Dynamic Task Allocation Approaches for Coordinated Exploration of Subterranean Environments

Matthew O'Brien[1*], Jason Williams[2], Shengkang Chen[1], Alex Pitt[2], Ronald Arkin[1] and Navinda Kottege[2]

[1*]Mobile Robot Lab, Georgia Institute of Technology, Atlanta, GA, USA.
[2]Robotics and Autonomous Systems Group, CSIRO Data61, Pullenvale, Qld, Australia.

*Corresponding author(s). E-mail(s): mjobrien@gatech.edu;
Contributing authors: jason.williams@csiro.au; schen754@gatech.edu; alex.pitt@csiro.au; arkin@cc.gatech.edu; navinda.kottege@csiro.au;

Abstract

This paper presents the methods used by team CSIRO Data61 for multi-agent coordination and exploration in the DARPA Subterranean (SubT) Challenge. The SubT competition involved a single operator sending teams of robots to rapidly explore underground environments with severe navigation and communication challenges. Coordination was framed as a multi-robot task allocation (MRTA) problem to allow for a seamless integration of exploration with other required tasks. Methods for extending a consensus-based task allocation approach for an online and highly dynamic mission are discussed. Exploration tasks were generated from frontiers in a map of traversable space, and graph-based heuristics applied to guide the selection of exploration tasks. Results from simulation, field testing, and the final competition are presented. Team CSIRO Data61 tied for most points scored and achieved second place during the final SubT event.

Keywords: multi-robot task allocation, exploration, field robots, subterranean challenge

1 Introduction

The DARPA Subterranean Challenge (SubT) [1] pushed the limits of teams of robots in complex, underground environments. These types of environments are notoriously dangerous during emergencies and disasters. Inspired by the needs of personnel and first responders in emergency situations, the competition involved heterogeneous teams of robots exploring tunnels, caves, and urban underground environments like subways. The objective for these robots was to localize survivors, gas leaks, and other important objects. One of the key requirements for a successful team was autonomy. A limit of a single operator and the difficulty of establishing wireless communications in underground environments meant that human supervision was limited. The volume to be searched along with a one-hour time limit meant that having multiple agents exploring in parallel was necessary to succeed.

This paper describes the approach taken by Team CSIRO Data61 (consisting of CSIRO Data61, Emesent and Georgia Tech) for coordination of exploration using multi-robot task allocation. Exploration tasks are created, bid upon, and executed autonomously by agents during a mission. The natural constraint of one agent per task ensures the team will pursue multiple goals at once, and the reward of a task provides a straightforward way to incorporate information on how valuable an area is expected to be for exploration. Coordinating exploration within a multi-robot task allocation (MRTA) framework provides two key benefits over methods that coordinate exploration at a lower level. First, other types of tasks can be easily integrated, as the framework is task-agnostic. Little work was required to introduce the capability for one agent to ferry data for multiple agents back to the base station, for example. Second, it provides useful ways for the operator to inject human knowledge and improve the team's performance. Allowing an operator to change assignments and rewards by task or region provides an intuitive interface into a complex system.

For this competition, a dynamic consensus-based task allocation system was developed. The consensus auction protocol, inspired by [2], provides robustness to communications breakdown. By having every agent reach a consensus, versus a single agent acting as an auctioneer, any sub-group of communicating agents will be able to coordinate. This is critical in underground environments, where concrete and rock can quickly block wireless communications. The dynamic nature also allows agents to continuously re-plan as the mission evolves, new tasks are found, and old tasks are completed.

To define exploration as discrete tasks, we use frontiers in an agent's traversability graph, which represent the areas in which it is possible to explore. Clusters of these frontiers become an exploration task. While frontiers do a good job of representing where an agent can explore, they offer little information about the best place to explore. Rather, the utility of a frontier is primarily determined by its location relative to all agents. In an unknown environment, spreading agents out is likely to minimize overlap and provide the fastest coverage. Based on a multi-robot tree exploration approach which was shown to

be optimal under some conditions [3], a set of heuristics were created to bias the rewards of individual explore tasks and recreate the team-level strategy in a more realistic and dynamic scenario.

In the final run of the DARPA Subterranean Challenge, Team CSIRO Data61 fielded four ground platforms running the task allocation system along with two UAVs that were launched by the operator and explored independently. This team tied for the most points scored (detecting 23 objects of interest), and finished in second place on the tie breaker criterion of the time of last detection (45 seconds after the winning team over the one-hour run). This paper will detail these methods and strategies used for multi-agent coordination by Team CSIRO Data61 during this final event.

2 Related Work

Exploration of unknown environments is a common task for mobile robots, and has been an active subject of research for decades. While a considerable amount of work has gone into exploration of graph structures (e.g., [3], [4]), the bulk of robotics literature focuses on continuous environments often represented as occupancy grids. Approaches range from pure behavior-based [5] to information-theoretic [6][7]. A seminal paper introduced the idea of using frontiers, the boundaries between unexplored and explored space, as the basis for exploration [8]. Frontier-based exploration approaches have been widely utilized due to their simplicity and effectiveness [9][10][11]. Recent work has also focused on hierarchical methods to separate local exploration and global coverage, and provide a scalable solution for large environments [12] [13].

Exploration is an ideal task for teams of robots, where multiple agents can work in parallel and provide robustness to individual platform failure. Strategies can focus on how best to distribute agents [14] or how to maintain communication constraints [15]. Frontiers are again a popular approach as they provide discrete goals for exploration to distribute among a team. The work in [16] provided benchmarks for some of the most popular approaches, including simple greedy assignment, heuristic-based methods, solving a multiple traveling salesman problem, or using task allocation approaches.

Multi-Robot Task Allocation (MRTA) is the problem of finding the assignment of a set of tasks to a set of agents that maximizes reward given the constraints of the agents and environment. Outside of the simplest case, where every robot is assigned one and only one known task, MRTA problems are NP-hard [17]. Approaches thus look for balances between finding effective solutions and minimizing computation and communication costs. The excellent survey of modern MRTA approaches found in [18] breaks them down along two main dimensions. Solutions fall broadly into either optimization-based or market-based approaches. Optimization approaches can take a range of approaches from integer linear programming [19] to simulated annealing [20]. Market-based draw inspiration from economic theory, and use auctions to allocate tasks by having agents make competing bids based on how suitable they are

to complete a task. Either approach can be implemented in a centralized or decentralized manner. Centralized systems have an off-platform base-station, or leader platform, find an assignment solution and distribute that to all agents, while decentralized approaches either cooperatively build the solution, or have agents each individually solve the team's assignment.

One of the earliest applications of MRTA approaches to multi-robot exploration is from [21]. In this work agents asynchronously bid on frontiers to visit based on path costs and expected information gain. A centralized executive program greedily assigns the highest bid, and then discounts all unassigned bids based on the expected overlap with assigned bids. A similar design was used to win the 2010 MAGIC competition [22], where a fourteen agent team performed an urban search and rescue mission to identify both friendly and hostile humans as well as bombs. As with the previous work, exploration is done by moving towards goal points on frontiers, with the expected information gain and distance being used for the reward and cost of the task. The centralized ground control station greedily assigned tasks based on bids.

Recent work looked at a different variation of the exploration problem, where potential locations for targets were known, and agents must destroy targets if found [23]. A two-phase auction based approach was implemented. Before the start of a mission, agents bid in phases, where the bid with the lowest travel distance wins out. This continues until all tasks are assigned, as agents are allowed to bid repeatedly and build an ordered list of tasks. During execution, new tasks are dynamically bid on, but agents only consider the addition of the new task within its list, without reordering previous assignments, to avoid combinatorial growth in computation.

One approach for MRTA is particularly relevant from the problem domain this paper focuses on. The Consensus-Based Bundle Algorithm (CBBA) [2] introduced a new distributed auction process where agents broadcast their bids globally, and each agent executes the same rules to determine who wins what task. As long as agents are communicating, they will form a consensus on the allocation of tasks. This work also demonstrated that when agents greedily build an ordered list of tasks (known as a bundle) one task at a time, the total team performance is guaranteed to reach at least half the reward of the optimal allocation.

For dynamic missions in underground environments, this offers two advantages. Consensus-based auctions offer coordination with minimal communication. Neither needing repeated back-and-forth messages (as in approaches with designated auctioneers) or the full mission state to be shared, as in approaches that repeat allocation on each agent independently. It smoothly allows for sub-groups of communicating agents to coordinate, without any additional overhead or logic. Second, the bundle building procedure offers a good balance of performance with reasonable computation loads. Missions into unknown areas are inherently dynamic, and the best solution can rapidly change. Pursuing higher quality solutions will often be wasted effort.

