Using Behavior Trees in Multi-Robot Systems

Michele Colledanchise, Alejandro Marzinotto, Dimos V. Dimarogonas, Petter Ögren,
Centre for Autonomous Systems - KTH - Sweden

Abstract

Multi-robot teams offer possibilities of improved performance and fault tolerance, compared to single robot solutions. In this paper, we show how to realize those possibilities when starting from a single robot system controlled by a Behavior Tree (BT). By extending the *single robot BT* to a *multi-robot BT*, we are able to combine the fault tolerant properties of the BT, in terms of built-in fallbacks, with the fault tolerance inherent in multi-robot approaches, in terms of a faulty robot being replaced by another one. Furthermore, we improve performance by identifying and taking advantage of the opportunities of parallel task execution, that are present in the single robot BT. Analyzing the proposed approach, we present results regarding how mission performance is affected by *minor faults* (a robot losing one capability) as well as *major faults* (a robot losing all its capabilities).

1 Introduction

In this paper we analyze the advantages of using Behavior Trees (BTs) in a multi-robot scenario. BTs are a recent alternative to Controlled Hybrid Systems (CHSs) for reactive fault tolerant execution of robot tasks, and they were first introduced in the computer gaming industry [1] to meet their needs of modularity, flexibility and reusability in artificial intelligence for in-game opponents as described in [2, 3]. Their popularity is due to BTs being easy to understand, having a recursive structure, and being easy to use, which is creating a growing attention in academia. A number of studies highlight the advantages of BTs over more classical CHSs in robotic applications [4, 5, 6], while other describe the advantages of BTs in various applications [7, 8, 9, 10, 11, 12]. BTs are often used to describe fully reactive systems in a convenient and compact way [6] as well as safe and robust behaviors [5]. However the literature lack studies on BTs in a multi-robot scenario.

Imagine a search and rescue robot for indoor environments. The robot can explore the environment and can recognize, resuscitate and evacuate found survivors. However, this robot is fairly complex and easily breaks down. Thus, it is desirable to replace this big versatile robot with a team of smaller specialized robots. This paper shows how to modify the single robot BT of the original robot, to a decentralized heterogeneous multi-robot BT, to be run on all new robots, thereby improving both fault tolerance and performance of the original controller. The fault tolerance of the single robot BT, in terms of *fallbacks*, is improved by also adding the tolerance afforded by the redundancy achieved through having multiple robots, and the performance of the single robot BT is improved by executing the right tasks in parallel. In detail, the BT includes both so-called *sequence* and *fallback* compositions, as explained in [6]. Some tasks that were earlier done in sequence, such as exploring different areas, are now done in parallel, while other tasks that were also done in sequence, such as resuscitating a survivor and then evacuating him, are still done in a sequence.

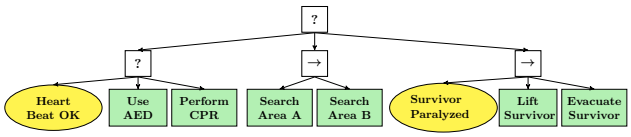


Figure 1 Single agent tree $\mathscr{T}_S$ of Example 3.

Similarly, some tasks that were earlier done as fallbacks, such as search for survivors if none needs a resuscitation, are now done in parallel, while other fallbacks, such as resuscitate a survivor performing a mechanical Cardio-Pulmonary Resuscitation (CPR) or using a Automated External Defibrillator (AED), are not done in parallel. Thus fault tolerance as well as performance is improved

The contribution of this paper is that we show how fault tolerance and performance can be improved in a single robot BT, by adding more robots to the team and extending the BT into a multi-robot BT, using two important modifications. First we add a task assignment functionality, and then we identify and adapt the sequences and fallback compositions of the BT that are possible to execute in parallel.

The outline of this paper is as follows. First, in Section 2 we review related work. Then, in Section 3.1 we give an overview of the classical formulation of BTs. The main problem is stated in Section 4 and the proposed solution is given in Section 5. We conclude the paper in Section 7.

2 Related Work

BTs are a recent alternative to Controlled Hybrid Systems (CHSs) for reactive fault tolerant execution of robot tasks, and they were first introduced in the computer gaming industry [1, 13, 14], to meet their needs of modularity, flexibility and reusability in artificial intelligence for in-game opponents. Their popularity lies in their ease of understanding, their recursive structure, and their usability, creating a growing attention in academia [15, 16, 17, 18, 19, 4, 20, 21]. In most cases, CHSs have memoryless transitions, i.e. there is no information where the transition took place from, a so-called *one way control transfer*. In BTs the equivalent of state transitions are governed by calls and return values being sent by parent/children in the tree structure, this information passing is called a *two way control transfers*. In programming languages, the replacement of a one way (e.g., GOTO statement) with a two way control transfer (i.e., Function Calls) made an improvement in readability and reusability [22]. Thus, BTs exhibit similar advantages as gained moving from GOTO to Function Calls in programming in the 1980s. Note however, that we do *not* claim that BTs are better than CHSs from a purely theoretical standpoint. Instead, the main advantage of using BTs comes from their modularity and (re)usability, [6]. BTs were first used in high profile computer games, such as the HALO series [1, 13]. Later work merged machine learning techniques with BTs' logic [15, 16], and made them more flexible in terms of parameter passing [17]. The advantage of BTs as compared to a general Finite State Machines (FSMs) was also the reason for extending the JADE agent Behavior Model with BTs in [18], and the benefits of using BTs to control complex missions of Unmanned Areal Vehicles (UAVs) was described in [6]. In [4] the modular structure of BTs were used to address the formal verification of mission plans. In [19], BTs were used to perform autonomous robotic grasping. In particular, it was shown how BTs enabled the easy composition of primitive actions into complex and robust manipulation programs. Finally, in [5] performance measures, such as success/failure probabilities and execution times, were estimated for plans using BTs.

