KT-BT: A Framework for Knowledge Transfer Through Behavior Trees in Multirobot Systems

Sanjay Sarma Oruganti Venkata, Ramviyas Parasuraman, *Senior Member, IEEE*, and Ramana Pidaparti

***Abstract*—Multirobot and multiagent systems demonstrate collective (swarm) intelligence through systematic and distributed integration of local behaviors in a group. Agents sharing knowledge about the mission and environment can enhance performance at individual and mission levels. However, this is difficult to achieve, partly due to the lack of a generic framework for transferring part of the known knowledge (behaviors) between agents. This article presents a new knowledge representation framework and a transfer strategy called KT-BT: knowledge transfer through behavior trees. The KT-BT framework follows a query-response-update mechanism through an online behavior tree framework, where agents broadcast queries for unknown conditions and respond with appropriate knowledge using a condition-action-control subflow. We embed a novel grammar structure called *stringBT* that encodes knowledge, enabling behavior sharing. We theoretically investigate the properties of the KT-BT framework in achieving homogeneity of high knowledge across the entire group compared to a heterogeneous system without the capability of sharing their knowledge. We extensively verify our framework in a simulated multirobot search and rescue problem. The results show successful knowledge transfers and improved group performance in various scenarios. We further study the effects of opportunities and communication range on group performance, knowledge spread, and functional heterogeneity in a group of agents, presenting interesting insights.**

***Index Terms*—Behavior trees (BTs), collective intelligence, heterogeneity, knowledge transfer, multirobot systems (MRSs), planning.**

Nomenclature

s_i/s	CURRENT state sequence.
s_q	Query state sequence.
s_{ka}	State sequence (knowledge action)
Q_r	List of action sub-trees received.
Q_m	List of queries received.
L_{ks}	Known states list.
L_{ka}	Known action list.
$\mathcal{T}^i_{\text{lbl}}$	Tree with label lbl and index i
$\mathcal{T}_{\text{control}}$	Control BT.
$\mathcal{T}_{\text{kt}}$	General tree (knowledge transfer).
$\mathcal{T}^*_{\text{child}}$	Child subtree.
$\mathcal{T}_{\text{teach}}$	Teach BT.
$\mathcal{T}_{\text{learn}}$	Learn BT.
$\mathcal{T}_{\text{CK}}$	Common knowledge sub-trees.
$\mathcal{T}_{\text{PK}}$	Prior Knowledge sub-trees.
$\mathcal{T}_{\text{NK}}$	New knowledge sub-trees.
$\mathcal{T}_k$	Knowledge subtree.
$\mathcal{T}_F$	Fallback sub-trees.
$\mathcal{T}_C$	Critical sub-trees.
$\mathcal{T}_{\text{ka}}$	Knowledge action subtree.
$\mathcal{T}^*_{\text{ka}}$	Response subtree.
t^1_{limit}	TT1 time limit.
t^2_{limit}	TT2 time limit.
G	Agent group.
P	Population size.
V_{COL_j}	Collision vector (jth object)
V_c	Resultant collision vector.

Manuscript received 26 March 2023; accepted 13 June 2023. Date of publication 13 July 2023; date of current version 4 October 2023. This paper was recommended for publication by Associate Editor Fabio Morbidi and Editor Wolfram Burgard upon evaluation of the reviewers' comments. *(Corresponding author: Ramviyas Parasuraman.)*

Sanjay Sarma Oruganti Venkata is with the School of Electrical and Computer Engineering, University of Georgia, Athens, GA 30602 USA (e-mail: sanjayovs.research@outlook.com).

Ramviyas Parasuraman is with the School of Computing, University of Georgia, Athens, GA 30602 USA (e-mail: ramviyas@uga.edu).

Ramana Pidaparti is with the School of Environmental, Civil, Agricultural and Mechanical Engineering, University of Georgia, Athens, GA 30602 USA (e-mail: rmparti@uga.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.2023.3290449.

Digital Object Identifier 10.1109/TRO.2023.3290449

I. Introduction

HUMANS and animals developed social communication as an evolutionary trait over thousands of years to help each other locate potential food opportunities, predators, migratory information, etc. [1]. These communications are ubiquitous and are crucial for decision-making under unknown circumstances and determine both the individual and group's survival and benefits [2], [3]. Depending on the type of information or knowledge transmitted, this may involve different auditory, visual, olfactory, or tactile communication methods and their combinations of modalities.

With the increasing pervasiveness of robots in industries, agriculture, transportation, defense, and security, the centralization of knowledge is sometimes complex and challenging. In addition, direct information exchange between the agents has advantages and enhances the system-level performance and robustness while also reducing the design complexities [4]. This is very important in mission control involving a variety of robots, where seamless transfer of knowledge requires a common acceptable framework across both homogeneous and heterogeneous groups in a multirobot system (MRS).

Knowledge and information-sharing strategies, similar to those in natural systems, have also been studied and applied in

information science, multiagent systems (MAS), machine learning, and IoT focused on developing collective intelligence [5]. Many important pieces of information are combined to generate knowledge from which inferences, action sequences, and predictions are made. While information is directly transferable, knowledge sharing may sometimes involve learning and require unique ways to transfer.

In robotics and artificial multiagent intelligent systems, such as in many animals, the knowledge can be factual, conceptual, metacognitive, or procedural [6]. Among these, procedural knowledge defines the robot's ability to perform a given task by synthesizing information from sensory data [7]. It can be the knowledge of performing a routine assembly operation performed in automobile manufacturing or cleaning a carpeted floor [8]. This knowledge can either be acquired through various learning strategies, such as observation and imitation [9], using evolutionary techniques [10], [11], machine learning [12], and transfer learning [13], knowledge sharing through query-response mechanisms [14], or preprogrammed. In many cases, the knowledge is limited or unique to a single robot or may involve multiple robots, learned from humans through demonstration or clarification, etc. [15], [16]. In the case of an MRS, a robust and unifying knowledge-sharing framework is currently lacking but is highly critical to facilitate robust knowledge transfer and improve group performance.

To remedy the gaps in knowledge representation and transfer in MRSs, we propose a new framework called knowledge transfer through behavior trees (KT-BT). The framework mimics how humans share knowledge by explicitly communicating the need-based behaviors through a query-response-update mechanism. Fig. 1 depicts an overview of the KT-BT framework. KT-BT is built over behavior trees (BTs), which are historically applied to automate nonplayer characters (NPCs) in games [17] and gathered recent applications to robotics and AI [18], [19]. The framework encapsulates knowledge in a hierarchical structure containing various sub-trees, each representing a particular knowledge (task-level condition-action tuples) that can be explicitly shared across the MRS.

The main contributions of this article are summarized as follows.

1) We define a novel query-based knowledge-sharing framework called KT-BT, where robots[1] explicitly communicate and collaborate to share parts of their exclusive knowledge base. Here, knowledge is a set of functional skills a robot possesses to execute an action plan based on the current state. The knowledge base of each robot could be preprogrammed or learned through some techniques such as reinforcement learning, but we assume they are preprogrammed (encoded) in their high-level state-action planning framework. Furthermore, to simplify the concepts, we assume that each robot has some part of the knowledge that is commonly present on all robots and some part of the knowledge that is unique to a robot.

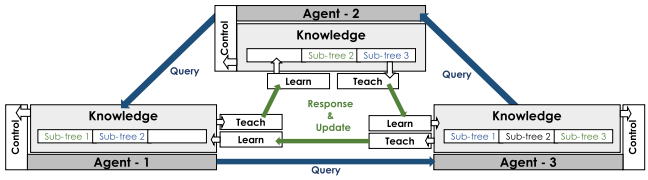

Fig. 1. Overview of the multiagent KT-BT framework, which facilitates the sharing of knowledge in the form of sub-trees between agents. The process involves querying, responding, and updating the agents' BTs. The figure summarizes the teach-learn process, where sub-trees from one agent can be communicated and integrated into the BT of another agent.

2) We utilize the modularity and hierarchy features of BTs in designing a query-response mechanism among multiple agents and embed a new mechanism to incorporate the acquired knowledge by updating the BTs on the go, enabling the framework to work online for robot control and collaboration.
3) We introduce a unique BT representation using novel grammar constructs called *stringBT* that enables the protocols for query, quick retrieval of sub-trees, and response for achieving knowledge sharing.
4) We theoretically analyze the properties of KT-BT in guaranteeing knowledge transfer between robots, increasing knowledge spread across the group, and enhancing opportunities to improve mission performance in a generic multiagent framework.
5) We present an application of the KT-BT framework on a search and rescue (SAR) problem simulation involving multiple robots.[2] Here, we validate the advantages and demonstrate the utility of the KT-BT framework in terms of homogenizing the knowledge spread and improving the overall mission performance and efficiency.

Finally, the rest of this article is organized as follows. Section II briefly reflects on the knowledge-sharing strategies proposed in the literature. In Section III, we discuss some background on BTs and formally introduce our KT-BT framework through definitions, architectures, and algorithms. Section IV presents the theorems that establish the knowledge spread in a multiagent group. We validate the KT-BT framework on an application considering a multirobot SAR or multitarget foraging problem in Section V. The results and analyzes on knowledge propagation from the simulations are presented in Section VI. Finally, Section VII concludes this article. The Nomenclature lists the important notations used in this article.

II. Related Work

We present an overview of various ontologies for knowledge representation and sharing in autonomous agents and discuss how our KT-BT framework differs from state-of-the-art.

Transfer of information via any modality, such as an agent communication language [20] generally involves identifying

[1]We use the term "robot" and "agent" interchangeably to represent an autonomous agent with some intelligence.

[2]We released an executable version of the simulator at https://github.com/herolab-uga/KTBT-Release to provide the readers a sandbox platform to configure the SAR simulation settings and verify the advantages of KT-BT.

queries, responding appropriately, and merging the response with existing knowledge. Hence, knowledge should be represented so that it is easily accessible, retrievable, and shared with the group [21] for individual and collective decision-making.

In this regard, ontologies have gained a reputation for their flexibility and robustness in knowledge representation, as they are designed to be unambiguous and can be reused, fragmented, or shared directly with other agents [22]. In ontologies, vocabularies for concepts [23] are defined along with their relationships and constraints in the form of axioms. Their logic is described using syntax and semantics, and operations, such as merging, mapping, alignment, unification, refinement, and inheritance are performed on relationship maps to dynamically update ontologies with new knowledge [24].

Research in MRS has seen many robust applications of ontologies for inter-robot knowledge transfer. The standards like core ontologies for robotics and automation (CORA) [25] accelerated the development of knowledge sharing in both homogeneous and heterogeneous multirobot teams and primarily focused on human–robot interactions, positioning systems, or industrial settings [26], [27].

In some hybrid techniques involving multiagent reinforcement learning, agents combine the policies, or knowledge with ontological representation for sharing with other agents [28]. For example, Qu et al. [29] proposed a framework with multiagent reinforcement learning for optimal scheduling of a multiskill workforce and multiple machines for a multistage manufacturing process. Similarly, Oprea et al. [30] used a combination of ontologies and Q-learning for agent adaptation. Taylor et al. [31] proposed a parallel transfer learning technique where the selected knowledge is shared with agents in parallel.

A robot receiving new knowledge shared by other robots can strategically decide on the need to merge by comparing the rewards of its experience. For instance, a confidence-based approach can be used to accommodate this knowledge [32]. Alternatively, agents can use a value function to share policies mutually, and each individual agent uses a common model to combine its expert policy with the multiagent network policy in deducing a joint policy [33]. A similar query-answer-based model-sharing was proposed by Jiang et al. [34] and Zhou et al. [35], in which an equilibrium-based sparse interaction framework that shares local Q-values with other agents called NegoSI was developed.

In all these works involving applications of ontologies in MAS and MRS research, we can observe that the knowledge does not directly represent the control actions at a lower level or instead supports only high-level decisions and inter-robot/agent communications. We identify a lack of a unifying model framework that works at all levels and that combines decision making, control, knowledge sharing, and communication. In addition, applying ontologies requires clear concept definitions, establishing relationships between concepts, and defining constraints, which requires a good amount of domain expertise and knowledge of using various tools.

On the other hand, there is growing interest in applying BT to robotics, multiagent, and multirobot control due to their properties of scalability, modularity, reactive, and safety guarantee [19], [36]. A summary of various nodes and components of BT design along with an application on a humanoid robot is presented [18], [37]. BTs have shown excellent advantages in robotics [38] and MRS [39], [40], [41]. For instance, in [42], a BT-based mechanism with explicit communication requests was proposed for multiagent event-driven coordination in NPCs of video games.

Furthermore, the synthesis of optimal controllers can be performed dynamically by exploiting the modularity of the tree structure in BT. For instance, Pacheck et al. [43] used the linear temporal logic specifications encoded as a game-theoretic structure to synthesize new (or revised) skills relevant to solving the tasks. Such a planning-level skill synthesis can be integrated with execution-level BT architecture, as demonstrated in [44].

