anders, A., Cruz, G., How, J. P., & Kaelbling, L. P. (2016). Policy search for multi-robot coordination under uncertainty. *International Journal of Robotics Research*, *35*(14), 1760–1778.

Arkin, R. C., & Balch, T. (1998). Cooperative multiagent robotic systems. In *Artificial intelligence and mobile robots*.

Ayanian, N., Fitch, R., Franchi, A., & Sabattini, L. (2017). Multirobot systems. *IEEE Robotics & Automation Magazine*, *24*(2), 12–16.

Beck, Z., Teacy, L., Rogers, A., & Jennings, N. R. (2016). Online planning for collaborative search and rescue by heterogeneous robot teams. In *Proceedings of the international conference on autonomous agents & multiagent systems* (pp. 1024–1033).

Bektas, T. (2006). The multiple traveling salesman problem: An overview of formulations and solution procedures. *Omega*, *34*(3), 209–219.

Best, G., Cliff, O., Patten, T., Mettu, R. R., & Fitch, R. (2018). Dec-MCTS: Decentralized planning for multi-robot active perception. *International Journal of Robotics Research*. https://doi.org/10.1177/0278364918755924.

Best, G., Forrai, M., Mettu, R. R., & Fitch, R. (2018). Planning-aware communication for decentralised multi-robot coordination. In *Proceedings of the IEEE International Conference on Robotics and Automation*.

Best, G., Huang, S., & Fitch, R. (2018). Decentralised mission monitoring with spatiotemporal optimal stopping. In *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems*.

Best, G., Faigl, J., & Fitch, R. (2018). Online planning for multi-robot active perception with self-organising maps. *Autonomous Robots*, *42*(4), 715–738.

Boddy, M., & Dean, T. L. (1989). *Solving time-dependent planning problems*. Providence: Department of Computer Science, Brown University.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., et al. (2012). A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, *4*(1), 1–43.

Chang, H. S., Fu, M. C., Hu, J., & Marcus, S. I. (2005). An adaptive sampling algorithm for solving Markov decision processes. *Operations Research*, *53*(1), 126–139.

Choi, H., Brunet, L., & How, J. P. (2009). Consensus-based decentralized auctions for robust task allocation. *IEEE Transactions on Robotics*, *25*(4), 912–926.

Chopra, S., Notarstefano, G., Rice, M., & Egerstedt, M. (2017). Distributed version of the Hungarian method for a multirobot assignment. *IEEE Transactions on Robotics*, *33*(4), 932–947.

Corah, M., & Michael, N. (2018). Distributed matroid-constrained submodular maximization for multi-robot exploration: Theory and practice. *Autonomous Robots*. https://doi.org/10.1007/s10514-018-9778-6.

Deng, Q., Yu, J., & Wang, N. (2013). Cooperative task assignment of multiple heterogeneous unmanned aerial vehicles using a modified genetic algorithm with multi-type genes. *Chinese Journal of Aeronautics*, *26*(5), 1238–1250.

Desrosiers, J., Dumas, Y., Solomon, M. M., & Soumis, F. (1995). Chapter 2: Time constrained routing and scheduling. In M. O. Ball, T. L. Magnanti, C. L. Monma, & G. L. Nemhauser (Eds.), *Handbooks in operations research and management science* (Vol. 8, pp. 35–139). Elsevier

DJI. Matrice 100 quadcopter for developers. https://www.dji.com/matrice100.

Faigl, J., Kulich, M., & Preucil, L. (2012). Goal assignment using distance cost in multi-robot exploration. In: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems* (pp. 3741–3746).

Fukuda, T., Nakagawa, S., Kawauchi, Y., & Buss, M. (1989). Structure decision for self organising robots based on cell structures. In

Proceedings of the IEEE international conference on robotics and automation.

Garivier, A., & Moulines, E. (2011). On upper-confidence bound policies for switching bandit problems. In *Proceedings of the international conference on algorithmic learning theory* (pp. 174–188). Berlin: Springer.

Gerkey, B. P., & Mataric, M. J. (2002). Sold!: Auction methods for multirobot coordination. *IEEE Transactions on Robotics and Automation*, *18*(5), 758–768.

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.

Karp, R.M. (1992). On-line algorithms versus off-line algorithms: How much is it worth to know the future? In *IFIP Congress (1)* (Vol. 12, pp. 416–429).

Kartal, B., Godoy, J., Karamouzas, I., & Guy, S. J. (2015). Stochastic tree search with useful cycles for patrolling problems. In *Proceedings of the IEEE international conference on robotics and automation (ICRA)* (pp. 1289–1294).

Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *Proceedings of the European conference on machine learning* (pp. 282–293).

Labbé, M., & Michaud, F. (2014). Online global loop closure detection for large-scale multi-session graph-based SLAM. In *Proceedings of the IEEE/RSJ international conference on intelligent robots and systems* (pp. 2661–2666).

Lan, X., & Schwager, M. (2016). Rapidly exploring random cycles: Persistent estimation of spatiotemporal fields with multiple sensing robots. *IEEE Transactions on Robotics*, *32*(5), 1230–1244.

Liu, Y., & Chopra, N. (2009). Controlled synchronization of robotic manipulators in the task space. In *Proceedings of the ASME dynamic systems and control conference* (pp. 443–450).

Liu, L., Michael, N., & Shell, D. A. (2015). Communication constrained task allocation with optimized local task swaps. *Autonomous Robots*, *39*(3), 429–444.

Liu, L., & Shell, D. A. (2012). Large-scale multi-robot task allocation via dynamic partitioning and distribution. *Autonomous Robots*, *33*(3), 291–307.

