Scalable and Robust Algorithms for Task-Based Coordination From High-Level Specifications (ScRATCHeS)

Kevin Leahy, *Member, IEEE*, Zachary Serlin, Cristian-Ioan Vasile, *Member, IEEE*, Andrew Schoer, Austin M. Jones, Roberto Tron, *Member, IEEE*, and Calin Belta, *Fellow, IEEE*

***Abstract*—Many existing approaches for coordinating heterogeneous teams of robots either consider small numbers of agents, are application-specific, or do not adequately address common real-world requirements, e.g., strict deadlines or intertask dependencies. We introduce scalable and robust algorithms for task-based coordination from high-level specifications (ScRATCHeS) to coordinate such teams. We define a specification language, capability temporal logic, to describe rich, temporal properties involving tasks requiring the participation of multiple agents with multiple capabilities, e.g., sensors or end effectors. Arbitrary missions and team dynamics are jointly encoded as constraints in a mixed integer linear program, and solved efficiently using commercial off-the-shelf solvers. ScRATCHeS optionally allows optimization for maximal robustness to agent attrition at the penalty of increased computation time. We include an online replanning algorithm that adjusts the plan after an agent has dropped out. The flexible specification language, fast solution time, and optional robustness of ScRATCHeS provide a first step toward a multipurpose on-the-fly planning tool for tasking large teams of agents with multiple capabilities enacting missions with multiple tasks. We present randomized computational experiments to characterize scalability and hardware demonstrations to illustrate the applicability of our methods.**

***Index Terms*—Formal methods, multiagent systems, planning, robotics.**

I. Introduction

ONE of the main challenges of multiagent systems is the deployment of teams of heterogeneous agents that must work together to complete a task that can be performed by neither a single agent nor a team of homogeneous agents. An example is coordination between airborne robots with downward-facing visual and infrared cameras and ground robots with manipulators capable of moving rubble to find survivors in an urban environment after a natural disaster. The problems of planning and coordination for these kinds of teams can be very complex, as heterogeneity prohibits the arbitrary exchange of one agent for another. This makes it challenging to develop scalable algorithms for these teams, as more distinct possibilities must be searched when generating a team plan. If agents are not interchangeable, then it is difficult to develop algorithms for robust "self-reorganizing" teams that can account for agent failure.

Most work in general planning and coordination algorithms for multiagent systems has assumed homogeneity of agent capabilities in order to avoid these complications [1], [2]. Typically, planning and coordination algorithms for teams with heterogeneous agents are *ad hoc* solutions that are heavily dependent on domain expertise, and are specialized to a single family of capabilities/platforms and a single, unique mission [3], [4], or do not consider temporal deployment requirements [5], [6].

In this work, we propose a framework called scalable and robust algorithms for task-based coordination from high-level specifications (ScRATCHeS) in which a human supervisor can task a team of heterogeneous agents on the fly by specifying a high-level mission, illustrated in Fig. 1. Specifically, we consider the problem of coordinating a large team of heterogeneous agents from a global high-level specification. The agents' capabilities, e.g., sensors or end effectors, are known *a priori*, and the agents work in a known shared environment that is partitioned into regions. Each region is labeled with the tasks that may be completed in that region. A task description consists of the labels of the regions where the task must be performed, the required number of agents with each type of capability that are required to perform the task, and the amount of time required for the agents to complete the task. From these tasks, an operator uses our specification language, called capability temporal logic (CaTL) to generate a specification that gives absolute or relative timing of task completion, repetition frequencies, and task interdependencies such as sequencing or synchronization. Our algorithm then encodes the dynamics of the agents moving throughout

Manuscript received July 23, 2021; accepted November 2, 2021. This work was supported by the Under Secretary of Defense for Research and Engineering under Air Force Contract FA8702-15-D-0001 and NSF under grant IIS-2024606. This paper was recommended for publication by Associate Editor Jason O'Kane and Editor Paolo Robuffo Giordano upon evaluation of the reviewers' comments. *(Corresponding author: Kevin Leahy.)*

Kevin Leahy and Austin M. Jones are with the MIT Lincoln Laboratory, Lexington, MA 02421 USA (e-mail: kevin.leahy@ll.mit.edu; austin.jones@ll.mit.edu).

Zachary Serlin and Andrew Schoer are with the MIT Lincoln Laboratory, Lexington, MA 02421 USA, and also with the Boston University, Boston, MA 02215 USA (e-mail: zserlin@bu.edu; aschoer@bu.edu).

Cristian-Ioan Vasile is with the Lehigh University, Bethlehem, PA 18015 USA (e-mail: cvasile@lehigh.edu).

Roberto Tron and Calin Belta are with the Boston University, Boston, MA 02215 USA (e-mail: tron@bu.edu; cbelta@bu.edu).

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TRO.2021.3130794.

Digital Object Identifier 10.1109/TRO.2021.3130794

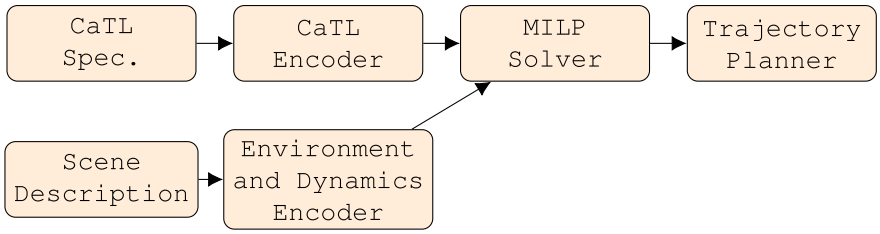


Fig. 1. Schematic overview of ScRATCHeS.

the environment, the specification, and the selected measure of robustness into a mixed integer linear program (MILP). The resulting plan from the MILP is handed to a motion planner to generate a collision-free motion plan for the team. Our framework can be applied to arbitrary distributions of capabilities among agents and to arbitrary missions expressed in CaTL. ScRATCHeS removes the need to design custom high-level planners for every new possible combination of robot platforms that can work in a team and allows a supervisor flexibility to accomplish new missions with the team of robots they have available.

We use tools from formal methods to develop planning and coordination algorithms that are scalable with the number of agents and robust to agent attrition or performance. We achieve scalability by defining and solving an MILP that is equivalent to the defined planning and coordination problem. Advances in solution techniques and heuristics make it possible to solve MILPs with hundreds of thousands of variables and constraints with limited computation resources [7]–[9]. We achieve robustness by defining and directly optimizing a measure of robustness to attrition that can be computed by mixed integer linear constraints derived from the temporal logic specifications. This approach is complete (at the level of a discrete-time abstraction, up to the precision of the numerical solver), meaning that if a solution exists, it will be found. Furthermore, modern MILP solvers can determine infeasibility very quickly [10]. In our experiments infeasibility was typically detected in under one second.

The main contribution of this work is an end-to-end framework for deploying a team of robots that 1) includes heterogeneous agents; 2) has strict timing requirements (i.e., concrete time); 3) is agent agnostic in that it specifies the capability that is required for completing tasks, but not which specific robot should accomplish the task; 4) is robust to agent attrition. The combined ability to reason about strict timing requirements, the number of agents, and the robustness of our solution come from the use of CaTL, a fragment of signal temporal logic (STL). Additionally, we provide extensions to our algorithm that address practical implementation concerns, namely: 1) regularizing travel time to avoid spurious motion; 2) an upper bound on our objective function to improve the speed of the optimization; 3) online replanning in the event of agent attrition. A preliminary version of this work appeared at the 2019 International Symposium of Robotics Research ([11]). The preliminary work introduced the CaTL specification language, an initial framework for deploying a team of robots from symbolic level specifications to low-level motion plans, and a robustness measure that can be used to optimize a plan's tolerance to agent attrition. This article expands upon that work by providing the following:

1) an on-line replanning algorithm in response to agent dropout;
2) an upper bound on robustness measure to speed up the MILP optimization;
3) a method for minimizing overall agent travel time;
4) updated randomized computational experiments and hardware demonstration;
5) expanded proofs;
6) additional information on the motion planning algorithms used.

The rest of the article is organized as follows. We give a short review of related literature in Section II. In Section III, we introduce a running example that illustrates the kinds of problems addressed in this article, and we introduce the models and definitions that are used throughout the article. We introduce our specification language in Section IV, and formulate our problem in Section V. Next, we describe our planning solution framework in Section VI. Finally, we present computational experiments and a hardware demonstration in Sections VIII and IX, respectively. Proofs of propositions and descriptions of motion planning algorithms are found in the appendix.

II. Literature Review

In our approach, we formulate tasks as high-level specifications using temporal logic (TL) formulas. Temporal logics, such as linear temporal logic (LTL), have seen success as specification languages for single agents systems [12], [13] and, increasingly, for multiagent systems [14]–[17], including for heterogeneous teams of agents [18].

Much of the work on planning from high-level specifications uses automata to model these specifications. While automata provide a useful framework for reasoning about specifications, they can lead to issues of computational complexity and scalability. In [19], [20], the authors use automata-based methods to determine independent subspecifications that can be assigned to individual agents, whose interleaved behavior is guaranteed to satisfy the global specification. Similarly, [18] uses an automaton representation of a specification to decompose tasks into "essential sequences" to reason about, allocate, and plan for tasks. Because we consider specifications with both counting and strict timing requirements, in our case the scalability of automata-based approaches limits their utility. We avoid the

computational complexity of automata-based methods by describing the joint state of the team as a vector of counts of agents with each capability in each region at a given time rather than as a product of automata. Sampling-based techniques show promise over the complexity of automata-based methods. In [21], [22], the authors use sampling to construct abstractions on-the-fly for large scale multiagent planning. While these methods incorporate heterogeneous teams, they are based upon LTL. LTL reasons over untimed sequences, so abstract timing requirements can be expressed (e.g., "perform task A before task B"), while concrete timing requirements cannot be expressed (e.g., "perform task A within 5 minutes of task B"). Further, implementing agent counting in LTL is also nontrivial, and cannot be easily included in the cost function of the sampling method used by the authors. In this work, we reason explicitly over counts of agents and concrete timing.

We use an MILP to encode the mission constraints and planning problem in order to mitigate the computational complexity associated with other multiagent planning approaches. This allows us to include counting and strict timing requirements in our framework. Such approaches have been used to solve similar planning problems. For example, [23] uses an MILP to solve the vehicle routing problem with abstract timing requirements. Likewise, the authors of [24] demonstrate an efficient MILP encoding for large teams of homogeneous agents. In contrast to these works, we use concrete timing requirements and consider heterogeneous teams.

The most closely related works to ours are [25], [26]. The authors propose a method for encoding a planning problem for large heterogeneous teams, including counting requirements, as an MILP. They adapt a logic called censusSTL [27], which was designed for inference over teams of agents, into a new logic called counting linear temporal logic (cLTL+). cLTL+ can specify the number of required capabilities for a team of agents to accomplish its task. Their solution is optimized for tolerance to asynchrony among the agents, so that it is robust to the late or early arrival of agents, as well as the number of agents that drop out. In this work, we define capability temporal logic (CaTL), a fragment of signal temporal logic (STL) [28]. Like cLTL+, CaTL can specify the number of required capabilities for a task, and can be optimized for robustness to attrition. Unlike cLTL+, we present a method that allows us to present concrete timing requirements on when agents should accomplish a task (e.g., "perform this task within 5 minutes").

III. Models

In this section, we define models for the kinds of teams we want to coordinate. We are motivated by the following hypothetical example from precision agriculture.

Example 1: Consider a large farm that grows diverse crops in spatially separated locations as illustrated in Fig. 2. Each colored region corresponds to a different type of crop, with the exception of the red regions that correspond to areas which the robots cannot traverse, e.g., areas where heavy equipment are in operation or where the terrain is too rough for the robots to traverse. To aid in monitoring these crops, a fleet of ground based robots with different sensing modalities has been deployed to keep track of plant health. The fleet as a whole has ultraviolet (UV), soil moisture (Mo), infrared (*IR*) and visual (Vis) sensors. Every robot in the fleet has at most two sensors and the assignment of sensors to robot is fixed *a priori*. Each of the crops in the field has distinct monitoring requirements. The tasks that the team of robots must perform during a 24 h deployment are listed in Table I.

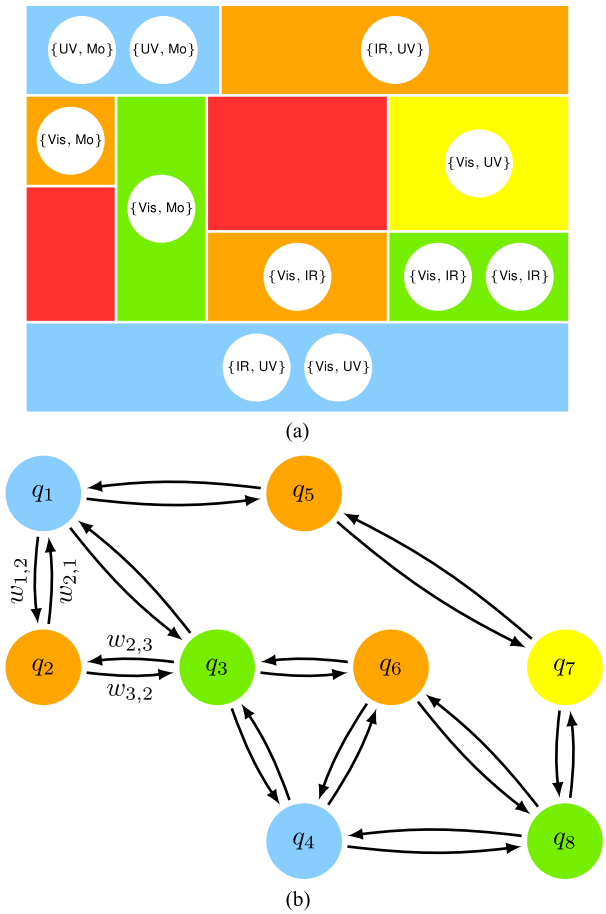


Fig. 2. (a) Schematic of the precision agriculture motion coordination problem. Colors of regions correspond to obstacles (red) or crop types (other colors). Discs are robots with their associated capabilities listed. The set of capabilities includes ultraviolet sensing (UV), moisture sensing (Mo), infrared sensing (*IR*), and vision (Vis). The team of robots is used for crop monitoring tasks, such as "Within 3 hours of deployment, two visual and two *IR* sensors must remain within each green crop region at the same time for at least 1 h." (b) Associated environment.

A. Environment

Definition 1: The *Environment* is given by a tuple $Env = (Q, E, W, AP, L)$ where:

1) Q is a finite set of states that correspond to regions of a workspace;
2) $E \subseteq Q \times Q$ is a set of edges such that $(q, q') \in E$ iff an agent in the environment can traverse from the region associated with q to the region associated with q' without passing through any other region;

TABLE I
LIST OF PRECISION AGRICULTURE TASKS

1) All robots must avoid the red regions at all times.
2) Within 10 hours of deployment, two visual and two IR sensors must reach each green crop region at the same time and remain there for at least half an hour.
3) Half hour soil moisture readings must be made in each of the blue crop regions every 10 hours.
4) Within 4 and 12 hours after deployment, two UV and two visual sensors must be in the yellow crop region and remain there for an hour.
5) Within 1 and 9 hours of deployment, and within 10 and 15 hours of deployment, two visual sensors must be in each of the orange regions of the environment and remain there for 1 hour.

