arXiv:2110.08436v4 [cs.RO] 21 Jun 2022

Extended Version of Reactive Task Allocation and Planning for Quadrupedal and Wheeled Robot Teaming

Ziyi Zhou[1], Dong Jae Lee[1], Yuki Yoshinaga[1], Stephen Balakirsky[2], Dejun Guo[3], and Ye Zhao[1]

***Abstract*— This paper takes the first step towards a reactive, hierarchical multi-robot task allocation and planning framework given a global Linear Temporal Logic specification. The capabilities of both quadrupedal and wheeled robots are leveraged via a heterogeneous team to accomplish a variety of navigation and delivery tasks. However, when deployed in the real world, all robots can be susceptible to different types of disturbances, including but not limited to locomotion failures, human interventions, and obstructions from the environment. To address these disturbances, we propose task-level local and global reallocation strategies to efficiently generate updated action-state sequences online while guaranteeing the completion of the original task. These task reallocation approaches eliminate reconstructing the entire plan or resynthesizing a new task. To integrate the task planner with low-level inputs, a Behavior Tree execution layer monitors different types of disturbances and employs the reallocation methods to make corresponding recovery strategies. To evaluate this planning framework, dynamic simulations are conducted in a realistic hospital environment with a heterogeneous robot team consisting of quadrupeds and wheeled robots for delivery tasks.**

I. Introduction

Mobile robots have been extensively investigated and deployed in various service applications such as assembly [1], surveillance, [2] and search and rescue [3]. In recent years, quadrupedal robots have been popularized for their superior traversability over unstructured terrains [4]. Nevertheless, even with exceptional locomotion capabilities, legged systems are often unstable, fragile, and less suitable for performing prolonged tasks compared to wheeled robots. However, distinct types of robots can form a heterogeneous team to compensate for their individual disadvantages.

Recent works on multi-robot systems have been focusing on mission planning problems with the assistance of formal languages such as Linear Temporal Logic (LTL) [5]. Originally proposed for model checking [6], LTL is a powerful tool used in the robotics community with a preponderance of research primarily conducted on wheeled robots [7], [8] and legged robots [9]–[11] for task and motion planning. There have also been works [7], [12]–[15] on multi-agent systems. However, objectives are explicitly assigned to individual robots rather than having one global specification. This can be challenging for a large team of robots [16], [17], where in most cases, a global task is simpler to define. Therefore, a simultaneous task allocation and planning (STAP) problem given a global LTL specification attracts more attention.

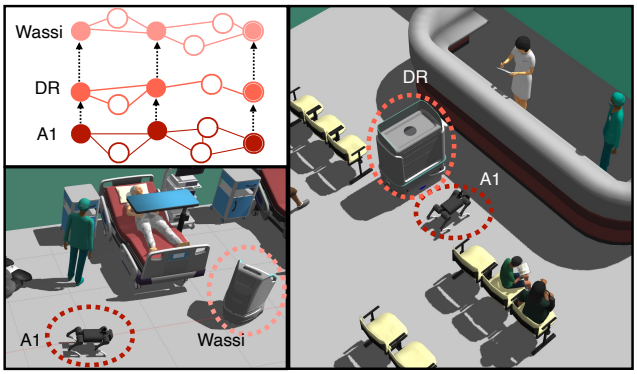

Fig. 1: A conceptual illustration of a three robots (Unitree A1, UBTECH DR, UBTECH Wassi) performing tasks in a simulated hospital setting. An illustration of team automaton is shown in the *top left*. The role of this team automaton is described in detail in Fig. 2.

A line of research exists where STAP problems have been solved with global LTL specifications. In [2], [18] a product model is constructed with an exponential complexity, while [19] proposed a team model automaton with a linear complexity, assuming that each robot conducts its task independently. Other works also focused on advanced search algorithms [16], [20], concrete time constraints [21], and collaborative tasks [22]. However, failure recovery during real-world deployment is rarely studied, especially considering unstructured environments such as rough terrain. Single and multi-robot scenarios have been demonstrated with disturbances caused by a change in the environment [23]–[26] or a failure to perform an action [27], but all have been limited to local LTL specifications. The work of [28] is conceptually similar to our goal, where each robot reallocates its tasks during execution failures, based on a Team Markov Decision Process. However, this work manually designs a set of LTL specifications in advance without identifying decomposable parts of a global task specification for optimal task sequences. Therefore, task reallocation capabilities during the online execution is desirable for STAP problems given global LTL specifications, which is addressed in this study.

In this paper, multiple reallocation approaches to a rich class of online disturbances are proposed to replan on the updated team model given one global LTL specification. A local reallocation efficiently outputs a new action sequence only for the problematic robot, while a global reallocation searches on the entire team model and generates optimal

[1]The authors are with the Laboratory for Intelligent Decision and Autonomous Robots, Woodruff School of Mechanical Engineering, Georgia Institute of Technology. {zhouziyi, dlee640, yyoshinaga3, yzhao301}@gatech.edu.

[2]The author is with Georgia Tech Research Institute, Atlanta, GA 30318. {Stephen.Balakirsky@gtri.gatech.edu}.

[3]The author is with the UBTECH North America Research and Development Center Corporation. {dejun.guo@ubtrobot.com}.

This work was supported by UBTECH, NSF grant #IIS-1924978, #CMMI-2144309, and Georgia Tech Research Institute IRAD Grant.

action sequences for all robots. Meanwhile, the connection between the high-level reallocation and the low-level disturbance detection is interfaced by a mid-level Behavior Tree (BT). BT is a commonly used graphical and mathematical controller for a robot that enables fault-tolerant task executions [29]–[34]. In our work, we structure the BT to accept an action sequence assigned by the LTL planner similar to [26], [35]. From an implementation perspective, our BT explicitly delegates all strategies while maintaining a simple structure that is extendable to other types of disturbances. In addition, disturbance detection rates at different frequencies are employed by leveraging the BT's modularity property.

Our contributions are summarized as follows:

- To handle potential disturbances from locomotion failures and environmental changes, we propose local and global task reallocation approaches given a global LTL specification. This approach eliminates the need to reconstruct the entire team model or resynthesize a new task.
- To select among reallocation strategies at run-time, we present a BT-based execution layer interfacing with low-level feedback. Our BT structure allows different types of disturbances to be detected at different rates.
- We evaluate our pipeline in a simulated hospital environment with a heterogeneous robot team consisting of both quadrupedal and mobile robots as shown in Fig. 1. An open-source software package[1] is provided for the proposed reactive multi-robot task allocation and planning framework.

II. Preliminaries

A. LTL basics

Linear temporal logic (LTL) has been widely used to encode temporal task specifications and automatically synthesize the system's transition behaviors. A specification ϕ is constructed from atomic propositions $\pi \in \Pi$, which is evaluated to be True or False and follows the syntax $\phi := \pi | \neg\phi | \phi_1 \wedge \phi_2 | \bigcirc\phi | \phi_1 \mathcal{U} \phi_2 | \phi_1 \mathcal{R} \phi_2$. Boolean operators $\neg$ "not" and $\wedge$ "and" in addition to a set of temporal operators $\bigcirc$ "next", $\mathcal{U}$ "until", and $\mathcal{R}$ "release", are denoted. To be concise, we omit the derivations of other boolean operators such as $\vee$ "or", $\rightarrow$ "implies", and $\leftrightarrow$ "if and only if", as well as temporal operators $\Diamond\phi$ "eventually ϕ" and $\Box\phi$ "always ϕ".