Luo, L., Chakraborty, N., & Sycara, K. (2012). Competitive analysis of repeated greedy auction algorithm for online multi-robot task assignment. In *Proceedings of the IEEE international conference on robotics and automation* (pp. 4792–4799).

Mathew, N., Smith, S., & Waslander, S. (2015). Multirobot rendezvous planning for recharging in persistent tasks. *IEEE Transactions on Robotics*, *31*(1), 128–142.

Meyer, J., Sendobry, A., Kohlbrecher, S., Klingauf, U., & von Stryk, O. (2012). Comprehensive simulation of quadrotor UAVs using ROS and Gazebo. In *Proceedings of the international conference on simulation, modeling and programming for autonomous robots (SIMPAR)*.

Mills-Tettey, G. A., Stentz, A., & Dias, M. B. (2007). *The dynamic Hungarian algorithm for the assignment problem with changing costs*. Technical Report, Carnegie Mellon University.

Mitrović-Minić, S., Krishnamurti, R., & Laporte, G. (2004). Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, *38*(8), 669–685.

O. S. R. Foundation. Robot operating system. http://www.ros.org/. Accessed 7 February 2017.

O. S. R. Foundation: Gazebo: Robot simulation made easy. http://gazebosim.org/. Accessed 13 October 2017.

Omidshafiei, S., Agha-Mohammadi, A., Amato, C., Liu, S., How, J. P., & Vian, J. (2017). Decentralized control of multi-robot partially observable Markov decision processes using belief space macro-actions. *International Journal of Robotics Research*, *36*(2), 231–258.

Petersen, K., Kleiner, A., & von Stryk, O. (2013). Fast task-sequence allocation for heterogeneous robot teams with a human in the loop. In *Proceedings of the international conference on intelligent robots and systems* (pp. 1648–1655).

Schaefers, L., & Platzner, M. (2015). Distributed Monte Carlo tree search: A novel technique and its application to computer Go. *IEEE Transactions on Computational Intelligence and AI in Games*, *7*(4), 361–374.

Shin, K. G., & Ramanathan, P. (1994). Real-time computing: A new discipline of computer science and engineering. *Proceedings of the IEEE*, *82*(1), 6–24.

Smith, A. Github/smithan7. https://github.com/smithan7.

Smith, R. N., Das, E. C., Heidarsson, H., Pereira, A. M., Arrichiello, F., Cetnic, I., et al. (2010). Usc CINAPS builds bridges. *IEEE Robotics & Automation Magazine*, *17*(1), 20–30.

Tumer, K., & Agogino, A. (2009). Multiagent learning for black box system reward functions. *Advances in Complex Systems*, *12*(04n05), 475–492.

Yoshizoe, K., Kishimoto, A., Kaneko, T., Yoshimoto, H., & Ishikawa, Y. (2011). Scalable distributed Monte-Carlo tree search. In *Proceedings of the fourth annual symposium on combinatorial search*.

Yu, J., Schwager, M., & Rus, D. (2016). Correlated orienteering problem and its application to persistent monitoring tasks. *IEEE Transactions on Robotics*, *32*(5), 1106–1118.

Zavlanos, M. M., Spesivtsev, L., & Pappas, G. J. (2008). A distributed auction algorithm for the assignment problem. In *Proceedings of the IEEE conference on decision and control* (pp. 1212–1217).

Zheng, X., & Koenig, S. (2009). K-swaps: Cooperative negotiation for solving task-allocation problems. In: *Proceedings of the international joint conference on artificial intelligence* (Vol. 9, pp. 373–378).

Zlot, R., Stentz, A., Dias, M. B., & Thayer, S. (2002). Multi-robot exploration controlled by a market economy. In *Proceedings of the IEEE international conference on robotics and automation* (Vol. 3, pp. 3016–3023).

Andrew J. Smith is a Robotics Ph.D. candidate at Oregon State University in the School of Mechanical, Industrial, and Manufacturing Engineering. His primary research focus is on distribute robotic systems an multiagent coordination. He received his Bachelors and Masters of Science in Mechanical Engineering from the University of Nevada, Reno where he focused on UAV design, path planning, and control.

Graeme Best is a Postdoctoral Scholar in the Collaborative Robotics and Intelligent Systems (CoRIS) Institute at Oregon State University. His research interests include planning algorithms, multi-robot coordination and robotic active perception. He received his Ph.D. (2018) at the Australian Centre for Field Robotics (ACFR) at The University of Sydney, Sydney, Australia, and his B.E. in Electrical and Computer Systems Engineering (2014) and B.Sc. in Computer Science (2014) at Monash University, Melbourne, Australia.

Javier Yu is a senior Mechanical Engineering student from the University at Buffalo with minors in mathematics and computer science. His research interests are in mobile robotics, bio-inspired robots, and multi-robot systems. He is a recipient of the NSF GRFP and will be pursuing his Ph.D. in Robotics at Stanford.

Geoffrey A. Hollinger is an Assistant Professor in the Collaborative Robotics and Intelligent Systems (CoRIS) Institute at Oregon State University. His current research interests are in adaptive information gathering, distributed coordination, and learning for autonomous robotic systems. He has previously held research positions at the University of Southern California, Intel Research Pittsburgh, University of Pennsylvania's GRASP Laboratory, and NASA's Marshall Space Flight Center. He received his Ph.D. (2010) and M.S. (2007) in Robotics from Carnegie Mellon University and his B.S. in General Engineering along with his B.A. in Philosophy from Swarthmore College (2005). He is a recipient of the 2017 Office of Naval Research Young Investigator Program (YIP) award.