3) $W : E \to \mathbb{R}$ is an edge weight such that $W(q, q')$ is the maximum amount of time required for an agent to traverse q before entering q';
4) AP is a set of atomic propositions that define what types of tasks may be performed in the environment;
5) $L : Q \to 2^{AP}$ is a mapping that labels each state in the environment according to which tasks may be performed in that region.

When constructing an environment from a partitioned workspace, forbidden regions are omitted from Q and transitions to/from those regions are omitted from E. We note that the choice of edge weight W as the *worst-case* travel time for an agent is conservative. This conservative formulation ensures that agents can plan to avoid collisions and react to noise and other disturbances without causing the plan to fail. For simplicity, we assume such times are given *a priori*, but less conservative bounds could be achieved through sampling trajectories from a motion planner that takes the agents' dynamics into account. We assume that agents are small relative to the regions in their environment, to avoid the need for limiting the number of agents in a region or edge at the same time, which might cause agents to interfere with each others' travel times. Additional constraints on the number of agents in a region or edge at any given time can easily be added to our MILP encodings to avoid such issues in dense environments.

Example 1 (Continued): The set of crops shown in Fig. 2(a) leads to the environment model shown in Fig. 2(b). This environment has eight regions $\{q_1, \ldots q_8\}$. An edge exists between regions if the two regions share a facet, i.e., if they are connected geographically. The weight between regions corresponds to the transit time required to travel from the point farthest away from the shared facet to the shared facet. The set of propositions $\{\pi_{\text{green}}, \pi_{\text{blue}}, \pi_{\text{orange}} \ldots\}$ in the model corresponds to the types of regions (colored crops/obstacles) and the labeling function applies the labels to the appropriate regions.

B. Agents

Let Cap be a finite set of capabilities that an agent can have and let J be a finite index set representing all agents.

Definition 2: An *Agent* $j \in J$ is given by a tuple $A_j = (q_{0,j}, Cap_j)$ where $q_{0,j} \in Q$ is the initial location of the agent in the shared environment and $Cap_j \subseteq Cap$ is a finite set of capabilities.

Example 1 (Continued): The set of capabilities is given by $Cap = \{Vis, UV, IR, Mo\}$. Agent A_1 located in the upper left hand corner of Fig. 2 is described by $A_1 = (q_1, \{UV, Mo\})$

Definition 3: An *input signal* for an agent j is a mapping $u_j : \mathbb{R} \to E \cup \{\emptyset\}$ where $u_j(t) = e$ indicates that agent j starts traversing edge e at time t. Each input signal has the properties $u_j(t) = e$ where $e = (q, q') \Rightarrow A_j$ is in state q at time t and $u_j(t) = e \Rightarrow u_j(\tau) = \emptyset,\ \forall \tau \in (t, t + W(e))$. The input signal u_j induces a piece-wise constant *trajectory* of agent A_j, denoted $s_j : \mathbb{R} \to Q \cup E$, such that $s_j(0) = q_{0,j}$ and $u_j(t) = (q, q') \Rightarrow s_j(\tau) = (q, q'),\ \forall \tau \in (t, t + W(e)) \wedge s_j(t + W(e)) = q'$.

Definition 4: Given a team of agents $\{A_j\}_{j \in J}$, let $G \subseteq 2^{Cap}$ be the set of unique combinations of capabilities present in the collection $\{Cap_j\}_{j \in J}$. The *team trajectory* is a mapping from each time t to the team state $s_J(t) = [n_{Q,G}(t), n_{E,G}(t)] \in \mathbb{Z}_{\geq 0}^{|G| \times (|Q| + |E|)}$. The matrix $n_{Q,G} = [n_{q,g}(t)]_{q \in Q, g \in G} \in \mathbb{Z}_{\geq 0}^{G \times |Q|}$ is defined such that

$$n_{q,g}(t) = \sum_{j \in J} I(s_j(t) = q) I(Cap_j = g) \tag{1}$$

where I is the indicator function. That is, $n_{q,g}$ is the number of agents with capability set g in state q. The matrix $n_{E,G}(t) = [n_{e,g}]_{e \in E, g \in G} \in \mathbb{Z}_{\geq 0}^{G \times |E|}$ is defined such that

$$n_{e,g}(t) = \sum_{j \in J} I(s_j(t) = e) I(Cap_j = g) \tag{2}$$

i.e., $n_{e,g}$ is the number of agents with capability set g that are traversing edge e.

That is, the team trajectory corresponds to how many agents with each type of capability are present in each region and traversing along each edge at each time.

IV. CAPABILITY TEMPORAL LOGIC

Here, we define the syntax and semantics of CaTL, a specification language for teams of heterogeneous agents. The atomic unit of a CaTL formula is a *task*. Note that CaTL differs from full STL [28] in that the core unit is a task rather than an arbitrary predicate.

Definition 5: A *counting proposition* $cp_i = (c_i, m_i) \in Cap \times \mathbb{N}$ is true if at least m_i agents with capability c_i are present and false otherwise [26].

Definition 6: A *task* is a tuple $T = (d, \pi, \{cp_i\}_{i \in \mathcal{I}_T})$ where $d \in \mathbb{R}$ is a duration of time, $\pi \in AP$ is an atomic proposition, each $cp_i \in Cap \times \mathbb{N}$ is a counting proposition corresponding to how many agents with each capability should be in each region labeled π, and $\mathcal{I}_T$ is the index set of counting propositions associated with task T.

In plain English, a task $T = (d, \pi, \{(c_i, m_i)\}_{i \in \mathcal{I}_T})$ is satisfied if for d time units, each of the regions labeled as π contains at least m_i agents with capability c_i for all $\{c_i\}_{i \in \mathcal{I}_T}$.

Definition 7: The *syntax* of CaTL is given in the Backus–Naur form [29] as

$$\phi := T|\phi_1 \wedge \phi_2|\phi_1 \vee \phi_2|\phi_1 \mathcal{U}_{[a,b)}\phi|\Diamond_{[a,b)}\phi|\Box_{[a,b)}\phi \tag{3}$$

where ϕ is a CaTL formula, T is a task, $\wedge$ is conjunction, $\vee$ is disjunction, $\mathcal{U}_{[a,b)}$ is time-bounded until, $\Diamond_{[a,b)}$ is time-bounded eventually, $\Box_{[a,b)}$ is time-bounded always, and $a, b \in \mathbb{R}$.

Definition 8: The *qualitative semantics* of CaTL are defined over pairs (s_J, t) where t is a time index. The semantics are defined recursively as

$$\begin{array}{lll}
(s_J,t) \models T & \Leftrightarrow & \forall \tau \in [t, t+d), \forall q \in L^{-1}(\pi) \\
 & & \forall \{cp_i = (c_i, m_i)\}_i \in \mathcal{I}_T \\
 & & \sum\limits_{g: c_i \in g} n_{q,g}(\tau) \geq m_i \\
(s_J,t) \models \phi_1 \wedge \phi_2 & \Leftrightarrow & (s_J,t) \models \phi_1 \text{ and } (s_J,t) \models \phi_2 \\
(s_J,t) \models \phi_1 \vee \phi_2 & \Leftrightarrow & (s_J,t) \models \phi_1 \text{ or } (s_J,t) \models \phi_2 \\
(s_J,t) \models \phi_1 \mathcal{U}_{[a,b)}\phi_2 & \Leftrightarrow & \exists t' \in [t+a, t+b)(s,t') \models \phi_2 \\
 & & \text{and } \forall t'' \in [t,t')s(t'') \models \phi_1 \\
(s_J,t) \models \Diamond_{[a,b)}\phi & \Leftrightarrow & \exists t' \in [t+a, t+b)(s,t') \models \phi \\
(s_J,t) \models \Box_{[a,b)}\phi & \Leftrightarrow & \forall t' \in [t+a, t+b)(s,t') \models \phi.
\end{array} \tag{4}$$

A team trajectory satisfies a CaTL formula ϕ, denoted $s_J \models \phi$, if $(s_J, 0) \models \phi$.

Example 1 (Continued): Each of the tasks from Table I may be expressed in CaTL as follows:

1) $\psi_2 = \Diamond_{[0,10)}(0.5, \pi_{\text{green}}, \{(IR, 2), (Vis, 2)\})$;
2) $\psi_3 = \Box_{[10,20)}\Diamond_{[0,5)}(0.5, \pi_{\text{blue}}, \{(Mo, 1)\})$;
3) $\psi_4 = \Diamond_{[4,12)}(1, \pi_{\text{yellow}}, \{(UV, 2), (Vis, 2)\})$;
4) $\psi_5 = \Diamond_{[1,9)}(1, \pi_{\text{orange}}, \{(Vis, 2)\}) \wedge$ $\Diamond_{[10,15)}(2, \pi_{\text{orange}}, \{(Vis, 2)\})$.

Note that Task 1 is accomplished by omitting the red regions and transitions from our construction of *Env*.

Our definition of CaTL is designed to take advantage of the quantitative semantics (i.e., robustness degree) of STL [30]. Because our predicates encode the counting propositions for each task, our robustness naturally expresses how well we are meeting the requirements of each task in our formula. Since CaTL is a proper fragment of STL by Proposition 1, we can use the existing tools from STL to track the availability of agents.

Proposition 1: CaTL is a proper fragment of STL.

Proof: See Appendix A-A.

To formalize the notion of robustness for CaTL, we define availability robustness, which measures the minimum number of agents J_R that can be removed from (added to) a given team in order to invalidate (satisfy) the given measure. Optimizing this quantity results in a plan that is robust to agent attrition. Formally, we have the following.

Definition 9: The *availability robustness* ρ of a trajectory is defined

$$\rho(s_J, t, \phi) = \begin{cases} \min |J_R| \text{ s.t. } (s_{J \setminus J_R}, t) \not\models \phi & \rho(s_J, t, \phi) \geq 0 \\ -\min |J_R| \text{ s.t. } s_{J \cup J_R}, t) \models \phi & \rho(s_J, t, \phi) < 0 \end{cases}. \tag{5}$$

The availability robustness for a given team trajectory s_J and formula ϕ can be computed recursively using a set of *quantitative semantics*.

Definition 10 (Quantitative semantics (availability robustness)): The availability robustness of a formula is computed recursively according to the following rules:

$$\begin{array}{lll}
\rho(s_J,t,T) & = & \min\limits_{i \in \mathcal{I}_T} \min\limits_{t' \in [t, t+d)} \min\limits_{q \in L^{-1}(\pi)} \\
 & & \sum\limits_{g: c_i \in g} n_{q,g}(t') - m_i \\
\rho(s_J,t,\phi_1 \wedge \phi_2) & = & \min(\rho(s_J,t,\phi_1), \rho(s_J,t,\phi_2)) \\
\rho(s_J,t,\phi_1 \vee \phi_2) & = & \max(\rho(s_J,t,\phi_1), \rho(s_J,t,\phi_2)) \\
\rho(s_J,t,\phi_1 \mathcal{U}_{[a,b)}\phi_2) & = & \max\limits_{t' \in [t+a, t+b)} \min\{\rho(s_J,t,'\phi_2) \\
 & & \min\limits_{t'' \in [t,t')} \rho(s_J,t,''\phi_1)\} \\
\rho(s_J,t,\Diamond_{[a,b)}\phi) & = & \max\limits_{t' \in [t+a, t+b)} \rho(s_J,t,'\phi) \\
\rho(s_J,t,\Box_{[a,b)}\phi) & = & \min\limits_{t' \in [t+a, t+b)} \rho(s_J,t,'\phi).
\end{array} \tag{6}$$

Proposition 2: Applying the recursive quantitative semantics listed in Definition 10 yields ρ as defined in Definition 9. *Proof* See Appendix A-B.

We note that in general for STL the recursive quantitative semantics are an estimate of the actual robustness (defined as the signed distance between a signal and the language of a formula; [31, Th. 13]). For CaTL, Proposition 2 states that these two Definitions 9 and 10 are equivalent. In other words, the robustness as computed via Definition 10 is an exact computation of availability robustness, which is not true for STL in general.

V. Problem Formulation and Approach

Here, we formalize the problem considered in this article. A list of the symbols used in the problem formulation is provided in Table II.

Problem 1 (Maximize availability): Given a team of agents $\{A_j\}_{j \in J}$ operating in a shared environment $Env = (Q, E, W, AP, L)$ and a CaTL specification ϕ, find a set of input signals $\{u_j\}_{j \in J}$ such that $\rho(s_J, 0, \phi)$ is maximized.

Problem 1 corresponds to finding a plan for the team of agents that tolerates the largest number of agents that drop out. This problem formulation is useful in situations where agent attrition is likely, or in which an agent's ability to complete its part of a task is uncertain.

We solve Problem 1 by formulating and solving an equivalent MILP. Tools from the multivehicle routing problem can be used to translate the environment model to a discrete-time linear system. The qualitative semantics of STL (and thus, CaTL) can be encoded as mixed integer linear constraints on trajectories of this linear system. In the next section we provide mixed integer linear formulations for the availability robustnesses as well as for the environment models. Formulating Problem 1 as an MILP allows us to use commercial off-the-shelf optimization software with optimized heuristics and solvers to find solutions faster than we could by using standard automata-theoretic graph search techniques.

In addition to preplanning to be robust to attrition, we would like to be able to replan on-line when agents drop out. That is, for cases in which the robustness of a plan with the surviving agents is negative, reallocation of the remaining agents may

TABLE II
TABLE OF SYMBOLS. THE TWO SECTIONS CORRESPOND TO PROBLEM VARIABLES AND MILP VARIABLES, RESPECTIVELY. SYMBOLS ARE LISTED IN ALPHABETICAL ORDER IN EACH SECTION

Problem Variables	
AP	set of atomic propositions
Cap_j	capability set of agent j
c_i	capability in cp_i
cp_i	i^{th} counting proposition
d	task duration
$e \in E$	edge in the environment
$g \in G$	capability class of agents
$\mathcal{I}_T$	index set of counting propositions for task T
J	index set of all agents
J_R	set of agents that can be removed to invalidate a plan
K	index set for discrete time horizon of CaTL formula
$k \in K$	discrete time index
m_i	required number of capability c_i in cp_i
$n_{e,g}(t)$	number of agents of class g in edge e at time t
$n_{q,g}(t)$	number of agents of class g in state q at time t
P	set of agents that drop out at time index k_{att}
$q \in Q$	state in the environment
s_J	team trajectory
s_j	trajectory of agent j
T	task
$u_{e,g}(k)$	number of agents entering edge e with capability set g at time index k
u_j	input signal to agent j
$W(q,q')$	environment edge weighting
$\mathcal{W}(q,q')$	discretized environment edge weighting
δt	environment discretization size
π	atomic proposition
ρ	availability robustness
ϕ, ψ	CaTL formula
MILP Variables	
$r_{k,\phi}$	availability robustness encoding of subformula ϕ at time k
$z_{q,cp_i}(k)$	encoding of satisfaction of counting proposition i in state q at time index k
$z_{q,g}(k)$	number of agents in state q with capability set g at time index k
$z_T(k)$	encoding of task T satisfaction at time index k
$z_{\pi,cp_i}(k)$	encoding of satisfaction of counting proposition i in all states labeled π at time index k
$z_{\pi,\mathcal{I}_T}(k)$	encoding of satisfaction of all counting propositions in $\mathcal{I}_T$ in all regions labeled π at time k index
$z_\phi(k)$	encoding of satisfaction of formula ϕ at time index k
α	weighting factor for τ
γ	relative weighting of τ
τ	total travel time of all agents

be able to generate a plan in which the robustness is nonnegative, i.e., the mission will be satisfied. To this end, we define Problem 2.