Contrary to the existing frameworks for knowledge representation and sharing, we use BTs to represent knowledge and propose a grammar protocol for sharing knowledge, similar to grammatical evolution-based BT synthesis in [10]. BTs are uniquely suited to our knowledge-sharing framework because they are capable of combining control, planning, and learning into a single unifying framework [11], [45], [46]. Also, in comparison to auction-based [47] or ontology-based [48] approaches that are mostly for either task or skill representation only, our framework using BTs provides the flexibility of knowledge representation, high-level decision-making, hierarchical state-action planning, and low-level control execution. However, the current BT-based methods lack tools to expand knowledge-sharing between multiple agents and to perform BT operations similar to ontologies. In addition, there is no consensus in the literature on a standard design template for task-agnostic BT design. Our work is in the direction of representing a knowledge-based BT template that is generalizable across various tasks and applications.

Therefore, we propose the new KT-BT framework that uses a query-response mechanism for explicitly sharing knowledge using communication in MRS. To the best of authors' knowledge, our KT-BT framework is the first work in the literature that incorporates the new knowledge (or intelligence) in a real-time manner (through live updates of their BTs) while the robots are performing their current control actions using their current BTs. Furthermore, present some first-of-a-kind investigations on the properties of knowledge sharing and its spread in an MRS group under various scenarios in a SAR simulation case study. We believe these advances will help advance the research in robotics and MAS/MRS by enabling explicit knowledge sharing.

III. Proposed KT-BT for Knowledge Transfer Using BTs in MASs

In this section, we first present a background on BT and discuss the query-response mechanism. Then, we formulate the knowledge representation using BTs and introduce a new protocol to enable explicit query, retrieval, and sharing of part of the knowledge between robots in an MRS.

A. Background on BTs

BTs were first introduced for the control design of NPCs in video games, in which the conditions and actions are mapped

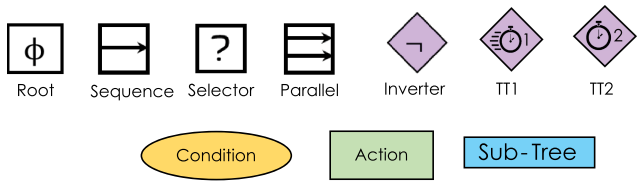

Fig. 2. Various BT nodes used in the current study.

TABLE I
SUMMARY OF THE FUNCTIONS OF CONTROL, DECORATORS, AND EXECUTION NODES USED IN THIS STUDY

Type of Node	Function
Sequence	Runs children nodes from left to right till a child node returns *failure*. Returns *success* when all children nodes return *success*.
Selector	Runs child nodes from left to right till a child node returns *success*. Returns *success* when at least one child node returns *success*.
Parallel	Runs all children in parallel. Returns *success*.
Inverter	Inverts the return value of the child.
TT1	Waits for a fixed number of ticks before executing the child. Returns *running* during the wait, and returns child return value after the wait.
TT2	Runs child for a fixed number of ticks. Returns child return value when running, otherwise, returns *failure* when execution is complete.
Condition	Returns *success* when the condition is true.
Action	Executes action or action sequence. Returns running during execution, *success* after completion.
Sub-tree	A smaller tree that can be merged with a larger tree.

using control and execution nodes. They provide excellent graphical design flexibility to the user to modify the control actions and define hierarchies in task planning for agents. Over time, they found their way into robotics and other AI applications [18].

BTs are directed trees that start with a root node and may have multiple control and execution nodes. Root nodes have no parents, execution nodes have no children, and control nodes have one parent and may have multiple children. In general, to represent BTs graphically, child nodes are represented under parent nodes, and all the execution nodes are shown as leaf nodes. Each execution of the BT happens at a certain frequency called ticks. In each tick, starting from the root node, the nodes are executed as per the control flow and from left to right. This article follows the convention of top-down tree flow representation and left-to-right priority in execution. And thus, the high-priority nodes can be placed with the leftmost nodes that are executed at the beginning of each tick. The node representations followed in the current work are presented in Fig. 2 and their summary in Table I.

1) Control Nodes: A control node may have multiple children that are executed according to logic. Commonly used control nodes are selector, sequencer, parallel, and decorator. A selector ticks children from left to right until a *success* is returned by a child, and a sequencer runs all the children from left to right till a child returns a *failure*. A parallel node executes all its child sub-trees in parallel and generally returns a *running* status [49]. Finally, a decorator node is designed to modify the child's response through a policy defined by the user. An inverter is a decorator that can only have one child node and flips the return status if it is different from running, e.g., a *success* to *failure*, and vice versa.

For our work, we propose two new decorators: a timer of type 1 (TT1) wait timer and timer of type (TT2) execution timer that are used to enable the query-response and online update mechanisms in the current work. Here, a TT1 timer can have only one child, and it waits for a fixed number of ticks before executing the child and returns running during the wait. After the delay, it returns the child return value. Similarly, a TT2 executes a child for a fixed number of ticks and returns the child status during the run time and failure thereafter.

2) Execution Nodes: Action and condition nodes fall under the execution category, which are the leaf nodes in a BT. An action node runs an action and returns a *success* if it is completed or a *failure* or *running* otherwise. On the other hand, a condition node verifies if a particular condition is satisfied and returns a *success* or returns a *failure* otherwise. Generally, all the condition variables frequently verified through a BT are maintained in a common location called a blackboard with (key, value) pairs. Similarly, in this study, for the SAR simulations, we maintain a state manager that keeps track of all the condition variables that a BT can access.

B. Overview of Query-Response Mechanism in KT-BTs

In our framework, each agent has a BT that defines its control, teaching (response), and learning (query and update) sequences that run in parallel. In general, each agent can exist either in a mission (executing an action using its current knowledge), teaching (responding to a query from other robots), or learning (incorporating new knowledge from other robots) modes, depending on its state and the conditions it encounters. Furthermore, every agent's control tree consists of critical, knowledge base, and fallback sub-trees. The term "fallback" we refer here in the sub-trees is different from the concept of "selector" control node in the BT structure.

While the critical and fallback sub-trees represent the agent's safety and fallback routines [50], respectively, the knowledge sub-trees representing the agent's current knowledge base are a primary focus of our work. The agent executes the knowledge sub-trees when a specific set of conditions are met in its environment. Furthermore, an agent also maintains a list of a known sequence of states and conditions that correspond to a new knowledge subtree. At any point during a mission, the agent verifies if the encountered state and condition sequences match with the sequences corresponding to its knowledge. When an unknown sequence is encountered, the agent broadcasts a query to its neighbors, thus initiating the query-response mechanism.

The agent sends out the unknown sequence as a query and awaits a response. Next, a receiving agent within the querying agent's communication range verifies the query sequence against

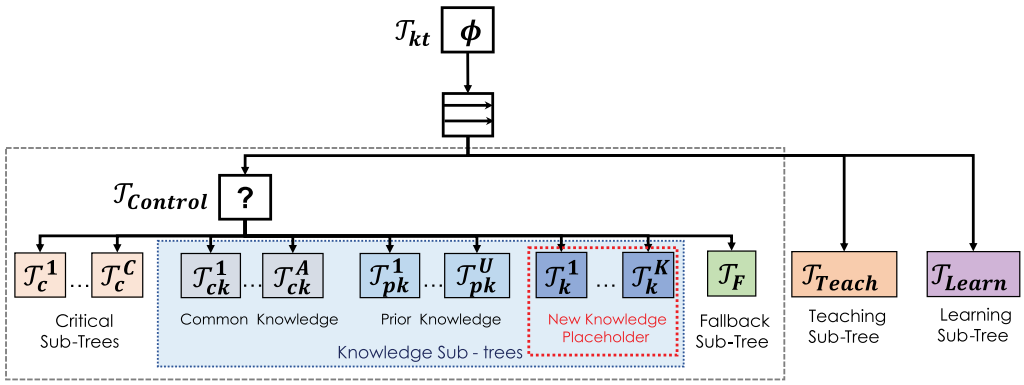

Fig. 3. Overall structure of BT designed for the current knowledge transfer study.

its known knowledge base and responds with the corresponding subtree encoded as a *stringBT* (described in Section III-D). Finally, the querying agent decodes the received response and merges it with its control tree, thus continuing with the appropriate execution process. We present an overview of the knowledge transfer in our current KT-BT framework in Figs. 1 and 3, where sub-trees are learned through query-response mechanisms between three functionally heterogeneous agents. Functional heterogeneity refers to robots with varied functional abilities, while structural heterogeneity involves robots that are structurally dissimilar, such as drones and manipulators.

This mechanism is advantageous when the agents demonstrate functional heterogeneity due to varying amounts of knowledge. For example, consider a functionally heterogeneous but structurally homogeneous team that contains only one agent with the knowledge of all the tasks. With the knowledge propagating across the groups, all the agents in the MAS can develop uniform capabilities in accomplishing the low-level tasks for mission-level success by learning from this one agent who knows all tasks. Similarly, consider a scenario where each agent in an MRS group contains unique knowledge that is complementary to other agents. Exploiting a KT-BT framework, this MRS group can propagate their knowledge, and each agent will harmonize their knowledge base by combining all of their knowledge. In another example, assume a robot has the ability to learn through interaction and encode this knowledge as a BT once learned [11], [51]. Other agents can acquire this knowledge using KT-BT framework without having to learn on their own.

C. Knowledge Formulation in KT-BT

A BT is defined as a three tuple, $\mathcal{T}_{\text{lbl}}^i = \{f^i, r^i, \Delta t\}_{\text{lbl}}$, where $i \in \mathbb{N}$ is the tree index, and lbl is a label that defines its class. f^i is the function that maps the system's current state $s^i \in S$ to the output actions a^i. Δt is a time step, and the return status is defined as $r^i : \mathbb{R}^n \rightarrow \{\mathcal{R}, \mathcal{S}, \mathcal{F}\}$, which can either be a running, successful, or failure status. Here, we go by any assumptions and definitions of sequence and selector as presented by Colledanchise et al., [18] in their state-space formulations for BTs.

For our current study, we designed a unique tree structure that facilitated the learning and teaching processes. We label this general tree structure as $\mathcal{T}_{\text{kt}}$ and is defined as follows.

Definition 1: A transfer learning tree $\mathcal{T}_{\text{kt}}$ has three sub-trees associated with control, learning, and teaching. All these three sub-trees are run in parallel

$$\mathcal{T}_{\text{kt}} = \text{Parallel}(\mathcal{T}_{\text{Control}}, \mathcal{T}_{\text{Teach}}, \mathcal{T}_{\text{Learn}}). \tag{1}$$

1) Control: A $\mathcal{T}_{\text{Control}}$ subtree is divided into critical, knowledge, and fallback sub-trees, each corresponding to their intended purposes, as shown in Fig. 3. For example, in the case of a mobile robot, a critical collision avoidance subtree with high priority is placed toward the left extreme, followed by lesser priority critical sub-trees for battery recharge or wait commands. Following the critical sub-trees, toward the right at a lower priority, are knowledge sub-trees.

A knowledge subtree can be classified either into common ($\mathcal{T}_{\text{CK}}$), prior ($\mathcal{T}_{\text{PK}}$), or new knowledge ($\mathcal{T}_{\text{NK}}$) sub-trees. The positions of these sub-trees may be varied depending on their order of priority. For the current framework, we maintain the priority order as common, prior, and new knowledge. A common knowledge subtree $\mathcal{T}_{\text{ck}}^i \in \mathcal{T}_{\text{CK}}$ is the knowledge that is common across all the agents in an MRS group.

In addition to common knowledge, an agent in a group may have prior knowledge $\mathcal{T}_{\text{PK}}$ that is inherent to the agent or may be acquired during a mission in the form of a new knowledge $\mathcal{T}_{\text{NK}}$. We create a placeholder in each agent's $\mathcal{T}_{\text{control}}$ where this new knowledge can be placed.

Finally, the tree $\mathcal{T}_F$ is a set of fallback sub-trees that follow the knowledge sub-trees segment. These trees are activated when none of the conditions toward the left under the selector in $\mathcal{T}_{\text{Control}}$ are met. Some examples of fallback routines include random walk, exploration, idle/sleep, etc. We present a formal definition of $\mathcal{T}_{\text{Control}}$ as follows.