In multi agent systems research, the problem of defining local tasks to achieve a global specification has been investigated [23, 24, 25]. Moreover, [26, 27] introduce the concept of task delegation among agents in a multi-agent system, dividing task specification in *closed delegation*, where goal and plan are predefined, and *open delegation* where either only the goal is specified while the plan can be chosen by the agent, or the specified plan describes abstractly what actions have to be taken, giving to the agent some freedom in terms of how to perform the delegated task. In [28], it is shown how verifying the truth of preconditions on single agents becomes equivalent to checking the fulfillment of a global robot network through recursive calls, using a tree structure called *Task Specification Trees*.

In this paper, we go beyond the related work, by showing how a BT for a multi-robot system can be created from a single robot BT, preserving the advantages aforementioned in addition to the scalability that distinguishes a general multi-robot system.

3 Background

In this section we describe BTs for single robot execution and one of the problems of using FSMs for multi robot execution.

3.1 Single Robot BTs

Here, we give a brief overview of BTs, and refer to [6] for a more detailed description.

A BT is defined as a directed rooted tree where nodes are grouped into control flow nodes, execution nodes, and a root node. In a pair of connected nodes we call the outgoing node *parent* and the incoming node *child*. Then, the root node has no parents and only one child, the control flow nodes have one parent and at least one child, and the execution nodes are the leaves of the tree (i.e. they have no children and one parent). Graphically, the children of a control flow node are sorted from its bottom left to its bottom right, as depicted in Figures 2-4. The execution of a BT starts from the root node, sending *ticks* [1] to its child. When a generic node in a BT receives a tick from its parent, its execution starts and it returns to its parent a status *running* if its execution has not finished yet, *success* if its execution is accomplished (i.e. the execution ends without failures), or *failure* otherwise.

Here we draw a distinction between three types of control flow nodes (selector, sequence, and parallel) and between two types of execution nodes (action and condition). Their execution is explained below.

Selector (also known as Fallback)

When the execution of a selector node starts (i.e. the node receives a tick from its parent), then the node's children are executed in succession from left to right, until a child returning success or running is found. Then this message is returned to the parent of the selector. It returns failure only when all the children return a status failure. The purpose of the selector node is to robustly carry out a task that can be performed using several different approaches (e.g. a motion tracking task can be made using either a 3D camera or a 2D camera) by trying each of them in succession until one succeeds. The graphical representation of a selector node is a box with a "?", as in Fig. 2.

A finite number of BTs $\mathscr{T}_1, \mathscr{T}_2, \ldots, \mathscr{T}_N$ can be composed into a more complex BT using the selector composition: $\mathscr{T}_0 = \text{Selector}(\mathscr{T}_1, \mathscr{T}_2, \ldots, \mathscr{T}_N)$.

Sequence

When the execution of a sequence node starts, then the node's children are executed in succession from left to right, returning to its parent a status failure (running) as

[1] A tick is a signal that enables the execution of a child.

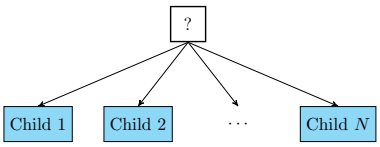


Figure 2 Graphical representation of a selector node with N children.

soon as the a child that returns failure (running) is found. It returns success only when all the children return success. The purpose of the sequence node is to carry out the tasks that are defined by a strict sequence of sub-tasks, in which all have to succeed (e.g. a mobile robot that has to move to a region “A" and then to a region “B"). The graphical representation of a sequence node is a box with a “$\rightarrow$", as in Fig. 3.
A finite number of BTs $\mathscr{T}_1, \mathscr{T}_2, \ldots, \mathscr{T}_N$ can be composed into a more complex BT using the sequence composition: $\mathscr{T}_0 = \text{Sequence}(\mathscr{T}_1, \mathscr{T}_2, \ldots, \mathscr{T}_N)$.

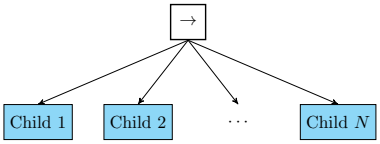


Figure 3 Graphical representation of a sequence node with N children.