However, the original CBBA algorithm cannot deal with the dynamic nature of the SubT challenge, as it is designed to be run offline before tasks are executed with full knowledge of the mission. Extensions exist to deal with this; one of the first is Asynchronous CBBA (ACBBA) which updated consensus rules to allow for running the bundle building and consensus phases simultaneously and asynchronously [24]. ACBBA also introduced the idea of an individual agent dropping its entire bundle to replan when faced with significant change in its environment. An alternative approach allowed for more limited replanning, dubbed CBBA with Partial Replanning (CBBA-PR) [25]. Here, agents could drop the tasks with the lowest bids from the bundle in each auction round.

The work in this paper extended the ideas described above in several ways. A graph-based exploration was integrated with the frontier-based task allocation by providing heuristics to weight the reward for certain agents to execute certain tasks. A new method for continuously replanning an agent's bundle and updating bids is introduced. This work also presents an effective rule for dealing with failed attempts at tasks, and a simplified set of consensus rules. The methods were rigorously tested with a range of simulated experiments, hardware experiments, and in physical competition.

3 Task Allocation

The approach for dynamic task allocation can be decomposed into the three processes that run continuously in parallel. First, a **task set** uses a consensus protocol to maintain a consistent set of tasks with every communicating agent and the base station. Second, a **task bundle** continuously selects the best available task for an agent and bids on it, building a bundle of owned tasks to execute. Finally, a **task monitor** continuously executes the leading task in the bundle, and monitors it until completion or failure.

3.1 The Task Set and Consensus

The Task Set is a list of all tasks and relevant information about them, and represents the current state of the mission as known by an agent. This includes information about the task itself (the type, location, base reward, etc). It also includes information related to the auction process: The state of the task (unassigned, assigned, executing, complete), the owning agent (i.e. the agent it is assigned to), the winning bid amount, the last update time, etc. As an agent creates, bids on, and executes tasks it can update its own task set directly. The key feature of the task set is the use of a consensus protocol to manage updates from other agents.

This consensus protocol acts as the auction mechanism for task allocation. Rather than having a single auctioneer, agents collaboratively fulfill that role. Updates to a task, such as new bids, are broadcast globally and all agents use the same rules to determine which update is valid and whose bid wins. As long as communication is maintained, agents will form a consensus. Figure 1 shows

how consensus is reached in a simple case of the base station broadcasting a new task. This decentralized approach allows agents to generate and distribute tasks among any sub-group that shares communication. The inherent robustness to limited communication makes the approach ideal for the SubT competition, where communication is lost frequently and for long durations.

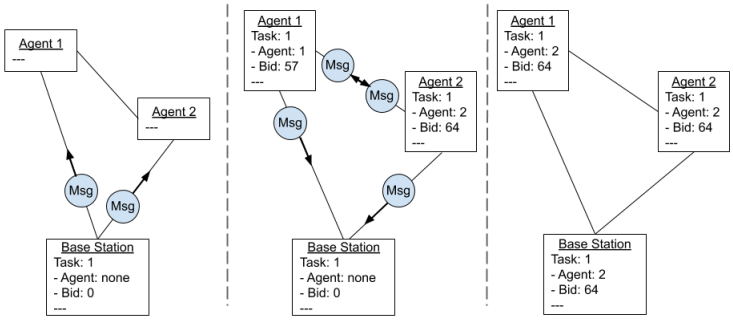

Fig. 1 (a) A task is generated at the base station and broadcast to two agents to execute. (b) Initially both agents bid on the task, assigning themselves as the owner in their own task set. (c) Each agent receives the other agent's bid, and the base station receives the bids from both agents. All come to consensus that agent 2, which has a higher bid, will execute Task 1.

The consensus protocol is used for more than an auction mechanism. Agents may renege on tasks to execute different ones, or platforms may stop running. Communication failures can lead to two agents executing the same task, an agent being unaware a task has been completed, etc. The consensus procedure must be able to resolve any conflict between the task sets of two agents. This can be done with four rules defining when an agent updates the information in its task set. From highest to lowest priority, an agent will update a task based on an incoming message when:

1. The new message has the task in a terminal state.
2. The new message has a more recent manual assignment time.
3. The current set and new message show the task as owned, and the new message has a higher bid.
4. The new message has a more recent auction update time

The first three rules are simple. Terminated tasks are finished and thus can't change, manual assignments by the operator supersede autonomous decisions, and agents contesting ownership of a task will have it go to the highest bidder (bidding ties are resolved by alphanumeric ordering of agent names). For all other conflicts, update time is used as the resolution, with more recent updates taking priority. The auction update time changes whenever an agent makes a change to the task information, usually due to bidding, executing, or releasing the task. The dynamic nature of the task allocation problem in this

work means that most changes to a task are equally valid in reverse. Consider if agent A believes some task is being executed by agent B who is no longer in communication range, and then agent C enters communication range and sends an update that the task is unassigned. Was the task released by agent B, or was agent C never informed about the task being executed in the first place? Based on the task status and bid information alone, it is impossible to tell.

The key mechanism for consensus is therefore the update time, i.e., the time the status or bid of a task was last changed. This simplifies the consensus protocol to the above set of rules. Note that while this provides a method for consensus, it does not answer the underlying problem of which agent was correct. When two agents out of communication conflict, update time does not necessarily provide the optimal resolution, but it provides a consistent rule for resolution every agent can use. In rare edge cases an agent may lose ownership of a task due to a consensus update, even though it is the best agent to execute said task, but it will immediately have the opportunity to rebid on the task.

The balance between updating the current solution, or resetting and building a new solution is a repeating theme in this work. Excessively re-planning can have agents repeatedly leaving tasks incomplete and wasting time, but locking in a task once started potentially ignores significant changes to the state of the mission or conflicts with others.

3.2 The Task Bundle and Task Selection

An agent continuously builds and executes a task bundle, i.e., an ordered list of tasks. Some tasks will be more efficient to execute together, primarily due to their location and the time/energy costs of traveling to different areas. Task bundles allow an agent to consider these interactions.

Finding the optimal bundle even for a single agent is a combinatorial problem, and given the rate at which bundles must be updated, it would not make sense to continuously re-solve it. Instead, the approach from [2] is adopted and extended. When an agent looks to select a new task to add to its bundle, it finds the available task in the task set with the highest profit and bids on it. Profit is related to a task's reward, and will be defined momentarily. A task is "available" for an agent to select if the agent can execute it, there is no higher bid already placed on it, and the task has not been completed or recently failed (failures will be discussed more in Section 3.3.1).

Let us define c_j as the base reward of task j. The vector b_i is the bundle of agent i, the list of tasks in order they were added to the bundle. In contrast, p_i is the "path," a list of the same tasks in the order the agent will execute them. Both orders are important and will be used. $S_i(p_i)$ is the expected total reward of agent i for executing the bundle (or path) p_i. $\tau_i^j(p_i)$ is the expected time for agent i to complete task j (including all prior tasks in the bundle), i.e., if $p_i = (j_1, \ldots, j_n)$ and $j = j_k$, then

$$\tau_i^j(p_i) = \sum_{k'=1}^{k} t(j_{k'-1}, j_{k'}) + T(j_{k'}) \tag{1}$$

where $t(j',')$ denotes the expected time to travel from task j' to task j, $t(j_0, j)$ denotes the time to travel to task j from the agent's current position, and $T(j)$ denotes the expected duration involved in executing task j.

The total reward of a bundle is the sum of the rewards of each task, with an increasing time discount applied to each successive task:

$$S_i(p_i) = \sum_j \lambda^{\tau_i^j(p_i)} c_j \tag{2}$$

where $0 < \lambda < 1$ is the time-discount parameter. The traversability costs $t(j, j')$ are calculated based on the global topometric graph described in [26]. Specifically, traversability data is collected and consolidated into submaps corresponding to each SLAM frame. Superpixel decompositions are computed for each submap, and a simplified graph formed with one node per superpixel, and edges between neighboring superpixels, as well as between superpixels that coincide in neighboring submaps. Edge costs are determined based on distance between superpixels, as well as the slope and roughness of the two superpixels (also including a risk-related penalty). Thus, task costs provide a prediction of the navigation cost that accounts for both the path distance and terrain difficulty. Part of the global topometric graph from the SubT finals event is illustrated in Figure 2.

As mentioned, a bundle is an ordered list of tasks, and the order will impact the total reward. When an agent is selecting a task, each position in the bundle is checked to find which maximizes reward. The profit of a task is the total increase in bundle reward from adding the task into its best index. Let us introduce the insertion operator, $\oplus_n$, used to designate the inclusion of a new task into a bundle at the index n. Therefore $p_i' = p_i \oplus_n j$ means the new bundle p_i' is made of the old bundle p_i with the task j inserted at index n. With this, we can write down the profit of a task j for agent i as:

$$c_{ij}[b_i] = \max_n \{S_i(p_i \oplus_n j)\} - S_i(p_i) \tag{3}$$

In other words, the profit is the reward of the best bundle created with the new task, minus the reward of the original bundle. This profit will always be less than the base reward of the new task, as its own reward is discounted due to the time needed to execute tasks in front of it, and its inclusion will further discount the reward of any task following it. The profit of a task is the bid an agent will place on it and send to other agents as part of the auction process.