Definition 2: A control subtree $\mathcal{T}_{\text{Control}}$ has selector with sub-trees in the order (priority) of critical sub-trees $\mathcal{T}_C$, knowledge trees $\mathcal{T}_{\text{CK}}$, $\mathcal{T}_{\text{PK}}$, $\mathcal{T}_{\text{NK}}$ and a fallback subtree $\mathcal{T}_F$

$$\mathcal{T}_{\text{Control}} = \text{Selector}(\mathcal{T}_C, \{\mathcal{T}_{\text{CK}}, \mathcal{T}_{\text{PK}}, \mathcal{T}_{\text{NK}}\}, \mathcal{T}_F). \tag{2}$$

Here, $\mathcal{T}_C$, $\mathcal{T}_{\text{CK}}$, $\mathcal{T}_{\text{PK}}$, and $\mathcal{T}_{\text{NK}}$ are ordered sets of critical, common knowledge, prior knowledge, and new knowledge

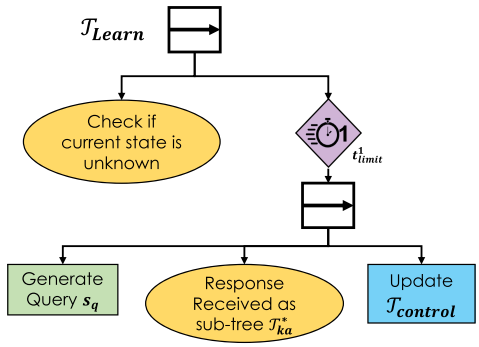

Fig. 4. Learning subtree transmits an unknown state-sequence s_q to other robots via broadcast and updates $\mathcal{T}_{\text{Control}}$ upon receiving a response $\mathcal{T}^*_{\text{ka}}$ from at least one robot.

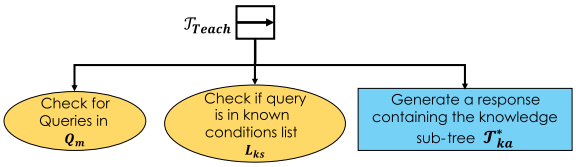

Fig. 5. Teaching subtree consisting of a sequence node that verifies if the queried sequence s_q is known and responds with the knowledge tree $\mathcal{T}^*_{\text{ka}}$ corresponding to the query s_q.

Algorithm 1: Pseudocode for **Learn** Process.

Input: $s_q, Q_r \leftarrow$ List of action sub-trees received
Data: $L_{ks}, L_{ka}, \mathcal{T}_{Control}$
Result: Merge received sub-tree $\mathcal{T}_k = Sequence(s_q, \mathcal{T}^*_{ka})$ with control sub-tree $\mathcal{T}_{Control}$ and return *success* or *failure*
▷ *If the current state sequence is not in the known-states list, then generate a query containing the current state sequence*
if s_q **not** in L_{ks} **then**
  $Broadcast(s_q)$
end if
▷ *Check if there is a response received.*
while not timeout **do**
  if $Q_r.length \neq 0$ **then**
    $\mathcal{T}^*_{ka} = Q_r.pop()$
    $L_{ks}.add(s_q)$
    $L_{ka}.add(\mathcal{T}^*_{ka})$
    $T_k \leftarrow Merge(Sequence(s_q, \mathcal{T}^*_{ka}))$
    $\mathcal{T}_{Control} \leftarrow Merge(\mathcal{T}_{Control}, \mathcal{T}_k)$
    return *success*
  end if
end while
return *failure*

sub-trees. A combined $\mathcal{T}_{\text{Control}}$ tree built from Definitions 1 and 2 is shown in Fig. 3.

The set of knowledge sub-trees in $\mathcal{T}_{\text{CK}}, \mathcal{T}_{\text{PK}}$, and $\mathcal{T}_{\text{NK}}$ are ordered. For example, a tree $\mathcal{T}^j_k \in \mathcal{T}_{\text{NK}}$ is defined as

$$\mathcal{T}^j_k = \text{Sequence}(s^j_{\text{ka}}, \mathcal{T}^j_{\text{ka}}) \tag{3}$$

where for a given knowledge subtree $\mathcal{T}^j_k$, s^j_{ka} denotes the state sequence, which is a sequence of conditions 1 through M denoted in their subscript $s^j = \{s_1, \ldots, s_M\}^j$. $\mathcal{T}^j_{\text{ka}}$ is an action subtree that is run when all the conditions corresponding to the sequence in the state sequence s^j are satisfied (for a jth knowledge subtree). We currently assume that the sequence of conditions and actions can be split into two separate sub-trees to simplify analysis. However, in real-world applications, these sub-trees can become more intricately intertwined.

2) Learn: Each agent is assumed to maintain a list of known states L_{ks} and their corresponding known action sub-trees in L_{ka}. A state-sequence s is generated by an agent by compiling a sequence of conditions encountered at a given instance. A learn tree runs in parallel continuously checking if $s \in L_{\text{ks}}$, treating s as a query-sequence if $s_q = s \notin L_{\text{ks}}$ (e.g., a new object detected but does not have an action sequence in the known knowledge sub-trees). The query is then broadcasted to obtain a response from other agents. We assume that the order of the state sequence follows a template that can otherwise also be modified using predefined sequencing strategies. We formally define learn as follows, and provide a pseudocode for the learning process in Algorithm 1 and the BT structure in Fig. 4.

Definition 3: A learning tree, $\mathcal{T}_{\text{Learn}}$ when faced with an unknown sequence s_q, broadcasts s_q, and waits for a response $\mathcal{T}^*_{\text{ka}}$. If received before a time-out, it is combined using a sequence operation on the query conditions to form a knowledge subtree $\mathcal{T}_k$. This subtree is merged to the new knowledge $\mathcal{T}_{\text{NK}}$ subtree segment in $\mathcal{T}_{\text{Control}}$. The query sequence s_q and $\mathcal{T}^*_{\text{ka}}$ are added to L_{ks} and L_{ka} sets, respectively, at the ith position in the new knowledge placeholder

$$\mathcal{T}_{\text{Learn}} = \text{Learn}(s_q, Q_r) \tag{4}$$

$$\text{Add}(\mathcal{T}^i_k, \{s_q, \mathcal{T}^*_{\text{ka}}\}), \text{ if } \mathcal{T}^*_{\text{ka}} \neq \emptyset. \tag{5}$$

3) Teach: A state-sequence query s_q is considered as known if it is present in the known states list L_{ks}, i.e., $s_q = L^i_{\text{ks}}$, for some i that maps the condition sequence to a state-action tree $\mathcal{T}^*_{\text{ka}} = \mathcal{T}^i_{\text{ka}} \in L_{\text{ka}} = \{\mathcal{T}^1_{\text{ka}}, \mathcal{T}^2_{\text{ka}}, \ldots\}$. A teaching tree $\mathcal{T}_{\text{Teach}}$ checks for any state-sequence query s_q received in a message buffer (Q_m) and responds with an appropriate state-action tree $\mathcal{T}^*_{\text{ka}}$ if the state-sequence query s_q is known in its knowledge base $\mathcal{T}_k$. A pseudocode for the teaching process is presented in the BT in Fig. 5 and its algorithm in Algorithm 2.

Definition 4: A teaching tree $\mathcal{T}_{\text{Teach}}$ upon receiving a query as a state s_q sequence in a message buffer Q_m, checks through a known states list in L_{ks} and if present, responds through an appropriate state-action tree $\mathcal{T}^*_{\text{ka}}$, where $\mathcal{T}^*_{\text{ka}} \in L_{\text{ka}}$, a list of known action sub-trees

$$\mathcal{T}_{\text{Teach}} = \mathcal{T}^*_{\text{ka}} = \text{Teach}(Q_m) = \begin{cases} \mathcal{T}^i_{\text{ka}} & \text{if } \exists\ s_q \in Q_m \mid s_q = L^i_{\text{ks}} \in L_{\text{ks}} \\ & \text{, for some } i \text{ and } \mathcal{T}^i_{\text{ka}} \in L_{\text{ka}} \\ \emptyset & \text{otherwise (no response).} \end{cases} \tag{6}$$

4) Timers: Finally, we define the two new timer decorators that are used in our KT-BT framework as follows. The

Algorithm 2: Pseudocode for **Teach** Process.

Input: $Q_m \leftarrow$ List of queries received from other agents
Data: L_{ks}, L_{ka}
Result: Transmit $\mathcal{T}^*_{ka}$ and return *success* or return *failure*
▷ *Check if a query is received.*
if $Q_m.length() \neq 0$ **then**
  $s_q \leftarrow Q_m.pop()$
▷ *Compare the query sequence against the sequences in known states list*
  for $i \leftarrow 1$ to $L_{ks}.length()$ **do**
▷ *If the query sequence is known, generate a response with the appropriate knowledge sub-tree*
    if $s_q = L^i_{ks}$ **then**
      $\mathcal{T}^i_{ka} \leftarrow L^i_{ka}$
      $\mathcal{T}^*_{ka} \leftarrow \mathcal{T}^i_{ka}$
      $Transfer(\mathcal{T}^*_{ka})$
      return Success
    end if
  end for
end if
return *failure*

Algorithm 3: Pseudocode for Timer Types **TT1** and **TT2**.

Input: $\mathcal{T}_{child}$, t^1_{limit} (TT1) or t^2_{limit} (TT2)
Result: TT1: Ticks a sub-tree $\mathcal{T}_{child}$ once after a duration of t^1_{limit} from the time the TT1 is ticked first.
Result: TT2: Ticks a sub-tree $\mathcal{T}_{child}$ for a duration of t^2_{limit}.
▷ TT1: *On the first tick of the timer, store start time*
▷ TT2: *When the timer is set, store start time*
if Timer.Start is *True* **then**
  $StartTime \leftarrow Time.current()$
else
  $t_{Elapsed} \leftarrow Time.current() - StartTime$
  if $t_{Elapsed} \geq t^1_{limit}$ (TT1) or $t_{Elapsed} \leq t^2_{limit}$ (TT2) **then**
    $Run(\mathcal{T}_{child})$ ▷Tick Child
    return Success
  else
    return Failure
  end if
end if

pseudocode versions of these two new timers are provided in Algorithm 3.

Definition 5: A TT1, runs its child subtree $\mathcal{T}_{\text{child}}$ once after the time elapsed is greater than a set limit t^1_{limit}, returns a success after successfully running the child tree and a failure otherwise

$$\mathcal{T}_{t1} = \text{TT1}(\mathcal{T}_{\text{child}}, t^1_{\text{limit}}). \tag{7}$$

Definition 6: A timer of type 2 (TT2) runs its child subtree till the time elapsed is less than t^2_{limit}, returns a success while running and a failure when stopped

$$\mathcal{T}_{t2} = \text{TT2}(\mathcal{T}_{\text{child}}, t^2_{\text{limit}}). \tag{8}$$

D. StringBT Representation of BT Grammar

The KT-BT framework requires a standard grammar for transmitting the response BT $\mathcal{T}^*_{\text{ka}}$ by a teaching tree. Through this grammar, a subtree as a whole is transmitted as a response to the queries posted by other robots. Therefore, we developed a unique *stringBT* representation similar to the grammatical representation of BTs by Neupen et al. [10] and Suddrey et al. [52].

In the *stringBT* representation, all the generic BT operators are designed to have shorthand tags for their equivalent code formats in BT constructs. The primary purpose of this grammatical representation is to simplify communication between agents and also to improve the human-readability aspect. For, e.g., a sequence operator in *stringBT* is represented as $<$ sq $>$ followed by other operations. A summary of various *stringBT* tags is presented in Section V-E along with an implementation of this grammar.

In KT-BT, when a condition sequence is queried, a teaching robot responds with $\mathcal{T}^*_{\text{ka}}$ formatted as a *stringBT*, and hence, the response is the form of *stringBT* sentences. On the receiving end, direct string manipulations like merge and append are performed using the received message at the *stringBT* equivalent of $\mathcal{T}_{\text{Control}}$ (specifically, at the new knowledge placeholder part of the *stringBT* grammar). The resultant $\mathcal{T}_{\text{Control}}$, which is also in *stringBT* form is converted into generic code representations for (re)compilation and ticking.

The string constructs in the *stringBT* grammar make it easier to search through its current BT during the teaching phase, as well as merge operations during the learning phase. Furthermore, this gives the capability to generalize this structure across multiple domains and applications in robotics and MRS. Also, with advanced string manipulation techniques, it is also possible to relax the condition-action splitting requirement for every knowledge as assumed in Section III-C3, as well as create the possibility of optimizing the BT and reorganizing the sub-trees (e.g., changing the priorities) in some applications.

IV. Formalization of the Knowledge Transfer

Here, the goal is to have knowledge shared between multiple agents involved in a mission. Having presented the definitions, we present more characteristics and technical analyzes of the knowledge transfer process.

Assumptions: The multiagent group is represented as an unweighted graph, $G = (V, E)$, with nodes V as the set of agents and edges E as the connectivity (communication possibility) between them. We assume that the graph is initially connected (not necessarily completely connected).