Problem 2 (On-line replanning): Given $\{A_j\}_{j\in J}$, $Env = (Q, E, W, AP, L)$, ϕ, and a set of input signals $\{u_j\}_{j\in J}$. Let $\{A_j\}_{j\in J}$ execute the plan until time $t_{k_{\text{att}}}$ at which point some set of agents $\{A_p\}_{p\in P}$ such that $|A_p| \geq \rho(s_J, 0, \phi)$ drop out. Find a new set of signals $\{u_j\}_{j\in J\setminus P}$ such that beginning at time $t_{k_{att}}$ the concatenated signal

$$s'_J = \begin{cases} s_J(t) & t < t_{k_{\text{att}}} \\ s_{J\setminus P}(t) & t \geq t_{k_{\text{att}}} \end{cases} \tag{7}$$

satisfies ϕ.

VI. INTEGER LINEAR PROGRAMMING ENCODING

In this section, we formulate Problem 1 as an MILP. For this purpose, we make the following assumption about the environment.

Assumption 1: The edge weight functions (transition times) are defined such that $W(q,q') = k\delta t$, $k \in \mathbb{N}$ where δt is a time step no larger than the minimum value of W. Further, $u_j(t) = \emptyset \forall t \notin \{k\delta t\}_{k\in\mathbb{N}}$, i.e., transitions can only happen at a set of discrete times. That is, the transition times can all be specified by integers.

To enable MILP encodings under Assumption 1, we define a mapping $\mathcal{W} : Q \times Q \to \mathbb{N}$ such that $W(q,q') = \mathcal{W}(q,q')\delta t$ for $q \neq q'$. To enable agents waiting at a state q, we define the weights for "self-loops" $\mathcal{W}(q,q) = 1$.

We denote the planning horizon for the team as K, representing the total number of time steps δt that we plan for our agents. We note that the planning horizon K must be longer than the horizon of our formula as defined in [28] in order to determine satisfiability of the formula.

A. Team Dynamics

Let $z_{q,g}(k) := n_{q,g}(k\delta t)$ be the number of agents with capability set g in the region associated with state q at time index k. Define $u_{e,g}(k)$ as the number of agents with capability set g entering e at time $k\delta t$. The initial positions of the agents are encoded in the equality constraints

$$z_{q,g}(0) = \sum_{j\in J} I(q_{0,j} = q) I(Cap_j = g) \forall q \in Q, g \in G. \tag{8}$$

We use node and edge balance equations

$$z_{q,g}(k) = \sum_{(q,'q)\in E} u_{(q,'q),g}(k - \mathcal{W}(q,'q)) \tag{9a}$$

$$\sum_{(q,q')\in E} u_{(q,q'),g,}(k) = \sum_{(q,'q)} u_{(q,'q),g}(k - \mathcal{W}(q,'q))$$

$$\forall q \in Q, g \in G, k = 0, \ldots, K \tag{9b}$$

where $u_{e,g}(k) = 0$ $\forall e \in E$, $q \in Q$, $k < 0$. These equations together form a linear system with $O((|Q| + |E|)|G|\Lambda)$ dimensions where $\Lambda := \max_{e\in E} \mathcal{W}(e)$. The inputs to the system $(u_{e,g}(k))$ as well as the states are all integer.

Proposition 3: Under Assumption 1, a team input signal $u = [u_j]_{j\in J}$ and the induced team trajectory s_J conform to Definitions 3–4 if and only if a set of variables

$$\{z_{q,g}(k)\}_{q\in Q, g\in G, k=0,\ldots K} \cup \{u_{e,g}(k)\}_{e\in E, g\in G, k=0,\ldots K}$$

satisfy constraints (8), (9).

Proof: See Appendix A-C.

B. Task Satisfaction

Here, we give the encodings for the satisfaction of tasks as functions of the variables $\{z_{q,g,k}\}_{q\in Q, g\in G, k=0,\ldots,K}$ defined in Section VI-A above. Satisfaction of a task depends on counting propositions being satisfied in the appropriate regions for the appropriate amount of time. Therefore, we need variables that

capture the following: 1) the counting propositions cp_i being satisfied in a given region q; 2) satisfaction of those counting propositions in all regions q with the same label π; 3) satisfaction for all counting propositions in $\mathcal{I}_T$; 4) satisfaction for an appropriate duration d. Thus, for a given region of a given task $T = (d, \pi, \{cp_i\}_{i\in\mathcal{I}_T})$, we define a variable $z_{q,cp_i}(k) \in \{0,1\}$ that we wish to be valued 1 if at least m_i agents with capability c_i are in region q at time k. This variable captures the satisfaction of cp_i in region q at time k, using the constraints

$$-Mz_{q,cp_i}(k) + \sum_{\{g|c_i\in g\}} z_{q,g}(k) \geq m_i - M \tag{10}$$

where M is a sufficiently large number, e.g., $M \geq 1 + \max\{\max_i\{m_i\}, |J|\}$. Under a simple assumption of feasibility of the CaTL formula, $\forall i\ m_i \leq |J|$, M can be taken larger than $1 + |J|$.

Next, we must determine satisfaction across each region q that is labeled π. We define integer variables $z_{\pi,cp_i}(k) \in \{0,1\}$ that we wish to be valued 1 if at least m_i agents with capability c_i are in each region $q \in L^{-1}(\pi)$ at time k. This can be accomplished with the set of constraints

$$z_{\pi,cp_i}(k) \geq \sum_{q\in L^{-1}(\pi)} z_{q,cp_i}(k) - |L^{-1}(\pi)| + 1$$

$$z_{\pi,cp_i}(k) \leq z_{q,cp_i}(k) \forall q \in L^{-1}(\pi). \tag{11}$$

Next, we define integer variables $z_{\pi,\mathcal{I}_T}(k) \in \{0,1\}$ that we wish to be valued 1 if at least m_i agents with capability c_i are in each region $q \in L^{-1}(\pi)\ \forall i \in \mathcal{I}_T$. This can be accomplished with the set of constraints

$$z_{\pi,\mathcal{I}_T}(k) \geq \sum_{i\in\mathcal{I}_T} z_{\pi,cp_i}(k) - |\mathcal{I}_T| + 1$$

$$z_{\pi,\mathcal{I}_T}(k) \leq z_{\pi,cp_i}(k) \forall i \in \mathcal{I}_T. \tag{12}$$

Finally, we define integer variables $z_T(k) \in \{0,1\}$ that we wish to be valued 1 if the task will be completed at time $k + d$ and 0 otherwise. This can be accomplished with the set of constraints

$$z_T(k) \geq \sum_{\ell=k}^{k+d} z_{\pi,\mathcal{I}_T}(\ell) - d + 1$$

$$z_T(k) \leq z_{\pi,\mathcal{I}_T}(\ell) \forall \ell = k, \ldots, k + d. \tag{13}$$

C. Formula Satisfaction

The satisfaction of a CaTL formula ϕ can be converted to a set of mixed integer linear constraints using encodings derived from STL encodings as given in [32], [33], and summarized in [34]. These encodings consist of binary variables $\{z_{\phi,k}\}_{k\in K}$ such that $z_{\phi,k} = 1 \Leftrightarrow (s_J, k\delta t) \models T$. CaTL formulae can be built from applying recursive encodings to the constraints $\{z_{T,k}\}$ defined in Section VI-B above.

D. Objective Function

Here, we present equivalent MILP encodings for the availability robustness.

1) Availability Robustness: The encodings recursively define intermediate variables $r_{k,\varphi}$ whose values are equivalent to $\rho(s_J, k\delta t, \varphi)$ where φ is a subformula of a given CaTL formula ϕ. When φ is nonatomic, the encodings for $r_{k,\varphi}$ are equivalent to standard recursive encodings of STL [24], [33]. Note that these encodings require the given formula to be in positive normal form (PNF), i.e., contain no negations ($\neg$).

Proposition 4: Every CaTL formula in PNF is equivalent to an STL formula in PNF.

Proof: See Appendix A-D.

In what follows, we give the encodings at the atomic task level T, i.e., $r_{k,T}$. Following the conventions of [33], we replace (10) with

$$\sum_{\{g|c_i\in g\}} z_{q,g}(k) - m_i + M(1 - z_{q,cp_i}(k)) \geq r_{0,\phi} \tag{14a}$$

$$\sum_{\{g|c_i\in g\}} z_{q,g}(k) - m_i - Mz_{q,cp_i}(k) \leq r_{0,\phi}$$

$$\forall k = 0, \ldots, K\ \forall (cp_i) \text{ appearing in } \phi. \tag{14b}$$

As pointed out in [34], applying the recursive quantitative semantics of any STL formula in PNF leads to compositions of minimum and maximum operators applied to predicate values over time. Thus, the value of $\rho(s_J, 0, \phi)$ must be equal to some value of the *margin* of a predicate at a certain time, i.e., by how much a function of the value of that signal exceeds (or falls below) the constant bound of a predicate. In our case, this corresponds to how many agents of a certain capability c_i exceed the required threshold m_i ($\sum_{\{g|c_i\in g\}} - m_i$) or how many more would need to be added to meet m_i ($m_i - \sum_{\{g|c_i\in g\}}$).

E. Optimization

Finally, we present the optimization performed to solve Problem 1

$$\begin{gathered}\max_{\{u_{e,g}(k)\}} r_{0,\phi} \\ \text{subject to} \\ \text{team dynamics (8), (9),} \\ \text{task satisfaction (11)–(13),} \\ \text{availability robustness (14).}\end{gathered} \tag{15}$$

This optimization can be solved using any existing solver and software tool for MILPs.

VII. Algorithmic Extensions

In this section, we expand our algorithm to address several practical implementation concerns. We consider the following: 1) regularizing by travel time to avoid spurious motion; 2) an upper bound on our objective function to improve the speed of the optimization; 3) online replanning in the event of agent attrition.

A. Total Travel Time Regularization

An optimal solution to Problem 1 does not explicitly consider agent travel time. Therefore, agents may take unnecessarily long paths to their goals, or may wander the environment when they

are not required to accomplish a task. An operator may wish to avoid these types of extraneous motion. In order to limit this spurious travel, we propose to regularize the optimization by the total travel time of the agents, defined as

$$\tau = \sum_{k=0}^{K} \sum_{g \in G} \sum_{(q,q') \in E, q \neq q'} u_{(q,q'),g}(k). \tag{16}$$

Note that τ is linear in the decision variables. The scale of τ, however, may be much larger than the scale of $r_{0,\phi}$. In order to prevent optimization of τ from overriding the optimization of $r_{0,\phi}$, we introduce the relative weighting term

$$\gamma = \frac{\alpha}{|J|K} \tag{17}$$

where $\alpha \in (0, 1)$.

In plain English, the following proposition states that minimizing the weighted total travel time with weighting γ will never come at the expense of maximizing robustness.

Proposition 5: Let $f(u, z) = r_{0,\phi}(u, z) - \gamma\tau(u, z)$. If $r_{0,\phi}(u_1, z_1) > r_{0,\phi}(u_2, z_2)$, then $f(u_1, z_1) > f(u_2, z_2)$.

Proof: See Appendix A-E

Proposition 6: Problem 1 is equivalent to solving the integer linear program

$$\begin{gathered} \max_{\{u_{e,g}(k)\}} r_{0,\phi} - \gamma\tau \\ \text{subject to} \\ (8)\text{–}(9),\ (11)\text{–}(13),\ (14), \end{gathered} \tag{18}$$

Proof: This follows directly from Propositions 3, 4, Theorem 1 from [34], and Proposition 5.

B. Initial Upper Bound

In practical settings, it may be desirable to obtain solutions to Problem 1 as quickly as possible. We consider an upper bound to the availability robustness that is efficient to compute. This allows the optimization software to consider a smaller set of possible solutions, which greatly reduces computation times.

The upper bound is given in terms of *capability excess* (Definition 11 below). Capability excess is similar to availability robustness, in that it has a recursive formulation for determining the excess number of agents for a formula. Unlike availability robustness, capability excess can be computed without finding a candidate solution for satisfying a CaTL specification.

We briefly explain the intuition behind capability excess here. Consider a task $T = (d, \pi, \{(c_i, m_i)\}_{i \in \mathcal{I}_T})$ and a team of agents $\{A_j\}_{j \in J}$. If the system were totally unconstrained, then all of the agents could service T simultaneously. Then the agents would be spread evenly over the regions labeled π. The difference between the number of agents in a region with capability c_i and the required number m_i provides an estimate of how well the team can satisfy the requirement on capability c_i. By finding the minimum across all capabilities c_i for a task, we can estimate an upper bound on robustness for that task. In Definition 11, the first row defines the robustness of a task.

Starting with the capability excess for each task in a CaTL formula, the capability excess is recursively computed. The resulting capability excess for the entire formula is an upper bound on the availability robustness of the MILP, as given in Proposition 7 below.

Definition 11: The *capability excess* of a CaTL formula ϕ with respect to set of agents $\{A_j\}_{j \in J}$, denoted $ce(\{A_j\}_{j \in J}, \phi)$, is given as

$$\begin{aligned} ce(\{A_j\}_{j \in J}, T) &= \min_{i \in \mathcal{I}_T} \lfloor \tfrac{\sum_{j \in J} I(c_i \in g_j)}{|L^{-1}(\pi)|} \rfloor - m_i \\ ce(\{A_j\}_{j \in J}, \phi_1 \wedge \phi_2) &= \min(ce(\{A_j\}_{j \in J}, \phi_1) \\ &\quad\ ce(\{A_j\}_{j \in J}, \phi_2)) \\ ce(\{A_j\}_{j \in J} \phi_1 \vee \phi_2) &= \max(ce(\{A_j\}_{j \in J}, \phi_1) \\ &\quad\ ce(\{A_j\}_{j \in J}, \phi_2)) \\ ce(\{A_j\}_{j \in J}, \phi_1 \mathcal{U}_{[a,b)} \phi_2) &= \min(ce(\{A_j\}_{j \in J}, \phi_1) \\ &\quad\ ce(\{A_j\}_{j \in J}, \phi_2)) \\ ce(\{A_j\}_{j \in J}, \Diamond_{[a,b)} \phi) &= ce(\{A_j\}_{j \in J}, \phi) \\ ce(\{A_j\}_{j \in J}, \Box_{[a,b)} \phi) &= ce(\{A_j\}_{j \in J}, \phi). \end{aligned} \tag{19}$$

Proposition 7: The capability excess $ce(\{A_j\}_{j \in J}, \phi)$ is an upper bound for $\max_{\{u'_{e,g}(k)\}} r_{0,\phi}$.

Proof: See Appendix A-F.

Capability excess is specific to CaTL, in that it is a semantically meaningful measure of robustness. For STL in general, bounds on robustness can be computed based on bounds on signals, if they are available. For CaTL, we use the set of agents to determine the upper bound, which is provided in terms of the number of agents. Indeed, because the capability excess of a task can be computed on a per-class basis, it is possible to compute a vector of capability excess to bound the z variables in the encoding that correspond to agent classes, task satisfaction, and formula satisfaction, leading to additional computational gains.

C. Online Replanning

During deployment, agents may become disabled, and the system needs to be able to replan accordingly. Here, we discuss the case of how to replan in the event of agent attrition, although a similar process could be used for agents deviating from their nominal plan (i.e., being late or going off-course). To address agent attrition, we consider replanning at time t_{att} using only a subset of the original team $\{A_j\}_{j \in J \setminus P}$, where P is the set of agents that attrit.