A common usage of LTL is for constructing an automaton. A non-deterministic automaton (NFA) is defined as a tuple $\mathcal{Q} = (S_\mathcal{Q}, S_{0,\mathcal{Q}}, \Sigma, \delta_\mathcal{Q}, F)$ such that $S_\mathcal{Q}$ is a set of states ($s_\mathcal{Q} \in S_\mathcal{Q}$), $S_{0,\mathcal{Q}} \subseteq S_\mathcal{Q}$ is a set of initial state, Σ is the input alphabet, $\delta_\mathcal{Q}$ is a set of transition relations such that $\delta_\mathcal{Q} : S_\mathcal{Q} \times \Sigma \rightarrow 2^{S_\mathcal{Q}}$, and F is a set of accepting final states. In addition, LTL formulas are evaluated over a sequence $\sigma : \mathbb{N} \rightarrow 2^\Pi$ where $\sigma(t) \subset \Pi$ represents all true propositions at time t [29].

For this framework, a transition system (TS) is created by combining data from the topological map and the robots' operating states. A TS is defined as a tuple $\mathcal{T} = (S_\mathcal{T}, s_{0,\mathcal{T}}, A_\mathcal{T}, \Pi_\mathcal{T}, \mathcal{L})$ such that $S_\mathcal{T}$ is a set of system states ($s_\mathcal{T} \in \mathcal{S}_\mathcal{T}$), $s_{0,\mathcal{T}} \in \mathcal{S}_\mathcal{T}$ is the initial system state, $A_\mathcal{T}$ is a set of available system actions, $\Pi_\mathcal{T}$ is the set of system propositions, and $\mathcal{L} : S_\mathcal{T} \rightarrow 2^{\Pi_\mathcal{T}}$ is a labeling function that assigns atomic propositions to states [16]. We use $\mathtt{Succ}(s_\mathcal{T}) = \{s'_\mathcal{T} \in S_\mathcal{T} | (s_\mathcal{T}, s'_\mathcal{T}) \in A_\mathcal{T}\}$ to denote the successors of $s_\mathcal{T}$ and $\mathtt{Pred}(s_\mathcal{T}) = \{s^*_\mathcal{T} \in S_\mathcal{T} | (s^*_\mathcal{T}, s_\mathcal{T}) \in A_\mathcal{T}\}$ as the predecessors of $s_\mathcal{T}$. By combining a TS with an NFA, a product automaton (PA) $\mathcal{P}$ composed of system states and mission specifications is generated. It is represented by $\mathcal{P} = \mathcal{Q} \otimes \mathcal{T} = (S_\mathcal{P}, S_{0,\mathcal{P}}, A_\mathcal{P})$ such that $S_\mathcal{P} = S_\mathcal{Q} \times S_\mathcal{T}$ is the set of states ($s_\mathcal{P} \in S_\mathcal{P}$), $S_{0,\mathcal{P}} = S_{0,\mathcal{Q}} \times \{s_{0,\mathcal{T}}\}$ is the set of initial states, and $A_\mathcal{P} = \{((s_\mathcal{Q}, s_\mathcal{T}), (s'_\mathcal{Q}, s'_\mathcal{T})) \in S_\mathcal{P} \times S_\mathcal{P} : (s_\mathcal{T}, s'_\mathcal{T}) \in A_\mathcal{T} \wedge s'_\mathcal{Q} \in \delta_\mathcal{Q}(s_\mathcal{Q}, \mathcal{L}(s_\mathcal{T}))\}$.

B. Offline task allocation

We introduce the baseline task allocation method and terminologies used throughout this paper. Readers are referred to [19] for more details. Given the LTL semantics above, a global task ϕ along with its corresponding NFA $\mathcal{Q}$ can be specified for a whole team of N agents, each of which has its own TS $\mathcal{T}^{(r)}$ and a corresponding PA $\mathcal{P}^{(r)} = \mathcal{Q} \otimes \mathcal{T}^{(r)}, r = \{1, \ldots, N\}$. Next, we will introduce a criterion that identifies the decomposed parts of ϕ based on the assumption that each agent executes its sub-task independently.

Definition 1 (Finite Decomposition [19])**.** *Let $\mathcal{J}_r$ with $r \in \{1, \ldots, N\}$ be a set of finite LTL task specifications and σ_i is any sequence s.t. $\sigma_r \models \mathcal{J}_r$. These tasks are called decomposition of the global finite LTL specification ϕ, iff:*

$$\sigma_{j_1} \ldots \sigma_{j_r} \ldots \sigma_{j_N} \models \phi \tag{1}$$

for all permutations of $j_r \in \{1, \ldots, N\}$ and all respective sequences σ_r.

Based on this criterion, a decomposition set $\mathcal{D} \subseteq S_\mathcal{Q}$ can be derived from $\mathcal{Q}$, which includes all NFA states that allocate tasks. Then the action-state sequence allocated to each agent is computed by first constructing a team automaton.

Definition 2 (Team automaton [19])**.** *The team automaton $\mathcal{G}$ is a union of N local PA $\mathcal{P}^{(r)}$ with $r \in \{1, \ldots, N\}$ and defined by $\mathcal{G} := (S_\mathcal{G}, S_{0,\mathcal{G}}, F_\mathcal{G}, A_\mathcal{G})$, where:*

- *$S_\mathcal{G} = \{(r, s_\mathcal{Q}, s_\mathcal{T}) : r \in \{1, \ldots, N\}, (s_\mathcal{Q}, s_\mathcal{T}) \in S^{(r)}_\mathcal{P}\}$ is the set of states;*
- *$S_{0,\mathcal{G}} = \{(r, s_\mathcal{Q}, s_\mathcal{T}) : r = 1, (s_\mathcal{Q}, s_\mathcal{T}) \in S^{(1)}_{0,\mathcal{P}}\}$ is the set of initial states;*
- *$F_\mathcal{G} = \{(r, s_\mathcal{Q}, s_\mathcal{T}) \in S_\mathcal{G} : s_\mathcal{Q} \in F\}$ is the set of final accepting states;*
- *$A_\mathcal{G} = \bigcup_r A^{(r)}_\mathcal{P} \cup \zeta$ is the set of actions including switch transitions ζ.*

We use $\delta_\mathcal{G} : S_\mathcal{G} \rightarrow S_\mathcal{G}$ to denote the transitions corresponding to $A_\mathcal{G}$. The set $\zeta \subset S_\mathcal{G} \times S_\mathcal{G}$ denotes the *switch transitions*, each of which $\varsigma = ((i, s_\mathcal{Q}, s_\mathcal{T}), (j, s'_\mathcal{Q}, s'_\mathcal{T}))$ is defined as a transition between two states in $\mathcal{G}$ iff: 1) $j = i + 1$: connects to the next agent; 2) $s_\mathcal{Q} = s'_\mathcal{Q}$: the

[1] https://github.com/GTLIDAR/ltl_multi_agent.