We now formalize the knowledge spread through the following theorems. First, we prove the knowledge transfer and propagation capability in KT-BTs (Theorem 1) and then we discuss the minimum latency and opportunity requirement for transferring knowledge.

Theorem 1 (Knowledge transfer between two agents): In a multiagent group connected by a graph G with $|V| = P$ number of robots, if there is only one agent k that has knowledge (i.e., knowledge subtree $\mathcal{T}_{\text{ka}}$) about a state sequence s_q, then the

knowledge corresponding to that skill/condition s_q is shared with all the agents in the group, as time $t \to \infty$

$\forall G_j \in G, s_q \subset L_{\text{ks}}(j), \mathcal{T}^*_{\text{ka}} \in L_{\text{ka}}(j)$ and $\mathcal{T}^*_{\text{ka}} \subset \mathcal{T}_{\text{Control}}(j)$ $\forall j = 1 \ldots P$.

Here, $\mathcal{T}^*_{\text{ka}}$ is the response received from agent k by agent j, which integrates the new knowledge subtree to its knowledge base and immediately updates its control BT, $\mathcal{T}_{\text{Control}}(j)$.

Proof: To prove this theorem, we first show that there will be knowledge transfer between the agent k (which has the knowledge) and an arbitrary agent l (which does not have this specific knowledge) in the graph G. Then, by induction, we show that all other agents in the graph G will eventually obtain this new knowledge, since the graph G is connected.

As agent l faces conditions in state-sequence s_q, the $\mathcal{T}_{\text{Learn}}$ tree verifies the condition is not in agent ls, $L_{\text{ks}}(l)$, and hence, generates a query s_q. As, the query is received by agent the interactive agent k in which, the $\mathcal{T}_{\text{Teach}}$ verifies in $k's$ known condition list $L_{\text{ks}}(k)$ and transmits the subtree $\mathcal{T}^*_{\text{ka}}$ in response according to Definition 4. Agent l merges this tree with the $\mathcal{T}_{\text{Control}}$ tree and adds the condition s_q to the list L_{ks} and $\mathcal{T}^*_{\text{ka}}$ to L_{ka}.

Now, we expand the abovementioned proof to all other agents in the group. Assuming enough opportunities exist for the knowledge to be propagated, which is equally (or randomly) distributed in the entire workspace, when an agent p faces a condition that is not in its known list of state sequence (knowledge database) $s_q \notin L_{\text{ks}}(p)$, also $s_q \in L_{\text{ks}}(p)$. By induction principle, the knowledge transfer from agent k to p will happen the same way as an arbitrary agent k, i.e., $s_q \in L_{\text{ks}}(p), \mathcal{T}^*_{\text{ka}} \in L_{\text{ka}}(p)$, and $\mathcal{T}^*_{\text{ka}} \subset \mathcal{T}_{\text{Control}}(p)$. This can also be proven to any agent j within the communication range of k or l (or any agent that has already obtained this knowledge). Through this one-to-one transmission of knowledge after a sufficient amount of time, the subtree $\mathcal{T}^*_{\text{ka}}$ related to the state sequence s_q is transferred to all the agents in the group G. ■

Corollary 1 (Latency of knowledge spread): For an environment with equal opportunity, the minimum latency (or speed) with which the transmission of knowledge happens is directly proportional to the diameter of the communication graph G, i.e., for a chain graph, the diameter would be equal to $P - 1$, where the knowledge transfer is the slowest. On the contrary, for a completely connected graph, the diameter is 1, where the knowledge transfer is the fastest.

Corollary 2 (Opportunity of knowledge spread): If there is only one agent k with knowledge of the state sequence s_q, then the minimum number of occurrences N_{occ} (queries) of s_q that are required for the knowledge T^*_{ka} to be transferred to all the agents in the group is equal to $P - 1$.

i.e. If $s_q \notin L_{\text{ks}}(j) \forall j \in [1, P] - \{k\}$, and $s_q \in L_{\text{ks}}(k)$, then $\min\{N_{\text{occ}}(s_q)\} = P - 1$.

Remark: The actual number of queries (or opportunities) would depend on the graph connectivity graph, the distributions of the opportunities and the agents in the workspace, the response rate, and the need to require the knowledge with s_q in the mission. For instance, if an agent k is at the center of the connectivity graph, the knowledge spread will be faster (Corollary 1) than this agent being at the end of a line graph, for example. In addition, the opportunity for propagation will be higher (i.e., fewer queries are needed) if more than one agent has the same knowledge that can be shared (Corollary 2).

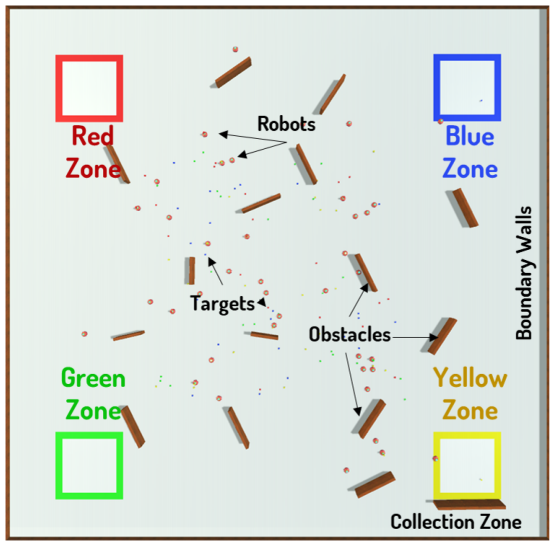

Fig. 6. Sample search space showing four collection zones at corners, randomly located targets, and 40 robots performing a simulated SAR mission.

V. Case Study Application: SAR

To provide an example of the concepts defined earlier and to analyze the framework, we consider a SAR problem with multiple robots. The SAR problem aims to collect different color-coded targets and move them to their corresponding collection zones. The generalized SAR problem we used here is analogous to multirobot foraging and multitarget search problems. These problems are predominantly used to test multirobot algorithms [53], [54].

A. Search Space

The search space is a rectangular space defined by $\mathcal{A} = [0, x] \times [0, y]$ dimensions. Targets are cubes colored red, green, yellow, and blue. There are four collection zones for each of the colored targets located at the four corners of the search space $\mathcal{A}$. The number of red, green, yellow, and blue targets are n_r, n_g, n_y, and n_b, respectively, and the total number of targets $n_t = n_r + n_g + n_y + n_b$. The targets are randomly scattered on the 2-D plane $\mathcal{A}$, and both the targets and the collection points are stationary. The search space may or may not have obstacles; however, every robot perceives other robots as obstacles. Fig. 6 presents a sample search space with randomly distributed targets.

B. KT-BT SAR Simulator

We developed a simulator tool for the SAR problem in the Unity 3-D game development environment (see Figs. 6 and 7). We used the Fluid BT library[3] and adapted them for the KT-BT

[3][Online]. Available: https://github.com/ashblue/fluid-behavior-tree

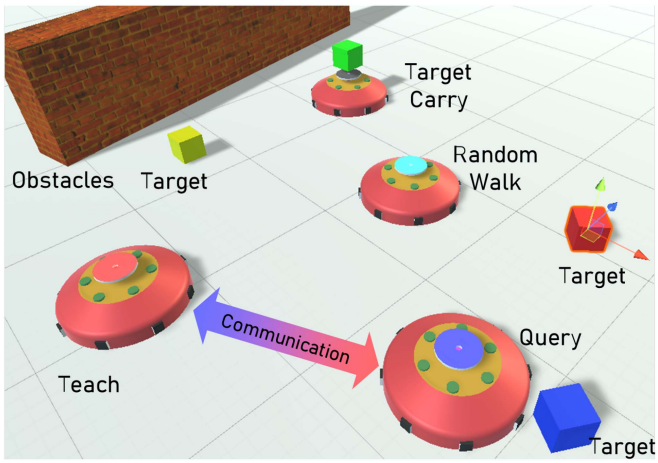

Fig. 7. Snapshot of simulation, showing various robot states. A robot indicator on the top blinks in red when teaching, blue during a query, and cyan during the random walk.

framework by combining them with the Roslyn[4] framework. We specify a code segment of BT called *LiveBT*, which is the compiled version of the knowledge base $\mathcal{T}_{\text{control}}$, and this *LiveBT* controls the robot based on its status. In general, the Fluid BT libraries are designed to have the trees precompiled before the start of simulations, like any other BT library for robotics.[5] However, in our KT-BT framework, the BT needs to be updated in real-time while the BT is being used for robot control. Specifically, the *LiveBT* should be recompiled every time a change is made in the form of new knowledge incorporated (through learning) without affecting its current execution. Therefore, we utilized the Roslyn framework's real-time compilation capabilities to address this challenge. Here, we use the *stringBT* version to recompile $\mathcal{T}_{\text{Control}}$ and update the *LiveBT*. For every change, a compiled BT is stored back in *LiveBT* and is ticked immediately.

The robots in the simulator indicate their states through the colored lights on the top, as shown in Fig. 7. For example, a robot in query mode blinks blue light, and a robot in teach mode blinks red. A complete simulator with interactive GUIs is available to test all simulation modes and strategies.[6]

C. Robot Model Architecture

A comprehensive model with the robot architecture is presented in Fig. 8. Mobile robots are equipped with sensors to detect targets, collection zones, collisions, and mechanisms for picking up and moving objects. In addition, they also feature a communications module for broadcasting and receiving queries and responses. A controller module executes the decision-making process, which generates control actions using a *LiveBT* controller. The control decisions by *LiveBT* are sent to the actuators for live actions, such as target pickup and movement.

[4][Online]. Available: https://github.com/dotnet/roslyn
[5][Online]. Available: https://www.behaviortree.dev/
[6][Online]. Available: https://github.com/herolab-uga/KTBT-Release

1) Sensing: The sensing module features a suite of four sensors for detecting targets, collection zones, robot position, and odometry. The target detector senses the presence of a target within a range D_T, as well as its color (red, green, yellow, or blue), and the collection point detector detects whether the robot is inside a collection zone. In addition, the robot's position in relation to a global coordinate system is determined through a position and odometry sensor suite, whose values are represented as a 3-tuple $\langle \text{Position}, \text{Orientation} \rangle$, where Position is a position vector and Orientation is a quaternion.

Finally, the collision detector identifies all potential collisions with neighboring robots and other objects within a designated collision field range, D_c. We utilize a vector-based approach [55], [56] for collision avoidance, such as the separation in Reynolds Flocks [57], as defined in the following:

$$F_{\text{collision}} = \begin{cases} 0 & : D_{\text{obj}} > D_c \\ 1 & : D_{\text{obj}} \leq D_c. \end{cases} \tag{9}$$

For all points of collision C in detection range D_c, a resultant vector is computed as

$$V_{\text{COL}_j} = \begin{cases} C_j - P & : F_{\text{collision}_j} = 1 \\ 0 & : F_{\text{collision}_j} = 0 \end{cases} \tag{10}$$

where $j = 1, \ldots, n(C)$ and subscripts are the indices of the jth object.

Finally, a resultant vector for all the collision vectors is computed as

$$V_c = \sum_{j=1}^{n(C)} V_{\text{COL}_j}. \tag{11}$$

2) Communications: The communications module enables generic communication channels between nearby robots within D_{coms} range. These channels can broadcast and receive messages, typically composed of queries, responses, and corresponding flags. For example, a learn tree may send a query to the message manager through the state manager for broadcasting, and the corresponding response, received through the same path, is handled by the learn tree accordingly. BTs are encoded as *stringBTs* during transfer.

3) Actions: The actions module consists of an action manager that converts the output signals from the controller into actions in the environment. Each robot is equipped with an omnidirectional movement actuator and a target pick-and-place actuation mechanism. The action manager is given various commands, such as the desired direction of movement, speed, angle of rotation, pick-up, and place. Targets that have been picked up are carried on the target carry stage located at the top of the robot, as depicted in Fig. 7.

