

Towards **Blended** Reactive Planning and Acting using Behavior Trees

Michele Colledanchise¹, Diogo Almeida², and Petter Ögren²

Abstract—In this paper, we show how a planning algorithm can be used to automatically create and update a Behavior Tree (BT), controlling a robot in a **dynamic** environment. The planning part of the algorithm is based on the idea of **back chaining**. Starting from a goal condition we **iteratively** select actions to achieve that goal, and if those actions have **unmet** preconditions, they are extended with actions to achieve them in the same way. The fact that BTs are inherently modular and reactive makes the proposed solution blend acting and planning in a way that enables the robot to effectively react to external disturbances. If an external agent undoes an action the robot re-executes it without re-planning, and if an external agent helps the robot, it skips the corresponding actions, again without re-planning. We illustrate our approach in two different robotics scenarios.

I. INTRODUCTION

Behavior Trees (BTs) were developed within the computer gaming industry as a modular and flexible alternative to Finite State Machines (FSMs). Their recursive structure and usability have made them very popular in industry, which in turn has created a growing amount of attention in academia [1]–[14]. However, the vast majority of BTs are still manually designed. In this paper, we show how to automatically create a BT using a planning algorithm. The resulting approach allows us to blend planning and acting in a reactive and modular fashion.

To illustrate how the proposed approach blends planning and acting, we use a simple example, depicted in Figure 1. A robot has to plan and execute the actions needed to pick up an object, and place it in a given location. The environment is however dynamic and unpredictable. After pickup, the object might slip out of the robot gripper, or, as shown in Figure 1(a), external objects might move and block the path to the goal location, and then move away again, forcing the robot to react once more, see Figure 1(b). **The BT includes reactivity, in the sense that if the object slips out of the robot gripper, it will automatically stop and pick it up again without the need to replan or change the BT.** The BT also supports iterative plan refinement, in the sense that if an object moves to block the path, the original BT is extended to include a removal of the blocking **obstacle**. Then, if the obstacle is removed by an external actor, the BT reactively skips the obstacle removal, and goes on to pick up the main object without having to change the BT.



Fig. 1. A simple example scenario where the goal is to place the green cube C onto the goal region G . But the fact that the sphere S intermittently blocks the path must be handled. In (a) the nominal plan is $MoveTo(C) \rightarrow Pick(C) \rightarrow MoveTo(G) \rightarrow Drop()$ when the sphere suddenly blocks the path. After replanning, the plan is $MoveTo(S) \rightarrow Push(S) \rightarrow MoveTo(C) \rightarrow Pick(C) \rightarrow MoveTo(G) \rightarrow Drop()$. In (b), an external agent moves the sphere before it is pushed by the agent. Thus the actions concerning the sphere S should be ignored.

Within the AI community, there has been an increased interest in the combination of planning and acting, [15], [16]. In particular, the community identified two key open challenges [15], summarized in the following quotes:

- **Challenge 1: “Hierarchically organized deliberation.** This principle goes beyond existing hierarchical planning techniques; its requirements and scope are significantly different. The actor performs its deliberation online”.
- **Challenge 2: “Continual planning and deliberation.** The actor monitors, **refines**, extends, updates, changes and repairs its plans throughout the acting process, using both descriptive and operational models of actions.”

Similarly, a recent book [16] describes the need for an agent that “reacts to events and extends, updates, and repairs its plan on the basis of its perception”. Finally, the authors of [15] also note that most of the current work in action planning yields a static plan, i.e., a sequence of actions that brings the system from the initial state to the goal state, and its **execution** is usually represented as a classical FSM. However, due to external agents creating changes in the environment, the outcome of an action can be unexpected. **This may lead to situations where the agent replans from scratch on a regular basis, which can be expensive in terms of both time and computational load.**

BTs are a graphical mathematical model for reactive fault tolerant task executions. They were first introduced in the computer gaming industry [17] to control in game opponents, and is now an established tool appearing in textbooks [18]–[20] and generic game-coding software such as Pygame, Craft AI, and Unreal Engine. BTs are appreciated for being highly modular, flexible and reusable, and have also been shown to generalize other successful control architectures such as the Subsumption architecture and the Teleo-reactive Paradigm [20]. So far, BTs are either created by human experts [2]–[5], [21], [22] or automatically designed using