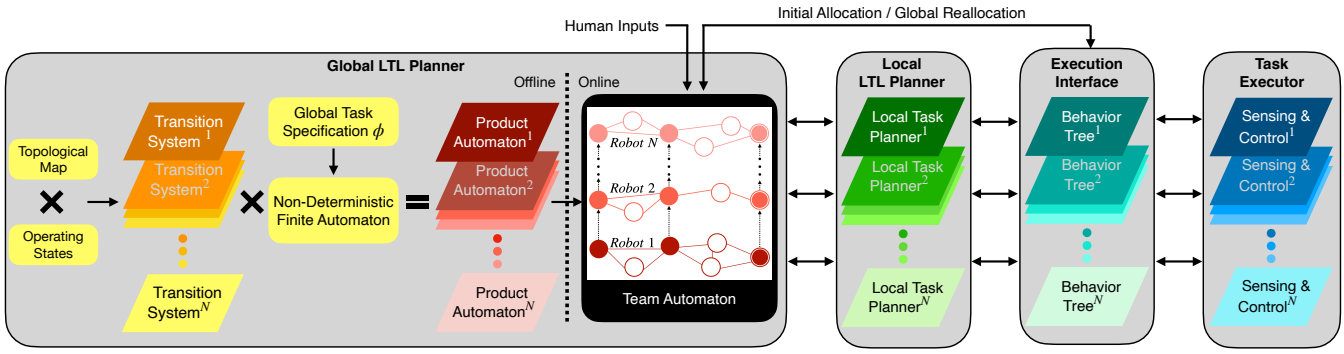

Fig. 2: An overview of the hierarchical planning framework. The global LTL planner portrays the high-level LTL task planner. Once the product automaton is created, it is sent to the team automaton, where a sequence of actions are assigned to individual robots. During execution, the local task planner, the behavior tree, and the hardware on the robot will react to disturbances and handle accordingly.

NFA state remains unchanged; 3) $s'_{\mathcal{T}} = s^{(j)}_{0,\mathcal{T}}$: points to an initial agent state; 4) $s_{\mathcal{Q}} \in \mathcal{D}$: the NFA state is inside the decomposition set.

The team automaton is a combination of every agent's PA $\mathcal{P}^{(r)}$ [2] with additional switch transitions which can reassign an agent's remaining task to another agent. As illustrated in Fig. 3(a), if a global action sequence β on the team automaton is found, we can state that task allocation and planning have been accomplished simultaneously (namely STAP [19]). By projecting β onto the PA of each agent, tasks can be executed in parallel. This process of finding an initial set of action-state sequence is called the *offline allocation*, whereas the state space complexity scales linearly with the number of agents. The rest of this paper will focus on addressing external disturbances during real-world deployment.

III. Problem Formulation

A. Disturbance characterization

In this section, we categorize disturbances into four classes in order to pair a reactive strategy that can efficiently resolve the problem. Unless specified, the following disturbances apply to both legged and wheeled robots.

- **Loss of balance** refers to a scenario where a *legged robot* falls due to an unstable gait or erratic controller output.
- **Critical failure** refers to an irrecoverable hardware or software malfunction such as a damaged motor or a software glitch.
- **Unexpected robot state change** refers to a situation where a robot detects a sudden shift in the robot's state.
- **Environmental change** refers to an environmental event preventing the robot from continuing its current task.

B. STAP reallocation

Given the baseline STAP approach (Sec. II-B), we seek to find an efficient reallocation strategy that is specific to each category of disturbance. For this problem, two aspects need to be investigated: 1) a formal guarantee to complete the global task; 2) a set of completed tasks by the whole team. To this end, we define a STAP reallocation problem as the following:

Problem Statement. *Given an initial task assignment and the current TS of every agent, one finds a set of new action-state sequences for the agents to accomplish the global task without restarting the whole mission or re-synthesizing a new mission.*

However, the task reallocation strategies are only compatible with high-level information such as changes in the NFA state or the TS state. Therefore, we propose a BT-based mid-level to organize different disturbance types into comprehensible inputs for the high-level planning framework.

Fig. 2 shows the planning architecture consisting of 1) a high-level task planner that performs offline allocation and online reallocation; 2) a mid-level BT interface to execute the assigned action plan for each agent; 3) low-level controllers that power the actuators on legged and wheeled robots.

IV. Reallocation Approach

To solve the STAP reallocation problem, we propose two approaches: a local and global approach, both of which are designed at the high level. Fig. 3(b) and 3(c) show the workflow and conceptual examples of both local and global reallocation.

A. Local reallocation addressing unexpected state changes

The offline task allocation process generates an action-state sequence for each agent, which assumes every action is performed successfully. During execution unexpected interventions could occur, which would undermine the original plan. For instance, if a human removes a load carried by the robot before the robot reaches its destination, an unforeseen robot state change occurs. To resolve this intervention, we introduce the local task reallocation approach. Suppose $\texttt{Path}^{(r)}$ is a planned state sequence for robot r, generated from the original team automaton. Let $\texttt{Path}^{(r)}(\text{acc})$ denote the agent's local accepting state and $s^{\triangle}_{\mathcal{T}}$ be the current state after the intervention. By comparing the state sequence execution history and $\texttt{Path}^{(r)}$, the last matched state is identified as $\texttt{Path}^{(r)}(m)$, whose NFA and TS are written as $\texttt{Path}^{(r)}_{\mathcal{Q}}(m)$ and $\texttt{Path}^{(r)}_{\mathcal{T}}(m)$. Thus,

[2]For simplicity, we abuse $\mathcal{P}^{(r)}$ to denote a sub-graph in $\mathcal{G}$ which contains the same states $S^{(r)}_{\mathcal{P}}$ and transitions except, the robot index r is appended.

$\texttt{Path}_{\mathcal{T}}^{(r)}(m+1) = s_{\mathcal{T}}^{\triangle}$. Now, the problem is reformulated to find a path on the local PA, starting from an up-to-date initial set defined as $S_{c,\mathcal{P}}^{(r)} = \{(r, s_{\mathcal{Q}}, s_{\mathcal{T}}) \in S_{\mathcal{P}}^{(r)} | s_{\mathcal{T}} = s_{\mathcal{T}}^{\triangle}, \forall s_{\mathcal{Q}} \in \delta_{\mathcal{Q}}(\texttt{Path}_{\mathcal{Q}}^{(r)}(m), \mathcal{L}(\texttt{Path}_{\mathcal{T}}^{(r)}(m)))\}$. The `find-path` method throughout this work is performed by Dijkstra's algorithm. Note that the original path can be reused if the current state happens to be on the agent's original path. This local task reallocation approach is summarized in Algorithm 1. $\texttt{Path}^{(r)}(i:)$ denotes a sub-path starting from the i^{th} element.