The satisfaction of CaTL constraints depends on trajectory history, and therefore replanning from the middle of a deployment requires the history to be encoded in the constraints of the problem as follows. Prior to t_{att}, the MILP variables are fixed to be equal to the output of the original solution to the MILP. This encodes the history of the agents up to time t_{att}. The optimizer then finds the first feasible solution of the updated system. The first feasible solution is taken due to the inherent time constraints of online planning. Depending on the time scale of the problem, the robust solution could be used if time permits for the computation.

In this section, the $\hat{}$ (hat) symbol indicates variables and constraints after replanning. We denote the discrete time index corresponding to t_{att} as k_{att}, i.e., $t_{\text{att}} = \delta_t k_{\text{att}}$.

Proposition 8: Let $a_{e,g,k_{\text{att}}}$ be the number of agents of class g that drop out along edge $e \in E$ at time t_{att}. Problem 2 is equivalent to solving the following mixed integer linear program:

$$\max_{\{\hat{u}_{e,g}(k)\}} r_{0,\phi} - \gamma\tau$$

subject to

$$(11)\text{–}(13),\ (14) \text{ using } \{\hat{z}_{q,g}(k)\} \tag{20a}$$

$$\hat{u}_{e,g}(k) = u_{e,g}(k), \forall e \in E, g \in G, k < k_{\text{att}} \tag{20b}$$

$$\hat{u}_{e,g}(k_{\text{att}}) = u_{e,g}(k_{\text{att}}) - a_{e,g,k_{\text{att}}}, \forall e \in E, g \in G \tag{20c}$$

$$\hat{z}_{q,g}(k) = z_{q,g}(k), \forall q \in Q, g \in G, k < k_{\text{att}} \tag{20d}$$

$$\hat{z}_{q,g}(k) = \sum_{(q,'q)\in E} \hat{u}_{(q,'q),g}(k - \mathcal{W}(q,'q))$$

$$\forall q \in Q, g \in G, k \geq k_{\text{att}} \tag{20e}$$

$$\sum_{(q,q')\in E} \hat{u}_{(q,q'),g}(k) = \sum_{(q,'q)} \hat{u}_{(q,'q),g}(k - \mathcal{W}(q,'q))$$

$$\forall q \in Q, g \in G, k \geq k_{\text{att}}. \tag{20f}$$

Proof: By diverting the $a_{(q,q'),g,k_{\text{att}}}$ agents at time k_{att} (20c), the updated edge and node balance equations enforce that the agents that drop out are not counted toward the solution. Setting the equality constraints for edge variables for times $k < k_{\text{att}}$ enforces that what has happened before attrition is accurately accounted in the optimization (20b), (20d). Because we have set the values of the edge and node variables via equality constraints, we do not need to check for motion consistency before k_{att}.

Remark 1: Because checking the satisfaction of a CaTL formula is trace-dependent rather than state dependent, we need to keep some state history in order to determine the best future set of actions. The desire for overall computational efficiency, however, suggests performing an additional procedure to truncate how far in the past the optimization should consider, and which parts of the CaTL formula are currently relevant (i.e., which part of the formula was not satisfied before attrition). Modern presolve methods for mixed integer solvers eliminate redundant constraints from the optimization [35]. In other words, these presolve routines already in effect truncate the history, removing motion constraints and the parts of CaTL satisfaction checking that are made redundant by setting the equality constraints of edge variables.

VIII. Computational Experiments

In this section, we characterize the computational requirements of our methodology via an extension of the precision agriculture case study used as a running example throughout this article. We consider a fixed specification $\phi_{pa} = \bigwedge_{i=2}^{5} \psi_i$ for all of the experiments (see Section IV). All computation times, number of variables, and number of constraints in this section are presented as the mean and maximum values. All robustness values are given as means. For the purpose of these experiments, we focus on the problem of region planning, and ignore low-level motion planning. All experiments were performed on a PC with 32 cores with 2.10 GHz processors and 64 GB of RAM. The off-the-shelf commercial MILP solver used was Gurobi Optimizer 8.1.1 [36].

A. Experiment 1—Effect of Algorithm Variants

The first experiment is used to quantify the performance changes incurred due to two of the contributions to this article, namely, augmenting the cost in the optimization with a total travel time regularization and by adding upper bounds to the optimization. We therefore consider four variants of the algorithm as summarized in Table III. The preliminary work of [11] corresponds to column 1 and rows 1 and 2 in Table III.

Characterization: We generate 50 random 3×3 grid transition systems. Edge weights are chosen uniformly from $W = \{1, 3\}$. The probability of a region being labeled was 0.2, and the label of each labeled region in the graph is drawn uniformly from $AP = \{\pi_{\text{blue}}, \pi_{\text{orange}}, \pi_{\text{yellow}}, \pi_{\text{green}}\}$.

For each randomly generated environment, we consider teams of 20 agents from four classes each with two capabilities drawn from $\{Vis, UV, IR, Mo\}$. We ensure all individual capabilities are covered in the classes. The initial states of each of the agents are selected uniformly at random.

We solve Problem 1 for each of these instances. We record the time to achieve the solution, the number of variables, number of constraints, and availability robustness ρ. Results from these experiments are shown in Table IV.

Results: These results indicate that although the encoded MILP for this problem can be quite large, the computation time is reasonably short. We note that there is a large difference in time between the time to find an optimal solution and the time to find the first feasible solution. Because there are excess agents available to perform this task, there are many satisfying solutions. However, finding the most robust solution requires more time to search through these solutions. Additionally, adding regularization increases the computation time by almost two orders of magnitude due to the fact that we need to search the entire set of possible solutions. In Table IV, the mean value for run time for robust regularized bounded (182 s) appears to be higher than the mean run time for robust regularized (158 s). However, a two-sided t-test suggests that the population means do not differ ($p = 0.76$), suggesting that the slow down due to regularization dominates any improvement provided by the upper bound. In future work, we may look for ways to add (approximate) regularization in postprocessing. For this set of experiments, the mean value of ρ was 1.2, whereas the mean value of ce was 1.3, indicating the ce upper bound on the optimization is fairly tight.

B. Experiment 2—Effect of Problem Dimensions

In our second computational experiment, we vary the size of the environment, the number of agents, and the number of agent

TABLE III
ALGORITHM VARIANTS TESTED IN CASE STUDY. VARIANTS INCLUDING ROBUSTNESS RETURN A SOLUTION WITH MAXIMUM AVAILABILITY ROBUSTNESS (SECTION VI-D1), WHILE VARIANTS THAT DO NOT INCLUDE ROBUSTNESS RETURN THE FIRST FEASIBLE SOLUTION. VARIANTS THAT INCLUDE TRAVEL TIME REGULARIZATION MINIMIZE EXTRANEOUS MOTION (SECTION VII-A). VARIANTS THAT INCLUDE OBJECTIVE BOUNDS PROVIDE AN UPPER BOUND TO THE OPTIMIZER (SECTION VII-B)

Name	Robustness	Travel Time Regularization	Objective Bounds
Feasible	×	×	×
Robust	✓	×	×
Regularized	×	✓	×
Robust Regularized	✓	✓	×
Robust Bounded	✓	×	✓
Robust Regularized Bounded	✓	✓	✓

TABLE IV
SUMMARY STATISTICS (MEAN / MAX) FOR EXPERIMENT 1. FOR EACH VARIANT, THE MEAN (MAX) NUMBER OF VARIABLES WAS 10585.4 (11604), AND THE MEAN (MAX) NUMBER OF CONSTRAINTS WAS 7480.3 (8074), EXCLUDING THE CONSTRAINT ADDED BY THE UPPER BOUND. THE MEAN (MAX) CAPABILITY EXCESS ce FOR EACH VARIANT WAS 1.3 (4.0)

Method	Computation Time (s)	ρ_a	$\frac{\tau}{\lvert J\rvert\lvert K\rvert}$	Timeouts
Feasible	$0.768/1.278$	$0/0$	$0.122/0.203$	0
Robust	$2.931/7.532$	$1.2/4$	$0.198/0.270$	0
Regularized	$155.310/603.392$	$0/0$	$0.014/0.025$	6
Robust Regularized	$158.066/601.414$	$1.2/4$	$0.035/0.107$	3
Robust Bounded	$0.889/2.188$	$1.2/4$	$0.179/0.270$	0
Robust Regularized Bounded	$182.080/602.036$	$1.2/4$	$0.035/0.107$	5

TABLE V
LIST OF TASKS USED IN (21)

Name	Duration (d)	Regions ($L^{-1}(\pi)$)	Counting Proposition (c)	Plain English
T_{red1}	3	q_3	$\{(CFD, 2)\}$	Survey for pests
T_{red2}	2	q_3	$\{(GRF, 1)\}$	Deter pests
T_{red3}	3	q_3	$\{(LDF, 1)\}$	Deter pests
T_{H1}	5	q_5	$\{(GRF, 3)\}$	Harvest crops
T_{H2}	5	q_6	$\{(GRF, 3)\}$	Harvest crops
T_{blue}	3	q_{10}	$\{(GRD, 1), (CFD, 1)\}$	Estimate water reserves
T_{I1}	3	q_8	$\{(GRF, 3)\}$	Inspect crops
T_{I2}	2	$\{q_0, q_2\}$	$\{(GRF, 1), (CFD, 1)\}$	Inspect crops

TABLE VI
DIFFERENCES BETWEEN REGION-LEVEL TRAJECTORY AND STATE SPACE TRAJECTORY TIMING AND TRAJECTORIES. NOTE THAT TWO AGENTS DID NOT ENTER ANY EXTRA REGIONS AND ARE NOT INCLUDED IN THE MEAN TIME IN EXTRA REGIONS. OVERALL TIME IN REGIONS IS PROVIDED FOR COMPARISON AND INCLUDES NOMINAL (CORRECT) REGIONS AS WELL AS EXTRA REGIONS

	Mean / Max (s)
Early arrival time	$89.4/168$
Number of extra regions	$3.3/9$
Time in extra regions	$22.6/241$
Overall time in regions	$131.4/415$

TABLE VII
OBSTACLES ENTERED BY AGENTS DURING EXECUTION. DIFFERENCES BETWEEN THE REGION-LEVEL AND STATE SPACE TRAJECTORY PLANNERS CAN RESULT IN SMALL INCURSIONS INTO OBSTACLE REGIONS

Agent	Obstacle	Duration (s)	Max Incursion (cm)
CF	q1	3	13.7
CF	q4	1	4.6
GR	q1	4	16.7

classes. We wish to determine how the effects of the parameters of the problem affect our algorithm's quality and speed. For simplicity of presentation, we present only the feasible and robust versions of our algorithm. We expect the effects of travel time regularization and upper bounds to be similar to the results seen in Experiment 1 above.

Scalability with environment size: For this experiment, we maintain the team size at 20 and vary the number of states in the environment. The results of this experiment are visualized in Fig. 3(a), (d), and (g) and summarized in Table IX in Appendix B. These experiments indicate that the size of the environment (and thus, the number of tasks that are required to be performed) can have a large effect on the computation time required to achieve a solution. This is due to an increase in the number of variables that must be tracked, the number of constraints used to describe the environment, and an increase in the length of the expected paths of the agents. In future work, this effect may be mitigated during the process of abstracting the workspace to the environment model by grouping together adjacent regions in which no service is required.

Scalability with number of agents: For this experiment, we maintain the environment size at 25 and vary the number of

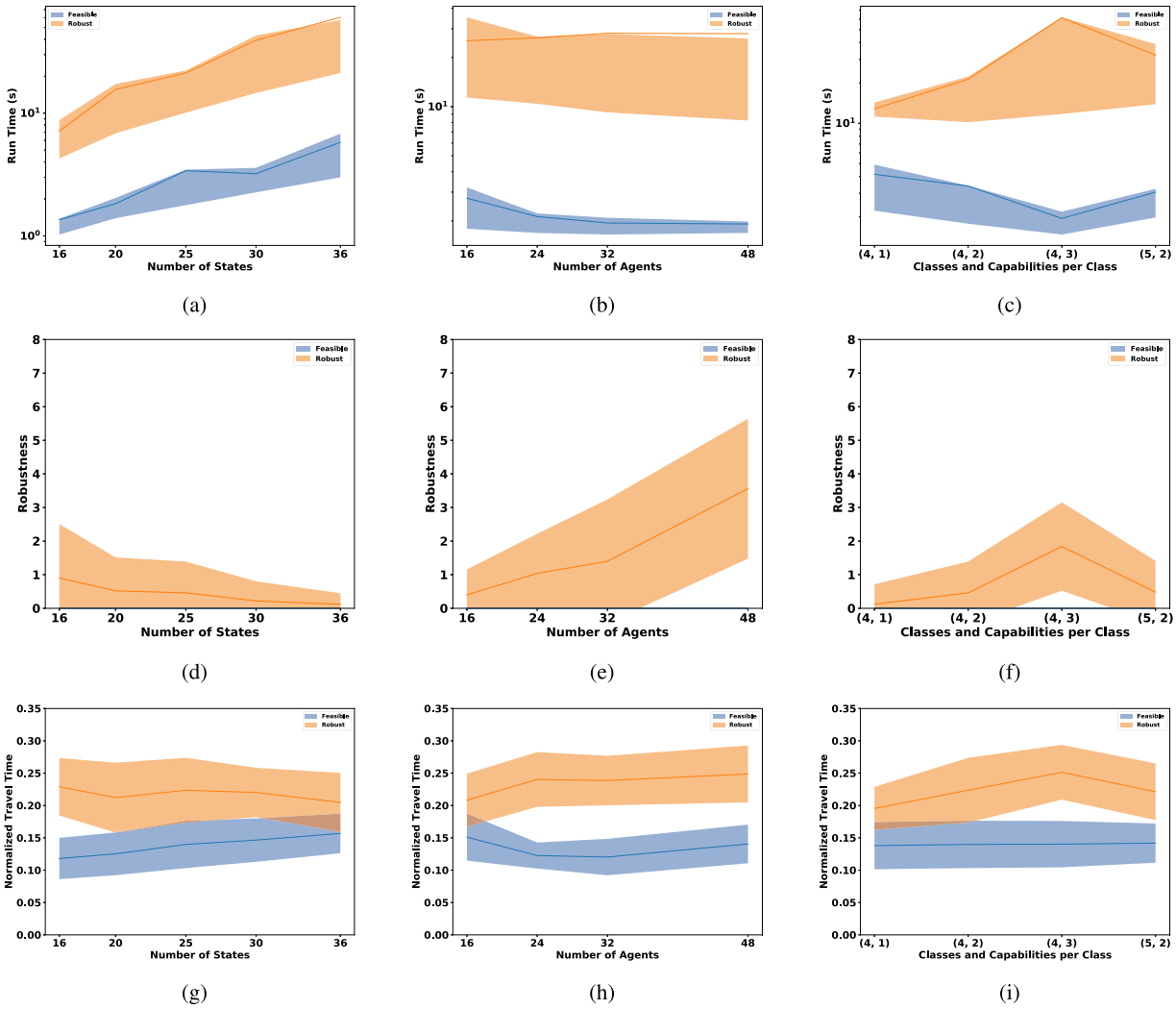


Fig. 3. Results for Experiment 2 varying the number of states, agents, and capabilities per class of agent. The solid line indicates the mean, and shaded regions indicate interquartile range (for run time) or standard deviation (for robustness and normalized travel time). *Top row:* Overall computational time to find the first feasible solution and maximally robust solutions compared to the number of states (3a), agents (3b), and capabilities per class (3c). Note that the presence of timeouts for the robust solution skews the statistics, so interquartile range is presented in the shaded area. *Middle row:* Availability robustness for both the maximally robust and first feasible solutions compared to the number of states (3b), agents (3e), and capabilities per class (3f). *Bottom row:* Normalized travel time for the first feasible solution and for the maximally robust solution compared to the number of states (3g), agents (3h), and capabilities per class (3i).

agents. The results of this experiment are visualized in Fig. 3(b), (e), and (h) and summarized in Table X in Appendix B. These results suggest that the number of agents has a relatively small effect on run time, but increasing the number of agents increases the robustness of the maximally robust solution. This is expected since our approach does not keep track of individual agents, but only of their number.