¹ iCub Facility, Istituto Italiano di Tecnologia - IIT, Genoa, Italy

² KTH - Royal Institute of Technology, Stockholm, Sweden. This work was partially sponsored by European Communitys Framework Programme Horizon 2020 - under grant agreement No 644938 SARAFun and the Swedish Foundation for Strategic Research, through the Swedish Maritime Robotics Center (SMaRC) (IRC15-0046).

machine learning techniques [23]–[25] maximizing some objective function.

In this paper, we propose an automated planning approach to synthesize a BT. The construction of the tree is based on the idea of backchaining. Starting from the goal condition we find actions that meet those conditions. We then look at the preconditions of those actions and try to find actions that satisfy them, and so on. This is a well known approach, but the novelty lies in the combination with BTs, exploiting their advantages in terms of reactivity and modularity, [1], as compared to e.g., FSM.

Looking back at the example above, the *reactivity* of BTs enable the robot to pick up a dropped object without having to replan at all. The *modularity* enables extending the plan by adding actions for handling the blocking sphere, without having to replan the whole task. Finally, when the sphere moves away, once again the *reactivity* enables the correct execution without changing the plan.

The main contribution of this paper is thus that we show how to iteratively create and refine BTs using a planning algorithm, and that the result is both reactive and modular, as described above. To the best of our knowledge, this has not been done before.

II. RELATED WORK

In this section, we briefly summarize related work and compare it with the proposed approach. We focus on Automated Planning as little work in the literature addresses our objective of automatically generating a BT.

The planning community has developed solid solutions for solving path-finding problems in large state spaces. Such solutions have found successful applications in a wide variety of problems searching in both state space and plan space. Plan space planners can also start from a partially defined plan (e.g. where the exact order of the action is not yet defined) and resolve incorrect action ordering known as *threats*. The solution can take the form of a sequence of actions [26], [27] or a tree [28], [29]. Nonetheless, numerous planning problems remain open challenges [15], [26], [30], [31]. For example, it was noted by Kaelbling et al. [28], that there is no systematic planning framework that can address an abstract goal such as “wash dishes” and reason on a long sequence of actions in dynamic or finite horizon environments.

In the robotic community most of the work focus, without loss of generality, on manipulation planning, where the objective is to have a robot operate in an environment and change the configuration of that environment by e.g., moving objects and opening doors. Early approaches treated the configuration space as continuous for both environment and robot but used discrete actions [26], [32], [33].

Recent approaches to robotic planning combine discrete task planning and continuous motion planning frameworks [34]–[36] pre-sampling grasps and placements producing a family of possible high level plans. These approaches use hierarchical architectures, but do not consider the continual update of the current plan. Other approaches [37]

consider two types of replanning: *aggressive replanning*, where replanning is done after every action; and *selective replanning*: where replanning is done whenever a change in the environment happens that enables a new path to the goal, that is shorter than the existing plan by a given threshold. In our approach we replan when needed. By using continually hierarchical monitoring, we are able to monitor the part of the environment that is relevant for goal satisfaction, disregarding environmental changes that do not affect our plan. This is particularly useful when the framework has to plan and act in highly dynamic environments and large state space.

The Hybrid Backward Forward (HBF) algorithm [38] was proposed as an action planner in infinite state space. HBF is a forward search in state space, starting at the initial state of a complete domain, repeatedly selecting a state that has been visited and an action that is applicable in that state, and computing the resulting state, until a state satisfying a set of goal constraints is reached. One advantage of this approach lies in the restriction to the set of useful actions, building a so-called *reachability graph*. A backward search algorithm builds the reachability graph working backward from the goal’s constraints, using them to drive sampling of actions that could result in states that satisfy them. Thus, HBF enables us to deal with infinite state space, but the resulting plan is static, and does not address issues related to acting and planning in dynamic environments.