4) Control: Robot planning, learning, and teaching are the key decisions made in the control module using BTs. The control module consists of three submodules: the state manager, high-level BT control, and the live BT module. The State Manager, akin to a blackboard [58], tracks the status of various internal and external flags and conditions. External flags correspond

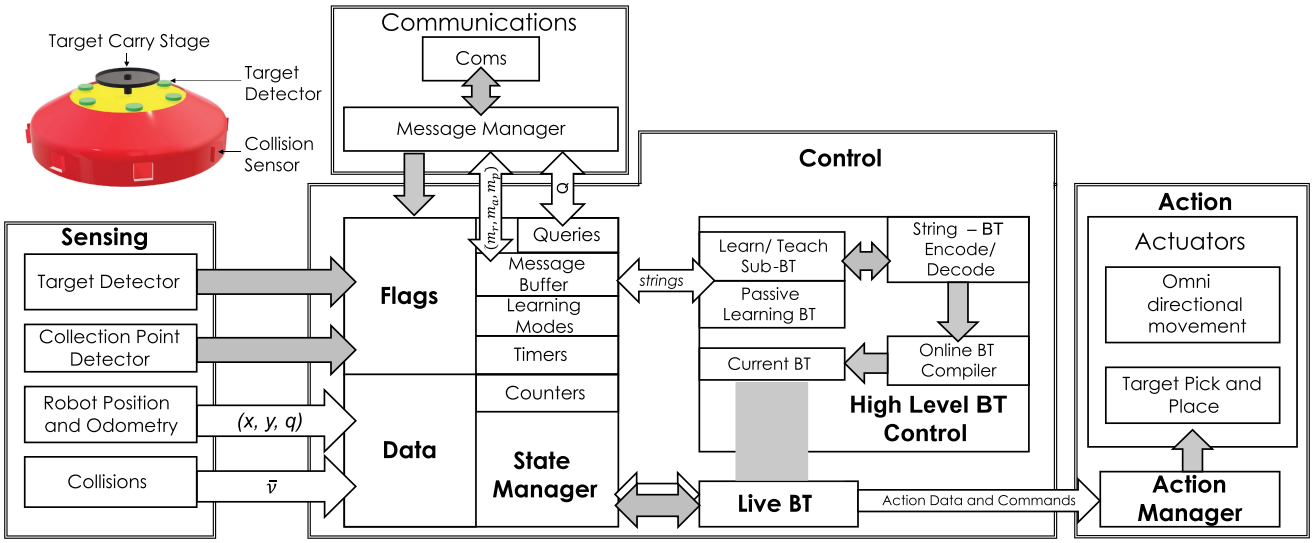

Fig. 8. 3-D Model of a Robot showing collision detectors, target detectors, and a target carry stage (inset). Robot's functional architecture shows various modules and connections between them. These modules can be broadly classified under sensing, communication, control, and action. The picture inset shows the robots interacting in the SAR simulator.

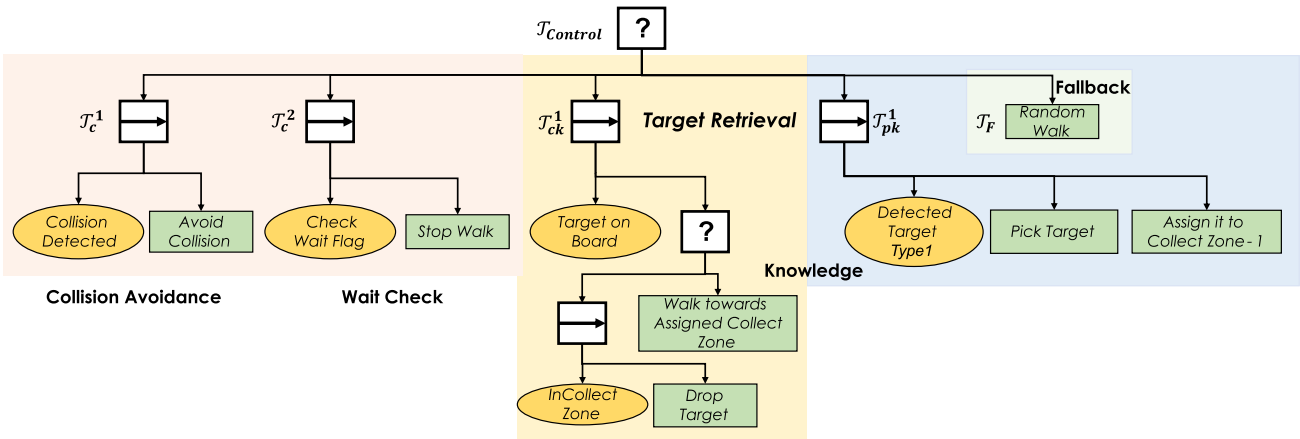

Fig. 9. Example control tree used on the robots for SAR simulation case study. (Refer to Fig. 2 for color/shape convention.).

to the state of the sensors and communicators, while internal flags represent the robot states for seamless decision-making at the BT level. In addition to these flags, State Managers also handle sensor data, such as collision vectors, robot position and odometry, communication queries, message buffers, counters, and timers.

A high-level BT control manages the core BT $\mathcal{T}_{\text{kt}}$ based on the BT outlined in Definition 1. The $\mathcal{T}_{\text{Control}}$ portion of the main BT in the high-level BT control is stored as a *stringBT*, and the corresponding actual $\mathcal{T}_{\text{Control}}$ is a compiled version of the represented *stringBT*, which is ticked at regular intervals. The controller module is designed to compile the *stringBT* whenever a change is detected in the *stringBT* version of $\mathcal{T}_{\text{Control}}$ of high-level BT control and stored as *LiveBT*. The following sections present further details on the BT design, along with a *stringBT* encoding example.

D. BT Design

The high-level BT Control submodule in the controller maintains a BT of a structure following Definition 1, i.e., the trees $\mathcal{T}_{\text{Control}}$, $\mathcal{T}_{\text{Teach}}$, and $\mathcal{T}_{\text{Learn}}$ running in parallel.

1) Control Subtree: According to Definition 2, a control subtree $\mathcal{T}_{\text{Control}}$ should contain a selector with sub-trees in the order of criticality followed by action, knowledge, and fallback sub-trees. The current robot models are designed to have two critical sub-trees. The first critical subtree $\mathcal{T}_c^1$ is designed for collision avoidance followed by the second wait subtree $\mathcal{T}_c^2$, as shown in the Fig. 9. In the collision avoidance subtree, when a collision flag is *true* in the state manager, its corresponding mean collision vector V_c is computed from (11). A unit vector in the direction $-V_c$ is computed in the *AvoidCollision* action and the corresponding control command is sent to the action manager.

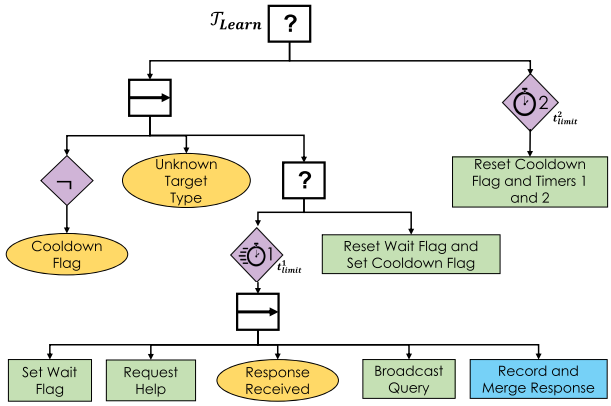

Fig. 10. Learning BT in SAR robots.

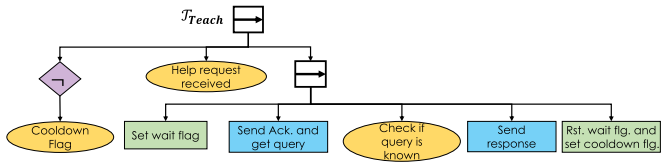

Fig. 11. Teaching BT in SAR robots.

TABLE II
BT OPERATOR EQUIVALENTS IN *FLUIDBT* AND *STRINGBT* FRAMEWORKS

BT Operator	*FluidBT*	stringBT
Sequence	.Sequence()	$< sq >$
Selector	.Selector()	$< sl >$
Parallel	.Parallel()	$< pl >$
Condition	.Condition(() $\Rightarrow$ State)	$< c >$*ConditionTag*
Action	.Action(() $\Rightarrow$Method, r)	$< a >$*ActionTag*
Wait	.Wait(WaitDuration)	$< w >$*WaitDuration*
Segmentation	.End()	$< e >$

The next subtree following the critical sub-trees sequence is the common knowledge subtree sequence $\mathcal{T}_{\text{CK}}$. In the current SAR problem, this is a target retrieval subtree, which is common across all the robots. This subtree ensures that if any target is picked up or on board, it is moved to its assigned collection zone. This action subtree $\mathcal{T}^1_{ck}$ is shown in Fig. 9.

The sub-trees following the common knowledge subtree sequence are for the prior knowledge $\mathcal{T}_{\text{PK}}$. This is a placeholder location for the prior knowledge sub-trees. For example, the knowledge subtree shown in Fig. 9, is for retrieving target type 1 (R-Red). As this subtree is already part of the control sequence, its condition sequence is also a subset of s_{ka}, and the robot, when queried, can respond with the $\mathcal{T}^1_{\text{pk}}$ as $\mathcal{T}^*_{\text{ka}}$.

Finally, the subtree to the extreme right is a fallback tree $\mathcal{T}_F$, which is executed when none of the sub-trees to the left return a success. For example, it could be a random walk routine, using which the robot chooses a random direction and walks for a certain duration.

2) Learn Tree: A Learn tree $\mathcal{T}_{\text{teach}}$ runs in parallel to the control and teaching trees. The Learn tree designed for the current SAR problem is shown in Fig. 10.

The designed learning tree, $\mathcal{T}_{\text{Learn}}$ from Definition 3, has two timers, which are decorator nodes with two different functionalities, as shown in Fig. 10. Timer type 1, is a pulse timer defined previously in Definition 5, and timer type 2 is a run timer as per Definition 6. In contrast to timer type 1, the type 2 timer runs the associated BT as long as the timer is running. Furthermore, an *unknown target type* condition in the BT in Fig. 10 verifies the sequence s when a target is encountered according to the Definition 3. If the encountered target is unknown to the robot, the resulting sequence s_q is also unknown, and the *request-help* sequence following the condition block is initiated.

The type 2 timer serves the purpose of query and wait, where the robot queries about an unknown target and waits for a duration t^2_{limit}. The BT, in this case, is designed to check the cool-down flag before executing the query sequence. This is a common flag shared between the Teach BT in Fig. 11, and hence, a cool-down flag set will run the teach and learn trees in the wait loops controlled by TT1 timers. This is to avoid repeated detection and queries on the same target when no response is received.

3) Teach Tree: The Teach subtree is similarly structured in all robots. From Definition 4, a Teach tree continuously checks for any queries being broadcast and further checks if the query is in its condition set. If found, it responds with the appropriate knowledge subtree, and additionally, a cool-down flag is checked every time a query is encountered to ensure the robot is not stuck in a teach loop when multiple robots are querying simultaneously. This cool-down flag is reset after a time t^1_{limit}, run by a TT1 decorator, as shown in Fig. 11.

E. StringBT Implementation of SAR Application

In our proposed grammar, we assume that the sets of conditions flags and actions are appropriately labeled in both state and action managers. For example, an action stating *RandomWalk* is an action routine that can be initiated with the tag *"RandomWalk."*

We developed a parser that facilitates effective communication and coordination among robots that use the *stringBT* formalism. This parser maps action labels in messages to their corresponding available action primitives in the action manager. The action labels may refer to abstract action definitions, which could involve complex action sequences that can further be decomposed into sub-trees. However, if an action label cannot be identified with an action primitive in the action manager library, the parser discards the corresponding message to prevent errors. In the future, it is possible to share the whole action primitives as well. While designing the grammar for the BT representation, the rules were written for BT encoding inline with the *FluidBT* library in Unity, and wrappers for these rules were written to convert the *stringBT* structures to the *FluidBT* codes. We summarize some of the grammar rules formulated for *stringBT* representation and *FluidBT* equivalent in Table II. An example BT representation of *stringBT* and *FluidBT* code is presented in Algorithm 4.

TABLE III
SUMMARY OF VARIOUS TYPES OF STUDIES CONDUCTED

Type of Study	Goals
No Transfer versus KT-BT	Compare performance of KT-BT and no transfer, and for configurations with and without obstacles.
Opportunities	Study the effect of opportunities on knowledge spread and performance.
Density and scalability	Study the effect of robot density and scalability on the performance.
Communication range	Study the effect of communication range on group performance, knowledge spread, and query efforts
Heterogeneity analysis	Estimate and analyze the functional heterogeneity in the group.