Scalability with number of capabilities per class: For this experiment, we maintain the team size at 20 and the environment size at 25. We vary the number of capabilities of each class of agent, as well as the number of classes (while holding the total number of agents constant). We held the number of agents fixed to control for the effects of varying the team size. Since our specification includes four capabilities, we were able to assess for 4 and 5 classes. The results of this experiment are visualized in Fig. 3(c), (f), and (i) and summarized in Table XI in Appendix B.

Results: In aggregate, these results suggest that the time to compute a feasible solution remains relatively flat across changes in environment size, number of agents, and capabilities and classes. The time to find a robust solution is most sensitive to the environment size and the number of capabilities per class. We observe that robust solutions tend to have a higher normalized travel time, indicating that more agents are being dispatched to the same task.

IX. Hardware Demonstration

To assess the real-world feasibility of ScRATCHeS, we performed two hardware demonstrations of the above algorithms using first, 10 heterogeneous robots with three unique platforms and four unique sensing capabilities, and second, 7 heterogeneous robots with agent dropout. All experiments are performed in an indoor 6 m by 9 m motion capture environment shown

TABLE VIII
LIST OF TASKS USED IN (21)

Name	**Duration** (d)	**Regions** ($L^{-1}(\pi)$)	**Counting Proposition** (c)	**Plain English**
T_1	5	W	$\{(C_3, 1)\}$	Estimate water reserves
T_2	3	H	$\{(C_1, 2), (C_2, 1)\}$	Harvest
T_3	3	D	$\{(C_1, 2), (C_2, 1)\}$	Unload harvest
T_4	5	E	$\{(C_1, 2), (C_2, 1)\}$	Unload harvest
T_5	5	A	$\{(C_3, 2)\}$	Survey for harvest
T_6	5	B	$\{(C_1, 2)\}$	Harvest
T_7	5	C	$\{(C_2, 2)\}$	Harvest
T_8	3	C	$\{(C_1, 2)\}$	Harvest
T_9	5	A	$\{(C_1, 1), (C_3, 1)\}$	Inspect and harvest
T_{10}	2	C	$\{(C_3, 2)\}$	Inspect crops

TABLE IX
SUMMARY STATISTICS (MEAN / MAX) FOR EXPERIMENT 2 WHILE VARYING THE ENVIRONMENT SIZE FOR 20 AGENTS. WHILE THE COMPUTATION TIME INCREASES FOR BOTH FEASIBLE AND ROBUST SOLUTIONS, THE ROBUST SOLUTION SCALES MORE POORLY, WITH THE MEAN COMPUTATION TIME ON THE ORDER OF 10X LONGER THAN THE FEASIBLE SOLUTION FOR 36 STATES

Method	Env. Size	Comp. Time (s)	Method	Env. Size	Comp. Time (s)	ρ	ce
feasible	9	0.62/0.91	robust	9	2.83/8.44	2.44/8	1.5/6
feasible	12	0.87/1.23	robust	12	5.11/30.92	1.60/8	1.4/6
feasible	16	1.35/3.35	robust	16	7.16/23.00	0.90/8	1.7/6
feasible	20	1.83/4.87	robust	20	15.60/60.48	0.52/3	1.4/6
feasible	25	3.40/15.79	robust	25	21.29/140.95	0.46/3	1.4/6
feasible	30	3.21/7.78	robust	30	39.29/211.53	0.22/3	1.2/6
feasible	36	5.79/23.92	robust	36	60.45/601.79	0.121	1.2/6

TABLE X
SUMMARY STATISTICS (MEAN / MAX) FOR EXPERIMENT 2 WHILE VARYING THE NUMBER OF AGENTS. THE ENVIRONMENT SIZE WAS FIXED AT 25. WE OBSERVE A DECREASE IN COMPUTATION TIME FOR A FEASIBLE SOLUTION AS THE NUMBER OF AGENTS INCREASES. LIKEWISE, THERE IS AN INCREASE IN ROBUSTNESS IN THE PRESENCE OF MORE AGENTS. THE AVERAGE COMPUTATION TIME REMAINS APPROXIMATELY THE SAME FOR THE ROBUST SOLUTION, BUT THE WORST-CASE COMPUTATION TIME APPEARS TO INCREASE AS THE NUMBER OF AGENTS INCREASES

Method	Team Size	Comp. Time (s)	Method	Team Size	Comp. Time (s)	ρ	ce
feasible	16	2.75/10.18	robust	16	25.33/140.46	0.40/2	2.4/6
feasible	24	2.13/8.04	robust	24	26.35/214.58	1.04/4	2.4/8
feasible	32	1.94/4.04	robust	32	28.14/330.12	1.40/6	2.5/10
feasible	48	1.91/3.00	robust	48	27.97/233.76	3.56/8	4.7/16

TABLE XI
SUMMARY STATISTICS (MEAN / MAX) FOR EXPERIMENT 2 WHILE VARYING THE NUMBER OF CLASSES AND THE NUMBER OF CAPABILITIES PER CLASS. ENVIRONMENT SIZE WAS FIXED AT 25 AND THE NUMBER OF AGENTS AT 20. THE TIME TO FIND A FEASIBLE SOLUTION APPEARS TO DECREASE WITH THE NUMBER OF CAPABILITIES PER CLASS, WHILE IT INCREASES IN THE ROBUST CASE

Method	Capabilities per Class	Classes	Comp. Time (s)	Method	Capabilities per Class	Classes	Comp. Time (s)	ρ	ce
feasible	1	4	4.15/20.83	robust	1	4	12.81/22.84	0.12/3	1.0/1.0
feasible	2	4	3.39/16.17	robust	2	4	21.34/141.29	0.46/3	1.4/6
feasible	3	4	1.95/4.65	robust	3	4	61.46/525.63	1.84/5	6.5/8
feasible	2	5	3.06/712.31	robust	2	5	32.27/128.54	0.48/4	1.7/4

in Fig. 5. The robots operate simultaneously, and the position of each robot is tracked with an Optitrack motion capture system.[1] All of the planning is done on a Intel i7-7800X CPU running Ubuntu 16.04 and the MILP solver is Gurobi 9.0 [36] using either the robust, regularized, bounded optimization for offline computation, and first feasible optimization for online replanning.

The robots used in the demonstrations include CrazyFlie 2.0 nano-UAVs, a large 220 mm custom drone, and iRobot Create2 Ground Robots. Each of these platforms communicates using the robot operating system (ROS) kinetic architecture [37].

[1]Natural Point Optitrack. [Online]. Available: https://www.optitrack.com

A. Motion Planning

Until this point, we have considered discrete, region-level trajectories. These trajectories need to be translated into individual robot motion plans, considering agent geometry, dynamic constraints, and inter-agent collisions. The state space motion planning for the team of robots is performed according to a sequential, timed, multiagent rapidly exploring random trees (RRT) algorithm ([38]) where an agent plans its entire trajectory, and is then considered an obstacle (at specific time instants) to future planning agents. This planning is also stratified based on operation height (i.e., drones do not consider ground robots as obstacles). The motion plans are calculated over then planning units for each CaTL time step. This allows for the trajectories

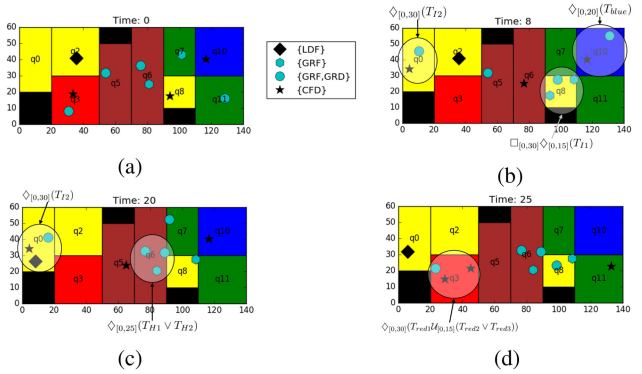


Fig. 4. Snapshots of solution to (21) in the demonstration environment. Times 8, 20, and 25 show where specific components of the specification are satisfied and these areas are highlighted in (b)–(d).

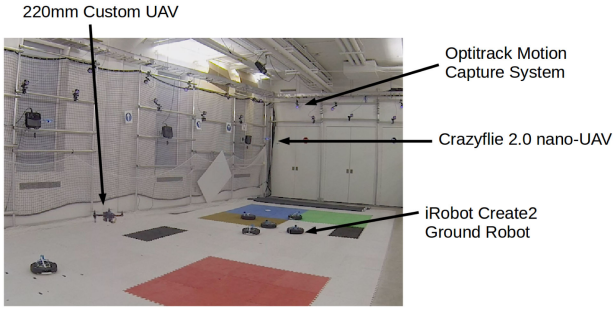


Fig. 5. Demonstration flight arena with motion capture system, colored regions representation simulation regions, and the three platform types performing the specification task. Note there is one large UAV, six ground robots, and 3 nano-UAVs deployed simultaneously.

to both have flexibility in avoiding obstacles and other agents, as well as better fitting the execution time to the size of the experimental space. More information about this approach may be found in Appendix C

Each robot follows a set of waypoints given by the multiagent RRT algorithm. The iRobot Create2 robots employ a pure pursuit controller to follow these points by controlling linear and angular velocity of the robots. The Crazyflie UAVs employ a PID controller to stabilize their flight given their current position and orientation using the CrazySwarm [39] software package. The custom UAV uses the PX4 flight stack [40] to control its flight in the space given its position from the motion capture system. Fig. 5 shows the robots operating in the experimental space.

B. Demonstration 1

A precision agriculture scenario is demonstrated, where a team of robots must inspect crops, harvest crops, survey for pests, deter pests, and estimate water reservoir levels simultaneously. To accomplish these tasks, agents must either take pictures of the prescribed region (inspect crops, survey for pests, and estimate water reserves), or simply be present in the prescribed region for the prescribed time (harvest crops and deter pests). The task definitions for this demonstration are shown in Table V and the specification for this demonstration is shown in (21).

The demonstration consists of three CrazyFlie 2.0 nano-UAV (CF), a Large 220 mm custom Drone (LD), and six iRobot Create2 Ground Robots (GR). Each platform carries a camera that can be oriented either forward (F) or downward (D). The set of capabilities for this demonstration is $Cap_{exp} = \{CFD, LDF, GRF, GRD\}$. The GRD capability is performed using a downward RGBC color sensor (considered as a single pixel camera), and coexists on four of the iRobot Create2 platforms with a forward facing camera. The optimization is run at time window of 12 seconds per time unit, and the transition weights in the optimization are 1/12 meters the largest distance between any two points between connected regions. This is an over approximation for transition time. The motion planning algorithms expand this 1/12 time scaling to consider actual travel times. This abstraction allows for faster optimization computation times because fewer time steps need to be considered.

The robots are divided among three altitudes to augment their sensor modalities and to separate them aerodynamically. The Crazyflie nano-UAVs fly at a higher altitude than the custom UAV to avoid its considerable down-wash

$$\begin{aligned}\phi_{demo_1} = &\diamondsuit_{[0,20)}(T_{blue}) \\ &\wedge \diamondsuit_{[0,25)}(T_{H1} \vee T_{H2}) \\ &\wedge \square_{[0,30)}\diamondsuit_{[0,15)}(T_{I1}) \\ &\wedge \diamondsuit_{[0,30)}(T_{I2}) \\ &\wedge \diamondsuit_{[0,30)}(T_{red1}\mathcal{U}_{[0,15)}(T_{red2} \vee T_{red3})). \end{aligned} \quad (21)$$

This demonstration does not include any replanning and is run in an open loop fashion. ScRATCHS was able to develop an initial motion plan to satisfy ϕ_{demo_1} in a total of 10.25 s (9.16 s to solve the MILP and 1.09 s to generate the motion plans). Note that this is substantially faster than most teams of humans can solve moderately sized planning and coordination problems by hand [41]. Snapshots of the solution to (21) are shown in Fig. 4.

When the trajectories of the agents are examined, some discrepancies between the discrete region-level plan and the state space trajectories followed by the agents. First, agents tend to arrive to regions much earlier than expected by their discrete region-level plan. This suggest that there is room to improve the abstraction and estimation of travel time, perhaps via sampling RRT examples between regions *a priori*. Agents also tend to clip regions and briefly enter them, due to properties of the edge checker in the RRT algorithm. If this impacts satisfaction (e.g., in the case of needing to avoid regions), the environment abstraction may require refinement, or other solutions could be employed to mitigate these effects. These results are summarized in Table VI. Likewise, we see that some agents have clipped the obstacle regions during run time (see Table VII). Again, the motion planner or abstraction of the environment can be refined to reduce the likelihood of such incursions.

C. Demonstration 2

A precision agriculture scenario is again demonstrated, where a team of robots must now survey crops, harvest crops, drop off

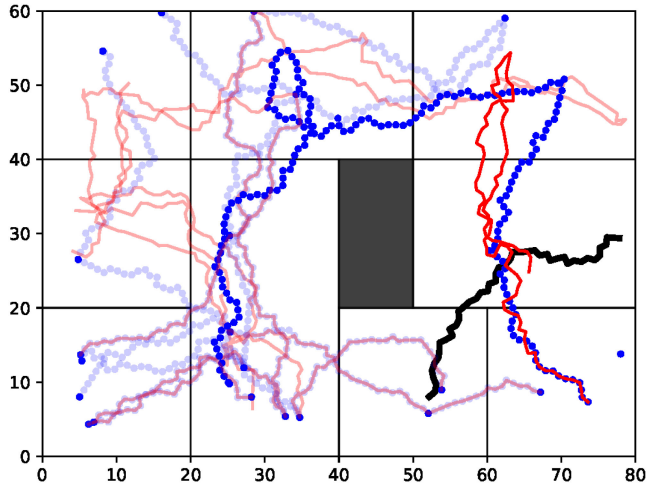


Fig. 6. Solution trajectories before and after replanning for ϕ_{demo_2}. Blue trajectories are before replanning, and red trajectories are postreplanning. The black trajectory is for the drone that drops out. The emphasized trajectory is of the agent that fills in for the lost agent.