When it comes to using planning to create BTs there is almost no previous work. The closest approach is the *A Behavior Language* (ABL) [39]. ABL planning was created for use in the dialogue game *Façade*, to automatically generate complex structures from a repository of simpler ones. The structures of ABL were predecessors of BTs, including the return statuses *success* and *failure*, but not *running*. This made reactivity much less straightforward and explicit constructions of so-called wait actions were used to respond to actions. Thus, ABL planning is not reactive and does not address the problems investigated in this paper.

III. BACKGROUND: BEHAVIOR TREES

In this section we briefly describe BTs, and refer the interested reader to the more detailed descriptions that can be found in the literature [20]. A BT can be seen as a graphical modeling language and a representation for execution of actions based on conditions and observations in a system.

Formally, a BT is a directed rooted tree where each node is either a *control flow node* or an *execution node*, see below. We use the standard definitions of *parent* (the node above) and *child* (the nodes below). The root is the single node without parents, whereas all other nodes have one parent. The control flow nodes have one or more children, and the execution nodes have no children. Graphically, the children of nodes are placed below it. The children nodes are executed in the order from left to right, as shown in Figure 2.

The execution of a BT begins from the root node. It sends *ticks*¹ with a given frequency to its child. When a parent

¹A tick is a signal that allows the execution of a child

sends a tick to a child, the child can be executed. The child returns to the parent a status *running* if its execution has not finished yet, *success* if it has achieved its goal, or *failure* otherwise.

There are four types of control flow nodes (fallback, sequence, parallel, and decorator) and two execution nodes (action and condition). Below we describe the execution of the nodes used in this paper.

Fallback: The fallback node ticks its children from the left, returning success (running) as soon as it finds a child that returns success (running). It returns failure only if all the children return failure. When a child returns running or success, the fallback node does not tick the next child (if any). The fallback node is graphically represented by a box with a “?”, as in Figure 2.

Sequence: The sequence node ticks its children from the left, returning failure (running) as soon as it finds a child that returns failure (running). It returns success only if all the children return success. When a child returns running or failure, the sequence node does not tick the next child (if any). The sequence node is graphically represented by a box with a “→”, as in Figure 2.

Action: The action node performs an action, returning success if the action is completed and failure if the action cannot be completed. Otherwise it returns running. An action node is graphically represented as a rectangle, as in Figure 2.

Condition: The condition node checks if a condition is satisfied or not, returning success or failure accordingly. The condition node never returns running. A condition node is graphically represented as an ellipse, as in Figure 2.

To get familiar with the BT notation, and prepare for the coming sections, we look at a BT construction addressing the simple example in Section I. The BT was created using the proposed approach as will be explained in Section V, but for now we just focus on how it is executed.

Example 1: The robot in Figure 1 is given the task to move the green cube into the rectangle marked GOAL. Ignoring the presence of the red sphere, a reactive plan BT can be found in Figure 3(e). Each time step, the root of the BT is ticked. The root is a fallback which ticks its first child, the condition $o_c \in GoalRect$ (cube on goal). If the cube is indeed in the rectangle we are done, and the BT returns Success. If not, the second child, a sequence, is ticked. The sequence ticks its first child, which is a fallback, which again ticks its first child, the condition $h = c$ (object in hand is cube). If the cube is indeed in the hand, the condition returns success, its parent, the fallback returns success, and its parent, the sequence ticks its second child, which is a different fallback, ticking its first child which is the condition $o_r \in \mathcal{N}_{p_g}$ (robot in the neighborhood of p_g). If the robot is in the neighborhood of the goal, the condition and its fallback parent returns success, followed by the sequence ticking its third child, the action $Place(c, p_g)$ (place cube in a position p_g on the goal), and we are done.

If $o_r \in \mathcal{N}_{p_g}$ does not hold, the action $MoveTo(p_g, \tau_g)$ (move to position p_g using the trajectory τ_g) is executed, given that the trajectory is free $\tau \subseteq C_{ollFree}$. Similarly, if the

Actions	Preconditions	Postconditions
A_1	$C_{11}^{Pre}, C_{12}^{Pre}, \dots$	$C_{11}^{Post}, C_{12}^{Post}, \dots$
A_2	$C_{21}^{Pre}, C_{22}^{Pre}, \dots$	$C_{21}^{Post}, C_{22}^{Post}, \dots$
\vdots	\vdots	\vdots

TABLE I. The input to Problem 1 is a set of actions and corresponding pre- and post conditions, as illustrated above.

cube is not in the hand, the robot does a *MoveTo* followed by a *Pick(c)* after checking that the hand is empty, the robot is not in the neighborhood of c and that the corresponding trajectory is free.