Algorithm 4: stringBT for $\mathcal{T}_{Control}$ in the SAR Case Study.

```
<Root>
  <sl>
    <sq><c> (_collisionDetectedF)
        <a> (CollisionAvoidance)<e>
    <sq><c> (_waitF)
        <a> (StopWalk)<e>
    <sq><c> (_treasureOnBoardF)
        <sl ><sq><c> (_inZoneF)
              <a> (PlaceTreasure)<e>
          < sq><c> (!_inZoneF)
              < a> (WalkToCollection)<e ><e><e>
    < a> (RandomWalk)<e>
```

TABLE IV
SIMULATION PARAMETERS FOR KT-BT STUDY

Parameter	Value	
Sim Mode	NT	KT-BT
Robots combination (*I, M, R, G, Y, B*)	(0, 40, 0, 0, 0, 0) (0, 0, 10, 10, 10, 10)	(0, 0, 10, 10, 10, 10)
Target combination (*R, G, Y, B*)	$(n_r, n_g, n_y, n_b) = (25, 25, 25, 25)$	
Obstacles	with and without	
Communication range	200 units	
Iterations	50000	
Trials	20	

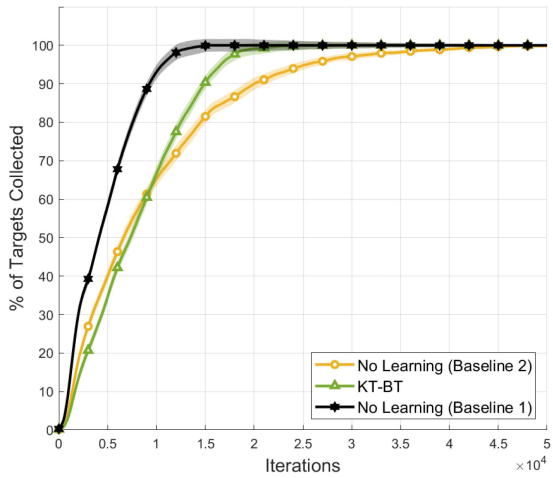

Fig. 12. Average percentage of targets collected over time over 20 trials, for No-transfer with $(0, 40, 0, 0, 0, 0)$ (BL1), No-transfer with $(0, 0, 10, 10, 10, 10)$ (BL2), and KT-BT with $(0, 0, 10, 10, 10, 10)$ group compositions.

VI. EXPERIMENTAL ANALYSIS

On the KT-BT SAR simulator, we conducted studies to understand the group performance, knowledge spread, query efforts, opportunities (number of targets available for search), the effect of communication range, and heterogeneity trends in various scenarios. These studies are summarized in Table III.

Across these studies, we maintain six different types of robots based on their prior knowledge levels. These are labeled as Ignorant (I), multitarget (M), target-red (R), target-green (G), target-yellow (Y), and target-blue (B). An ignorant robot has no prior knowledge of handling any target type. And on the other hand, a multitarget robot can handle any target type. The rest of the robot types have prior knowledge about the color they are associated with. For the current study, we use different combinations of these robots to evaluate the groups' performance. For example, a combination of (10,10,5,5,5,5) has robots of numbers in the sequence (I, M, R, G, Y, B). To maintain sufficient space for movement and avoid crowding, we kept the total number of robots at 40 across all our studies. The attached video depicts the teach-learn process, as well as provides a real-time view of the current BT status for some of the example scenarios.

A. No Transfer Versus KT-BT Study

In this study, we compare the performance of three different groups that differed in their knowledge transfer capabilities, as shown in Table IV. Base line 1 (BL1) group consists of agents with knowledge of handling any target type, and base line 2 (BL2) has agents that are evenly grouped to retrieve each target type. In BL2, agents cannot transfer knowledge; otherwise, the agents cannot query other agents for help with unknown conditions (Queries in BL1 do not arise as all the agents have complete knowledge). We compare the performance of these baseline groups with a KT-BT group that contained agent composition similar to BL2 and additionally is enabled with the knowledge transfer ability. Additionally, simulation trials were conducted in two different search spaces that varied in the presence of obstacles, as shown in Fig. 6. In both search spaces, each target type was fixed at 25, and the position of the targets was randomly varied across all the trials.

The time series graphs for the total percentage of targets collected are presented in Fig. 12, and the performance comparison is made in Fig. 13. The percentage of target collection, shown in Fig. 12, is the average in 20 trials conducted for the same robots and target compositions, but the initial positions of the targets and robots were randomly varied. It can be noted that the worst performer was the BL2 group, which lacked any knowledge transfer capabilities. Consequently, the best performers were the BL1 groups that had knowledge about all target types. The true advantage of knowledge transfer can be noticed in the performance of the KT-BT groups that are similar in composition to the BL2 groups but also could query and respond. From the time series graph Fig. 12, it can be noted that the KT-BT groups

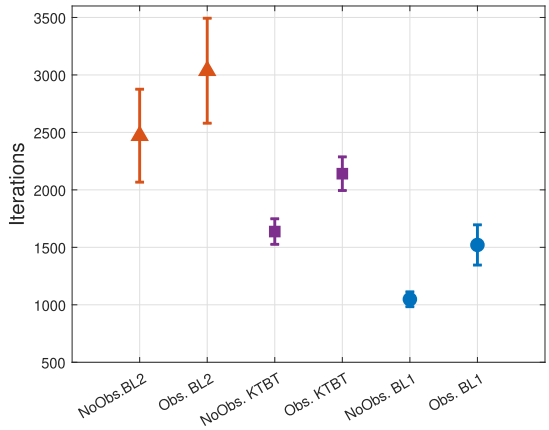

Fig. 13. Efficiency of collecting 99% of targets averaged over 20 trials for scenarios with and without obstacles.

TABLE V
SIMULATION SUMMARY FOR KNOWLEDGE SPREAD AND OPPORTUNITIES STUDY

Parameter	Value
Sim mode	KT-BT
Robots combination (*I, M, R, G, Y, B*)	(39,1,0,0,0,0)
Target combination (*R, G, Y, B*)	(10,10,10,10), (25,25,25,25) (50,50,50,50), (100,100,100,100)
Obstacles	without
Communication range	200 units
Iterations	50000
Trials	20

lagged BL2 groups initially, as the query-response process in the robots introduced delays. But going further, the performance of the KT-BT group surpassed that of BL2 as more robots learned to deal with multiple target types.

The mean performance graph over 20 trials measuring the number of iterations the groups took to collect 99% of the targets is presented in Fig. 13. The graph also shows the collection rate decreased (higher number of iterations) in with-obstacle scenarios across all the groups but followed a similar trend as the no-obstacle scenario.

B. Knowledge Spread and Opportunities Study

The objective of this study is to analyze the impact of opportunities on the dissemination of knowledge in robots. Here, opportunities refer to the number of targets present in the search space. In this study, we varied target counts between 10 and 100 for each color type, as shown in Table V. We varied the target counts in the simulations that contained a single group with one multitarget and 39 ignorant robots (called 39I-1 M group).

Whenever a target is encountered, an ignorant robot posts a query with its adjacent neighbors and awaits a response. In this case, at the beginning of the simulations, only one robot can respond to any query. As the simulation progresses, the knowledge about various targets is shared among the groups; thus, the robots learn to handle multiple targets.

We segregate robots into different levels according to the number of targets they can handle. For example, a robot that knows how to handle two types of targets is grouped under

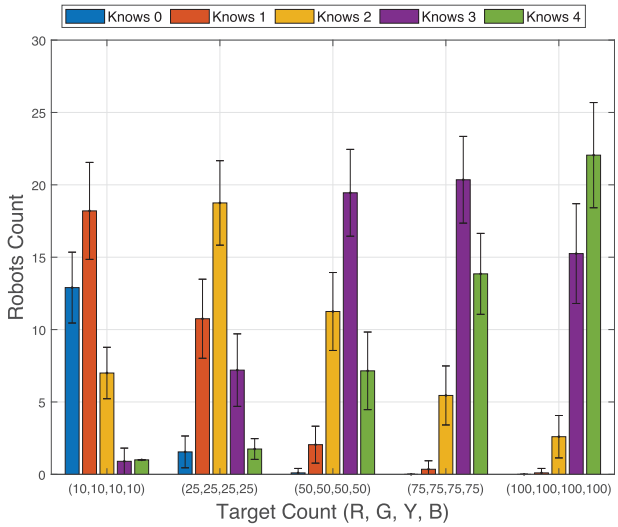

Fig. 14. Graphs showing the counts of robots at various knowledge levels and at the end of a simulation (39I-1 M).

"knows – 2"; similarly, a robot that knows how to handle all targets is grouped under "knows – 4." A robot starting in a "knows – 0" group progresses to higher level groups as more knowledge is acquired. In each trial, the final number of robots in all four groups is counted for different target counts (opportunity counts). The results of this experiment averaged over 20 trials are presented in Fig. 14.

With increasing number of targets, more robots had the opportunity to gain knowledge of multiple targets, that is, the count of robots with the knowledge to handle all four targets increased monotonically with increasing number of target opportunities, as seen in Fig. 14. The rise and drop in the counts of robots that know 3, 2, and 1 target types are because of the shift in numbers across groups when more opportunities were made available. The presented results confirm that the KT-BT framework was efficient in achieving knowledge transfer and spread in a MAS.

C. Effect of Robot Density and Scalability

We varied the number of robots to verify the scalability of our approach. In addition, this study also helps us analyze the effect of robots' density (number of robots per square meter) on the knowledge improvements in the system. Specifically, we performed two experiments. In the first experiment, we considered the baseline case of 10R-10G-10Y-10B used in Section VI-A. Here, we analyzed the performance by increasing all robots in each target knowledge from 10 per target knowledge to 50 per target knowledge in multiples of 10. In the second experiment, we consider the 39I-1 M group in Section VI-B as the baseline, and change the Ignorant robot count from 9 to 79. There were no obstacles in both experiments, and the communication range was 200 units. Also, the environment size is the same and the number of targets is set as R:25, G:25, Y:25, and B:25, same as in the capability study in Section VI-A. Accordingly, the main difference between the configurations is the number of robots per square meter (density).

TABLE VI
RESULTS OF THE SCALABILITY EXPERIMENTS

Configuration I-M-R-G-B-Y	Iterations or %Targets	Knows 0 # Robots	Knows 1 # Robots	Knows 2 # Robots	Knows 3 # Robots	Knows 4 # Robots
/-/-10-10-10-10	17620 Iter.	0	5 (12.5%)	24 (60%)	9 (22.5%)	2 (5%)
/-/-20-20-20-20	10480 Iter.s	0	18 (22.5%)	51 (63.75%)	10 (12.5%)	1 (1.25%)
/-/-30-30-30-30	9560 Iter.	0	36 (30%)	69 (57.5%)	14 (11.6%)	1 (0.8%)
/-/-40-40-40-40	8070 Iter.	0	59 (36.87%)	85 (53.12%)	16 (10%)	0
/-/-50-50-50-50	7500 Iter.	0	90 (45%)	102 (51%)	8 (4%)	0
9I-1M-/-/-/-/	12% Targets	5 (50%)	4 (40%)	0	0	1 (10%)
19I-1M-/-/-/-/	29% Targets	8 (40%)	9 (45%)	1 (5%)	1 (5%)	1 (5%)
39I-1M-/-/-/-/	50% Targets	8 (20%)	21 (52.5%)	10 (25%)	0	1 (2.5%)
59I-1M-/-/-/-/	68% Targets	16 (26.7%)	31 (51.7%)	12 (20%)	0	1 (1.6%)
79I-1M-/-/-/-/	42% Targets	38 (47.5%)	35 (43.75%)	6 (7.5%)	0	1 (1.25%)

The results of these experiments are presented in Table VI. The first five rows in the table provide the key data for the first experiment, where the 10R-10G-10Y-10B configuration from Section VI-A is the baseline. As we can see, the higher the density, the faster the target collection. This is not just due to the mere addition of robots into the system but also to the transfer of knowledge occurring in the KT-BT framework. For instance, the percentage of robots that know to handle 1 target kept increasing, with more robots able to transfer the knowledge acquired from only one multitarget robot in the beginning. Also, over time, all ignorant robots developed at least one behavior, and the efficiency improved with the number of robots. However, there was a decrease in heterogeneity (spread between robots with different knowledge levels) when more robots were in the system. This can be attributed to the fact that the higher robot density will naturally have a negative effect on the learning opportunity, confirming the reverse of the trend observed in Section VI-B. The second experiment also resulted in a similar trend, but an interesting observation here is that the knowledge transfer saturates at a certain density, and increasing the density beyond an optimal point may negatively affect (or produce reduced knowledge gain) on the system.

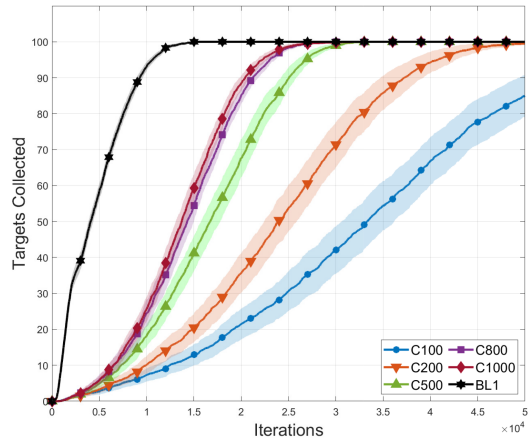

Fig. 15. Target collection rate for different communication ranges in a group with 39 Ignorant and 1 Multi-target robot.