Fig. 7. Overhead view of Demonstration 2. Note 6 ground robots and 1 aerial robot. Two ground robots have camera and small harvester capabilities (one in H and one in M), the 4 other ground robots have the large harvester and small harvester capabilities, and the drone has only the camera capability.

their harvest, and estimate water reservoir levels simultaneously. A smaller team of seven robots, comprised of two platforms (the large custom drone and six iRobot Create2 ground robots) and three capabilities (camera, large harvester, and small harvester) are used for this demonstration. The task definitions are shown in Table VIII and the specification for this demonstration is shown in (22). The set of capabilities for this demonstration is $Cap_{exp2} = \{C_1, C_2, C_3\}$. C_1 corresponds to the *small harvester* capability, C_2 corresponds to the *large harvester* capability, and C_3 corresponds to the *camera* capability

$$\begin{aligned}\phi_{\text{demo}_2} = &\Box_{[0,75)}\Diamond_{[0,25)}(T_1)\\ &\wedge \Box_{[0,75)}\Diamond_{[0,25)}(T_2)\\ &\wedge \Box_{[25,75)}\Diamond_{[0,20)}(T_3 \vee T_4)\\ &\wedge \Diamond_{[0,50)}(T_5)\mathcal{U}_{[0,25)}(T_6 \vee T_7)\\ &\wedge \Diamond_{[0,40)}(T_8)\\ &\wedge \Diamond_{[30,60)}(T_9)\\ &\wedge \Diamond_{[15,45)}(T_{10}).\end{aligned} \tag{22}$$

At time $t = 16$, we simulate the drone dropping out. This robot ceases its motion (flies to a safe waypoint outside the environment boundary) and online replanning [see (21)] is triggered, enabling the remaining platforms to continue the mission. After replanning, the drone is replaced by a ground robot with a camera capability. In the experimental demonstration, that robot then goes on to fulfill a different set of propositions than originally planned. This sequence of events is shown in Fig. 6, where the drone trajectory is shown in black, and the ground robot that replaces it is shown in bold. Fig. 7 shows a snapshot of the second hardware demonstration.

This demonstration includes replanning due to agent dropout. ScRATCHeS was able to develop an initial motion plan to satisfy ϕ_{demo_2} in a total of 159.28 s (145 s to solve the MILP and 14.28 s to generate the motion plans) using the travel regularized robustness and returned the robust solution ($\rho = 0.076$). The replanning phase is triggered at t=16, and the replanning took 47.09 s (41 s to solve the MILP and 6.09 s to generate the motion plans) to find the first feasible solution using the travel regularized robustness ($\rho = 0.00051$).

X. Conclusion

In this article, we had developed a framework for scalable and robust deployment of teams of heterogeneous agents. ScRATCHeS was able to build plans based on rich, temporal logic specifications involving tasks that require the participation of multiple capabilities, e.g., sensing modalities, distributed across the team of agents. This framework encoded the planning problem as a large mixed integer linear program, which could be efficiently solved using modern commercial off-the-shelf solvers. We validated this method using a series of randomized computational experiments, which showed the potential scalability of the method, and via a hardware demonstration, which illustrated the potential implementability and applicability of our approach.

Our initial results indicated the possibility to develop real-time planning tools to allow supervisors in charge of large teams of heterogeneous robots to task them by giving intent rather than by manually planning paths or scheduling tasks. The gap between the time to compute first feasible solutions and the time to compute optimal solutions for this case indicated that an "any-time" planner in which iteratively improved solutions were presented to the supervisor may be useful. This also indicated that there was potential for improving the performance by smoothing the availability robustness measure, i.e., by including "partial credit" for cases in which some but not all tasks have excess agents available.

In the future, we will work to extend this approach to be more reactive so as to reduce the load on the supervisor. That is, in addition to generating plans that are robust to attrition, we will equip the team with a monitor to keep track of progress to plans and reactive synthesis techniques to alter the plan on-the-fly when attrition or delays occur. Finally, we will also investigate parallelization by breaking the team into subteams

and equipping each subteam with its own monitor and plan, thus allowing subteams to operate quasi-independently and reduce the amount of communication required to execute and alter the plan.

APPENDIX A
PROOFS OF PROPOSITIONS

A. Proof of Proposition 1

Proposition: CaTL is a proper fragment of signal temporal logic (STL) [28].

Proof: (CaTL $\subseteq$ STL) CaTL is a fragment of STL that requires all predicates be part of a task T. For any given task $T = (d, \pi, \{cp_i\}_{i \in \mathcal{I}_T}))$, define an STL formula

$$\phi(T) = \bigwedge_{i \in \mathcal{I}_T} \Box_{[0,d)} \left(\bigwedge_{q \in L^{-1}(\pi)} n_{q,c_i}(t) \geq m_i \right) .$$

By applying the recursive semantics of STL, we have that

$$(s_J, t) \models \phi(T) \Leftrightarrow \begin{array}{l} \forall \tau \in [t, t+d) \\ \forall q \in L^{-1}(\pi) \\ \forall \{cp_i = (c_i, m_i)\}_i \in \mathcal{I}_T \\ n_{q,c_i}(\tau) \geq m_i \end{array}$$

which is exactly the same condition for $(s_J, t) \models T$. Therefore,$(s_J, t) \models T \Leftrightarrow (s_J, t) \models \phi(T)$ where $\phi(T)$ is the STL formula defined above. The remainder of syntax and semantics for CaTL is included in STL. Thus, any CaTL formula has an equivalent STL formula.

($STL \not\subseteq CaTL$) The formula $\phi_s = \Diamond_{[a,b)}(s \leq 0)$ is an STL formula but not a CaTL formula.

B. Proof of Proposition 2

Proposition: Applying the recursive quantitative semantics listed in Definition 10 yields ρ as defined in Definition 9.

Proof: We reason about the robustness ρ in terms of subtracting and adding agents to a team. By showing how the addition or subtraction of agents affects the satisfaction of one or more formulas, we can prove the proposition. To prove the proposition for each task and operator in CaTL, we consider the case in which $\rho(s_J, t, T) \geq 0$ and the case in which $\rho(s_J, t, T) < 0$.

We prove the proposition by structural induction. First, we demonstrate the base case corresponding the atomic unit of CaTL, i.e., we show that ρ is equivalent to Definition 9 for tasks. Then, using the induction hypothesis, we prove the equivalence between the definitions for each of the operators in the language. Specifically, we show that if ρ is equivalent to Definition 9 for the operands, then ρ of the overall formula is equivalent to Definition 9.

Task (T): Here, we demonstrate that ρ is equivalent to Definition 9 for a CaTL task T. Let

$$\delta_{i,s_J,t,T} := \min_{t' \in [t,t+d)} \min_{q \in L^{-1}(\pi)} \sum_{\{g | c_i \in g\}} n_{q,g}(t') - m_i$$

denote the excess agents with capability c_i at time t for completing task T from the team trajectory s_J. The excess $\delta_{i,s_J,t,T}$ may be negative, in which case it denotes the number of missing agents with capability c_i needed at time t to satisfy task T. Then

$$(s_J, t) \models \phi \Leftrightarrow \delta_{i,s_J,t,T} \geq 0, \ \forall i \in \mathcal{I}_T \ .$$

By definition, $\rho(s_J, t, T) = \min_{i \in \mathcal{I}_T} \delta_{i,s_J,t,T}$ is the least number of agents in excess with any capability involved in task T. Let $i^* = \arg\min_{i \in \mathcal{I}_T} \delta_{i,s_J,t,T}$ be the index of the minimizing capability.

We begin with the case that $(s_J, t) \models T$, or equivalently $\rho(s_J, t, T) \geq 0$ and $\delta_{i^*,s_J,t,T} \geq 0$. Removing an agent j from J decrements all excess values $\{\delta_{i,s_J,t,T}\}_{c_i \in g_j}$ associated with the capabilities g_j of agent j. Thus, in the worst case, removing any agent with all capabilities $c_i \in C$ decrements all $\{\delta_{i,S_J,t,T}\}_{i \in \mathcal{I}_T}$. Let J_R contain $\rho(s_J, t, T)$ agents with all capabilities $c_i \in Cap$, and let $J_{R'}$ contain $\rho(s_J, t, T) + 1$ such agents. Then, $\delta_{i^*,s_{J \setminus J_R},t,T} = 0 \Rightarrow (s_{J \setminus J_R}, t) \models T$ and $\delta_{i^*,s_{J \setminus J_{R'}},t,T} = -1 \Rightarrow (s_{J \setminus J_{R'}}, t) \not\models T$. Thus, $\rho(s_J, t, T) = \min |J_R|$ such that $(s_{J \setminus J_R}, t) \models T$.

To prove the case in which $(s_J, t) \not\models T$, or equivalently $\rho(s_J, t, T) <$ and $\delta_{i^*,s_J,t,T} < 0$, the same procedure as above is followed. Instead of decrementing $\{\delta_{i,S_J,t,T}\}_{i \in \mathcal{I}_T}$, we increment it instead.

Now we consider the remaining quantitative semantics outlined in (6).

Conjunction ($\wedge$):: Here, we demonstrate that subtracting (adding) ρ agents from the less robust of two formulas does not affect satisfaction (nonsatisfaction) of the conjunction of the two formulas. Without loss of generality, let

$$\rho(s_J, t, \phi_1) \leq \rho(s_J, t, \phi_2)$$

i.e.,

$$\rho(s_J, t, \phi_1) = \min(\rho(s_J, t, \phi_1), \rho(s_J, t, \phi_2)) \ .$$

If $\rho(s_J, t, \phi_1) \geq 0$, then we can remove any $\rho(s_J, t, \phi_1)$ agents and both $(s_J, t) \models \phi_1$ and $(s_J, t) \models \phi_2$. If $\rho(s_J, t, \phi_1) < 0$, then at least $\rho(s_J, t, \phi_1)$ agents must be added in order for $s_J \models \phi_1$. If $\rho(s_J, t, \phi_2) \geq 0$, adding $\rho(s_J, t, \phi_1)$ will maintain $(s_J, t) \models \phi_2$. If $\rho(s_J, t, \phi_2) < 0$, then by adding $\rho(s_J, t, \phi_1)$ agents with all capabilities, $(s_J, t) \models \phi_2$.

Disjunction ($\vee$): Here, we demonstrate that subtracting (adding) ρ agents from the more robust of two formulas does not affect satisfaction (nonsatisfaction) of the disjunction of the two formulas. Without loss of generality, let

$$\rho(s_J, t, \phi_1) \geq \rho(s_J, t, \phi_2)$$

i.e.,

$$\rho(s_J, t, \phi_1) = \max(\rho(s_J, t, \phi_1), \rho(s_J, t, \phi_2)) \ .$$

If $\rho(s_J, t, \phi_1) > 0$ and $\rho(s_J, t, \phi_1)$ agents are removed, then $(s_J, t) \models \phi_1 \Rightarrow (s_J, t) \models \phi_1 \vee \phi_2$. If $\rho(s_J, t, \phi_1) \leq 0$, then if $\rho(s_J, t, \phi_1)$ agents are added, again, $(s_J, t) \models \phi_1 \Rightarrow (s_J, t) \models \phi_1 \vee \phi_2$.

Until ($\mathcal{U}_{[a,b)}$): For the until operator, ϕ_1 and ϕ_2 are ordered. Therefore, we must examine four cases. In the first two cases, ϕ_1 is more robust than ϕ_2, for both the satisfying on nonsatisfying condition. The next two cases are those in which ϕ_2 is more robust than ϕ_1, for both the satisfying and nonsatisfying cases.

For $\phi_1 \mathcal{U}_{[a,b)} \phi_2$ to be true, ϕ_2 must be true at some time $t' \in [t+a, t+b)$, and ϕ_1 must be true for all $t'' \in [t, t')$. Our proof therefore relies on defining robustness in terms of t' and t'', depending on whether satisfaction of ϕ_1 or ϕ_2 is more robust.

We begin with the two cases in which ϕ_1 is more robust than ϕ_2. Let

$$t^* = \underset{t' \in [t+a,t+b)}{\arg\max} \ \min\{\rho(s_J, t,' \phi_2), \underset{t'' \in [t,t')}{\min} \rho(s_J, t,'' \phi_1)\}$$

and let

$$0 \leq \rho(s_J, t^*, \phi_2) < \underset{t'' \in [t,t^*)}{\min} \rho(s_J, t,'' \phi_1) \ .$$

Then, we can remove any $\rho(s_J, t^*, \phi_2)$ agents at time t and $(s_J, t^*) \models \phi_2$. Since

$$\rho(s_J, t^*, \phi_2) < \underset{t'' \in [t,t^*)}{\min} \rho(s_J, t,'' \phi_1)$$

removing $\rho(s_J, t^*, \phi_2)$ means that $s(_J, t'') \models \phi_1 \forall t'' \in [t, t^*)$, and thus, $(s_J, t) \models \phi_1 \mathcal{U}_{[a,b)} \phi_2$. If $\rho(s_J, t^*, \phi_2) < 0$, then adding $-\rho(s_J, t^*, \phi_2)$ at time t will ensure $(s_J, t^*) \models \phi_2$ and, since

$$\rho(s_J, t^*, \phi_2) < \underset{t'' \in [t,t^*)}{\min} \rho(s_J, t,'' \phi_1)$$

also ensures $s(_J, t'') \models \phi_1 \forall t'' \in [t, t^*)$ and thus $(s_J, t) \models \phi_1 \mathcal{U}_{[a,b)} \phi_2$.

Now, we examine the cases in which ϕ_2 is more robust than ϕ_1. Let

$$t^{**} = \underset{t'' \in [t,t^*)}{\arg\min} \rho(s_J, t,'' \phi_1)$$

and let

$$0 \leq \underset{t'' \in [t,t^*)}{\min} \rho(s_J, t,'' \phi_1) < \rho(s_J, t^*, \phi_2) \ .$$

If we remove $\rho(s_J, t^{**}, \phi_1)$ agents at time t, then $(s_j, t'') \models \phi_1 \forall t'' \in t, t^*$ and $(s_J, t^*) \models \phi_2$, as $\rho(s_J, t^{**}, \phi) > 0$ and thus $(s_J, t) \models \phi_1 \mathcal{U}_{[a,b,)} \phi_2$. Finally, if

$$\underset{t'' \in [t,t^*)}{\min} \rho(s_J, t,'' \phi_1) < 0$$

then if we add $-\rho(s_J, t^{**}, \phi_1)$ agents at time t means that $\rho(s_J, t^{**}, \phi_1) \geq 0 \forall t'' \in [t, t^*)$, i.e., $(s_J, t) \models \phi_1 \forall t'' \in [t, t^*)$. Further, this implies $\rho(s_J, t^*, \phi_2) \geq 0$, i.e., $(s_J, t^*) \models \phi_2$. Again in this final case, $(s_J, t) \models \phi_1 \mathcal{U}_{[a,b)} \phi_2$.

Eventually ($\Diamond_{[a,b)}$): For the eventually operator, a trajectory must satisfy the formula for at least one time step in the interval $[a, b)$. Therefore, we may determine the maximally satisfying time step in order to compute robustness. Let

$$t^* = \underset{t' \in [t+a,t+b)}{\arg\max} \ \rho(s_J, t,' \phi) \ .$$

If $\rho(s_J, t^*, \phi) > 0$, then we can remove any $\rho(s_J, t^*, \phi)$ agents and still $(s_J, t^*) \models \phi$, and therefore $(s_J, t) \models \Diamond_{[a,b)} \phi$. If $\rho(s_J, t^*, \phi) \leq 0$, then we must add $\rho(s_J, t^*, \phi)$ so that $(s_J, t^*) \models \phi$, and therefore $(s_J, t) \models \Diamond_{[a,b)} \phi$.