IV. PROBLEM FORMULATION

We now describe the main problem addressed in this paper.

Problem 1: Given a set of actions, with corresponding preconditions and postconditions, as in Table I, the current state, a set of goal conditions, $C_1^{Goal}, C_2^{Goal}, \dots$ create and iteratively extend a BT that strives to satisfy the goal conditions. The BT should be reactive to changes brought about by external agents in the following senses:

First, if an external agent reverses an action executed by the main agent, such as taking an item from the agent and putting it on the floor, the main agent should pick it up again without having to replan in terms of expanding the BT.

Second, if an external agent carries out an action that the main agent was planning to do, such as opening a door, the main agent should take advantage of this fact, and traverse the door without trying to open it, and without having to replan in terms of expanding the BT.

Third, if there are several actions that result in a common post condition, the BT should include these so that if one fails due to external conditions, the other ones can be tried instead, without having to replan in terms of expanding the BT.

Finally, the BT should be able to be expanded during runtime. If, e.g., a condition that was earlier assumed to hold turns out to not hold, actions for achieving this conditions can be added on the fly.

V. PROPOSED APPROACH

Formally, the proposed approach is described in Algorithms 1, but before going into detail, we will first describe the two main ideas of the algorithms, i.e., **creating atomic BTs and iteratively replacing failed conditions with these atomic BTs**. Then we will see how the algorithms are applied to the problem to iteratively create the BTs of Figure 3. Finally, we will discuss the key steps in detail.

A. Atomic BTs for each postcondition

The first step of the algorithm converts the list of actions in Table I into a list of atomic BTs, each aimed at satisfying a given condition, but invoking the actions only when the condition is not met. This construction is what enables the reactivity needed in Problem 1 of Section IV.

Assume that the table includes a postcondition C that can be achieved by either action A_1 or action A_2 , that in turn

have preconditions C_{11}, C_{12} and C_{21}, C_{22} respectively. Then we create an atomic BT aimed at achieving the condition C by composing the actions and conditions in the generic way displayed in Figure 2, i.e., each action A_i in sequence after its preconditions C_{ij} , and these sequences in a fallback composition after the main condition C itself. Finally we create similar BTs for each postcondition C of Table I.

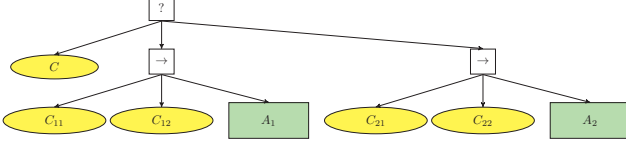


Fig. 2. General format of an atomic BT. The Postcondition C can be achieved by either one of actions A_1 or A_2 , which have Preconditions C_{11}, C_{12} and C_{21}, C_{22} respectively.

The order of actions A_1, A_2 and conditions C_{11}, C_{12} were arbitrary in the BT of Figure 2. We will later enable the re-ordering of the pre-conditions based on so-called conflicts.

B. Iteratively expanding a BT from the Goal Conditions

Having a list of atomic BTs we can now iteratively build a deliberative BT by starting with a **minimalistic BT**, made up of a single sequence composition of all the goal conditions. Then we execute this BT. If it returns success all conditions are met and we are done. If not we replace each condition that returned failure with the corresponding atomic BT, of the form shown in Figure 2. Note that this BT returns success immediately if the condition is met, but tries to satisfy the condition if it is not. The new BT is again executed. As long as it returns running we let it run. If it succeeds we are done, and if it fails, we replace the failing condition with a new atomic BT. In the next paragraph we will see that this is how the BTs of Figure 3 where created.