Proposition 1. *In the presence of one robot experiencing unexpected state changes and all other agents not interfered (i.e., can successfully accomplish their tasks), the global specification ϕ will be fulfilled if a path is found by the local reallocation method in Algorithm 1 on the local PA $\mathcal{P}^{(r)}$.*

Algorithm 1 Local reallocation: unexpected state change

Input: $\mathcal{P}^{(r)}, \texttt{Path}^{(r)}$
Output: A new path $\texttt{Path}^{(r)+}$

```
Path^(r)+ ← empty path
Path-set^(r) ← empty set
for s in S^(r)_{c,P} do
  if s in Path^(r) then
    i ← getIndex(Path^(r)(s))
    Path-set^(r).append(Path^(r)(i:))
    continue
  else
    Path-set^(r).append(find-path(s, Path^(r)(acc)))
  end if
end for
Path^(r)+ ← find-best(Path-set^(r))
```

Proof. According to Algorithm 1 and the definition of $S_{c,\mathcal{P}}^{(r)}$, the new state sequence $\texttt{Path}^{(r)+}$ (i) connects to agent r's executed state sequence through a valid NFA transition since $s_{\mathcal{Q}} \in \delta_{\mathcal{Q}}(\texttt{Path}_{\mathcal{Q}}^{(r)}(m), \mathcal{L}(\texttt{Path}_{\mathcal{T}}^{(r)}(m))$; and (ii) ends with the original local accepting state $\texttt{Path}^{(r)}(\text{acc})$. In other words, the originally assigned sub-task for agent r is fulfilled again. Given the assumption that the rest of the agents are not interfered by any disturbances, agent r's newly concatenated sequence, along with the other agents' planned sequences, consist a global action sequence β on the team automaton $\mathcal{G}$ again. Then according to the correctness property in [29], the projected global path onto the NFA satisfies the mission specification ϕ. □

B. Local reallocation addressing environmental changes

In the previous section, the disturbance shifts the robot's state but does not modify the environment, which will be addressed in this section. Such a disturbance will directly impact the TS. For instance, if the floor is occupied by an impassable object, the mobile robot would encounter a navigation failure and would not be able to transition to its next expected state. In this case, the robot will receive the changes to be made on TS called $Info(t)^{(r)}$. Each update contains three types of information: 1) $(s_{\mathcal{T}}, s'_{\mathcal{T}}) \in Add(t)$ if $s_{\mathcal{T}}$ is allowed to transit to $s'_{\mathcal{T}}$; 2) $(s_{\mathcal{T}}, s'_{\mathcal{T}}) \in Delete(t)$ if $s_{\mathcal{T}}$ is not allowed to transit to $s'_{\mathcal{T}}$; 3) $(b, s_{\mathcal{T}}) \in Relabel(t)$ if the labeling function of state $s_{\mathcal{T}}$ is updated to $b \subseteq 2^{AP}$.

The TS change can be reflected by directly modifying the team automaton using the PA revision strategy in [23]. When a single agent r receives an update, the latest team automaton is revised by only updating corresponding $\mathcal{P}^{(r)}(t)$[3]. All deleted transitions, i.e., edges, are added into a set $R(t)$.

Definition 3 (Updating rules)**.** *$\mathcal{G}(t)$ (more specifically, only $\mathcal{P}^{(r)}(t)$) is updated given the $Info(t)$ from agent r following the rules:*

- *If $(s_{\mathcal{T}}, s'_{\mathcal{T}}) \in Add(t)$, $(r, s^n_{\mathcal{Q}}, s'_{\mathcal{T}})$ is in $\delta_{\mathcal{G}}((r, s^m_{\mathcal{Q}}, s_{\mathcal{T}}))$ for $\forall s^n_{\mathcal{Q}}, s^m_{\mathcal{Q}}$ satisfying $s^n_{\mathcal{Q}} \in \delta_{\mathcal{Q}}(s^m_{\mathcal{Q}}, \mathcal{L}(s_{\mathcal{T}}))$;*
- *If $(s_{\mathcal{T}}, s'_{\mathcal{T}}) \in Delete(t)$, $(r, s^n_{\mathcal{Q}}, s'_{\mathcal{T}})$ is deleted from $\delta_{\mathcal{G}}((r, s^m_{\mathcal{Q}}, s_{\mathcal{T}}))$ for $\forall s^n_{\mathcal{Q}}, s^m_{\mathcal{Q}} \in S^{(r)}_{\mathcal{Q}}$;*
- *If $(b, s_{\mathcal{T}}) \in Relabel(t)$, then $\forall s^*_{\mathcal{T}} \in Pred(s_{\mathcal{T}})$: $(r, s^n_{\mathcal{Q}}, s^*_{\mathcal{T}})$ is added to $\delta_{\mathcal{G}}((r, s^m_{\mathcal{Q}}, s_{\mathcal{T}}))$ for $\forall s^n_{\mathcal{Q}} \in \delta_{\mathcal{Q}}(s^m_{\mathcal{Q}}, b)$; $(r, s^n_{\mathcal{Q}}, s^*_{\mathcal{T}})$ is deleted from $\delta_{\mathcal{G}}((r, s^m_{\mathcal{Q}}, s_{\mathcal{T}}))$ for $\forall s^n_{\mathcal{Q}} \notin \delta_{\mathcal{Q}}(s^m_{\mathcal{Q}}, b)$*

If a disturbance was detected on an agent's TS, a new type of task reallocation algorithm is necessary. Note that no unexpected robot state is assumed in this case and the last matched state is equivalent to the current state, i.e. $\texttt{Path}_{\mathcal{T}}^{(r)}(m) = s_{\mathcal{T}}^{\triangle}$. Given the revised local PA $\mathcal{P}^{(r)}(t)$, we propose a different replanning approach in Algorithm 2, compared to the one in Sec. IV-A.

Proposition 2. *In the presence of environmental change and all other agents that are not interfered (i.e., can successfully accomplish their tasks), the global specification ϕ will be fulfilled if a path is found by the local reallocation method in Algorithm 2 on the revised local PA $\mathcal{P}^{(r)}(t)$.*

Proof. According to Algorithm 2, the new state sequence $\texttt{Path}^{(r)+}$ starts from the last state $\texttt{Path}^{(r)}(m)$ in agent r's execution history and reaches the same local accepting state $\texttt{Path}^{(r)}(\text{acc})$. Since the PA updating rules preserve valid NFA transitions, the originally assigned sub-task for agent r is still fulfilled by the newly found path on $\mathcal{P}^{(r)}(t)$. Same as Proposition 1, a global action sequence β is formed assuming the rest agents are not interrupted by any disturbances. Consequently, the global specification ϕ is satisfied again. □

Algorithm 2 Local reallocation: environmental change

Input: $\mathcal{P}^{(r)}(t), \texttt{Path}^{(r)}, Info(t)$
Output: A new path $\texttt{Path}^{(r)+}$

```
Path^(r)+ ← empty path
P^(r)(t), R(t) ← UpdatePA(P^(r)(t), Info(t))
if R(t) ∩ edge(Path^(r)) ≠ ∅ then
  Path^(r)+ ← find-path(Path^(r)(m), Path^(r)(acc))
else
  Path^(r)+ ← Path^(r)(m:)
end if
```

[3]We use $\cdot(t)$ to denote an updated automaton at time t given the transition relation is changed.