3.2.1 Updating Bundle Bids

An agent may renege on a task, or have a task stolen by another agent. One issue this creates is that the profit of a task, and thus the bid placed on it,

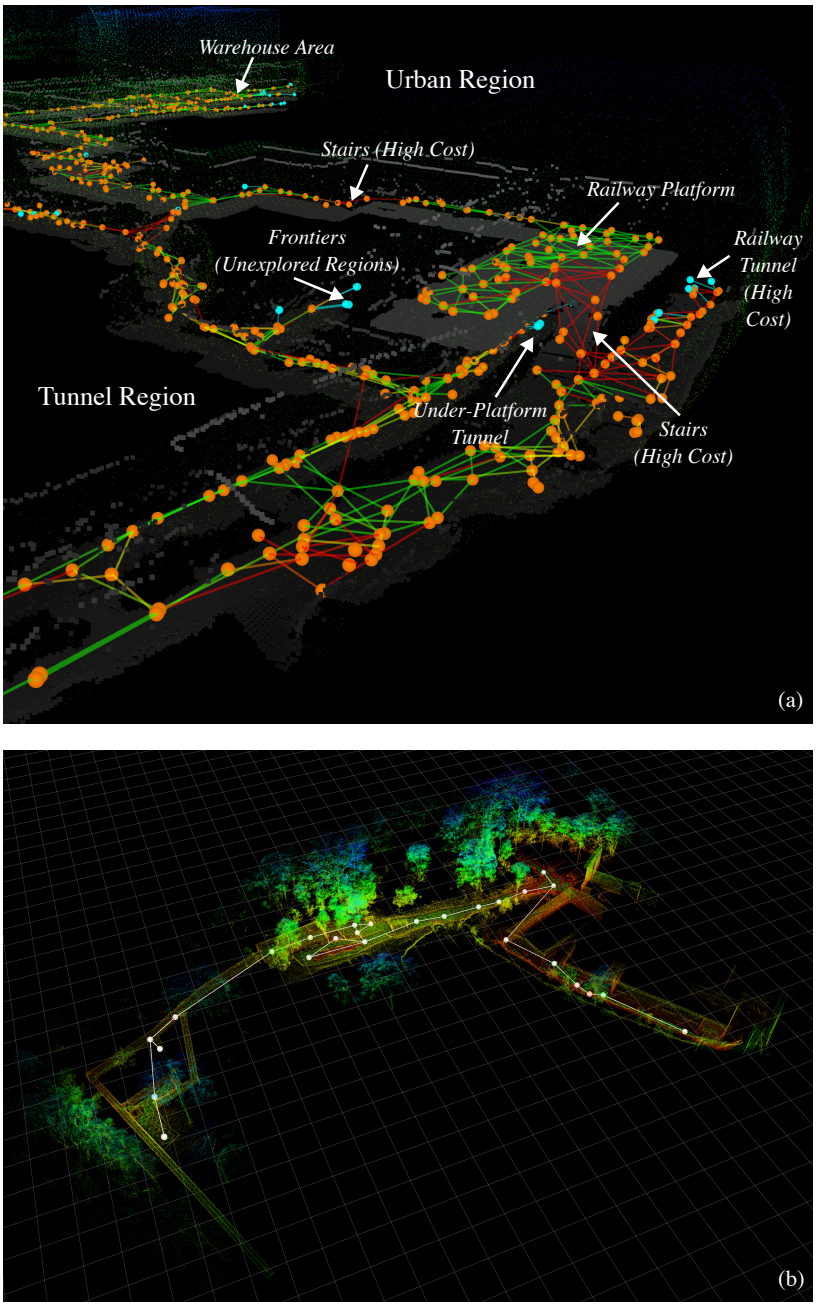

Fig. 2 (a) portion of topometric graph from finals course, showing railway platform, railway tunnel, tunnel underneath platform, and other parts of urban and tunnel courses. Orange circles denote traversable superpixels, while cyan circles denote frontiers, i.e., boundaries to unknown space. Edges color code traversability cost, with green denoting low cost and red denoting high cost. (b) topological graph from field test described in Section 6.2 (shown in white).

Algorithm 1 Bundle Update algorithm, for re-evaluating rewards when a task is removed from the bundle. Note that we do not explicitly show the execution order of the bundle, but this is maintained alongside the bundle and used in reward evaluations as previously described.

Require: Bundle $b_i = (j_1, \ldots, j_n)$, ordered by when the tasks were inserted into the bundle; path $p_i = \{j : j \in b_i\}$ ordered by when tasks will be executed; order of task to be removed k, $1 \leq k \leq n$; cached reward of the bundle prior to inserting k-th element $S_i^{[b_i]_{1:k-1}}$.

Ensure: Bundle with the profit for each task updated.

$b \leftarrow (j_1, \ldots, j_{k-1})$
$p \leftarrow p_i \setminus j_k$
$S \leftarrow S_i^b$
for $k' \in \{k+1, \ldots, n\}$ **do**
    $b' \leftarrow b \oplus j_{k'}$
    $S' \leftarrow S_i(b')$ (preserving previous path order)
    $c_{ij_{k'}}[b'] \leftarrow S' - S$
    $S \leftarrow S'$
    $b \leftarrow b'$
end for

depends on the other tasks in the bundle. When a task is removed from the bundle for any reason, every task that was added after it no longer has a valid value for profit. It is important to clarify that here, "after" does not mean further back in the queue of the task bundle, but that the task was inserted into the bundle later in time. Thus later in the list b_i, not p_i. We will refer to this as the insertion order.

The solution used in [2] is to drop all tasks with a higher insertion order then the task being removed (i.e., all tasks later in b_i) and let agents rebid on the released tasks. While effective, this creates additional communication costs. When task allocation is performed online, it could also interrupt and drop the task being executed by an agent. The alternative used in this work is to have an agent update the profit, and therefore bids, of the tasks based on the current bundle without removing them.

The process is shown in Algorithm 1. Starting with the bundle b before any task with insertion order k or later was added, we can add the tasks that were not removed, one at a time, based on their insertion order. Let $p_i(j)$ provide the index of task j in the path p_i, i.e. the order it is to be executed. This order does not change, all tasks after the one being removed simply move one step forward. Each step, the total reward for agent i, $S_i(b)$, and therefore the new profit and bid for the task, are recalculated.

Except for unusual circumstances where the agent has been relocated in a manner inconsistent with the task bundle, the profit for any individual task will increase when another task is removed, so an agent will generally not lose ownership due to this process decreasing a bid. As discussed, the profit of a task is how much it increases the bundles total reward (equation 2), which will

be less than the base reward c_j of the task. A task ahead in execution order of task j will increase the time $\tau_i^j(p_i)$, and therefore decrease the discount factor $\lambda^{\tau_i^j(p_i)}$ applied to the base reward c_j. Similarly, when task j is inserted it increases all the times and discounts for tasks after it, reducing how much reward they provide and thus the total profit of task j. Whether they will be executed before or after, a task with non-zero time to travel and execute reduces the reward gained from other tasks in the bundle, thus its removal increases the profits of other tasks.

This process corrects individual bids, but drops the guarantees on solution quality provided by the original CBBA algorithm when building a task bundle. In this case we choose to update the known solution, rather then resetting and rebuilding a potentially better solution. The guarantees on a bundle's performance are based on a mission with a priori known and fixed tasks/agents, and thus are invalid anyway without frequent full resets due to the dynamic nature of the mission. The next section describes a specific replanning-procedure used to update bundles without needing to reset them entirely.

3.2.2 Rebuilding Bundle

In a highly dynamic mission, creating large bundles that plan far into the future is likely to be a waste. The mission will usually change significantly before the bundle completes. This is particularly true for exploration, where the endpoint of tasks is unknown a priori. Instead, limits are placed on a bundle, both for the number of tasks and the estimated time to execute them. This introduces a new issue: How does the agent respond to changing conditions after its bundle is full?

A simple approach proved to be effective. Dropping the last task in the bundle path (i.e., execution order, not insertion order) allows for the agent to bid on new tasks. This does not limit an agent to swapping only the final task in p_i, as a new task can be inserted in any location in the bundle. An agent continuously checks if dropping a task allows it to pickup another task that creates a bundle with higher reward. To add some hysteresis to the bundle, the last task is only dropped if the the replacement task improves the overall bundle reward by more then a small threshold.