Always ($\Box_{[a,b)}$): For the always operator, a trajectory must satisfy the formula for all time steps in the interval $[a, b)$. Therefore, we must determine the least satisfying time step in order to compute robustness. Let

$$t^* = \underset{t' \in [t+a,t+b)}{\arg\min} \ \rho(s_J, t,' \phi) \ .$$

If $\rho(s_J, t^*, \phi) > 0$, then we can remove any $\rho(s_J, t^*, \phi)$ agents and still $(s_J, t') \models \phi \ \forall t' \in [t+a, t+b)$, and therefore $(s_J, t) \models \Box_{[a,b)} \phi$. Likewise, if $\rho(s_J, t^*, \phi) \leq 0$, then we must add $\rho(s_J, t^*, \phi)$ agents such that $(s_J, t^*) \models \phi \Rightarrow (s_J, t) \models \Box_{[a,b)} \phi$.

C. Proof of Proposition 3

Proposition: Under Assumption 1, a team input signal $u = [u_j]_{j \in J}$ and the induced team trajectory s_J are valid, i.e., conform to Definitions 3 and 4, only if a set of variables a set of variables

$$\{z_{q,g}(k)\}_{q \in Q, g \in G, k=0,\ldots K} \cup \{u_{e,g}(k)\}_{e \in E, g \in G, k=0,\ldots K}$$

satisfy constraints (8), (9).

Proof: Under Assumption 1

$$n_{q,g}(t) = n_{q,g}(\lfloor \frac{t}{\delta t} \rfloor \delta t) \forall t \neq k\delta t, k \in \mathbb{N}$$

$$n_{q,e}(t) = n_{q,e}(\lfloor \frac{t}{\delta t} \rfloor \delta t) \forall t \neq k\delta t, k \in \mathbb{N}. \tag{23}$$

Thus, a trajectory s_J such that $s_J(t) = [n_{q,g}(t), n_{e,g}(t)]_{q \in Q, e \in E}$ is uniquely specified by the sequences $\{[n_{q,g}(k\delta t), n_{e,g}(k\delta t)]_{q \in Q, e \in E}\}_{k=0}^{K}$.

Substituting $z_{q,g}(k)$ into the requirements

$$s_j(t) = q_{0,j}$$

$$n_{g,q}(0) = \sum_{j \in J} I(s_j(0) = q) I(Cap_j = g) \tag{24}$$

from Definitions 3 and 4 yield (8).

The requirements from Definitions 3 and 4 that $u_j(t) = (q, q') \Rightarrow s_j(t) = q \wedge s_j(t') = (q_1, q_2) \forall t' \in (t, t + W(e)) \wedge s_j(t + W(q, q')) = q'$ can be stated as

$$\begin{aligned} n_{q,g}(k\delta t) &= n_{q,g}((k-1)\delta t) \\ &\quad + \sum_{(q,'q)} I(u_j((k - \mathcal{W}(q,' q))\delta t) \\ &= (q,' q)) I(g = Cap_j) \\ &\quad - \sum_{(q,q'')} I(u_j(k\delta t) = (q,' q)) I(g = Cap_j) \\ n_{e,g}(k\delta t) &= -I(u_j((k-\ell)\delta t) = (q,' q)) I(g = Cap_j) \\ &\quad + \sum_{\ell = -\mathcal{W}(e)+1}^{0} I(u_j((k-\ell)\delta t) \\ &= (q,' q)) I(g = Cap_j). \end{aligned} \tag{25}$$

Substituting in $z_{q,g}(k)$ and $u_{e,g}(k)$ yields

$$z_{q,g}(k) = z_{q,g}(k-1) + \sum_{(q,'q)} u_{(q,'q),g}(k - \mathcal{W})$$

$$- \sum_{(q,q'')} u_{(q,q''),g}(k) \tag{26a}$$

$$n_{e,g}(k\delta t) = -u_{e,q}(k) + \sum_{\ell=-\mathcal{W}(e)+1}^{0} u_{e,q}(\ell). \tag{26b}$$

Now, if we add into our formulation a set of artificial edges $(q,q)\ \forall q \in Q$ with weight $W((q,q)) = \delta t\ \forall q \in Q$, then

$$z_{q,g}(k-1) - \sum_{(q,q'')} u_{(q,q''),g}(k) = \sum_{(q,q)} u_{(q,q),g}(k - \mathcal{W}(q,q)). \tag{27}$$

Substituting this expression into (27) yields (9a). Further, the addition of the artificial weights means that at every time $k\delta t$, every agent j in a node $q \in Q$ has to choose an edge to traverse along. In order to reflect the fact that the set of agents in the environment remains constant, we require that the number of agents flowing into a region must be the same as the number of agents flowing out of the region. Thus, we can replace (26b) with the equivalent flow conservation condition (9a).

Therefore, all valid state trajectories can be recovered from a set of variables $\{z_{q,g}(k)\}_{q\in Q, g\in G, k=0,\dots K} \cup \{u_{e,g}(k)\}_{e\in E, g\in G, k=0,\dots K}$ that satisfy (8) and (9).

D. Proof of Proposition 4

Proposition: Every CaTL formula in PNF is equivalent to an STL formula in PNF.

Proof: If a CaTL formula ϕ is in PNF, then the only possibility for the equivalent STL formula not to be in PNF is if any task in the formula contains a negation when translated to STL. Since the equivalent STL formula for any task T, $\phi(T) = \bigwedge_{i\in\mathcal{I}_T} \Box_{[0,d)}(\bigwedge_{q\in L^{-1}(\pi)} n_{q,c_i}(t) \geq m_i)$, does not contain a negation, the equivalent STL formula for a CaTL formula in PNF is in PNF.

E. Proof of Proposition 5

Proposition: Let $f(u,z) = r_{0,\phi}(u,z) - \gamma\tau(u,z)$. If $r_{0,\phi}(u_1,z_1) > r_{0,\phi}(u_2,z_2)$, then $f(u_1,z_1) > f(u_2,z_2)$.

Proof: $r_{0,\phi}$ is integer-valued, so $r_{0,\phi}(u_1,z_1) > r_{0,\phi}(u_2,z_2) \Rightarrow r_{0,\phi}(u_1,z_1) - r_{0,\phi}(u_2,z_2) \geq 1$. The value $\tau(u,z)$ is maximized at $\tau_{\max} = |J|K$, i.e., every agent is always traveling and minimized when $\tau_{min} = 0$, i.e., all agents are always idle. Therefore

$$\begin{aligned}\gamma\tau(u_1,z_1) - \gamma\tau(u_2,z_2) &\leq \gamma(\tau_{\max} - \tau_{\min}) \\ &= \alpha\tfrac{|J|K}{|J|K} \\ &= \alpha.\end{aligned}$$

Thus

$$\begin{aligned}f(u_1,z_1) - f(u_2,z_2) &= (r_{0,a,\phi} - r_{0,a,\phi}) \\ &\quad +(\gamma\tau(u_2,z_2) - \gamma\tau(u_1,z_1) \\ &\geq 1-\alpha \\ &> 0\end{aligned}$$

since $\alpha < 1$. Equivalently, $f(u_1,z_1) > f(u_2,z_2)$, thus, establishing the proposition.

F. Proof of Proposition 7

Proposition: The capability excess $ce(\{A_j\}_{j\in J}, \phi)$ is an upper bound for $\max_{\{u'_{e,g,k}\}} r_{0,\phi}$.

Proof: The robustness $\max_{\{u'_{e,g,k}\}} r_{0,\phi}$ would reach its theoretical maximum if there were no motion constraints and all agents are available to service tasks at any time, i.e., if $\mathcal{W}(q,q') = 0 \forall (q,q') \in E$. Denote the team signal resulting from these conditions as $s_{0,J}$. We now prove by induction that $ce(\{A_j\}_{j\in J}, \phi) \geq \rho(s_{0,J}, t, \phi)$

Base case: As $\rho(s_{0,J}, t, T)$ is defined as a minimum over capabilities and over regions, it would reach its maximum if all of the agents with required capabilities $\{c_i\}_{i\in\mathcal{I}_T}$ were equally divided among the regions $q \in L^{-1}(\pi)$ at the appropriate time. This maximum value is given by $ce(\{A_j\}_{j\in J}, T)$, i.e., $ce(\{A_j\}_{j\in J}, T) \geq \rho(s_{0,J}, t, T) \forall t$.

Recursion: For $\wedge$ and $\vee$, we apply the same recursive relationships to ce as we do to ρ. Therefore, if $ce(\{A_j\}_{j\in J}, \phi_d), d \in \{1,2\}$ are upper bounds, then $ce(\{A_j\}_{j\in J}, \phi_1 \cdot \phi_2) \geq \rho(s_{0,J}, t, \phi_1 \cdot \phi_2) \forall t, \cdot \in \{\vee, \wedge\}$.

For the temporal operators, since $s_{0,J}$ considers the case when we are not motion or time-constrained, we can ignore the maximization and minimization with respect to temporal arguments in the recursive semantics. This yields the form of the recursive relations in (19). Therefore, if $ce(\{A_j\}_{j\in J}, \phi_d), d \in \{1,2\}$ is an upper bound, then $ce(\{A_j\}_{j\in J}, \phi_1 \mathcal{U}_{[a,b)} \phi_2) > \rho(s_{0,J}, t, \phi_1 \mathcal{U}_{[a,b)} \phi_2), \forall t$ and $ce(\{A_j\}_{j\in J}, \sim \phi) > \rho(s_{0,J}, t, \sim \phi), \forall t, \sim \in \{\Diamond_{[a,b)}, \Box_{[a,b)}\}$.

Therefore, $ce(\{A_j\}_{j\in J}, \phi) \geq \rho(s_{0,J}, 0, \phi) \forall \phi \in CaTL$.

Appendix B
Detailed Tables of Results

Tables IX–XI present the results of Experiment 2, varying the size of the environment, number of agents, and classes of agents, respectively.

Appendix C
Motion Planning

Individual motion plans are initially generated for each agent as region level trajectories from the output of the mixed integer linear program. These trajectories are found using an assignment algorithm shown in Algorithm 1. Once the region level trajectories are determined from the MILP solution (see Section XIII-A), a local planner can be used for each agent to determine trajectories between and within regions. Specific points within a region are determined *a priori* or chosen at random from the free workspace (see Section XIII-B). Agents are assigned to these locations based first on sensor requirements for the task, and then greedily based on distance for agents within each region (if applicable). Once goal locations are determined in the workspace, a multiagent RRT planner that uses a sequential planning method determines continuous space agent trajectories (see Section XIII-C). These trajectories are designed to not be in collision with other agents or objects in the environment.

A. Region Level Agent Trajectory Generation

The output of the MILP encoding does not specify individual trajectories, but instead numbers of agents of each type required in a region (or transitioning between regions) at each time. We denote the set of these outputs from the MILP as $Z_{Q,G,K}$ and $U_{E,G,K}$, respectively. The set of time steps appearing in the MILP solution is denoted K. An assignment matching algorithm (see Algorithm 1) is employed to take these agent requirements at each time, the transition system of the environment, the current agent positions, and required capabilities and generate a region level trajectory for each agent.

We begin by introducing a region-level agent trajectory as a sequence of states or edges $\sigma_j = \{q_{0,j}\} \dots \{q, q'\} \dots \{q^{t_{end}}\}$, where $j \in J$ is the agent index and $[k]$ represents the discrete time index in the trajectory. Each trajectory is initialized to q_0 from A_j and indicates if an agent is in a region or transitioning between two regions (line 1). We introduce a function $\texttt{occupied}(\sigma_j, k)$ that returns whether agent j is already assigned to a state/transition at time $[k]$. The function returns False if it is unassigned at that time. For each time step $[k]$, we take the state and transition decision variables $z_{q,g}(k)$ and $u_{(q,q'),g}(k)$ from the MILP solution (lines 4 and 12). For each state variable greater than zero ($z_{q,g}(k) > 0$), we find agents that are currently in region q, have capability g, and who are unoccupied (line 8). There are always sufficient agents $j \in J$, because the MILP solves for this condition. Therefore, we assign the agent(s) of lowest index (i.e. $\min(j)$) to that region at that time (i.e., $\sigma_j[k] = q$) in the quantity of $z_{q,g}(k)$ (line 9). For each edge variable greater than zero ($u_{(q,q'),g}(k) > 0$), we again find agents that are currently in region q, have capability g, and who are unoccupied (line 16). This condition can always be satisfied, because it is satisfied in the MILP solution. We assign the agent(s) of lowest index (i.e., $\min(j)$) to that transition at that time (i.e. $\sigma_j[k] = \{q, q'\}$) in the quantity of $z_{q,g}(k)$ (line 17). Unlike state assignments however, we assign this transition value from time k to $k + W(e)$ and at time $k + W(e)$ we assign the agent(s) to region q'. This is run for each time step, generating the discrete, region-level, trajectory of each agent. This trajectory is then used to generate a continuous space trajectory.

B. Region Level Task Allocation

Before generating a continuous space trajectory, agent positions for each task must be allocated from discrete trajectories. The assignment algorithm finds the discrete region level trajectories, however these must be related to specific points in space depending on the requirements of the task. These points are defined as a tuple of capability and position. From the individual agent region level trajectories, the first agent to enter a region where its capability can be employed will be assigned to that capability's predetermined position. If another agent enters the region with the same capability set, and is not needed for the task, that agent is diverted to a random location in the region that also does not obstruct other agents. This process occurs at each time step.

Algorithm 1: MILP Solution Region-Level Assignment.

Input: $Z_{Q,G,K}; U_{E,G,K}; W; A_J; K; G; Q$
Output: $\sigma_j \forall j \in J$

1: $\sigma_j[0] \leftarrow q_{0,j}\ \forall j \in J$
2: **for** $k = 1$ to K **do**
3: **for** $(g, q) \in G \times Q$ **do**
4: **if** $z_{q,g}(k) > 0$ **then**
5: $count \leftarrow 0$
6: **while** $count < z_{q,g}(k)$ **do**
7: **for** $j \in J$ **do**
8: **if** $\sigma_j[k-1] = q$ **and** $\neg\ \texttt{occupied}(\sigma_j, k)$ **and** $g \in A_j$ **and** $count < z_{q,g}(k)$ **then**
9: $\sigma_j[k] \leftarrow \{q\}$
10: $count++$
11: **for** $q' \in Q$ **do**
12: **if** $u_{(q,q'),g}(k) > 0$ **then**
13: $count \leftarrow 0$
14: **while** $count < u_{(q,q'),g}(k)$ **do**
15: **for** $j \in J$ **do**
16: **if** $\sigma_j[k-1] = q$ **and** $\neg\ \texttt{occupied}(\sigma_j, k)$ **and** $g \in A_j$ **and** $count < u_{(q,q'),g}(k)$ **then**
17: $\sigma_j[k : k + W(q, q')] \leftarrow \{q, q'\}$
18: $count++$
19: **return** $\sigma_j \forall j \in J$

C. Ranked Multiagent RRT

To generate continuous trajectories for each agent, a variant of the rapidly-exploring random trees (RRT) algorithm is used. This variant considers both obstacles and previously planned agents in planning trajectories. To do this, it ranks the robots randomly and plans each trajectory sequentially by decreasing rank. As each node is created, it inherits a time step value from its parent node that is incremented. This time step value is used to determine if a node is in collision with another trajectory at a given time. When checking for such collisions, the distance to each prior computed trajectory node at that time is determined, and the node is discarded if it is within a safety distance to any of these prior nodes. Beyond this, a randomization structure is used to get samples from the configuration space that create more efficient trajectories (the sample distribution is a hybrid between a Gaussian around the goal and a uniform distribution). If the number of consecutively failed nodes exceeds a threshold, the algorithm continues but then uses only a uniform sampling distribution. This protects against the Gaussian distribution trying to exploit rather than explore the workspace. See Fig. 8 for an example with 10 agents in a cluttered environment.