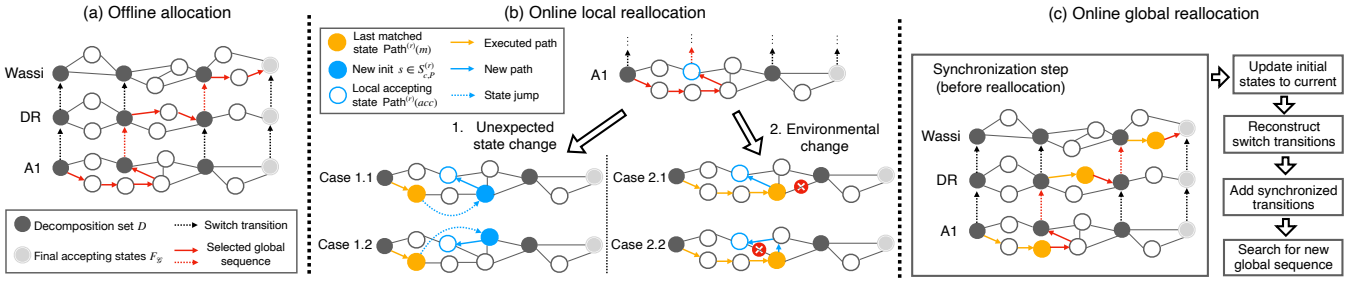

Fig. 3: Conceptual illustrations of local and global reallocations: (a) offline allocation given the team automaton. A global sequence (the red lines with arrows) is found during the offline phase as an initial task allocation. Assuming A1 undergoes a disturbance such as *unexpected state change* and *environmental change* as shown in subfigure (b). In correspondence to Algorithm 1, Case 1.1 refers to a scenario where the new initial state after the state jump is on the original offline-generated global sequence, while Case 1.2 corresponds to a completely new initial state and requires a replanning. Similarly in Algorithm 2, Case 2.1 demonstrates a scenario where the transition change doesn't affect the execution of the original global path, while a replanning is required in Case 2.2 due to a deleted edge on the original global path (represented by the cross marker). Subfigure (c) reveals a block diagram on the essential steps for a global reallocation. The synchronization step notifies the planner of each agent's current state (denoted by the yellow circle) and execution history to proceed with the remaining steps. More details are explained in Sec. IV-C.

C. Global reallocation

The two aforementioned local task reallocation approaches do not consider replanning for the whole team, which results in a sub-optimal strategy. Furthermore, if the local task reallocation fails to find a new plan for the agent, a succeeding global task reallocation over the entire team is activated. First, a synchronization step will be executed where the task planner requests for each agent's current TS state and sets it to be the latest initial TS state $s^{(r)}_{0,\mathcal{T}}(t)$. Then the initial PA set $S^{(r)}_{0,\mathcal{P}}(t)$ is updated accordingly by keeping $S^{(r)}_{0,\mathcal{Q}}$ the same. Since each $\mathcal{P}^{(r)}(t)$ has been updated during local reallocation if needed, the team automaton is only modified by updating the initial set of states and switch transitions, in addition to appending the *synchronized transitions*.

Definition 4 (Synchronized team automaton)**.** *The synchronized team model $\mathcal{R} := \mathcal{G}(t)$ is a union of N product automata $\mathcal{P}^{(r)}(t)$ given the updated product states after synchronization, where $r \in \{1,\dots,N\}$ and $\mathcal{R} := (S_\mathcal{R}, S_{0,\mathcal{R}}, F_\mathcal{R}, A_\mathcal{R})$ consists of:*

- $S_\mathcal{R} = \{(r, s_\mathcal{Q}, s_\mathcal{T}) : r \in \{1,\dots,N\}, (s_\mathcal{Q}, s_\mathcal{T}) \in S^{(r)}_\mathcal{P}(t)\}$ *is the set of states;*
- $S_{0,\mathcal{R}} = \{(r, s_\mathcal{Q}, s_\mathcal{T}) : r = 1, (s_\mathcal{Q}, s_\mathcal{T}) \in S^{(1)}_{0,\mathcal{P}}(t)\}$ *is the set of initial states;*
- $F_\mathcal{R} = \{(r, s_\mathcal{Q}, s_\mathcal{T}) \in S_\mathcal{R} : q \in F\}$ *is the set of final accepting states, which remains unchanged since the NFA accepting states are fixed;*
- $A_\mathcal{R} = \bigcup_r A^{(r)}_\mathcal{P}(t) \cup \zeta(t) \cup \xi(t)$ *is the set of actions that include the updated switch transitions $\zeta(t)$ and the newly proposed synchronized transitions $\xi(t)$.*

Suppose $\texttt{ExePath}^{(r)}$ is the executed state sequence acquired from each agent r and $\texttt{ExePath}^{(r)}_\mathcal{Q}$ is the projected NFA state sequence. The definition of a synchronized transition is as follows:

Definition 5 (Synchronized transition)**.** *The set $\xi \subset S_\mathcal{R} \times S_\mathcal{R}$ denotes synchronized transitions. Each element $\varepsilon = ((i, s_\mathcal{Q}, s_\mathcal{T}), (j, s'_\mathcal{Q}, s'_\mathcal{T}))$ satisfies:*

- *$i = j$: connects the same agent;*
- *$s_\mathcal{Q} = \texttt{ExePath}^{(r)}_\mathcal{Q}(\text{init}), s'_\mathcal{Q} = \texttt{ExePath}^{(r)}_\mathcal{Q}(\text{final})$, $r \in \{1,\dots,N\}$ starts from the initial NFA state and points to the most recent NFA nodes upon request for synchronization of each agent;*
- *$s_\mathcal{T} = s'_\mathcal{T}$: TS state is preserved.*

This synchronized transition allows a new transition between two NFA states inside each agent's $\mathcal{P}^{(r)}(t)$. Once this is complete, each agent will be aware of the task completion status of the whole team and avoid performing redundant tasks. In the original team automaton [29], the four properties including correctness, independence, completeness, and ordered sequence are proposed to justify the rationale of finding a global path on the team automaton for a task allocation. Here we claim that our synchronized team automaton preserves these properties, so that a new global action sequence β can be found by applying the same search algorithm performed during the offline phase (as presented in Sec. II). This process leads to a global task reallocation that assigns new sub-tasks to all agents.

Proposition 3. *The synchronized team automaton preserves the properties of the original team automaton, i.e., correctness, independence, completeness and ordered sequence.*

Proof. The synchronized team automaton is distinguished from the original team automaton in four folds: 1) the set of initial states $S_{0,\mathcal{R}}$ is different since the initial PA set $S^{(1)}_{0,\mathcal{P}}(t)$ is updated; 2) the set of actions $A^{(r)}_\mathcal{P}(t)$ from each agent r is updated according to the latest $\mathcal{P}^{(r)}(t)$; 3) the switch transitions $\zeta(t)$ are removed and then reconstructed after setting the current TS state as the latest initial state $s^{(r)}_{0,\mathcal{T}}(t)$ for each agent r; 4) the synchronized transitions are added according to the execution history. The first three changes won't affect the properties of the team automaton in that every state $s_\mathcal{R} \in S_\mathcal{R}$ still has an NFA component $s_\mathcal{Q}$ constructed from ϕ, and the switch transitions $\zeta(t)$ are reconstructed under the same definition. As for the fourth change, according to the second condition of Definition 5, new transitions between NFA states are added by connecting the initial and final NFA states from each agent r's execution history $\texttt{ExePath}^{(r)}_\mathcal{Q}$. Although these transitions are not