Empirically, dropping the final task is particularly effective. When a non-final task is removed, the agent's bundle path must be updated to connect the prior and subsequent tasks. The new path segment may not be efficient in terms of reward per distance without the original task in the middle. Since only one task is being dropped at a time (to limit the number of bundles to continuously evaluate) the new path segment is likely near the dropped task, and often no better option will exist. This can be thought of as a local minimum. The final task, by contrast, is only connected with one other task. As it removes a path segment without adding any, this leaves more flexibility in terms of tasks that will provide a net increase in reward.

Message Type	Channel Count	Send Condition
Single Task Definition	One Shared Channel	Update (high-freq)
Bundle Auction Info	Channel Per Agent	Update (high-freq)
All Task Definition	Channel Per Agent	Timer (low-freq)
All Auction Info	Channel Per Agent	Timer (low-freq)

Table 1 Agents update each other frequently by broadcasting changes to individual tasks, or their own task bundles. The full task set is broadcast out at a regular but slower interval to balance synchronization with communication bandwidth.

3.2.3 Reducing Bandwidth

Due to the potential that a neighboring agent was out of communication range for some time, agents need to share all the information in their task set to ensure other agents are up-to-date. Agents must also update each other frequently as they continuously bid on and execute tasks. Though task messages are not particularly large, we want to avoid naively broadcasting the entire task set at a high frequency in a communication-constrained challenge where all bandwidth is valuable.

One method of limiting excess communications bandwidth is the division of task information into a definition of the task itself (task type, position, reward, etc) and the auction-related information (task state, owning agent, highest bid, update time, etc). The auction information is likely to change far more frequently as agents bid on tasks, while the definition of a task is usually constant. When agents broadcast new bids or renege on tasks, they only broadcast the auction message.

Communication channels are also split into high/low frequency options. High-frequency channels, for both task definitions and auction information, are broadcast any time there is an update: A new task is generated, a new bid is made, etc. For the auction channels, agents send their whole bundle each time the bundle changes. This allows an agent to quickly receive the full bundle of every connected agent, and get the current state of which agents own which tasks. These bundles are still significantly smaller than broadcasting the full task set.

Low-frequency channels broadcast the entire set of definitions or auction states on a timer (in competition, every 30 seconds). These ensure the full task set will be shared without an agent having to wait too long. Table 1 summarizes the four communication channels used.

3.3 Task Execution and Monitoring

Task execution is a parallel process to the maintenance of the task set and the construction of the task bundle. Whenever the agent is not executing a task, and the task bundle is not empty, it will execute the leading task. Each task generates the appropriate list of commands: A list of primitive and composite behaviors the agent needs to perform to complete a task.

When a task completes, either successfully or not, the task's information in the task set is updated and it is removed from the agent's bundle. The next

task in the bundle is then executed. Executing tasks can also be stopped and released if the agent finds a better task to bid on and reneges on its original task, or if another agent provides a better bid and steals ownership.

3.3.1 Exponential Back-off for Failed Tasks

With real robots in complex environments, it is common for attempts to complete a task to struggle or stall, but this does not necessarily mean that the task cannot be completed. Balance is needed between stopping a robot from wasting too much time trying to complete a task, and choosing to ignore tasks too quickly. A notion of a failed attempt is used for tasks. Most commonly, this is due to the agent being unable to progress towards the goal for too long. The relevant question is then how these failures should be treated by the task allocation system.

Inspiration from communications networks is used to answer this question. Exponential back-off is a protocol for managing transmissions when interference can cause transmissions to fail [27]. After each failure, the average amount of time before the next attempt at transmission increases exponentially. In this work, a task that fails is released from ownership, becoming unassigned, and blocked from being selected for some time. The time T^j_{blocked} the task j is blocked increases exponentially with the number of failures f_j, based on the equation

$$T^j_{\text{blocked}} = 30 \times 2^{f_j} \tag{4}$$

The first failure will only block the task briefly, which gives an agent the opportunity to move out of a potentially problematic location before attempting it again if no similar value tasks are around. The blocked duration will rapidly increase with repeated failures, ensuring that problematic tasks do not waste too much time, but allowing them to potentially be re-attempted later in the mission.

4 Task Types

In the SubT context, the vast majority of the agent's time is spent exploring. Accordingly, explore tasks will be described in detail in this section. Other tasks include "return to synchronize", "drop communications node" and "go to". Synchronization tasks require the agent to return towards the base station until all data are uploaded to and downloaded from the base, ensuring that the two contain equivalent data sets. These utilize the multi-agent coordination, allowing one agent to bid on another agent's synchronization task if it holds the data being synchronized. This avoids having multiple agents needing to return to the base simultaneously. Mechanisms were developed for autonomously generating drop node tasks, but were not employed due to the risk of a poorly placed communications node rendering narrow passageways impassable. Therefore both drop node and go to tasks are manually generated by the operator.

The task allocation system allocates tasks to agents based on the rewards that agents receive for doing them. That alone will not lead to effective exploration of an environment. Exploration tasks must be created in a way that provides both complete and efficient options for exploration. If the rewards for these tasks are poorly chosen, even an optimal allocation of tasks (in terms of net reward) could lead to poor team performance during actual execution. The remainder of this section provides an overview of how exploration tasks are created based on frontiers, before detailing how rewards are modified to drive the MRTA system to perform efficient exploration.

4.1 Creating Exploration Tasks

Exploration of environments such as those in SubT involves a range of complex 3D structures such as steep ascents and descents, stairs, rubble piles, partially see-through barriers, and passages above and below each other. Initially, exploration tasks were generated by forming a 3D mesh describing the boundary of observed space [28], but this was found to be undesirable for ground agents as it results in a loose tie between frontiers and the path to observe them. For example, a region may be entirely observed from the top of a platform, but it is still necessary to traverse the lower part of the region to find paths to subsequent areas. Subsequently, the focus switched to analysis of traversability frontiers, i.e., a boundary between traversable and unknown space. Thus the goal of the system is to progressively expand the map of known traversable space. Three dimensional features are accommodated by allowing submaps to contain multiple layers.

As described in [26], global traversability submaps are tied to SLAM submaps, which are communicated and solved independently on each agent, allowing each agent to develop a representation of the global traversability map combining information from all agents. Each agent develops its own set of traversability frontiers based on its local solution of the global traversability map. A key advantage of this approach is that it accommodates the late arrival of data from different agents due to communications dropout. For example, if agent 1 traverses a region and loses communications with agent 2, exploration tasks for agent 2 will lie at the traversability frontier from the latest submap received from agent 1. If agent 2 continues in this direction, its known traversable space will be expanded based on its own observations. When communications are reestablished between the agents, each agent will update its own traversability frontiers (and hence exploration tasks) based on the combined data from the two agents. By fusing raw data, we avoid the need to agree on task labels (e.g., agent 1's frontier #13 is equivalent to agent 2's frontier #45). An example of a fused global traversability map from the final event is shown in Figure 2.

Frontier pixels in the traversability submaps are initially clustered based on the superpixel methods described in [26]. Clusters of frontier superpixels are then formed based on an k-means approach adapted to incremental updates. Specifically, we incrementally add updated frontiers to the cluster containing

the nearest existing frontier, creating a new cluster if the diameter of the cluster will become too large. By using the global traversability graph to measure distances, we avoid clustering frontiers that are near by Euclidean distance, but far by paths, e.g., frontiers in adjacent rooms that require long traversal, or frontiers on different floors of a building. The diameter threshold is set to 15m, which in the environments of interest will likely capture all frontiers in a particular tunnel or room, for example, but split tasks between different tunnels/rooms.

A consequence of this process is that different agents will have their own set of equivalent explore tasks. For this reason, the bidding process is amended such that each agent can only bid on explore tasks within its own set, but two tasks from different agents will be considered equivalent if they are within a distance threshold of each other, and thus a bid on one task will only be successful if it outbids a nearby explore task that is assigned to another agent (i.e., the two tasks are considered as one within the bidding process). When communications are strong and agents are fully synchronized, their submaps and generated explore tasks will be nearly identical, but as communications breakdown this provides a graceful way for agents to rectify tasks over diverging maps.

4.2 Utility of an Exploration Task

It is common for the reward of an exploration task to factor in estimates of the size (e.g., volume) of the space that may lie behind the frontier. However, this often leads to undesirable behavior. For example, when progressing down a narrow tunnel, the new volume being discovered seems small compared to a nearby task in an open area, so it may seem beneficial to interrupt the current task and move to the open area. Similarly, small doorways will rarely seem sufficiently rewarding to enter compared to the larger open area to which they are attached. Without awareness of this semantic information (e.g., the specific value of narrow tunnels and doorways in the context), size-based rewards were found to be counter-productive, and were thus dropped.