Remark 2: The RRT algorithm we use will converge for point robots. It is possible that the physical extent of the robots can cause the RRT to fail to converge. Such a case would occur if robots are large relative to the size of the regions in the environment. In that case, the abstraction can be refined, or limits can be placed on the number of agents that may simultaneously occupy a region in the MILP. In practice, we did not encounter the need for such modifications.

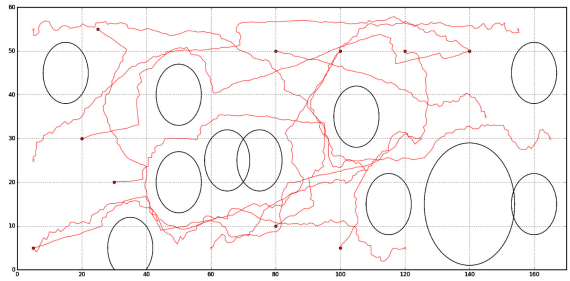


Fig. 8. Sample trajectories from ranked multiagent RRT algorithm—start location is on one side, and the goal region is in the opposite quadrant. Circles are obstacle regions, and each agent considers the trajectories off all agents calculated before it as moving obstacles.

ACKNOWLEDGMENT

Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Under Secretary of Defense for Research and Engineering.

REFERENCES

[1] J. Cortes and M. Egerstedt, "Coordinated control of multi-robot systems: A survey," *SICE J. Control, Meas. Syst. Integr.*, vol. 10, no. 6, pp. 495–503, 2017.

[2] B. Schlotfeldt, D. Thakur, N. Atanasov, V. Kumar, and G. J. Pappas, "Anytime planning for decentralized multirobot active information gathering," *IEEE Robot. Autom. Lett.*, vol. 3, no. 2, pp. 1025–1032, Apr. 2018.

[3] R. Simmons *et al.*, "Coordinated deployment of multiple, heterogeneous robots," in *Proc. 2000 IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS 2000)(Cat. No. 00CH37113)*, vol. 3, 2000, pp. 2254–2260.

[4] J. Kiener and O. Von Stryk, "*Cooperation of heterogeneous, autonomous robots: A case study of humanoid and wheeled robots*" in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2007, pp. 959–964.

[5] A. Prorok, M. A. Hsieh, and V. Kumar, "Fast redistribution of a swarm of heterogeneous robots," in *Proc. 9th EAI Int. Conf. Bio-inspired Inf. Commun. Technol.*, 2016, pp. 249–255.

[6] H.-L. Choi, L. Brunet, and J. P. How, "Consensus-based decentralized auctions for robust task allocation," *IEEE Trans. Robot.*, vol. 25, no. 4, pp. 912–926, Aug. 2009.

[7] H. D. Mittelmann, "Selected benchmark results," in *INFORMS Annual Meeting*, 2016. [Online]. Available: http://plato.asu.edu/talks/informs2016-bench.pdf

[8] V. Tjeng, K. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," in *Proc. Int. Conf. Learn. Representations*, 2018.

[9] J. Karlsson, C.-I. Vasile, J. Tumova, S. Karaman, and D. Rus, "Multi-vehicle motion planning for social optimal mobility-on-demand," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 7298–7305.

[10] H. D. Mittelmann, "The state-of-the-art in optimization software," in *Proc. 22nd Int. Symp. Math. Program.*, 2015. [Online]. Available: http://plato.asu.edu/talks/ismp2015.pdf

[11] A. Jones *et al.*, "ScRATCHS: Scalable and robust algorithms for task-based coordination from high-level specifications," in *Proc. Int. Symp. Robot. Res.. Int. Federation Robot. Res.*, 2019.

[12] C. Baier and J.-P. Katoen, *Principles of Model Checking*. Cambridge, MA, USA: MIT Press, 2008.

[13] C. Belta, B. Yordanov, and E. A. Gol, *Formal Methods for Discrete-Time Dynamical Systems*, vol. 89. Berlin, Germany: Springer, 2017.

[14] M. Guo and D. V. Dimarogonas, "Multi-agent plan reconfiguration under local LTL specifications," *Int. J. Robot. Res.*, vol. 34, no. 2, pp. 218–235, 2015. [Online]. Available: https://doi.org/10.1177/0278364914546174

[15] M. Guo and D. V. Dimarogonas, "Task and motion coordination for heterogeneous multiagent systems with loosely coupled local tasks," *IEEE Trans. Autom. Sci. Eng.*, vol. 14, no. 2, pp. 797–808, Apr. 2017.

[16] Y. Diaz-Mercado, A. Jones, C. Belta, and M. Egerstedt, "Correct-by-construction control synthesis for multi-robot mixing," in *Proc. 54th IEEE Conf. Decis. Control*, 2015, pp. 221–226.

[17] Y. V. Pant, H. Abbas, R. A. Quaye, and R. Mangharam, "Fly-by-logic: Control of multi-drone fleets with temporal logic objectives," in *Proc. ACM/IEEE Int. Conf. Cyber- Phys. Syst.*, 2018, pp. 186–197.

[18] P. Schillinger, M. Bürger, and D. V. Dimarogonas, "Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems," *Int. J. Robot. Res.*, vol. 37, no. 7, pp. 818–838, 2018.

[19] Y. Chen, X. C. Ding, A. Stefanescu, and C. Belta, "Formal approach to the deployment of distributed robotic teams," *IEEE Trans. Robot.*, vol. 28, no. 1, pp. 158–171, Feb. 2012.

[20] K. Leahy, A. Jones, M. Schwager, and C. Belta, "Distributed information gathering policies under temporal logic constraints," in *Proc. 54th IEEE Conf. Decis. Control*, 2015, pp. 6803–6808.

[21] Y. Kantaros and M. M. Zavlanos, "Stylus*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems," *Int. J. Robot. Res.*, vol. 39, no. 7, pp. 812–836, 2020.

[22] X. Luo, Y. Kantaros, and M. M. Zavlanos, "An abstraction-free method for multirobot temporal logic optimal control synthesis," *IEEE Trans. Robot.*, vol. 37, no. 5, pp. 1487–1507, Oct. 2021.

[23] S. Karaman and E. Frazzoli, "Vehicle routing problem with metric temporal logic specifications," in *Proc. 47th IEEE Conf. Decis. Control*, 2008, pp. 3953–3958.

[24] I. Haghighi, S. Sadraddini, and C. Belta, "Robotic swarm control from spatio-temporal specifications," in *Proc. IEEE 55th Conf. Decis. Control*, 2016, pp. 5708–5713.

[25] Y. E. Sahin, P. Nilsson, and N. Ozay, "Provably-correct coordination of large collections of agents with counting temporal logic constraints," in *Proc. ACM/IEEE 8th Int. Conf. Cyber-Physical Syst.*, 2017, pp. 249–258.

[26] Y. Sahin, P. Nilsson, and N. Ozay, "Multirobot coordination with counting temporal logics," *IEEE Trans. Robot.*, vol. 36, no. 4, pp. 1189–1206, Aug. 2020.

[27] Z. Xu and A. A. Julius, "Census signal temporal logic inference for multiagent group behavior analysis," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 1, pp. 264–277, Jan. 2018.

[28] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Proc. Int. Symp. Formal Techn. Model. Anal. Timed Fault-Tolerant Syst.*, 2004, pp. 152–166.

[29] P. Linz, *An Introduction to Formal Languages and Automata*. Burlington, MA, USA: Jones & Bartlett Learning, 2006.

[30] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals," in *Proc. Int. Conf. Formal Model. Anal. Timed Syst.*, 2010, pp. 92–106.

[31] G. E. Fainekos and G. J. Pappas, "Robustness of temporal logic specifications for continuous-time signals," *Theor. Comput. Sci.*, vol. 410, no. 42, pp. 4262–4291, 2009.

[32] V. Raman, A. Donzé, M. Maasoumy, R. M. Murray, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Model predictive control with signal temporal logic specifications," in *Proc. IEEE 53rd Annu. Conf. Decis. Control*, 2014, pp. 81–87.

[33] S. Sadraddini and C. Belta, "Robust temporal logic model predictive control," in *Proc. 53rd Annu. Allerton Conf. Commun. Control Comput.*, 2015, pp. 772–779.

[34] S. Sadraddini, "Formal methods for resilient control," Ph.D. dissertation, Dept. Mech. Eng., Boston Univ., Boston, MA, USA, 2018. [Online]. Available: https://open.bu.edu/handle/2144/27455

[35] A. Mahajan, "Presolving mixed-integer linear programs," in *Wiley Encyclopedia of Operations Research and Management Science*. Hoboken, NJ, USA: Wiley, 2010, Art. no. 44.

[36] L. Gurobi Optimization, "Gurobi optimizer reference manual," 2020. [Online]. Available: http://www.gurobi.com

[37] Stanford Artificial Intelligence Laboratory, "Robotic Operating System," *ROS Melodic Morenia*, May 2018, https://www.ros.org.

[38] D. Connell and H. M. La, "Extended rapidly exploring random tree-based dynamic path planning and replanning for mobile robots," *Int. J. Adv. Robotic Syst.*, vol. 15, no. 3, 2018, Art. no. 1729881418773874.

[39] J. Preiss, W. Hönig, G. Sukhatme, and N. Ayanian, "*Crazyswarm: A large nano-quadcopter swarm*," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 3299–3304.

[40] L. Meier, D. Honegger, and M. Pollefeys, "PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, pp. 6235–6240.

[41] J. Kim, C. J. Banks, and J. A. Shah, "Collaborative planning with encoding of users' high-level strategies," in *Proc. 31st AAAI Conf. Artif. Intell.*, 2017, pp. 955–961.

Kevin Leahy (Member, IEEE) received the B.A. degree in economics in 2009 and the M.S. and Ph.D. degrees in mechanical engineering in 2017 from Boston University, Boston, MA, USA.

He is a Technical Staff Member with the Artificial Intelligence Technology Group, MIT Lincoln Laboratory, Lexington, MA, USA. His current work involves AI for autonomous systems, with an emphasis on formal methods and multiagent systems. For his dissertation, he studied correct-by-construction techniques for multiagent persistent surveillance and information gathering.

Zachary Serlin received the B.S. and M.S. degrees in 2015 and 2016, respectively, from Tufts University, Medford, MA, USA, and the Ph.D. degree from Boston University, Boston, MA, USA, in 2020, all in mechanical engineering.

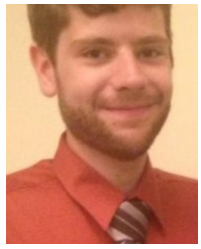
He is a Technical Staff Member with MIT Lincoln Laboratory, Lexington, MA, USA. His research interests include algorithms for decision-making, motion planning, and control for autonomous systems, formal methods, distributed multirobot systems, distributed computer vision algorithms, safe reinforcement learning techniques, and tendon-driven bioinspired soft robotics.

Cristian-Ioan Vasile (Member, IEEE) received the B.Sc. degree in computer science, M.Eng. degree in control and systems engineering, and Ph.D. degree in systems engineering from the Politehnica University of Bucharest, Bucharest, Romania in 2009, 2011, and 2015, respectively, and the Ph.D. degree in systems engineering from Boston University, Boston, MA, USA in 2016.

He was a Postdoctoral Associate with the Laboratory for Information and Decision Systems (LIDS), and the Computer Science and Artificial Intelligence Laboratory (CSAIL), Massachusetts Institute of Technology (MIT), Cambridge, MA, USA. He is an Assistant Professor with the Mechanical Engineering and Mechanics Department, Lehigh University, Bethlehem, PA, USA. He leads the Explainable Robotics Lab (ERL). He is also affiliated with the Computer Science and Engineering Department and the Autonomous and Intelligent Robotics Laboratory (AIRLab), Lehigh University. His research interests include enabling robot autonomy via scalable automated synthesis of explainable plans using motion planning and machine learning.

Andrew Schoer received the B.S. degree in electrical engineering and systems engineering from Washington University, St. Louis, MO, USA, in 2015, and the M.S. degree in systems engineering from Boston University, Boston, MA, USA, in 2021.

While not in school, he has worked on a variety of projects with MIT Lincoln Laboratory, Lexington, MA, USA. Currently, his work focuses on safety methods for autonomous systems, and reinforcement learning.



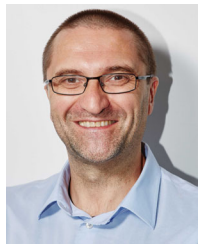
Austin M. Jones received the B.S. and M.S. degrees in systems science from Washington University, St. Louis, MO, USA, in 2010 and the Ph.D. degree in systems engineering from Boston University, Boston, MA, USA, in 2015.

He joined Numerica Corporation, Fort Collins, CO, USA, in 2010–2011 and was a Postdoctoral Fellow with the Schools of Mechanical Engineering and Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA, in 2015. From 2016–2020, he was a Member of the technical staff with MIT Lincoln Laboratory, Lexington, MA, USA. He is currently a Staff Software Engineer with Arbor Biotechnologies, Cambridge, MA, USA. His research interests include robotics, formal methods, and stochastic systems.

Roberto Tron (Member, IEEE) received the B.Sc. degree in 2004 and the M.Sc. degree (highest honors) in 2007 both in telecommunication engineering from the Politecnico di Torino, Turin, Italy, the Diplome d'Engenieur from the Eurecom Institute, Biot, France, and the DEA degree from the Université de Nice Sophia-Antipolis, Nice, France in 2006, and the Ph.D. degree in electrical and computer engineering from The Johns Hopkins University, Baltimore, MD, USA, in 2012.

He was Postdoctoral Researcher with the GRASP Lab, University of Pennsylvania, Philadelphia, PA, USA, until 2015. He is an Assistant Professor of Mechanical Engineering and System Engineering with Boston University, Boston, MA, USA. His research interests include intersection of automatic control, robotics and computer vision, with particular interest in applications of Riemannian geometry and linear programming in problems involving distributed teams of agents, or geometrical and spatio-temporal constraints.

Dr. Tron was recognized at the IEEE Conference on Decision and Control with the "General Chair's Interactive Presentation Recognition Award" in 2009, the "Best Student Paper Runner-up" in 2011, and the "Best Student Paper Award" in 2012. He is Associate Editor for IEEE TRANSACTION ON ROBOTICS.



Calin Belta (Fellow, IEEE) received the B.S. and M.Sc. degrees in control and computer science from the Technical University of Iasi, Iasi, Romania, the M.Sc. degree in electrical engineering from Louisiana State University, Baton Rouge, and the M.Sc. and Ph.D. degrees in mechanical engineering from the University of Pennsylvania, Philadelphia.

He is a Professor with the Department of Mechanical Engineering, Boston University, Boston, MA, USA, where he holds the Tegan Family Distinguished Faculty Fellowship. He is the Director of the BU Robotics Lab and of the Center for Autonomous and Robotic Systems (CARS), and is also affiliated with the Department of Electrical and Computer Engineering and the Division of Systems Engineering, Boston University. His research interests include dynamics, control, and formal methods, with particular emphasis on hybrid and cyber-physical systems, robotics, and systems biology.

Dr. Belta's notable awards include the 2005 NSF CAREER Award, the 2008 AFOSR Young Investigator Award, and the 2017 IEEE TCNS Outstanding Paper Award. He is a distinguished Lecturer of the IEEE Control System Society.