Parallel

When the execution of a parallel node starts, then the node’s children are executed in succession from left to right without waiting for a return status from any child before ticking the next one. It returns success if a given number of children $M \in \mathbb{N}$ return success, it returns failure when the children that return running and success are not enough to reach the given number, even if they would all return success. It returns running otherwise. The purpose of the parallel node is to model those tasks separable in independent sub-tasks performing non conflicting actions (e.g. the tracking of multiple objects can be performed using several cameras). The parallel node is graphically represented by a box with “$\rightrightarrows$" with the number M on top left, as in Fig. 4.
A finite number of BTs $\mathscr{T}_1, \mathscr{T}_2, \ldots, \mathscr{T}_N$ can be composed into a more complex BT, with them as children, using the parallel composition: $\mathscr{T}_0 = \text{Parallel}(\mathscr{T}_1, \mathscr{T}_2, \ldots, \mathscr{T}_N, M)$.

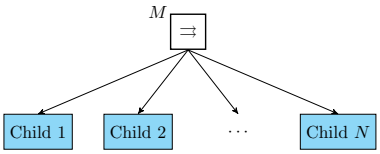


Figure 4 Graphical representation of a parallel node with N children.

a Action node. **b** Condition node.

Figure 5 Graphical representation of action and condition nodes.

Action

When an action node starts its execution, then it returns success if the action is completed and failure if the action cannot be completed. Otherwise it returns running. The action node is represented in Fig. 5a

Condition

The condition node checks if a condition is satisfied or not. The return status is success or failure accordingly and it is never running. The condition node is represented in Fig. 5b.

Root

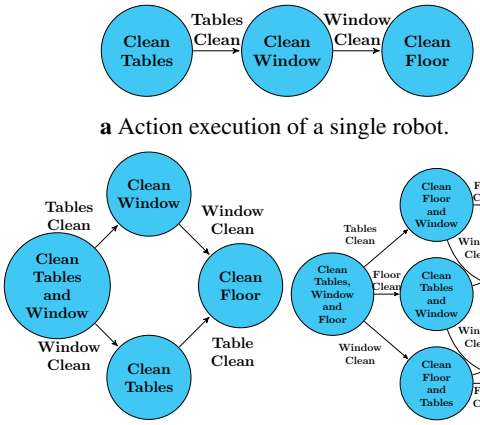
The root node is the node that generates ticks. It is graphically represented by a box labeled with “$\varnothing$”.

3.2 FSMs for Multi Agent Mission Execution

Here we describe the drawbacks of using FSMs to control a multi robot system. Let $\mathscr{S}$ be a FSM with $n \in \mathbb{N}$ states describing an action execution carried out by a single robot. Now let $m \in \mathbb{N}$ be the number of robots of a multi-robot system that has to perform the task done in $\mathscr{S}$. The classical way to derive an action execution for the multi-robot system is to compute the product of $\mathscr{S}$ with itself m times and removing the states describing a situation where two or more robots are performing the same action. This operation requires to compute n^m new states. Once we have computed the new states, we have to analyze each of them to evaluate which state has to be removed. This process, despite being trivial to understand, is impractical for large systems. In fact, as highlighted in [29], this is a case of the *curse of dimensionality* described in [30].

Example 1 *To illustrate the aforementioned drawback, we consider the following problem. A cleaning robot has to clean three surfaces: table, window, and floor. The execution order can be disregarded. The FSM in Fig. 6a describes a possible execution. Since there is only one single robot, it has to clean the table first then the window then the floor. Now lets consider the case where we have a two or three robots that have to perform the same task. Fig. 6b and Fig. 6c shows the corresponding FSMs.*

As can be seen from the example above. The control system complexity grows significantly with the number of robots.



a Action execution of a single robot.

b Action execution of a two-robot system.

c Action execution of a three-robot system.

Figure 6 I will make these with the black magic

4 Problem Formulation

The first problem considered in this work is to define a systematic procedure to translate a single robot BT to a set of BTs, one for each robot of a heterogeneous multi-robot system, deployed to achieve the same global goal as the initial single robot.

Problem 1 *Given a multi-robot system and a BT $\mathcal{T}_S$ defined for a single robot system, design a BT for the multi-robot system*

As described above, replacing a single robot system with a multi robot system can improve performance and robustness. However, to be economically viable, the individual robots often needs to be made smaller, and less capable than the initial robot. Thus we need to design a system where some tasks require the cooperation of several robots to be completed.