If the goal is for the entire environment to be explored, one could assign all explore tasks equal reward, and allow the task allocation system to determine an efficient way to allocate them. In a situation with a static mission, that would be effective. For dynamic missions into unknown environments, the task allocation system tries to optimize received reward by completing *known* tasks as fast as possible. It does not consider where new undiscovered tasks might be. For basic situations, like a tunnel with two distinct branches, tasks will naturally allocate agents to different areas. However as environments get more complicated with loops, large areas containing multiple explore tasks, and agents that are doing more than exploring (e.g., returning to offload data), agents may end up executing tasks nearby each other. This is usually less efficient then having one agent explore a region independently.

An intuitive strategy would be to instead distribute agents to different areas as much as possible; work on multi-robot exploration of trees and graphs

[3] bears this out. In this work, individual agents explore depth-first, while hypothetical book-keeping devices at each vertex in a graph track which edges agents have traversed. When encountering a visited node, an agent chooses edges that have not been explored first, and then follows edges with the least agents downstream if no alternatives are available. This was shown to be optimal for two agents on a tree, and provide good performance in more general cases. In summary, the idea is to make individual agents explore depth-first, while the team explores breadth-first.

To leverage this idea, there are several differences with the SubT competition that must be taken into account. The most important is that agents are not exclusively exploring. They have other tasks, and must occasionally return to establish communications with the base station and dump data. Thus a sequential algorithm is not directly applicable. However, modifying the rewards of exploration tasks provides a natural way to bias agents to select tasks that best approximates the behavior. There are no bookkeeping devices, instead agents share their trajectory with each other, which includes their current position. Finally, a topological graph of the environment, expanding from the base station outwards, is created to represent the structure of the environment. Exploration tasks within this graph are the options for exploration.

The topological graph is built from an underlying dense topometric graph used for path planning and navigation via skeletonization [29]. The minimum distance from each topometric vertex to a boundary (wall or unknown space) is measured, and vertices that are local maxima with respect to this measurement become the vertices of the topological graph. This provides the high-level representation needed to reason over. An example from a real run is shown in Figure 2. As described above, each agent independently builds its own topometric map from local and shared cost map information. This means the topological maps will be unique, though generally the maps of different agents will be very similar.

Three heuristics are used to capture the desired behavior of the depth-search for individuals, breadth-search for the team. First, a basic depth reward increases the reward of exploration tasks based on how much further they are from the base station, compared to the agent. Second, a branch penalty applies to tasks on the same branch. When a task is evaluating its reward, it checks the path from itself to the base station in the topological graph against the equivalent path for any other explore task being executed. When these paths overlap, the overlap distance is added up to create a net penalty. Therefore the longer the overlap, or the more tasks it overlaps with, the more penalty accumulated. The ideal explore tasks therefore branch immediately at the root of the graph (the base station) into a section with no other active explore tasks. Finally, there is a simple nearest neighbor penalty. When the topometric distance to the nearest active explore task is within a threshold, every meter below that threshold increases the penalty. This has the same goal as the branching penalty, i.e., to spread agents out. On large scales, underground environments are usually well approximated by graphs, but on smaller scales

something like an open room can break the graph-based formulation. The nearest neighbor penalty helps push agents in a large room, for example, to take opposite sides.

Formalizing this, consider a task j from the viewpoint of agent i. Let $E(j)$ be the set of edges of the topological path from the base to task j. Let $\mathcal{X}_i$ denote the set of exploration tasks in the bundles of other agents (note that any agent can only hold a single exploration task), and $t(e)$ denote the cost (time) of traversing edge e. Then the depth reward ρ_d^j, branch penalty $\rho_b^{i,j}$ and nearest neighbor penalty $\rho_n^{i,j}$ are given by:

$$\rho_d^j = \sum_{e \in E(j)} t(e) \tag{5}$$

$$\rho_b^{i,j} = \sum_{j' \in \mathcal{X}_i} \sum_{e \in E(j) \cap E(j')} t(e) \tag{6}$$

$$\rho_n^{i,j} = \sum_{j' \in \mathcal{X}_i} \max\{0, \eta - t(j, j')\} \tag{7}$$

Where η is the threshold distance under which the nearest neighbor penalty applies. The overall heuristic penalty $\rho^{i,j}$, and the new modified task reward $\hat{c}_j$, is then given by:

$$\rho^{i,j} = w_d \rho_d^j + w_b \rho_b^{i,j} + w_n \rho_n^{i,j} \tag{8}$$

$$\hat{c}_j = \lambda^{\rho^{i,j}} c_j \tag{9}$$

where $-\infty < w_d \leq 0$ denotes the weight for the depth reward, $0 \leq w_b < \infty$ denotes the weight for the branching penalty, and $0 \leq w_n < \infty$ denotes the weight for nearest neighbor. λ is the same time discount used in equation 2. All penalties are functions of distance, and impact the reward as if adding/subtracting distance required to travel to it in the bundle path.

This common currency of distance provides an intuitive way to tune the rewards, grounding the value of associated weights for each heuristic. For example, the depth penalty's weight w_d specifies how much distance is added per meter of depth. If $w_d = -1$, tasks downstream of an agent (relative to the base) will have no penalty for being further away, as the travel cost and depth penalty cancel out. To make the agent prioritize the deeper tasks, regardless of travel distance, then w_d could be set below -1. If the agent should still minimize distance traveled, a value $-1 < w_d < 0$ will prioritize tasks closer to the agent and downstream from the base station.

5 Operator Interface

Humans still have significant knowledge and insight that cannot be replicated in autonomous systems, and leveraging that knowledge can improve

performance. However, understanding and controlling a multi-robot team is challenging. It is important to have a well-designed interface to allow the operator to have a clear overview about the multi-robot team status and effective control.

To provide a clear overview for the operator, the interface adapts a map-based approach where the robots (models highlighted by octagons with segments denoting status), tasks (pin makers), and priority regions (gray shapes) are displayed in a real-time 3D map constructed using SLAM data and navigation data (Figure 3). As a result, the operator has a good understanding of the current status of the mission at a glance.

One of the key features of the operator interface is the effective control of the task allocation process at both the single-task level and the region (multi-task) level. At the single task level, the operator can create, delete, and manually assign tasks by interacting with the task markers directly. Manually assigned tasks are given higher priority than tasks that agents independently bid on. A priority system avoids the difficult task of having an operator manually tune rewards during a mission. Every bid has an associated priority level. Within the task allocation system, a higher priority is treated as infinitely higher reward, and thus higher priority tasks are always executed first. This simplifies interaction with the task allocation system, providing a direct way to specify to do certain promising tasks first or dangerous tasks last.

For modifying multiple tasks at once, the operator can create different types of priority regions: geometric priority regions and graph-based priority regions. The geometric priority regions prioritize/de-prioritize tasks within the region bounds (e.g., within a rectangular prism), while the graph-based ones not only change the priorities of affected tasks within the regions but also tasks downstream from the regions, where "downstream" is defined as away from the base station in the global graph. By setting up these priority regions, the operator can guide the multi-robot team toward important areas or "push" them away from dangerous or uninteresting areas.

More details of the user interface for the task allocation system can be found in [30].

6 Results

This section discusses the performance of the system across three tests: a simulated experiment, field testing, and the SubT final competition itself. These range from more realistic (the final competition) to more repeatable (simulations).

6.1 Simulated Experiments

After competition, a set of experiments to compare performance were executed using the Gazebo robotics simulator. This is a high fidelity simulation utilizing the full perception and navigation stack. An example of the starting area is shown in Figure 4(a). The experiment mimics the design of the SubT

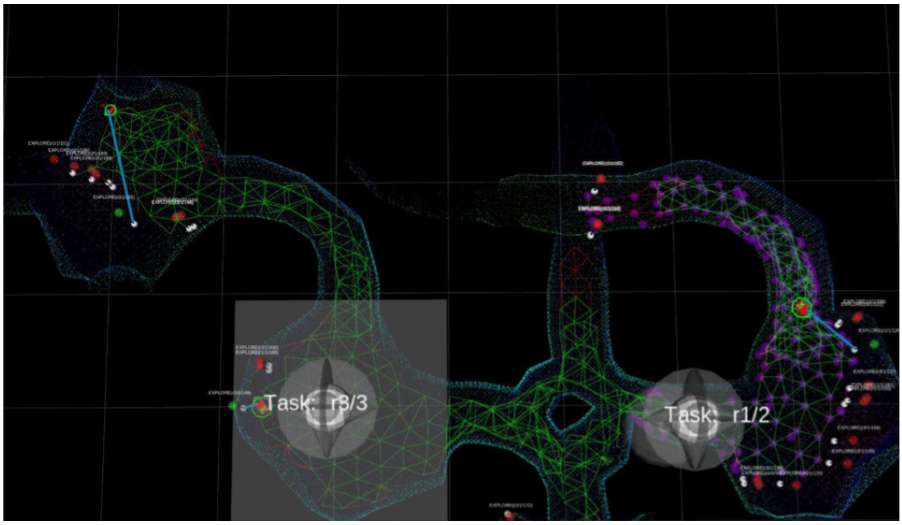

Fig. 3 Part of the map-based operator interface for controlling a multi-robot team. The map is constructed using SLAM and navigation data, including a point cloud map and a traversability map. Tasks are represented as pin markers (red ones are unassigned tasks and green ones are executing tasks), and robots are highlighted by the green octagons. The large gray cuboid region ("r3/3") is a geometric priority region, where tasks within the region have reward or priority altered according to the region parameters. The large spherical region ("r1/2") is a graph-based priority region, where task lying within or beyond the region (i.e., any attached to the vertices colored purple) have reward or priority altered according to the region parameters.

competition, with agents entering through a narrow gate into an underground environment approximating either man-made tunnels, urban spaces, or natural caves. The simulated environments are shown in Figure 4(b)-(d).