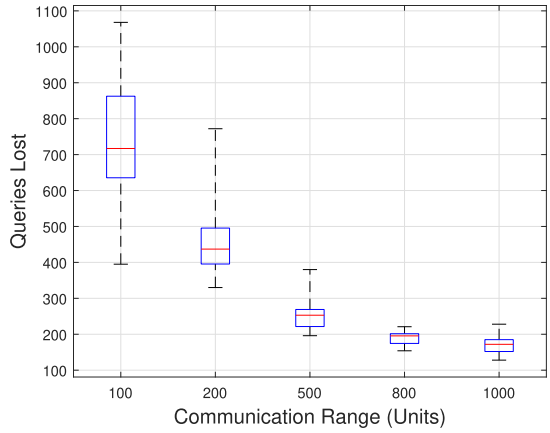

Fig. 16. Effect of communication range on the losses in the number of queries.

D. Effect of Communication Range on Knowledge Transfer

In this analysis, we varied the communication range of robots from 100 units to 1000 units. We maintained the population constant with 39 ignorant and 1 multitarget robot. The target counts and other settings are the same as set in Section VI-B. We compared the target retrieval performance with BL1 and the cumulative counts of lost queries for all different communication ranges. In target retrieval performance comparison (see Fig. 15), the performance of the test group progressively improved with the increase in communication range, approaching the ideal BL1 performance.

From our study comparison of effective communication plotted from the query loss graph in Fig. 16, it is inferred that lower communication ranges resulted in higher query losses than larger communication ranges. This suggests the need for reliable and long-range communication to improve knowledge transfer and, eventually, better group performance.

E. Knowledge and Functional Heterogeneity

We extend the knowledge propagation study to estimate the functional heterogeneity in the group. Heterogeneity is measured as a product of complexity and disparity, as proposed by Twu et al. [59], where complexity estimates how distributed the group is in its knowledge, and disparity measures how distinct these group members are in their knowledge. Complexity is computed as entropy for the distribution of agents across different species, and the disparity is computed from Rao's quadratic entropy using interspecies distance, as shown in the following equations:

$$\text{Heterogeneity} = \text{Complexity} \times \text{Disparity} \tag{12}$$

$$\text{Complexity} = E(p) = -\sum_{i=1}^{M} p_i \times \log p_i \tag{13}$$

$$\text{Disparity} = Q(p) = \sum_{i=1}^{M}\sum_{j=1}^{M} p_{ij} \times d(i,j)^2. \tag{14}$$

Here, p_i is the ratio of a species count to the total population, $d(i,j)$ is interspecies distance between agents i and j.

In the current SAR problem, heterogeneity is functionally defined through the difference in knowledge of the agents. This is similar to the computation of heterogeneity from the BTs presented in our previous work [60]. As shown in the following, we segregate the robots into four groups, each with the ability

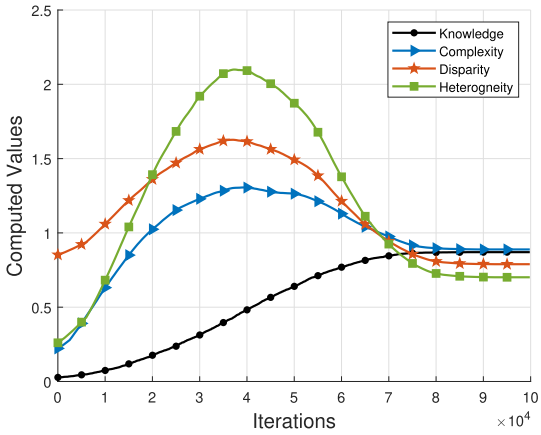

Fig. 17. Complexity, disparity, heterogeneity, and mean knowledge score changes over iterations for 39 Ignorant and 1 multitarget robot group (39I-1 M) searching for a target combination $(R, G, Y, B) = (100, 100, 100, 100)$.

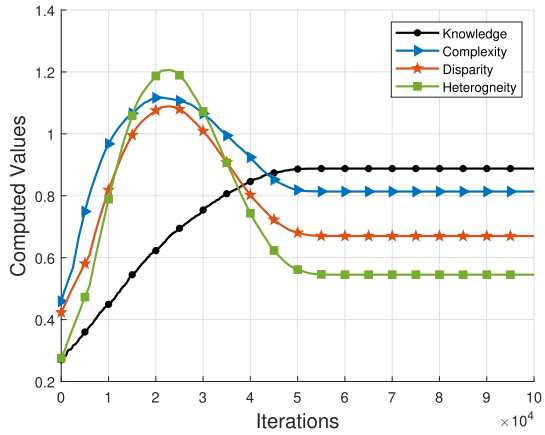

Fig. 18. Complexity, disparity, heterogeneity, and mean knowledge score changes over iterations for 10 each of R, G, Y, B knowledge robots in a group (10RGYB) searching for a target combination $(R, G, Y, B) = (100, 100, 100, 100)$.

to deal with a combination of targets:

$$\begin{aligned} g_0 &= \{\phi\} \\ g_1 &= \{k_R, k_G, k_Y, k_B\} \\ g_2 &= \{k_{RG}, k_{RY}, k_{RB}, k_{GY}, k_{GB}, k_{YB}\} \\ g_3 &= \{k_{RGY}, k_{RYB}, k_{GYB}\} \\ g_4 &= \{k_{RGYB}\} \end{aligned} \tag{15}$$

where k_R is knowledge of the red target, k_{RG} is knowledge of red and green, etc.

We define each group type as a species and interspecies distance as the knowledge distance between each set. For, e.g., the knowledge distance between g_0 and g_1 is 1 and between g_0 and g_4 is 4. This distance estimate is based on the assumption that the knowledge about all target types is similar. If the knowledge for each target type is dissimilar, the groups can be broken further and scored accordingly. Furthermore, for ease of computation, we maintain that the total ability sums to unity. E.g., in the current case, as the knowledge about the targets is similar, we assign $ks_R = ks_G = ks_Y = ks_B = 0.25$.

Based on the abovementioned knowledge scores, we compute the mean knowledge score in the group as follows:

$$\text{Mean Knowledge Score} = \frac{\sum_{i=1}^{P} ks_i}{P} \tag{16}$$

where ks_i is the total knowledge score of ith agent, and P is the total population.

We analyzed the complexity, disparity, and heterogeneity measure based on previously presented equations and compared them against the knowledge factor, as shown in Figs. 17 and 18. In Fig. 17, we present the results obtained for a group with 39 ignorant and one multitarget robot (39I-1 M). In Fig. 18, we present the results for a group with 40 population size, with members equally distributed with the knowledge to handle the R, G, Y, and B targets (10 each), (10RGYB).

In the 39I-1 M group, at the start of the simulations, the system had low complexity as there were 39 homogeneous agents and a high disparity as the knowledge level difference between the ignorant and multi-target robots is high. As robots shared knowledge, more agents moved from lower to higher levels of intelligence. Approximately halfway, while opportunities lasted, both complexity and disparity peaked as the group is now comprised of multiple robots with various levels of intelligence. Finally, the system slowly became homogeneous as all the robots' knowledge levels converged at level 4. The stagnation of heterogeneity beyond 70 k iterations is due to the lack of opportunities, which is also evident through the saturation observed in the knowledge factor.

A similar trend can also be observed in the 10RGYB combination, as shown in Fig. 18. In contrast to the previous 39I-1 M combination, the group starts with slightly higher complexity than disparity as there are four different types of robots but with a comparably lower distinction in knowledge, thus demonstrating lower functional heterogeneity. Trends similar to the 39I-1 M composition are observed in complexity, disparity, and knowledge graphs. In both cases, as the knowledge factor saturated, the heterogeneity remained constant, thus, supporting the argument of functional heterogeneity's association with knowledge and opportunities. When more opportunities are provided, when sufficient knowledge is shared, the heterogeneity measure settles at zero as all the agents have the same knowledge factor and thus forming a homogeneous group. This demonstrates the applicability of the KT-BT framework for explicit knowledge sharing tightly integrated with robot control.

F. Limitations and Implications on Hardware Implementations

The current work primarily focuses on abstract definitions of action primitives rather than the granularity of complex action primitives. By taking this approach, we can explore the current problem from the perspective of a structurally homogenous group of robots that share the same pool of action primitives. For real-world implementations, using composition of abstract action labels, the proposed *stringBT* formalism can be extended to structurally heterogeneous groups. For instance, a pick-and-place action for a manipulator involves a series of granular actions that differ from those of a mobile manipulator. Despite these differences, both robots/agents can still share

knowledge/behaviors) using the current framework by utilizing the composition of abstract action primitives.

Furthermore, some of the states/variables/flags (e.g., cool-down flag) are being accessed by multiple concurrent sub-trees (control, teach, and learn) that are executed in parallel. Here, proper design of BT nodes, such as proposed in the work [49] must be considered to make sure the concurrency issues are handled well to guarantee predictable performance.

Moreover, we chose to validate the framework in a simulation environment, since there are a few technical challenges to deploying the KT-BT framework on real-world robots. Most real-world robots use Linux-based robot operating systems (ROS[7]) as their software framework. ROS-compatible software tools[8] available currently for the design and visualization of BTs support only precompiled BT structures. This limitation does not allow dynamic (real time) updates or recompilation of BTs for knowledge updates while the BT is being used by the robot for execution. We plan to overcome this challenge by developing wrappers similar to the Roslyn framework compatible with ROS and implementing them on a swarm robotics test bed. In principle, the KT-BT framework is feasible for real-world robots by addressing the above technical challenges.

VII. Conclusion

This article introduced a new framework called KT-BT, which uses BTs to transfer knowledge (functional behaviors) between robots through direct communication. This framework can propagate and expand intelligence within a multirobot and MAS, ultimately achieving homogeneous high-potent knowledge starting from heterogeneous low-potent knowledge spread in individual robots. We established the rules for a query-response mechanism for knowledge sharing and presented mathematical analysis on knowledge transfer, knowledge spread, and opportunities. We also introduced a *stringBT* grammatical representation of BTs to facilitate BT transfer.

We demonstrated an application of the KT-BT framework on a SAR problem involving a variety of robots that search for various targets and deposit them at their corresponding collection zones. In addition, we developed a unique simulator for conducting studies on knowledge transfer, spread, the effect of knowledge transfer on overall group performance, the effect of opportunity count, and the effect of communication range. The results demonstrate successful knowledge transfers and improved group performance in various scenarios. In our future work, we plan to analyze the KT-BT framework under the contexts of memory-limited computing resources on robots and passive transfer without explicit queries.

References

[1] G. Kaplan, "Animal communication," *Wiley Interdiscipl. Reviews: Cogn. Sci.*, vol. 5, no. 6, pp. 661–677, 2014.

[2] D. Rendall, M. J. Owren, and M. J. Ryan, "What do animal signals mean?," *Animal Behav.*, vol. 78, no. 2, pp. 233–240, 2009.

[3] J. Van Diggelen, R.-J. Beun, F. Dignum, R. M. Van Eijk, and J.-J. Meyer, "Ontology negotiation in heterogeneous multi-agent systems: The anemone system," *Appl. Ontol.*, vol. 2, no. 3/4, pp. 267–303, 2007.

[4] A. Sharma, D. Srinivasan, and D. S. Kumar, "A comparative analysis of centralized and decentralized multi-agent architecture for service restoration," in *Proc. IEEE Congr. Evol. Comput.*, 2016, pp. 311–318.

[5] A. Whiten, D. Biro, N. Bredeche, E. C. Garland, and S. Kirby, "The emergence of collective knowledge and cumulative culture in animals, humans and machines," *Philos. Trans. Roy. Soc. B*, vol. 377, no. 1843, 2022, Art. no. 20200306.

[6] L. O. Wilson, "Anderson and Krathwohl Bloom's taxonomy revised understanding the new version of Bloom's taxonomy," in *Second Princ.*, vol. 1, no. 1, pp. 1–8, 2016.

[7] M. O. Riedl, "Human-centered artificial intelligence and machine learning," *Hum. Behav. Emerg. Technol.*, vol. 1, no. 1, pp. 33–36, 2019.

[8] J. Hautala, "Can robots possess knowledge? Rethinking the DIK(W) pyramid through the lens of employees of an automotive factory," *Humanities Social Sci. Commun.*, vol. 8, no. 1, pp. 1–10, 2021.

[9] T. Fitzgerald, K. McGreggor, B. Akgun, A. K. Goel, and A. L. Thomaz, "A visual analogy approach to source case retrieval in robot learning from observation," in *Proc. Workshops 28th AAAI Conf. Artif. Intell.*, 2014.

[10] A. Neupane and M. A. Goodrich, "Learning swarm behaviors using grammatical evolution and behavior trees," in *Proc. Int. Joint Conf. Artif. Intell.*, 2019, pp. 513–520.