Problem 2 (Less capable robots) *Solve Problem 1 with robots that have the following capabilites.*
Let $\mathcal{S}$ be a set of symbols describing atomic actions (e.g. perform grasp, go to position) and let $\mathcal{R} \in \mathbb{N}$ be the set of robots in a multi-robot system, each of which can perform a finite collection of local tasks $\mathcal{L}_i \subseteq \mathcal{S}$, $i \in \mathcal{R}$ and $n = |\mathcal{R}|$ and let $\bar{\mathcal{L}} = \bigcup_{i \in \mathcal{R}} \mathcal{L}_i$ be the set of all the local tasks. Now let $\mathcal{G} = \{g_1, g_2, \ldots, g_v\}$ be a finite collection of global tasks executable by the multi-robot system and P_k be the set of global tasks running in parallel with g_k. We define $\psi(g_k)$ as the finite set of local tasks which have to return success for the global task g_k to succeed.
Finally, let $\nu : \mathcal{G} \times \bar{\mathcal{L}} \to \mathbb{N}$ and $\mu : \mathcal{G} \times \bar{\mathcal{L}} \to \mathbb{N}$ respectively be functions that give the minimum number of robots needed, and the maximum number of robots assignable, to perform a local task, in order to accomplish a global task.

In order for the problem above to have a solution, we need to make the following assumptions:

Assumption 1 *The minimum number of robots required to execute global tasks running in parallel does not exceed the total number of robots in the system: $\sum_{g_k \in P_h} \sum_{l_j \in \psi(g_k)} \nu(g_k, l_j) \leq n \; \forall g_h \in \mathcal{G}$.*

Assumption 2 *Each global task in $\mathcal{G}$ can be performed by assigning to some robots in $\mathcal{R}$ one of their local tasks: $\psi(g_k) \subseteq 2^{\bar{\mathcal{L}}} \; \forall g_k \in \mathcal{G}$.*

5 Proposed Solution

In this section, we will first give an informal description of the proposed solution, involving the three subtrees $\mathcal{T}_G$ (global tasks), $\mathcal{T}_A$ (task assignment) and $\mathcal{T}_{L_i}$(local tasks). Then, we describe the design of these subtrees in detail. To illustrate the approach we give a brief example.

Algorithm

1. Create $\mathcal{T}_A$ according to the definition below.
2. Create $\mathcal{T}_G$ according to the definition below.
3. Create $\mathcal{T}_{Li}$ according to the definition below.
4. Let the overall BT be given by

$$\mathcal{T}_i = \text{Parallel}(\mathcal{T}_A, \mathcal{T}_G, \mathcal{T}_{Li}, 3) \tag{1}$$

As can be seen, the multi agent BT is composed of three subtrees running in parallel, returning success only if all three subtrees return success. $\mathcal{T}_G$ is illustrated in Fig. 8b, $\mathcal{T}_A$ is illustrated in Fig. 8a, $\mathcal{T}_{Li}$ is illustrated in Fig. 9.

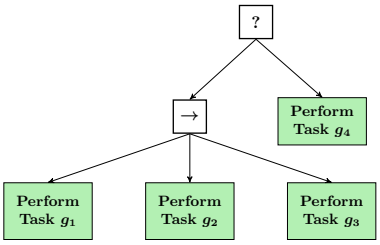
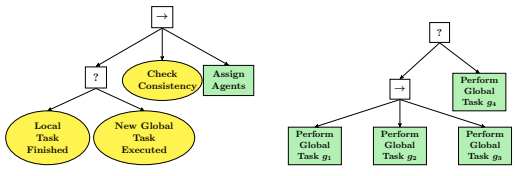


Figure 7



a **b**

Figure 8 The Task Assignment tree $\mathcal{T}_A$ (a) and an example of a global tasks tree $\mathcal{T}_G$ (b)

Remark 1 *Note that all robots run identical copies of $\mathcal{T}_i$, including the computation of assignments for all robots in the team, but each robot only executes the task that they themselves are assigned to, according to their own computation.*

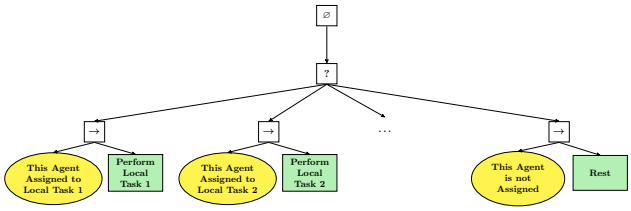


Figure 9 The Local task execution tree $\mathscr{T}_{Li}$ (assuming a numbering of the tasks).

The first subtree, $\mathscr{T}_A$ is doing task assignment. Parts of the performance and fault tolerance of the proposed approach comes from this feature, making sure that a broken robot is replaced and that robots are assigned to the tasks they do best.
The second subtree, $\mathscr{T}_G$ takes care of the overall mission progression and is created from the given single robot BT. This tree includes information on task that needs to be done in sequence (using sequence composition), tasks that could either be done in sequence or in parallel (using parallel $M = N$ compositions), fallback tasks that should be tried in sequence (using selector compositions), and fallbacks that can be tried in parallel (using parallel $M = 1$ compositions). The execution of $\mathscr{T}_G$ provides information to the assignment in $\mathscr{T}_A$. The fault tolerance of the proposed approach is improved by the fallbacks encoded in $\mathscr{T}_G$, and the performance is improved by parallelizing actions whenever possible, as described above.
The third subtree, $\mathscr{T}_{L_i}$ uses output from the task assignment to actually execute the proper actions. We will now describe the three subtrees in more detail.