C. Algorithm Example Execution

This example starts with a single goal condition, shown in Figure 3(a). Running Algorithm 1 we have the set of goal constraints $\mathcal{C}_{goal} = \{o_c \in \{\text{GoalRect}\}\}$, thus the BT \mathcal{T} is initially composed of a single condition $o_c \in \{\text{GoalRect}\}$, as shown in Figure 3(a). The first iteration of the loop starting on Line 4 now produces the next BT shown in Figure 3(b), and the second iteration produces the BT in Figure 3(c) and so on until the final BT in Figure 3(e).

In detail, running \mathcal{T} on Line 7 returns a failure, since the cube is not in the goal area. Trivially, the failed condition is $c_f = (o_c \in \{\text{GoalRect}\})$, and a call to `ExpandTree` is made on Line 10. On Line 14 we get $A_T = \text{Place}$. Then, on Line 19 and 20, a sequence \mathcal{T}_{seq} is created of the conditions of `Place` (the hand holding the cube $h = c$ and the robot being near the goal area $o_r \in \mathcal{N}_{p_g}$) and `Place` itself. On Line 21 a fallback \mathcal{T}_{fall} is created of c_f and the sequence above. Finally, a BT is returned where this new sub-BT is replacing c_f . The resulting BT is shown in Figure 3(b).

Note that Algorithm 1 describes the *core principle* of the proposed approach. The BT is iteratively extended as described in Sections V-A to V-B.

Algorithm 1: Implementation of the approach

```

1  $\mathcal{T} \leftarrow \emptyset$ 
2 for  $c$  in  $\mathcal{C}_{goal}$  do
3    $\mathcal{T} \leftarrow \text{SequenceNode}(\mathcal{T}, c)$ 
4 while True do
5    $T \leftarrow \text{RefineActions}(T)$ 
6   do
7      $r \leftarrow \text{Tick}(T)$ 
8     while  $r \neq \text{Failure}$ 
9        $c_f \leftarrow \text{GetConditionToExpand}(T)$ 
10       $\mathcal{T}, \mathcal{T}_{new\_subtree} \leftarrow \text{ExpandTree}(T, c_f)$ 
11      while  $\text{Conflict}(\mathcal{T})$  do
12         $\mathcal{T} \leftarrow \text{IncreasePriority}(\mathcal{T}_{new\_subtree})$ 
13 Function ExpandTree ( $\mathcal{T}, c_f$ )
14    $A_T \leftarrow \text{GetAllActTemplatesFor}(c_f)$ 
15    $\mathcal{T}_{fall} \leftarrow c_f$ 
16   for  $a$  in  $A_T$  do
17      $\mathcal{T}_{seq} \leftarrow \emptyset$ 
18     for  $c_a$  in  $a.con$  do
19        $\mathcal{T}_{seq} \leftarrow \text{SequenceNode}(\mathcal{T}_{seq}, c_a)$ 
20        $\mathcal{T}_{seq} \leftarrow \text{SequenceNode}(\mathcal{T}_{seq}, a)$ 
21        $\mathcal{T}_{fall} \leftarrow \text{FallbackNode}(\mathcal{T}_{fall}, \mathcal{T}_{seq})$ 
22    $\mathcal{T} \leftarrow \text{Substitute}(\mathcal{T}, c_f, \mathcal{T}_{fall})$ 
23   return  $\mathcal{T}, \mathcal{T}_{fall}$ 
24 Function GetConditionToExpand ( $\mathcal{T}$ )
25   for  $c_{next}$  in GetConditionsBFS() do
26     if  $c_{next}.status = \text{Failure}$  and
27        $c_{next} \notin \text{ExpandedNodes}$  then
28        $\text{ExpandedNodes.push\_back}(c_{next})$ 
29       return  $c_{next}$ 
30   return None

```

Running the next iteration of the main loop (Lines 4-12), a similar expansion of the condition $h = c$ transforms the BT in Figure 3(b) to the BT in Fig. 3(c). Then, an expansion of the condition $o_r \in \mathcal{N}_{o_c}$ transforms the BT in Figure 3(c) to the BT in Figure 3(d). Finally, an expansion of the condition $o_r \in \mathcal{N}_{p_g}$ transforms the BT in Figure 3(d) to the BT in Figure 3(e), and this BT is able to solve *Example 1*.