[11] M. Colledanchise, R. Parasuraman, and P. Ögren, "Learning of behavior trees for autonomous agents," *IEEE Trans. Games*, vol. 11, no. 2, pp. 183–189, Jun. 2019.

[12] Y. Liang and B. Li, "Parallel knowledge transfer in multi-agent reinforcement learning," 2020, *arXiv:2003.13085*.

[13] T. Fitzgerald, K. Bullard, A. Thomaz, and A. Goel, "Situated mapping for transfer learning," in *Proc. 4th Annu. Conf. Adv. Cogn. Syst.*, 2016, pp. 1–14.

[14] G. Flórez-Puga, M. A. Gómez-Martín, P. P. Gómez-Martín, B. Díaz-Agudo, and P. A. González-Calero, "Query-enabled behavior trees," *IEEE Trans. Comput. Intell. AI Games*, vol. 1, no. 4, pp. 298–308, Dec. 2009.

[15] M. Racca, A. Oulasvirta, and V. Kyrki, "Teacher-aware active robot learning," in *Proc. IEEE/ACM 14th Int. Conf. Hum.-Robot Interact.*, 2019, pp. 335–343.

[16] H. Ravichandar, A. S. Polydoros, S. Chernova, and A. Billard, "Recent advances in robot learning from demonstration," *Annu. Rev. Control, Robot., Auton. Syst.*, vol. 3, pp. 297–330, 2020.

[17] Y. A. Sekhavat, "Behavior trees for computer games," *Int. J. Artif. Intell. Tools*, vol. 26, no. 2, 2017, Art. no. 1730001.

[18] M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and AI: An Introduction*.Boca Raton, FL, USA: CRC Press, 2018.

[19] M. Iovino, E. Scukins, J. Styrud, P. Ögren, and C. Smith, "A survey of behavior trees in robotics and AI," *Robot. Auton. Syst.*, vol. 154, 2022, Art. no. 104096.

[20] G. K. Soon, C. K. On, P. Anthony, and A. R. Hamdan, "A review on agent communication language," *Lecture Notes Elect. Eng.*, vol. 481, pp. 481–491, 2019.

[21] V. Tamma and T. Bench-Capon, "An ontology model to facilitate knowledge-sharing in multi-agent systems," *Knowl. Eng. Rev.*, vol. 17, no. 1, pp. 41–60, 2002.

[22] A. Dorri, S. S. Kanhere, and R. Jurdak, "Multi-agent systems: A survey," *IEEE Access*, vol. 6, pp. 28573–28593, 2018.

[23] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowl. Acquisition*, vol. 5, no. 2, pp. 199–220, 1993.

[24] S. Staab and R. Studer, *Handbook on Ontologies*. Berlin, Germany: Springer, 2010.

[25] C. Schlenoff et al., "An IEEE standard ontology for robotics and automation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 1337–1342.

[26] J. I. Olszewska et al., "Ontology for autonomous robotics," in *Proc. RO-MAN 26th IEEE Int. Symp. Robot Hum. Interactive Commun.*, 2017, pp. 189–194.

[27] S. R. Fiorini et al., "A suite of ontologies for robotics and automation [industrial activities]," *IEEE Robot. Autom. Mag.*, vol. 24, no. 1, pp. 8–11, Mar. 2017.

[28] H. Kono, A. Kamimura, K. Tomita, and T. Suzuki, "Transfer learning method using ontology for heterogeneous multi-agent reinforcement learning," *Int. J. Adv. Comput. Sci. Appl.*, vol. 5, no. 10, pp. 156–164, 2014.

[7][Online]. Available: https://www.ros.org/

[8][Online]. Available: https://github.com/BehaviorTree/BehaviorTree.CPP

[29] S. Qu, J. Wang, S. Govil, and J. O. Leckie, "Optimized adaptive scheduling of a manufacturing process system with multi-skill workforce and multiple machine types: An ontology-based, multi-agent reinforcement learning approach," *Procedia CIRP*, vol. 57, pp. 55–60, 2016.
[30] M. Oprea, "Agent-based modelling of multi-robot systems," in *Proc. IOP Conf. Ser.: Mater. Sci. Eng.*, vol. 444, no. 5. 2018, Art. no. 052026.
[31] A. Taylor, I. Dusparic, M. Gueriau, and S. Clarke, "Parallel transfer learning in multi-agent systems: What, when and how to transfer?," in *Proc. Int. Joint Conf. Neural Netw.*, 2019.
[32] S. Chernova and M. Veloso, "Confidence-based multi-robot learning from demonstration," *Int. J. Social Robot.*, vol. 2, no. 2, pp. 195–215, 2010.
[33] Y. Liu, Y. Hu, Y. Gao, Y. Chen, and C. Fan, "Value function transfer for deep multi-agent reinforcement learning based on n-step returns," in *Proc. Int. Joint Conf. Artif. Intell.*, 2019, pp. 457–463.
[34] W.-C. Jiang, V. Narayanan, and J.-S. Li, "Model learning and knowledge sharing for cooperative multiagent systems in stochastic environment," *IEEE Trans. Cybern.*, vol. 51, no. 12, pp. 5717–5727, Dec. 2021.
[35] L. Zhou, P. Yang, C. Chen, and Y. Gao, "Multiagent reinforcement learning with sparse interactions by negotiation and knowledge transfer," *IEEE Trans. Cybern.*, vol. 47, no. 5, pp. 1238–1250, May 2017.
[36] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ogren, "Towards a unified behavior trees framework for robot control," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2014, pp. 5420–5427.
[37] E. Giunchiglia, M. Colledanchise, L. Natale, and A. Tacchella, "Conditional behavior trees: Definition, executability, and applications," in *Proc. IEEE Int. Conf. Syst., Man Cybern.*, 2019, pp. 1899–1906.
[38] F. Rovida, B. Grossmann, and V. Kruger, "Extended behavior trees for quick definition of flexible robotic tasks," in *Proc. IEEE Int. Conf. Intell. Robots Syst.*, 2017, pp. 6793–6800.
[39] M. Colledanchise, A. Marzinotto, D. V. Dimarogonas, and P. Oegren, "The advantages of using behavior trees in multi-robot systems," in *Proc. ISR: 47st Int. Symp. Robot.*, 2016, pp. 1–8.
[40] Q. Yang, Z. Luo, W. Song, and R. Parasuraman, "Self-reactive planning of multi-robots with dynamic task assignments," in *Proc. Int. Symp. Multi-Robot Multi-Agent Syst.*, 2019, pp. 89–91.
[41] Q. Yang and R. Parasuraman, "Needs-driven heterogeneous multi-robot cooperation in rescue missions," in *Proc. IEEE Int. Symp. Saf., Secur., Rescue Robot.*, 2020, pp. 252–259.
[42] R. A. Agis, S. Gottifredi, and A. J. García, "An event-driven behavior trees extension to facilitate non-player multi-agent coordination in video games," *Expert Syst. With Appl.*, vol. 155, 2020, Art. no. 113457.
[43] A. Pacheck, S. Moarref, and H. Kress-Gazit, "Finding missing skills for high-level behaviors," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2020, pp. 10335–10341.
[44] S. Li, D. Park, Y. Sung, J. A. Shah, and N. Roy, "Reactive task and motion planning under temporal logic specifications," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2021, pp. 12618–12624.
[45] J. Styrud, M. Iovino, M. Norrlöf, M. Björkman, and C. Smith, "Combining planning and learning of behavior trees for robotic assembly," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2022, pp. 11511–11517.
[46] E. Ferrante, A. E. Turgut, E. Duéñez-Guzmán, M. Dorigo, and T. Wenseleers, "Evolution of self-organized task specialization in robot swarms," *PLoS Comput. Biol.*, vol. 11, no. 8, 2015, Art. no. e1004273.
[47] M. Nanjanath and M. Gini, "Dynamic task allocation for robots via auctions," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2006, pp. 2781–2786.
[48] Z. A. Saigol et al., "Facilitating cooperative AUV missions: Experimental results with an acoustic knowledge-sharing framework," in *Proc. OCEANS - San Diego*, 2013, pp. 1–7.
[49] M. Colledanchise and L. Natale, "Handling concurrency in behavior trees," *IEEE Trans. Robot.*, vol. 38, no. 4, pp. 2557–2576, Aug. 2022.
[50] M. Colledanchise and P. Ögren, "How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees," *IEEE Trans. Robot.*, vol. 33, no. 2, pp. 372–389, Apr. 2017.
[51] K. French, S. Wu, T. Pan, Z. Zhou, and O. C. Jenkins, "Learning behavior trees from demonstration," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 7791–7797.
[52] G. Suddrey, B. Talbot, and F. Maire, "Learning and executing re-usable behaviour trees from natural language instruction," vol. 7, no. 4, pp. 10643–10650, Oct. 2022.
[53] D. A. Shell and M. J. Mataric, "On foraging strategies for large-scale multi-robot systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2006, pp. 2717–2723.
[54] J. Harwell and M. Gini, "Broadening applicability of swarm-robotic foraging through constraint relaxation," in *Proc. IEEE Int. Conf. Simul., Model., Program. Auton. Robots*, 2018, pp. 116–122.
[55] V. Kunchev, L. Jain, V. Ivancevic, and A. Finn, "Path planning and obstacle avoidance for autonomous mobile robots: A review," in *Proc. 10th Int. Conf. Knowl.-Based Intell. Inf. Eng. Syst.*, 2006, pp. 537–544.
[56] J. A. Oroko and G. Nyakoe, "Obstacle avoidance and path planning schemes for autonomous navigation of a mobile robot: A review," in *Proc. Sustain. Res. Innov. Conf.*, 2022, pp. 314–318.
[57] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proc. 14th Annu. Conf. Comput. Graph. Interactive Techn.*, 1987, pp. 25–34.
[58] R. Ghzouli, T. Berger, E. B. Johnsen, S. Dragule, and A. Wąsowski, "Behavior trees in action: A study of robotics applications," in *Proc. 13th ACM SIGPLAN Int. Conf. Softw. Lang. Eng.*, 2020, pp. 196–209.
[59] P. Twu, Y. Mostofi, and M. Egerstedt, "A measure of heterogeneity in multi-agent systems," in *Proc. Amer. Control Conf.*, 2014, pp. 3972–3977.
[60] S.S. O.V.R. Parasuraman and R. Pidaparti, "Impact of heterogeneity in multi-robot systems on collective behaviors studied using a search and rescue problem," in *Proc. IEEE Int. Symp. Saf., Secur., Rescue Robot.*, 2020, pp. 290–297.

Sanjay Sarma Oruganti Venkata received the Ph.D. degree in electrical and computer engineering from the University of Georgia, Athens, GA, USA, in 2022.

He is currently a Postdoctoral Researcher with Rensselaer Polytechnic Institute, Troy, NY, USA, where he is researching integrating knowledge-based systems with transformer-based LLMs in the field of cognitive sciences. Formerly, he researched heterogeneity and knowledge transfer strategies in multi-agent and multirobot systems. He was with GVP-SIRC, Indore, India, where he developed opto-mechatronic systems for environmental monitoring. His research interests include heterogeneous multirobot systems, knowledge representation, large language models, robotics, mechatronics, and optomechatronic systems.

Ramviyas Parasuraman (Senior Member, IEEE) received the Ph.D. degree in robotics and automation from TU Madrid, Madrid, Spain, in 2014.

He is currently an Assistant Professor with the University of Georgia, Athens, GA, USA. He directs the Heterogeneous Robotics Research Lab, which conducts cutting-edge research in heterogeneous multirobot systems, search and rescue robotics, swarm robotics, and human–robot interfaces. He conducted doctoral research with the European Organization for Nuclear Research (CERN), Switzerland. Before joining the University of Georgia as an Assistant Professor in 2018, he was a Postdoctoral Researcher with Purdue University, West Lafayette, IN, USA, and the KTH Royal Institute of Technology, Stockholm, Sweden.

Dr. Parasuramane was a recipient of the prestigious Marie Skłdowska Curie ESR Fellowship.

Ramana Pidaparti received the Ph.D. degree in aeronautics and astronautics from Purdue University, West Lafayette, IN, USA, in 1989.

Previously, he has been on the faculty at Virginia Commonwealth University, Richmond, VA, USA, as well as with the Purdue University School of Engineering and Technology, Indianapolis, IN, USA. He is currently a Professor and Distinguished Faculty Scholar with the College of Engineering, University of Georgia, Athens, GA, USA. His research interests include broad areas of multidisciplinary design innovation, computational biomechanics and informatics, bioinspired materials and structures, and STEM education.

Dr. Pidaparti is a member of several professional societies including Fellow of American Association for the Advancement of Science, Fellow of Royal Aeronautical Society, Fellow of American Society of Mechanical Engineers, Associate Fellow of American Institute of Aeronautics and Astronautics, and a member of American Society of Engineering Education.