5.1 Description of $\mathscr{T}_A$

This tree is responsible for the reactive task assignment by solving an optimization problem, deploying robots to different local tasks according to the given scenario. The reactiveness lies in changes of constraints' parameters in the optimization problem, reflecting the fact that the number of robots needed to perform a given task changes during its execution.
Depending on constraints and objective, different optimization formulations can be used to compute the assignment. Here we suggest the following, to account for the limited capabilities of Problem 2.

$$\begin{aligned} \text{minimize} \quad & -\sum_{i\in\mathscr{R}}\sum_{l_j\in\bar{\mathscr{L}}} p(i,l_j)r(i,l_j) \\ \text{subject to:} \quad & a_j \le \sum_{i\in\mathscr{R}} r(i,l_j) \le b_j \;\forall l_j \in \mathscr{L}_i \\ & \sum_{l_j\in\bar{\mathscr{L}}} r(i,l_j) \le 1 \;\forall i \in \mathscr{R} \\ & r(i,j) \in \{0,1\} \end{aligned} \tag{2}$$

where $r : \mathscr{R} \times \bar{\mathscr{L}} \to \{0,1\}$ is a function that represents the assignment of robot i to a local task l_j, taking value 1 if the assignment is done and 0 otherwise; $p : \mathscr{R} \times \bar{\mathscr{L}} \to \bar{\mathbb{R}}$ is a function assigning a performance to robot i at executing the task l_j (if the robot i cannot perform the task l_j i.e., $l_j \notin \mathscr{L}_i$ then $p(i,l_j) = -\infty$). $a_j, b_j \in \mathbb{N}$ are respectively the minimum and the maximum number of robots assignable to the task l_j and they change during the execution of (1) and $a_j \in \mathbb{N}$ is set to a positive value upon requests from $\mathscr{T}_G$. At the beginning they are initialized to 0, since no assignment is needed. When the execution of a global task g_k starts, a_j and b_j are set respectively to $\nu(g_k,l_j)$ and $\mu(g_k,l_j)\ \forall l_j \in \psi(g_k)$. Then when a local task l_j finishes its execution (i.e., it returns either success or failure), a_j and b_j decrease by 1 until $a_j = 0$, since the robot executing l_j can be assigned to another task while the other robots are still executing their local task. When the task g_k finishes, both a_j and b_j are set back to zero.
The tree $\mathscr{T}_A$ is shown in Fig. 8a. The condition "Local Task Finished" returns success if a robot has succeeded or failed a local task, failure otherwise. We assume that a robot can communicate to all the other when it has finished a local task. The condition "New Global Task Executed" returns success if it is satisfied, failure otherwise.
The condition "Check Consistency" checks if the optimization problem is feasible, this is needed since the parameters change during execution. The addition of a constraint in (2) effects the system only if a solution exists. This condition is crucial to run a number of global tasks in parallel according to the number of available robots (see Example 2 below).
The action "Assign Agents" is responsible for the task assignment, it returns running while the assignment routine is executing, it returns success when the assignment problem is solved and it returns failure if the optimal value is ∞.

5.2 Description of $\mathscr{T}_G$

The high level choice of what actions the multi robot system should do is decided in $\mathscr{T}_G$ which is created from the original single robot BT $\mathscr{T}_S$ by adding possibilities afforded by parallelization and accounting for some tasks needing mutiple robots by converting the actions to global tasks to be handled by $\mathscr{T}_A$. An example of $\mathscr{T}_G$ is depicted in Fig. 8b. To formalize this we need the following definitions:

Definition 1 (Parallelizable Sequences) *A Sequence of actions in the single robot BT $\mathscr{T}_S$, where the actions are such that they can be done in parallel, is called a Parallelizable Sequence.*

Definition 2 (Parallelizable Fallbacks) *A fallback combination of actions in the single robot BT $\mathscr{T}_S$, where the actions are such that they can be done in parallel, is called a Parallelizable Fallback.*

Then $\mathscr{T}_G$ is created by making a copy of $\mathscr{T}_S$ and replacing all Parallelizable Sequences and Parallelizable Fallbacks with Parallel nodes. Finally, all actions are replaced by global tasks, providing constraints to $\mathscr{T}_A$.
The execution of the action "Perform Global Task g_k" requires that some robots have to be assigned, then it sets $b_j = \mu(g_k,l_j)\ \forall l_j \in \psi(g_k)$. In the tree $\mathscr{T}_A$ the condition