D. The Algorithm Steps in Detail

1) **Refine Actions** (Line 5): This process implements an action refinement as described in [31], that is, we map template actions and conditions (e.g. `Place(c, Pg)`) into grounded actions and conditions (e.g. `Place(c, [0, 0])`). Grounded actions can be executed by the robot. We assume that a valid action refinement always exists for the entire BT, see Section V-F.

2) **Get Deepest Failed Condition and Expand Tree** (Lines 9 and 10): If the BT returns failure, Line 9 finds the deepest condition returning failure. This will then be expanded, as shown in the example of Figure 3. \mathcal{T} is expanded until it can perform an action (i.e. until \mathcal{T} contains an action template whose condition are supported by the initial state). If there exists more than one valid action that satisfies a condition, their respective trees (sequence composition of the action and its conditions) are collected in a fallback

effect the action *Pick* sets the object in the end effector to i (i.e. $h = i$); Finally, the descriptive model of the action *Place* is parametrized over object i and final position p . It requires the robot to hold i , (i.e. $h = i$), and the robot to be in the neighbourhood of the final position p . As effect the action *Place* places the object i at p (i.e. $o_i = p$).

E. Do Algorithms 1-3 solve the problem in Section IV?

A solution to Problem 1 needs to be reactive in three ways. Looking at the solution to Example 1, shown in Figure 3(e), we see that if the cube c is removed from the agent it will pick it up again without replanning. We also see that if the cube is somehow placed in the hand of the agent, the agent will skip moving to the proper place and picking it up. The example does not include postconditions that can be achieved by several actions, but it is clear from Algorithm 1 (lines 16-21) that such functionality is included. Finally, it is clear from Example 2 that the algorithm can respond to unexpected events by extending the BT when necessary.

F. Assumptions and Performance

In this section we will briefly discuss assumptions made, as well as what performance can be expected from the proposed algorithm.

First, we assume that valid action refinements for the entire BT can be found. For example, the general action of placing an object in a goal area must be refined to a specific action of placing the object in a specific position and orientation, within the goal area, from which there exist valid action refinements for the subsequent actions, and so on. This is an important restriction, but similar problems have been studied in the literature [34]–[36], [38], [40], and an extension using for example a planner-independent interface layer [36] might be possible, but is outside the scope of this paper. Second, we assume that a non conflicting order always exists, so that we can re-order the BT, as described in Section V-D.3. Handling cases where it does not is beyond the scope of this paper. Off-the-shelf solutions can be used to address this problem. Third, we assume that the action templates, with pre- and post- conditions exist and are correct. This is similar to PDDL descriptions of tasks that are used in e.g. [37], [38], [41]. If the templates are wrong, the algorithm will not work.

Regarding performance, we first note that the proposed approach is not optimal. This is clear from the fact that we allow an arbitrary ordering of actions meeting a given post condition, as well as pre-conditions needed for those actions. Performance can be improved by optimizing this order or using more sophisticated techniques, such as [37], but that is beyond the scope of the paper and, as is noted in [16], the cost of non optimal action execution is often lower than the cost of the extensive modeling needed to achieve optimality.

Regarding completeness, we note that the proposed approach is not complete as it relies on the assumptions above, including the existence of an action refinement, and algorithms finding those are usually only probabilistically complete [27], [38].

VI. SIMULATIONS

In this section we show how the proposed approach scales to complex problems using two different scenarios. First, a KUKA Youbot scenario, where we show the applicability of our approach on dynamic and unpredictable environments, highlighting the importance of continually planing and acting. Second, an ABB Yumi industrial manipulator scenario, where we highlight the applicability of our approach to real world plans that require the execution of a long sequence of actions. The experiments were carried out using the physics simulator V-REP, in-house implementations of low level controllers for actions and conditions and our open source BT library [20]. The action refinement algorithm used is a modified version of the one used in the HBF algorithm [38].