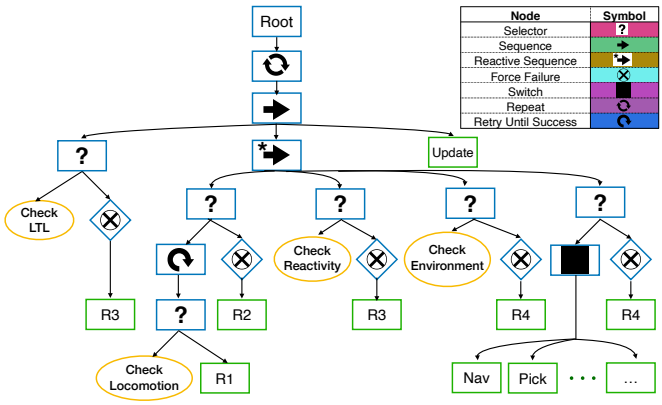

Fig. 4: A behavior tree for three levels of reactive strategies is shown. Basic types of BT nodes are thoroughly described in [36]. Yellow oval nodes are condition nodes, and green rectangular nodes are action nodes. R1 is the recovery stand for legged robots, R2 is the reallocation for critical failure, R3 is the reallocation for unexpected robot state change, and R4 is the reallocation for environmental change. Force Failure custom decorator node terminates the BT to execute an appropriate reactive strategy for the specific condition node.

directly provided by ϕ, since the states in $\texttt{ExePath}^{(r)}$ have already been traversed by agent r, there always exists a sequence of valid NFA transitions between $\texttt{ExePath}_{\mathcal{Q}}^{(r)}(\text{init})$ and $\texttt{ExePath}_{\mathcal{Q}}^{(r)}(\text{final})$. Therefore, synchronized transitions do not change the validness of the original NFA transitions, but only skip the executed transitions. Consequently, all of the original properties, including correctness, independence, completeness, and ordered sequence, are preserved. □

V. BT Execution Layer

As described in Sec. IV, both local and global task reallocation approaches enable robot recovery from various disturbances and failures. We leverage the reactivity property of BT at the middle level to rapidly select appropriate reactive strategies.

A. Reactive strategies

The decision to execute one reactive strategy over another is determined by the categoried disturbances in Sec. III. Four corresponding types of reactive strategies are specified below.

- **Recovery stand**. This scenario responds to a loss of balance which is specific to legged robots. In this case, the low-level controller on the robot attempts to recover from the fallen state without triggering the high-level LTL planner.
- **Task reallocation: critical failure**. Under this scenario, the robot is deemed incapable of working and therefore, a local task reallocation will be impractical. To resolve this issue, the robot will directly request a global task reallocation assuming a disability to transit to any TS state, to allow the remaining agents to take over its task.
- **Task reallocation: unexpected robot state change**. If this type of disturbance is detected, the planner will locally reallocate its task sequence without the assistance of other robots. If no local plans are feasible, the planner will perform a global task reallocation.

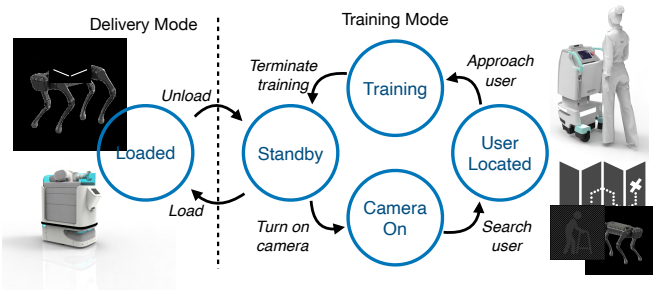

Fig. 5: Operating state diagram for legged robot capable of both delivery and training task. Each node refers to robot operating states, whereas each edge refer to an action performed by the robot. Unitree A1 quadruped *(top left)* and UBTECH DR *(bottom left)* are shown as delivery robots, and A1 and UBTECH Wassi *(top right)* are shown as training robots.

- **Task reallocation: environmental change**. If a robot encounters an environmental change, it will perform a local task reallocation. However, if a solution is not found, a global task reallocation will be performed.

B. Behavior tree structure

The BT structure is constructed in the form of precondition $\rightarrow$ action $\rightarrow$ effect, which is similar to the works of [26], [35]. This structure allows us to encode all of the reactive strategies into separate nodes within a single BT. Two special types of nodes, Repeat and Reactive Sequence, are utilized in our BT. The former node allows a simpler BT construction instead of concatenating all actions into one big tree, while the latter node enables condition nodes to check for disturbances at varying frequencies (i.e., detection rates). For example, as shown in Fig. 4, checking locomotion failure, reactive state change, and environmental change would be executed continuously under the Reactive Sequence node, while checking LTL precondition would only occur prior to receiving a new task. The BT for a wheeled robot is similar but omitted due to space limits.

VI. Evaluation and Discussion

A. Experiment set-up for simulation and hardware

To evaluate the feasibility and robustness of the proposed multi-robot task allocation and planning framework, we first establish a simulation of a hospital environment in Gazebo [37] and create a topological map for defining the TS, as shown in Fig. 6. The simulation architecture is composed of a high-level LTL planning layer based on a ROS package from [38], a mid-level execution interface using BehaviorTree.CPP [39], and a low-level navigation and controller layer using ROS navigation stack and appropriate controllers for each robot model. A convex model predictive controller from MIT Mini Cheetah [40], [41] is used to control a Unitree A1 quadruped over rough terrain, while a conventional holonomic drive model is used on the UBTECH DR and Wassi robot. For global path planning, the robot navigates between regions using the A* algorithm [42] and performs collision avoidance with the dynamic window approach [43].

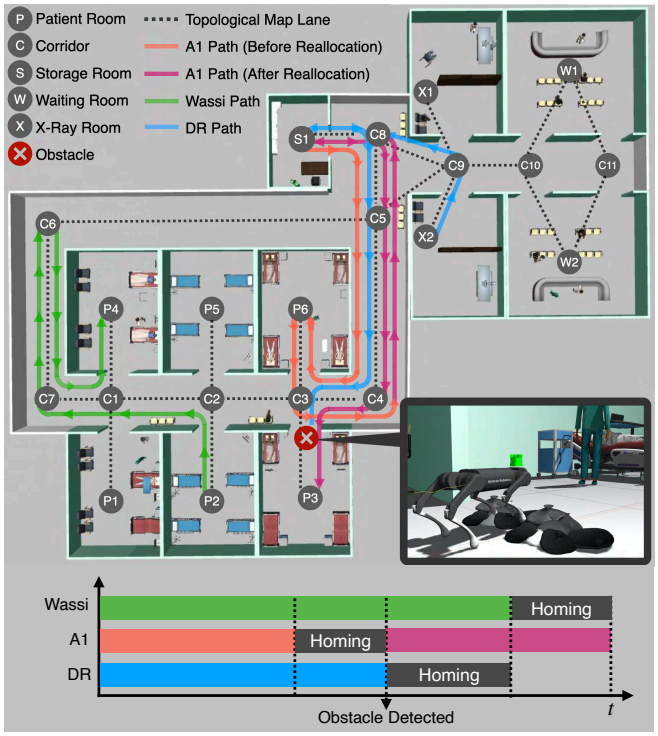

Fig. 6: Topological map of a hospital environment consisting of locations that are explicitly defined in each robot's transition system. The execution history for each robot is depicted for achieving global task ϕ under a scenario where an unexpected garbage heap is detected. The timeline for task execution and reallocation is illustrated at the bottom.

B. Case study

We evaluate our framework on a heterogeneous team of robots consisting of a delivery robot DR, a walk training robot Wassi, and a quadrupedal robot A1 with both capabilities. As shown in Fig. 5, a delivery robot consists of two simple operating states (*Loaded, Standby*), where the *Standby* state is equivalent to *Unloaded State*. Likewise, a training robot consists of 4 operating states (*Standby, Camera On, User Located, Training*), where the robot visually locates and helps seniors who need walking training assistance. These operating states are encoded into TS for each type of robot.