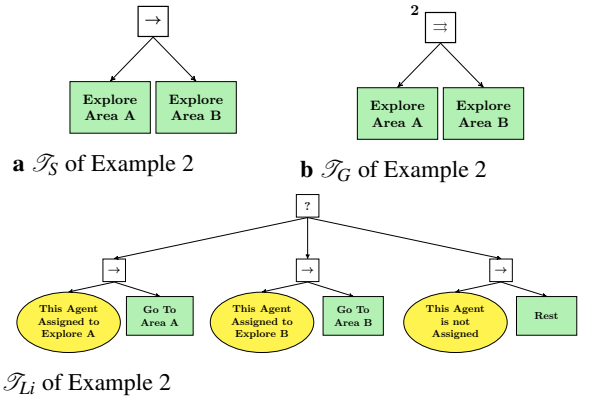
"New Global Task Executed" is now satisfied, making a new assignment if the constraints are consistent with each other. Finally action "Perform Global Task g_k" returns success if the number of local tasks $l_j \in \psi(g_k)$ that return success is greater than $v(g_k, l_j)$ (i.e the minimum amount of robots needed have succeeded) it returns failure if the number of local tasks in $\psi(g_k)$ that return failure is greater than $\sum_{i \in \mathscr{R}} r(i,j) - v(g_k, l_j)$ (i.e., some robots have failed to perform the task, the remaining ones are not enough to succeed), it returns running otherwise.

5.3 Description of $\mathscr{T}_{Li}$

This tree is illustrated in Fig. 9 and executes whatever actions that was assigned in $\mathscr{T}_A$. If needed, a safety feature can be added to this BT, as described in [5]. This could be used to prevent overheating, or to make sure the robot leaves the group to go and recharge when needed.
We conclude this section with two examples illustrating the proposed approach.

Example 2 *By way of example, we consider a system that is to explore* 2 *different areas. Intuitively, different areas can be explored by different robots at the same time but if only a single robot is available those areas can only be explored in a sequence.*
Considering first the case in which there is only one robot available. When the execution of Explore Area *A in* $\mathscr{T}_G$ *starts, in* (2) *the constraint that one robot has to be assigned to explore area A is added (i.e. the related* a_j *and* b_j *are both set to* 1*). The optimization problem is feasible hence the assignment takes place. When the execution of* Explore Area *B starts, the related constraint is not consistent with the other constraints, since the only available robot cannot be assigned to two tasks, hence the assignment does not take place.*
Considering now the case where a new robot is introduced in the system, it is possible to assign two robots. Both constraints: the one related to Explore Area *A and the one related to* Explore Area *B, can be introduced in the optimization problem without jeopardizing its consistency. Those two tasks can be executed in parallel.*
Note that the two different executions (i.e. Explore Area *A first, then* Explore Area *B;* Explore Area *A and* Explore Area *B in parallel) depend only on the number of available robots, the designed tree does not change.*

Example 3 *Consider a system that is to perform a search and rescue task. The BT for the single robot system* $\mathscr{T}_S$ *is shown in Fig. 1. As stressed above, some nodes in* $\mathscr{T}_S$ *can be turned into a parallel node for the multi-robot system. The actions that a single robot can perform are many, for simplicity we abstract them in a general "Task i".*
We now describe the execution of the multi-robot system. All the robots in the system run a copy of the three BTs aforementioned. Considering first the case in which there is only one robot available having found a survivor that needs help. When the execution of Use AED *in* $\mathscr{T}_G$ *starts,*



a $\mathscr{T}_S$ of Example 2 **b** $\mathscr{T}_G$ of Example 2

c $\mathscr{T}_{Li}$ of Example 2

Figure 10 Trees of Example 2

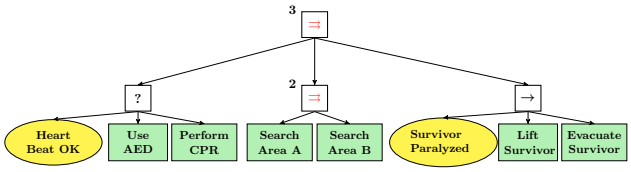


Figure 11 Global tasks tree $\mathscr{T}_G$ of Example 3

in $\mathscr{T}_L$ *the constraint that one robot has to be assigned to "Task use AED" is added (i.e. the related constraint is added in the task assignment problem). The optimization problem is feasible (the condition* Check Consistency *returns true), hence the assignment takes place. When the execution of* Search Area *A starts, the related constraint is not consistent with the other constraints, since the only available robot cannot be assigned to two tasks, hence the assignment does not take place (the condition* Check Consistency *returns false and the action* Assign Agents *is not executed). Considering now the case where a new robot is introduced in the system, it is possible to assign two robots. Both constraints, the one related to* Use AED *and the one related to* Search Area *A, can be introduced in the optimization problem without jeopardizing its consistency. The constraines related to* Search Area *B, is then added (the two searching task are done in parallel but the introduction of the related constraint are still done in sequence). This constraint is not consistent with the ones already present in the optimization problem as there is no available robot. Such constraint will be consistent whenever a robot finishes its task or a new robot is introduced. The argument is easily extendable to the next action to perform.*

6 Analysis of Fault Tolerance

The fault tolerance capabilities of a multi-robot system are due to the redundancy of tasks executable by different robots. In particular, a local task l_j can be executed by every robot $i \in \mathscr{R}$ satisfying the condition $l_j \in \mathscr{L}_i$. Here we make the distinction between *minor* faults and *major* faults of a robot according to their severity. A minor fault

involves a single local task l_j, i.e. the robot is no longer capable to perform the task l_j, here the *level* of a minor fault is the number of local tasks involved. A major fault of a robot implies that the robot can no longer perform any of its local task.