Four simulated robots were used in each trial based on the BIA5 ATR platform used by Team CSIRO Data61. At the start of a trial, these agents were given waypoints to move through the gate one by one (shown in Figure 4(a)), and then start automatic exploration. After the first agent passed the threshold to the environment, the experiment time started, and agents had thirty minutes to explore as much of the space as possible. Communication limitations were approximated based on distance, with data transfer rates being reduced after 40m, and cut off after 80m, for both agent-to-agent and agent-to-base communications.

To compare the potential benefit of the task allocation approach, two other strategies were tested. A baseline "no coordination" strategy had agents exploring simultaneously but fully independently (i.e., not sharing data). An "implicit coordination" strategy had agents share their maps and area explored, but not perform any collaborative allocation or use the graph-based heuristics when selecting tasks. The final condition, "explicit coordination," used the full task-allocation approach as described. All relevant parameters in each condition were identical to what was used in the final competition. The major parameters are shown in Table 2.

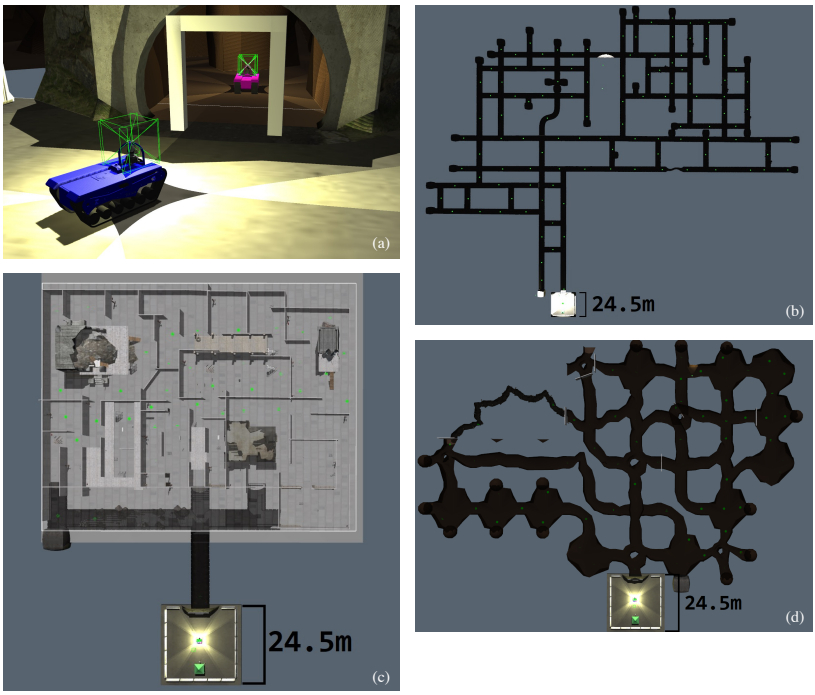

Fig. 4 (a) detailed view of BIA5 ATR platforms entering a simulated course; (b) tunnel simulation world with scale shown for agent's starting area at bottom of figure; (c) and (d) similar for urban and cave worlds respectively.

Time Discount per Minute	λ	0.774
Depth Penalty	w_d	-0.3
Branch Penalty	w_b	0.25
Nearest Neighbor Penalty	w_n	1.2
Nearest Neighbor Max Distance (m)	η	30

Table 2 Key parameters for task allocation and exploration.

The key metric tracked was the percent of the map explored. Exploration markers were spread uniformly throughout the environments. Once an agent has reached within 5m of a marker (a distance where real camera detections of competition artifacts was reliable), it is logged as explored. Trajectories were also tracked for qualitative analysis of the team's exploration. Sixteen trials per condition were run. The total area explored for all nine conditions is shown in Table 3.

A second version was run requiring agents to sync after they had been out of communications with the base station for 15 min, causing agents to drive back to towards the start until they could connect with the base station, upload and download data, and then restart exploration. This provided a better representation of the requirements of SubT events. Results of this experiment are shown in Table 4.

Coordination Method	Tunnel	Urban	Cave
None	56.6%	86.7%	88.3%
Implicit	61.4%	95.0%	96.8%
Explicit	**66.3**%	**97.4**%	**97.1**%

Table 3 Results of simulation testing, shown as the average percentage of markers for which an agent passed within 5m during the 30 min run. For this first set of experiments, agents are not required to return to synchronize data with base.

Coordination Method	Tunnel	Urban	Cave
None	32.2%	85.6%	83.4%
Implicit	37.0%	**95.4**%	95.9%
Explicit	**45.9**%	95.1%	**97.6**%

Table 4 Results of simulation testing, shown as the average percentage of markers for which an agent passed within 5m during the 30 min run. For this second set of experiments, agents must return to synchronize data with the base after being out of communications for 15 min.

The data show explicit coordination performing significantly better than no coordination in all environments. Compared to implicit coordination, the task allocation approached used with explicit coordination significantly increases performance in the tunnel environment, but the difference is small in the cave and urban environments. As shown in Figures 4, the scale of the tunnel environment is the largest, and it best represents the tree structure exploited in the graph-based penalization. Both the cave and urban environments also have a clear initial branch point with three options leading to distinct areas. The implicit coordination was able to divide agents between these branches, as paths already taken had further out frontiers, leading to more penalized explore tasks. This initial division into different areas seemed to be the most important step, after which exploration can run in parallel without coordination for some time until agents overlap.

6.2 Field Testing

A field test was conducted at the CSIRO Queensland Centre for Advanced Technologies (QCAT) site to evaluate the performance of the proposed method when utilized in a purely autonomous manner, i.e., without operator input. The course included indoor and outdoor segments of an industrial area, gravel and concrete roads, and an artificial tunnel environment. Photographs of segments of the course are shown in Figure 5.

The agents utilized in both the field test and the SubT final event are shown in Figure 5. Three agents were utilized in the field test, including two BIA5 OzBot ATRs, and one Boston Dynamics Spot quadruped, and the mission time was limited to 30 mins. Drones were not utilized in the field test. Communications nodes were pre-positioned to enable communication across most of the course (during the SubT competition, these were deployed from the agents, but this process was not automated due to the risk of poor node placement rendering narrow tunnels impassible). The paths followed by the

Fig. 5 Robot fleet used in experiment ()and SubT final event), and course utilized in experiment. (a) shows robot fleet and components (drone was not uilized in experiment). (b)-(g) show examples of parts of the course utilized in the experiment, and placement of artifacts that the agents needed to discover: (b) backpack in indoor section; (c) fire extinguisher in outdoor industrial area; (d) helmet in industrial barrel storage area; (e) survivor in outdoor terrain park; (f) rope in synthetic tunnel environment; (g) rope under mezzanine within synthetic tunnel region.