We conduct a series of case studies in a hospital environment simulation to evaluate the reactive strategies proposed in Sec. V. The global mission is defined as:

Scenario 1. *"Deliver medicines to locations p3 and p6; meet a patient at c1; complete a walk training along the corridor between c1 and c6, and then send the patient back to p4."*

$$\begin{aligned}\phi_1 =& \Diamond(p3 \wedge Standby) \wedge \Box((\neg p3 \wedge \bigcirc p3) \rightarrow Loaded) \\ & \wedge \Diamond(p6 \wedge Standby) \wedge \Box((\neg p6 \wedge \bigcirc p6) \rightarrow Loaded) \\ & \wedge \Diamond(p4 \wedge Standby) \wedge \Box((\neg p4 \wedge \bigcirc p4) \rightarrow Training) \\ & \wedge \Diamond(c1 \wedge Training \wedge \Diamond(c7 \wedge Training \wedge \Diamond(c6 \wedge Tr \\ & \text{-}aining \wedge \Diamond(c7 \wedge Training \wedge \Diamond(c1 \wedge Training)))))\end{aligned}$$

Three robots, A1, DR and Wassi, are placed at various locations in the hospital before the start of the simulation. Next, the task planner decomposes the specification and assigns sub-tasks to each robot offline, which takes 21 seconds in total. 92% of the computation time is taken by the generation of PA for each robot, which only needs to be performed once.

TABLE I: Triggered times and averaged computation time for four different recovery strategies R1 – R4.

	R1	R2	R3	R4
Triggered times	18	13	10	16
Local reallocation time (s)	-	-	0.023	0.018
Global reallocation time (s)	-	3.31	2.93	3.42

During the simulation, each robot will encounter a disturbance. Since the integration with exteroceptive systems is out of this paper's scope, all the disturbances except the locomotion failure are triggered by manually setting the checkers in BT to be false. A diagram of this simulation is displayed in Fig. 6. More details can be found in the video[4].

1) External force is applied to A1 and induces a loss of balance. A handcrafted whole-body recovery stand trajectory is tracked by a PD controller to assist A1 to resume its task.

2) A critical failure is induced for Wassi when performing the walk training task from $c7$ to $c6$. A global reallocation strategy assigns A1 to complete its delivery task first, and then proceeds to finish Wassi's incomplete walk training task.

3) The *Loaded* state of DR suddenly becomes a *Standby* state. This simulates a situation in which the robot unexpectedly loses its cargo. A local reallocation succeeds in instructing the robot to return to $s1$ and pick up another cargo. Then DR is instructed to complete the original delivery task.

4) A garbage heap is placed in front of $p3$ to simulate an environmental change. This obstruction can only be traversed by the legged robot A1. The routes taken by each robot and the timeline for obstacle detection and task allocation are portrayed in Fig. 6. While performing its task, DR encounters the obstruction and fails to find an alternate plan via local reallocation. While A1 is returning home after completing the delivery task to $p6$, it is assigned to take over DR's incomplete delivery task at $c4$. As a result, A1 goes to $s1$ to retrieve the object for delivery, and completes the task by traveling to $p3$.

In addition to applying individual disturbances, we also evaluate the the same scenario ten times with multiple manually triggered adversarial disturbances. Furthermore, the initial configuration for each robot is modified to prompt a different offline allocation result, for evaluating the generalization of our approach. At run-time, signals indicating robot state and environmental changes are sent to each robot's BT and prompt their reactive behaviors. As shown in Table I, we report the number of times that the reactivity strategy is triggered and the average computation time it takes for each trial. The LTL planner is not responsible for locomotion failure (R1) and the critical failure (R2) can only be handled at global level. Although a global task reallocation in principle could result in an optimal task sequence according to Proposition 3, it is observed to be more computational expensive compared with the local

[4]https://youtu.be/AO7Ms3GKzYQ

reallocation. This is caused by a communication among all agents and extra computation for constructing the synchronized team automaton. Our framework demonstrates an 80% success rate of completing the global mission ϕ. The 20% failed cases are caused by mission-level failures when more than one robot undergoes a critical failure, which causes the remaining task to be unachievable.

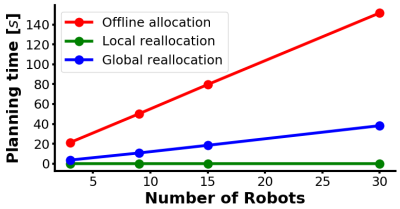

Fig. 7: The recorded planning time for offline allocation, local and global reallocation given different numbers of robots.

To evaluate the scalability of our proposed framework, we deploy 3 (described above), 9, 15, and 30 robots in the same aforementioned scenario. Due to the high computation demand from the quadruped controller, we don't run the simulation but directly trigger the LTL planner to perform reallocation strategies. In all scalability tests, the delivery robot is assumed to encounter an abrupt state change that triggers local reallocation and a critical failure that triggers global reallocation. Without significant code optimization, Fig. 7 reveals a linear complexity for both offline allocation and global reallocation, while the time required for local reallocation maintains at the same level. An alternative but *ad hoc* way for reallocation is to reconstruct the entire team model with an updated specification, which is similar to repeating the expensive offline process and requires human designer knowledge in the loop.

VII. Conclusion and Discussions

In this work, we present a heterogeneous, multi-robot task allocation and planning framework equipped with a hierarchically reactive mechanism from extensive disturbances. A local and global task reallocation is performed at the high level where an LTL-based team automaton is generated to follow a formal guarantee. At the middle level, a BT framework is incorporated to promptly select different replanning strategies which can be executed at different rates. Lastly, all the work mentioned is showcased in a dynamic simulation of a hospital scenario involving quadrupeds and wheeled robots.

In certain scenarios, a global task reallocation could result in a more optimal task sequence than the one generated from a local reallocation. For instance, if a robot encounters a blocked path, a local strategy will first attempt to find a detour. However, if the task could be transferred over to a different robot closer to the destination, the global task reallocation will outperform the local one. Our current framework does not consider optimality over computational efficiency. For our future work, we will incorporate a method to take into account optimality to determine whether replanning for the whole team over a single robot is favourable.

References

[1] D. Halperin, J.-C. Latombe, and R. H. Wilson, "A general framework for assembly planning: The motion space approach," *Algorithmica*, vol. 26, no. 3, pp. 577–601, 2000.

[2] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, "Optimality and robustness in multi-robot path planning with temporal logic constraints," *The International Journal of Robotics Research*, vol. 32, no. 8, pp. 889–911, 2013.

[3] J. S. Jennings, G. Whelan, and W. F. Evans, "Cooperative search and rescue with a team of mobile robots," in *International Conference on Advanced Robotics. Proceedings.* IEEE, 1997, pp. 193–200.

[4] P. Biswal and P. K. Mohanty, "Development of quadruped walking robots: a review," *Ain Shams Engineering Journal*, 2020.

[5] Y. Shoukry, P. Nuzzo, A. Balkan, I. Saha, A. L. Sangiovanni-Vincentelli, S. A. Seshia, G. J. Pappas, and P. Tabuada, "Linear temporal logic motion planning for teams of underactuated robots using satisfiability modulo convex programming," in *2017 IEEE 56th annual conference on decision and control (CDC)*. IEEE, 2017, pp. 1132–1137.