Remark 2 *A minor fault of level λ is different from λ minor faults, since the latter might involve different robots.*

Remark 3 *A major fault involving a robot $i \in \mathscr{R}$ is equivalent to a minor fault of level $|\mathscr{L}_i|$ involving every task in $\mathscr{L}_i$.*

Definition 3 *A multi-robot system is said to be* weakly fault tolerant *if it can tolerate any minor fault.*

Definition 4 *A multi-robot system is said to be* strongly fault tolerant *if it can tolerate any major fault.*

Lemma 1 *A multi-robot system is weakly fault tolerant if and only if for each robot $i \in \mathscr{R}$ and for each local task $l_j \in \mathscr{L}_i$ there exists another robot $h \in \mathscr{R}$ such that $l_j \in \mathscr{L}_i \cap \mathscr{L}_h$ and the problem* (2) *is consistent under the constraint $r(i,j) \cdot r(h,j) = 0$.*

Proof 1 *(*if*) When $r(i,j) \cdot r(h,j) = 0$ either robot i or h is not deployed. Assume that the robot i is involved in a minor fault, related to the local tasks l_j. Since $l_j \in \mathscr{L}_i \cap \mathscr{L}_h$, $l_j \in \mathscr{L}_h$ then the robot h can perform the local task l_j in place of robot i, since it is not deployed.*
*(*only if*) if the system can tolerate a minor fault then for each robot i there exists another robot h such that the local task related to the fault can be performed by both of them since one took over the other. Hence $l_j \in \mathscr{L}_i$ and $l_j \in \mathscr{L}_h$, this implies that $l_j \in \mathscr{L}_i \cap \mathscr{L}_h$ holds. Moreover, if the fault can be tolerated it means that when l_j is performed either the robot i or robot h is not deployed otherwise the reassignment would not be possible. Hence the condition $r(i,j) \cdot r(h,j) = 0$ holds.*

Corollary 1 *A multi-robot system can tolerate a minor fault of level λ if Lemma 1 holds for the local tasks involved in the fault.*

Lemma 2 *A multi-robot system is strongly fault tolerant if and only if for each robot i there exists a set of robots I such that $\mathscr{L}_i \subseteq \cup^n_{h \in I, h \neq i} \mathscr{L}_h$ and the problem* (2) *is consistent under the constraints $r(i,j) \cdot r(h,j) = 0 \; \forall j : l_j \in \mathscr{L}_i, h \in I$.*

Proof 2 *(*if*) Since $\mathscr{L}_i \subseteq \cup_{h \in I, h \neq i} \mathscr{L}_h$ all the local task of the robot i can be performed by some other robots in the system. Then when the robot i is asked to perform a local task l_j, exists at least another robot that can perform such task, this robot is not deployed since $r(i,j) \cdot r(h,j) = 0 \; \forall j : l_j \in \mathscr{L}_i, h \in I$.*
*(*only if*) if the system can tolerate a major fault then exists a robot i that is redundant, i.e. its tasks can be performed by other robots. Hence there exists a set of robots I, in which i is not contained, such that $\mathscr{L}_i \subseteq \cup_{h \in I, h \neq i} \mathscr{L}_h$, moreover since each local task l_j of the robot i can be performed by a robot $h \in I$, robot h is not deployed when i is performing l_j otherwise it could not take over robot i, hence $r(i,j) \cdot r(h,j) = 0 \; \forall j : l_j \in \mathscr{L}_i, h \in I$.*

7 Conclusions

We have shown a possible use of BTs as a distributed controller for multi-robot systems working towards global goals. This extends the fields where BTs can be used, with a strong application in robotic systems. We have also shown how using BTs as a framework for robot control becomes natural going from a plan meant to be executed by a single robot to a multi-robot plan. Moreover since BTs are modular, whenever the multi-robot system changes only a subtree is affected. For example if the same multi-robot system has to achieve a different global goal, only $\mathscr{T}_G$ is affected. On the other hand, if a robot i is replaced by a different one , only $\mathscr{T}_{Li}$ is affected.

Acknowledgment

This work has been supported by the Swedish Research Council (VR) and the European Union Project RECONFIG, (FP7-ICT-2011-9), the authors gratefully acknowledge the support.

8 Literature

[1] D. Isla, "Handling Complexity in the Halo 2 AI," in *Game Developers Conference*, 2005.

[2] I. Millington and J. Funge, *Artificial Intelligence for Games*. Taylor & Francis, 2009. [Online]. Available: https://books.google.com/books?id=1OJ8EhvuPXAC

[3] S. Rabin, *Game AI Pro: Collected Wisdom of Game AI Professionals*. Natick, MA, USA: A. K. Peters, Ltd., 2013.

[4] A. Klökner, "Interfacing Behavior Trees with the World Using Description Logic," in *AIAA conference on Guidance, Navigation and Control, Boston*, 2013.

[5] M. Colledanchise, A. Marzinotto, and P. Ögren, "Performance Analysis of Stochastic Behavior Trees," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, June 2014.