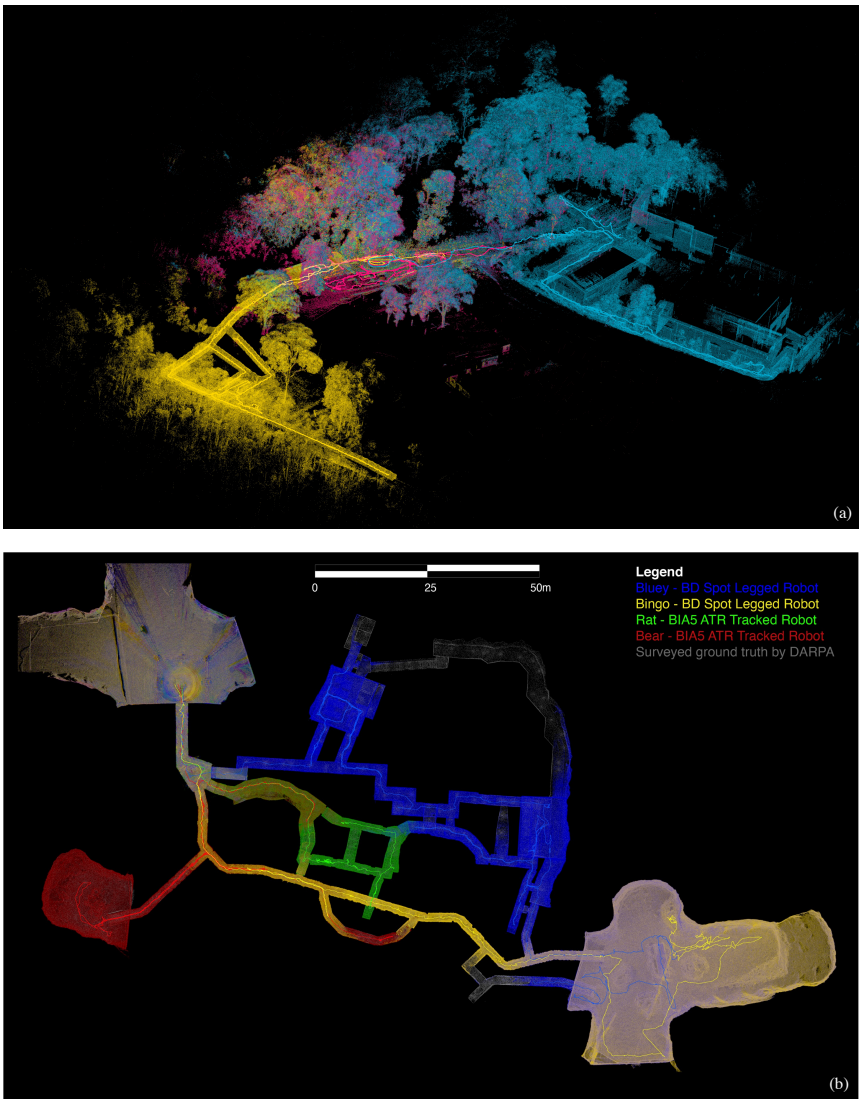

Fig. 6 (a) shows point cloud and agent trajectories for experiment, both colored by agent, with Spot agent colored in yellow, and ATR agents colored in red and green. Spot agent covered urban sections of course (shown in Figure 5(b)-(d)); red ATR covered terrain park region (shown in 5(e)); and green ATR covered tunnel region (shown in 5(f) and (g)). (b) shows similar for the SubT Final Event, where the robots start from the top-left region; upper area (i.e., the upper part covered by Bluey) is the urban environment, middle area (i.e., green and lower part of blue, covered by Rat and Bluey) is tunnel, and lower area (yellow/red, covered by Bear and Bingo) is cave.

three agents and resulting point cloud are shown in Figure 6(a), with each agent's point cloud and trajectory shown in a different color. The total distance traveled by the three agents was 2263m. Collectively, the agents covered essentially the entire course.

6.3 SubT Final Competition

The final event of the DARPA Subterranean Challenge was hosted in the Louisville Mega Cavern. Within it, a course was built to simulate urban underground environments like subways, mining tunnels, and natural caves. This course provided severe navigation and communication challenges for robot teams, which had 60 min to explore and localize artifacts placed throughout the environment. Team CSIRO Data61 fielded four ground vehicles: Two BIA5 OzBot ATRs and two Boston Dynamic Spots. All ground vehicles ran the task-allocation system described in this paper. Two Emesent quadrotor UAVs were mounted and launched from the ATRs, however once launched these explored independently.

The point clouds and trajectories for the ground agents in the final prize run of the event are shown in Figure 6(b). During the final competition a single operator was able to command and control the agents. This, by design, provided too high a workload for one person to directly control agents, so autonomous operation was essential. Figure 7(b) shows a breakdown of the mode of operation of each robot during the 60 minute run. Bingo (Spot) entered the cave segment of the course and explored under a prioritization region, falling in rough terrain in the end cavern after 20 min. Bluey (Spot) entered the urban area and was carefully managed to prevent it from entering the railway tunnel where it had fallen on high tracks in the two preliminary runs. Beyond communications range and under a task prioritization region, it traversed two sets of stairs to explore the subway platform and the tunnel course connected to the railway tunnel. In doing so it connected to Bingo and subsequently relayed parts of Bingo's data back to the base. Bluey was then manually assigned an exploration task following a different route heading towards the back cave. After exploring the back cave, it began to follow a different route back towards the base before falling, still out of communications range. Rat (ATR) explored much of the tunnel section before losing a track soon after becoming tangled in a loose fire hose. Bear (ATR) was initially autonomous, but gained the operator's close attention when it was the one remaining operational robot within communications range. Due to time constraints, the operator manually operated it through a combination of waypoints and teleoperation to bridge the connection from Bluey to the base, overcoming a dynamic obstacle, an alternative route blocked by Rat, and finally launching the drone to reduce the clearance sufficiently to be able to reach a point where communications were bridged.

While the initial allocations were heavily influenced by the operator in line with a priori knowledge gained from preliminary runs, the majority of operation is autonomous, and there are several extended periods where different

robots are beyond communications. An example is shown in Figure 7, where Bingo and Bluey (green and yellow trajectories) are out of communications from the base, but communicating with each other.

7 Conclusion

This paper has described the multi-agent coordination approach used by Team CSIRO Data61 for exploration of underground environments in the DARPA Subterranean Challenge. Framing multi-agent exploration as a multi-robot task allocation problem allowed for non-exploration tasks and operator input to be combined with autonomous exploration into one cohesive framework. To better handle an unknown and changing mission, new methods for updating an agent's set of owned tasks and bids were presented. Techniques to deal with some of the challenges of physical robots, such as inconsistent failures, were also discussed.

The approach described was effective in competition, and Team CSIRO Data61 achieved second place. The simulated experiments show that the approach provides significant benefit for larger scale environments with long tunnels and tree-like structure, but less benefit for smaller environments with denser connections or more open areas. Estimating the value of exploring an area with unknown structure is a challenging problem. Characterizing different types of environments to better predict which frontiers should be explored first is necessary for efficient exploration across diverse domains.

Declarations

Contributions. M.O. was the lead author. M.O., J.W. and R.A. contributed to the system design. M.O., J.W., S.C. and A.P. contributed to system implementation, experiment and analysis, and R.A. and N.K. provided guidance and oversight. The authors also thank Fletcher Talbot, Brendan Tidd, Brett Wood and Tom Hines for assistance with the field experiment.

Funding. This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

Conflicts of interest. The authors declare they have no conflicts of interest.