[6] A. Pnueli, "The temporal logic of programs," in *Annual Symposium on Foundations of Computer Science*. IEEE, 1977, pp. 46–57.

[7] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE transactions on robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[8] J. A. DeCastro, J. Alonso-Mora, V. Raman, D. Rus, and H. Kress-Gazit, "Collision-free reactive mission and motion planning for multi-robot systems," in *Robotics research*. Springer, 2018, pp. 459–476.

[9] J. Warnke, A. Shamsah, Y. Li, and Y. Zhao, "Towards safe locomotion navigation in partially observable environments with uneven terrain," in *IEEE Conference on Decision and Control*, 2020, pp. 958–965.

[10] Y. Zhao, Y. Li, L. Sentis, U. Topcu, and J. Liu, "Reactive task and motion planning for robust whole-body dynamic locomotion in constrained environments," *The International Journal of Robotics Research*, p. 02783649221077714, 2022.

[11] S. Kulgod, W. Chen, J. Huang, Y. Zhao, and N. Atanasov, "Temporal logic guided locomotion planning and control in cluttered environments," in *American Control Conference*. IEEE, 2020, pp. 5425–5432.

[12] M. E. Cao, J. Warnke, Y. Han, X. Ni, Y. Zhao, and S. Coogan, "Leveraging heterogeneous capabilities in multi-agent systems for environmental conflict resolution," *arXiv preprint arXiv:2206.01833*, 2022.

[13] M. Guo and D. V. Dimarogonas, "Multi-agent plan reconfiguration under local ltl specifications," *The International Journal of Robotics Research*, vol. 34, no. 2, pp. 218–235, 2015.

[14] J. Tumova and D. V. Dimarogonas, "Multi-agent planning under local ltl specifications and event-based synchronization," *Automatica*, vol. 70, pp. 239–248, 2016.

[15] Y. Kantaros and M. M. Zavlanos, "Stylus*: A temporal logic optimal control synthesis algorithm for large-scale multi-robot systems," *The International Journal of Robotics Research*, vol. 39, no. 7, pp. 812–836, 2020.

[16] C. Banks, S. Wilson, S. Coogan, and M. Egerstedt, "Multi-agent task allocation using cross-entropy temporal logic optimization," in *IEEE International Conference on Robotics and Automation*. IEEE, 2020, pp. 7712–7718.

[17] A. Kolling, P. Walker, N. Chakraborty, K. Sycara, and M. Lewis, "Human interaction with robot swarms: A survey," *IEEE Transactions on Human-Machine Systems*, vol. 46, no. 1, pp. 9–26, 2015.

[18] Y. Chen, X. C. Ding, A. Stefanescu, and C. Belta, "Formal approach to the deployment of distributed robotic teams," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 158–171, 2011.

[19] P. Schillinger, M. Bürger, and D. V. Dimarogonas, "Decomposition of finite ltl specifications for efficient multi-agent planning," in *Distributed Autonomous Robotic Systems*. Springer, 2018, pp. 253–267.

[20] P. Schillinger, M. Bürger, and D. Dimarogonas, "Multi-objective search for optimal multi-robot planning with finite ltl specifications and resource constraints," in *IEEE International Conference on Robotics and Automation*. IEEE, 2017, pp. 768–774.

[21] K. Leahy, A. Jones, and C.-I. Vasile, "Fast decomposition of temporal logic specifications for heterogeneous teams," *arXiv preprint arXiv:2010.00030*, 2020.

[22] X. Luo and M. M. Zavlanos, "Temporal logic task allocation in heterogeneous multi-robot systems," *arXiv preprint arXiv:2101.05694*, 2021.

[23] M. Guo, K. H. Johansson, and D. V. Dimarogonas, "Revising motion planning under linear temporal logic specifications in partially known workspaces," in *IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 5025–5032.

[24] M. Guo and D. V. Dimarogonas, "Reconfiguration in motion planning of single- and multi-agent systems under infeasible local ltl specifications," in *52nd IEEE Conference on Decision and Control*. IEEE, 2013, pp. 2758–2763.

[25] S. C. Livingston, R. M. Murray, and J. W. Burdick, "Backtracking temporal logic synthesis for uncertain environments," in *IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 5163–5170.

[26] S. Li, D. Park, Y. Sung, J. A. Shah, and N. Roy, "Reactive task and motion planning under temporal logic specifications," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 12 618–12 624.

[27] J. Tumova, A. Marzinotto, D. V. Dimarogonas, and D. Kragic, "Maximally satisfying ltl action planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 1503–1510.

[28] F. Faruq, D. Parker, B. Laccrda, and N. Hawes, "Simultaneous task allocation and planning under uncertainty," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2018, pp. 3559–3564.

[29] P. Schillinger, M. Bürger, and D. V. Dimarogonas, "Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems," *The international journal of robotics research*, vol. 37, no. 7, pp. 818–838, 2018.

[30] M. Colledanchise, D. Almeida, and P. Ögren, "Towards blended reactive planning and acting using behavior trees," in *International Conference on Robotics and Automation*. IEEE, 2019, pp. 8839–8845.

[31] R. H. Abiyev, I. Günsel, N. Akkaya, E. Aytac, A. Çağman, and S. Abizada, "Robot soccer control using behaviour trees and fuzzy logic," *Procedia Computer Science*, vol. 102, pp. 477–484, 2016.

[32] M. Kim, M. Arduengo, N. Walker, Y. Jiang, J. W. Hart, P. Stone, and L. Sentis, "An architecture for person-following using active target search," *arXiv preprint arXiv:1809.08793*, 2018.

[33] V. Berenz and S. Schaal, "The playful software platform: Reactive programming for orchestrating robotic behavior," *IEEE Robotics & Automation Magazine*, vol. 25, no. 3, pp. 49–60, 2018.

[34] E. Coronado, X. Indurkhya, and G. Venture, "Robots meet children, development of semi-autonomous control systems for children-robot interaction in the wild," in *IEEE International Conference on Advanced Robotics and Mechatronics*. IEEE, 2019, pp. 360–365.

[35] M. Lan, S. Lai, T. H. Lee, and B. M. Chen, "Autonomous task planning and acting for micro aerial vehicles," in *IEEE International Conference on Control and Automation*. IEEE, 2019, pp. 738–745.

[36] M. Colledanchise and P. Ögren, "Behavior trees in robotics and AI: an introduction," *CoRR*, vol. abs/1709.00084, 2017.

[37] OpenRobotics. (May) Hospital. Open Robotics. [Online]. Available: https://fuel.ignitionrobotics.org/1.0/OpenRobotics/fuel/collections/Hospital

[38] R. Baran, X. Tan, P. Varnai, P. Yu, S. Ahlberg, M. Guo, W. S. Cortez, and D. V. Dimarogonas, "A ros package for human-in-the-loop planning and control under linear temporal logic tasks," in *IEEE International Conference on Automation Science and Engineering*, 2021.

[39] D. Faconti and M. Colledanchise, "BehaviorTree.CPP," 2019. [Online]. Available: https://github.com/BehaviorTree/BehaviorTree.CPP

[40] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic locomotion in the mit cheetah 3 through convex model-predictive control," in *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2018, pp. 1–9.

[41] D. Kim, J. Di Carlo, B. Katz, G. Bledt, and S. Kim, "Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control," *arXiv preprint arXiv:1909.06586*, 2019.

[42] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[43] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.