[6] P. Ögren, "Increasing Modularity of UAV Control Systems using Computer Game Behavior Trees," in *AIAA Guidance, Navigation and Control Conference, Minneapolis, MN*, 2012.

[7] R. d. P. Pereira and P. M. Engel, "A framework for constrained and adaptive behavior-based agents," *arXiv preprint arXiv:1506.02312*, 2015.

[8] J. Yao, Q. Huang, and W. Wang, "Adaptive human behavior modeling for air combat simulation," in *2015 IEEE/ACM 19th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, Oct 2015, pp. 100–103.

[9] K. R. Guerin, C. Lea, C. Paxton, and G. D. Hager, "A framework for end-user instruction of a robot assistant for manufacturing," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, May 2015, pp. 6167–6174.

[10] M. Colledanchise, R. Parasuraman, and P. Ögren, "Learning of behavior trees for autonomous agents," *arXiv preprint arXiv:1504.05811*, 2015.

[11] A. Klöckner, "Behavior trees for uav mission management," in *INFORMATIK 2013: Informatik angepasst an Mensch, Organisation und Umwelt*. Köllen Druck + Verlag GmbH, Bonn, 2013, pp. 57–68.

[12] D. Hu, Y. Gong, B. Hannaford, and E. J. Seibel, "Semi-autonomous simulated brain tumor ablation with ravenii surgical robot using behavior tree," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, May 2015, pp. 3868–3875.

[13] A. Champandard, "Understanding Behavior Trees," *AiGameDev. com*, vol. 6, 2007.

[14] D. Isla, "Halo 3-building a Better Battle," in *Game Developers Conference*, 2008.

[15] C. Lim, R. Baumgarten, and S. Colton, "Evolving Behaviour Trees for the Commercial Game DEFCON," *Applications of Evolutionary Computation*, pp. 100–110, 2010.

[16] D. Perez, M. Nicolau, M. O'Neill, and A. Brabazon, "Evolving Behaviour Trees for the Mario AI Competition Using Grammatical Evolution," *Applications of Evolutionary Computation*, 2011.

[17] A. Shoulson, F. M. Garcia, M. Jones, R. Mead, and N. I. Badler, "Parameterizing Behavior Trees," in *Motion in Games*. Springer, 2011.

[18] I. Bojic, T. Lipic, M. Kusek, and G. Jezic, "Extending the JADE Agent Behaviour Model with JBehaviourtrees Framework," in *Agent and Multi-Agent Systems: Technologies and Applications*. Springer, 2011, pp. 159–168.

[19] J. A. D. Bagnell, F. Cavalcanti, L. Cui, T. Galluzzo, M. Hebert, M. Kazemi, M. Klingensmith, J. Libby, T. Y. Liu, N. Pollard, M. Pivtoraiko, J.-S. Valois, and R. Zhu, "An Integrated System for Autonomous Robotics Manipulation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2012, pp. 2955–2962.

[20] A. Klöckner, *Advances in Aerospace Guidance, Navigation and Control: Selected Papers of the Third CEAS Specialist Conference on Guidance, Navigation and Control held in Toulouse*. Cham: Springer International Publishing, 2015, ch. Behavior Trees with Stateful Tasks, pp. 509–519.

[21] M. Colledanchise and P. Ogren, "How Behavior Trees Modularize Robustness and Safety in Hybrid Systems," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, Sept 2014, pp. 1482–1488.

[22] E. W. Dijkstra, "Letters to the editor: go to statement considered harmful," *Commun. ACM*, vol. 11, pp. 147–148, March 1968.

[23] X. C. Ding, M. Kloetzer, Y. Chen, and C. Belta, "Automatic deployment of robotic teams," *Robotics Automation Magazine, IEEE*, vol. 18, no. 3, pp. 75–86, 2011.

[24] S. Karaman and E. Frazzoli, "Vehicle routing problem with metric temporal logic specifications," in *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, 2008, pp. 3953–3958.

[25] A. Ulusoy, S. L. Smith, and C. Belta, "Optimal multi-robot path planning with ltl constraints: Guaranteeing correctness through synchronization," *CoRR*, vol. abs/1207.2415, 2012.

[26] C. Castelfranchi and R. Falcone, "Towards a Theory of Delegation for Agent-based Systems," *Robotics and Autonomous Systems*, vol. 24, pp. 141–157, 1998.

[27] R. Falcone and C. Castelfranchi, "C.: The human in the loop of a delegated agent: The theory of adjustable social autonomy," *IEEE Transactions on Systems, Man and Cybernetics–Part A: Systems and Humans*, pp. 406–418, 2001.

[28] P. Doherty, D. Landén, and F. Heintz, "A distributed task specification language for mixed-initiative delegation," 2012.

[29] C. G. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Springer Science & Business Media, 2008.

[30] R. Bellman, *Dynamic Programming*, ser. Dover Books on Computer Science Series. Dover Publications, 2003. [Online]. Available: http://books.google.se/books?id=fyVtp3EMxasC