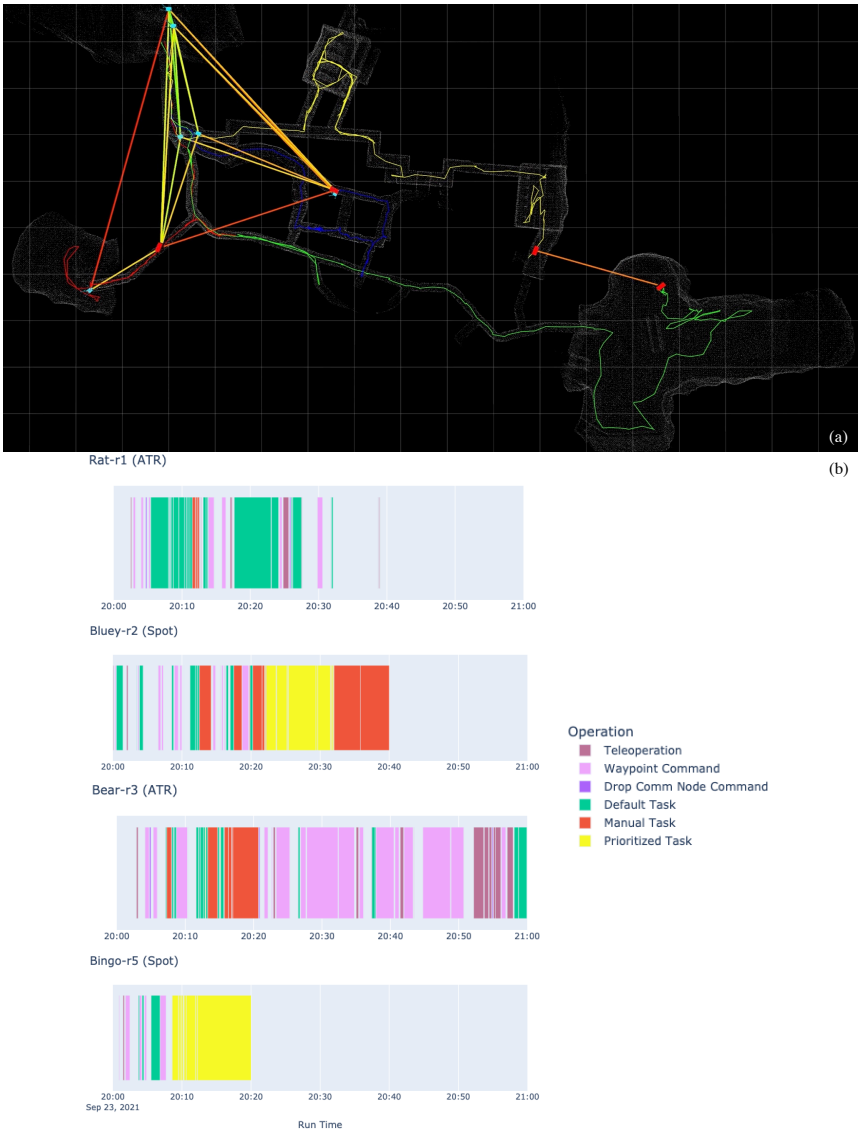

Fig. 7 (a) shows snapshot of communications links around half-way through final prize run. Agents colored by trajectory, as Rat (blue), Bluey (yellow), Bingo (green) and Bear (red). Communications links shown with straight lines, with green indicating good link quality, ranging to yellow, orange and finally red indicating the poorest quality. (b) shows mode of operation of each robot during SubT final prize run. Rat (ATR), Bluey (Spot) and Bingo (Spot) were mostly utilized in autonomous task allocation mode, with prioritization regions being used extensively on Bluey and Bingo, and manual task allocation being used on Bluey. Bear soon became the final robot operational robot in contact, and was utilized with manual control, through waypoints and (in the final time critical period) teleoperation.

References

[1] Chung, T.: DARPA Subterranean (SubT) Challenge (2019). https://www.darpa.mil/program/darpa-subterranean-challenge

[2] Choi, H.-L., Brunet, L., How, J.P.: Consensus-based decentralized auctions for robust task allocation. IEEE transactions on robotics **25**(4), 912–926 (2009)

[3] Brass, P., Cabrera-Mora, F., Gasparri, A., Xiao, J.: Multirobot tree and graph exploration. IEEE Transactions on Robotics **27**(4), 707–717 (2011)

[4] Fraigniaud, P., Gasieniec, L., Kowalski, D.R., Pelc, A.: Collective tree exploration. Networks: An International Journal **48**(3), 166–177 (2006)

[5] Cepeda, J.S., Chaimowicz, L., Soto, R., Gordillo, J.L., Alanís-Reyes, E.A., Carrillo-Arce, L.C.: A behavior-based strategy for single and multi-robot autonomous exploration. Sensors **12**(9), 12772–12797 (2012)

[6] Bourgault, F., Makarenko, A.A., Williams, S.B., Grocholsky, B., Durrant-Whyte, H.F.: Information based adaptive robotic exploration. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, vol. 1, pp. 540–545 (2002). IEEE

[7] Yang, K., Keat Gan, S., Sukkarieh, S.: A gaussian process-based rrt planner for the exploration of an unknown and cluttered environment with a uav. Advanced Robotics **27**(6), 431–443 (2013)

[8] Yamauchi, B.: A frontier-based approach for autonomous exploration. In: CIRA, pp. 146–151 (1997)

[9] Burgard, W., Moors, M., Stachniss, C., Schneider, F.E.: Coordinated multi-robot exploration. IEEE Transactions on robotics **21**(3), 376–386 (2005)

[10] Keidar, M., Kaminka, G.A.: Efficient frontier detection for robot exploration. The International Journal of Robotics Research **33**(2), 215–236 (2014). https://doi.org/10.1177/0278364913494911

[11] Jain, U., Tiwari, R., Godfrey, W.W.: Comparative study of frontier based exploration methods. In: 2017 Conference on Information and Communication Technology (CICT), pp. 1–5 (2017). IEEE

[12] Dang, T., Mascarich, F., Khattak, S., Papachristos, C., Alexis, K.: Graph-based path planning for autonomous robotic exploration in subterranean environments. In: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 3105–3112 (2019). IEEE

[13] Kim, S.-K., Bouman, A., Salhotra, G., Fan, D.D., Otsu, K., Burdick, J., Agha-mohammadi, A.-a.: Plgrim: Hierarchical value learning for large-scale exploration in unknown environments. In: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 31, pp. 652–662 (2021)

[14] Nieto-Granda, C., Rogers III, J.G., Christensen, H.I.: Coordination strategies for multi-robot exploration and mapping. The International Journal of Robotics Research **33**(4), 519–533 (2014)

[15] Arkin, R.C., Diaz, J.: Line-of-sight constrained exploration for reactive multiagent robotic teams. In: 7th International Workshop on Advanced Motion Control. Proceedings (Cat. No. 02TH8623), pp. 455–461 (2002). IEEE

[16] Faigl, J., Kulich, M.: On benchmarking of frontier-based multi-robot exploration strategies. In: 2015 European Conference on Mobile Robots (ECMR), pp. 1–8 (2015). IEEE

[17] Gerkey, B.P., Matarić, M.J.: A formal analysis and taxonomy of task allocation in multi-robot systems. The International journal of robotics research **23**(9), 939–954 (2004)

[18] Khamis, A., Hussein, A., Elmogy, A.: Multi-robot task allocation: A review of the state-of-the-art. Cooperative robots and sensor networks 2015, 31–51 (2015)

[19] Atay, N., Bayazit, B.: Mixed-integer linear programming solution to multi-robot task allocation problem (2006)

[20] Mosteo, A.R., Montano, L.: Simulated annealing for multi-robot hierarchical task allocation with flexible constraints and objective functions. In: Workshop on Network Robot Systems Toward Intelligent Robotic Systems Integrated with Environments (2006). Citeseer

[21] Simmons, R., Apfelbaum, D., Burgard, W., Fox, D., Moors, M., Thrun, S., Younes, H.: Coordination for multi-robot exploration and mapping. In: Aaai/Iaai, pp. 852–858 (2000)

[22] Olson, E., Strom, J., Morton, R., Richardson, A., Ranganathan, P., Goeddel, R., Bulic, M., Crossman, J., Marinier, B.: Progress toward multi-robot reconnaissance and the magic 2010 competition. Journal of Field Robotics **29**(5), 762–792 (2012)

[23] Dai, W., Lu, H., Xiao, J., Zeng, Z., Zheng, Z.: Multi-robot dynamic task allocation for exploration and destruction. Journal of Intelligent & Robotic Systems **98**(2), 455–479 (2020)

[24] Johnson, L., Ponda, S., Choi, H.-L., How, J.: Asynchronous decentralized task allocation for dynamic environments. In: Infotech@ Aerospace 2011, p. 1441 (2011)

[25] Buckman, N., Choi, H.-L., How, J.P.: Partial replanning for decentralized dynamic task allocation. In: AIAA Scitech 2019 Forum, p. 0915 (2019)

[26] Hudson, N., Talbot, F., Cox, M., Williams, J., Hines, T., Pitt, A., Wood, B., Frousheger, D., Lo Surdo, K., Molnar, T., Steindl, R., Wildie, M., Sa, I., Kottege, N., Stepanas, K., Hernandez, E., Catt, G., Docherty, W., Tidd, B., Tam, B., Murrell, S., Bessell, M., Hanson, L., Tychsen-Smith, L., Suzuki, H., Overs, L., Kendoul, F., Wagner, G., Palmer, D., Milani, P., O'Brien, M., Jiang, S., Chen, S., Arkin, R.C.: Heterogeneous ground and air platforms, homogeneous sensing: Team CSIRO Data61's approach to the DARPA Subterranean Challenge. Field Robotics **2**, 595–636 (2022). https://doi.org/10.55417/fr.2022021

[27] Goodman, J., Greenberg, A.G., Madras, N., March, P.: Stability of binary exponential backoff. Journal of the ACM (JACM) **35**(3), 579–602 (1988)

[28] Williams, J., Jiang, S., O'Brien, M., Wagner, G., Hernandez, E., Cox, M., Pitt, A., Arkin, R., Hudson, N.: Online 3D frontier-based ugv and uav exploration using direct point cloud visibility. In: 2020 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI), pp. 263–270 (2020). IEEE

[29] Saha, P.K., Borgefors, G., di Baja, G.S.: A survey on skeletonization algorithms and their applications. Pattern recognition letters **76**, 3–12 (2016)

[30] Chen, S., O'Brien, M.J., Talbot, F., Williams, J.L., Tidd, B., Pitt, A., Arkin, R.C.: Multimodal user interface for multi-robot control in underground environments. In: IEEE International Conference on Intelligent Robots And Systems (2022)