Since capturing long reactive action sequences is difficult in pictures, we provide a video showing both the scenarios.³

In the first scenario, which is an extension of Examples 1 and 2, a KUKA Youbot has to place a green cube on a goal area. The robot is equipped with a single arm with a simple parallel gripper. Additional objects may obstruct the feasible paths to the goal, and the robot has to plan when to pick and where to place to the obstructing objects. Moreover, two external agents move around in the scene and force the robot to replan by modifying the environment.

In the second scenario, an ABB Yumi has to assemble a cellphone, whose parts are scattered across a table. The robot is equipped with two arms with simple parallel grippers, which prevents any kind of dexterous manipulation. Some parts must be grasped in a particular position. For example the opening on the cellphone's chassis has to face away from the robot's arm, exposing it for the assembly. However, the initial position of a part can be such that it requires multiple grasps transferring the part to the other gripper, effectively changing its orientation w.r.t the grasping gripper.

VII. CONCLUSIONS

In this paper we proposed an approach to automatically create and update a BT using a planning algorithm. The approach combines the advantages of BTs, in terms of modularity and reactivity with the synthesis capability of automated planning. The reactivity enables the system to both skip actions that were executed by external agents, and repeat actions that were undone by external agents. The modularity enables the extension of BTs to add new actions, when previously satisfied conditions are violated by external agents. Finally, the approach was illustrated in a dynamic and challenging scenario.

³https://youtu.be/b_Ug2My9_Xw

REFERENCES

- [1] M. Colledanchise and P. Ögren, "How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees," *IEEE Transactions on robotics*, vol. 33, no. 2, pp. 372–389, 2017.
- [2] A. Klöckner, "Interfacing Behavior Trees with the World Using Description Logic," in *AIAA conference on Guidance, Navigation and Control*, Boston, 2013.
- [3] M. Colledanchise and P. Ögren, "How Behavior Trees Modularize Robustness and Safety in Hybrid Systems," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, June 2014.
- [4] D. Hu, Y. Gong, B. Hannaford, and E. J. Seibel, "Semi-autonomous simulated brain tumor ablation with raven ii surgical robot using behavior tree," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.
- [5] K. R. Guerin, C. Lea, C. Paxton, and G. D. Hager, "A framework for end-user instruction of a robot assistant for manufacturing," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.
- [6] F. Rovida, B. Grossmann, and V. Krüger, "Extended behavior trees for quick definition of flexible robotic tasks," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 6793–6800.
- [7] S. Pütz, J. S. Simón, and J. Hertzberg, "Move base flex," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3416–3421.
- [8] D. Shepherd, P. Francis, D. Weintrop, D. Franklin, B. Li, and A. Afzal, "[engineering paper] an ide for easy programming of simple robotics tasks," in *2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2018, pp. 209–214.
- [9] C. I. Sprague, Ö. Özkahraman, A. Munafo, R. Marlow, A. Phillips, and P. Ögren, "Improving the modularity of auv control systems using behaviour trees," *arXiv preprint arXiv:1811.00426*, 2018.
- [10] C. I. Sprague and P. Ögren, "Adding neural network controllers to behavior trees without destroying performance guarantees," *arXiv preprint arXiv:1809.10283*, 2018.
- [11] X. Neufeld, S. Mostaghim, and S. Brand, "A hybrid approach to planning and execution in dynamic environments through hierarchical task networks and behavior trees," in *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2018.
- [12] F. Rovida, D. Wuthier, B. Grossmann, M. Fumagalli, and V. Krüger, "Motion generators combined with behavior trees: A novel approach to skill modelling," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 5964–5971.
- [13] B. Banerjee, "Autonomous acquisition of behavior trees for robot control," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3460–3467.
- [14] E. Coronado, F. Mastrogiiovanni, and G. Venture, "Development of intelligent behaviors for social robots via user-friendly and modular programming tools," in *2018 IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO)*. IEEE, 2018, pp. 62–68.
- [15] M. Ghallab, D. Nau, and P. Traverso, "The actor's view of automated planning and acting: A position paper," *Artif. Intell.*, vol. 208, pp. 1–17, Mar. 2014. [Online]. Available: <http://dx.doi.org/10.1016/j.artint.2013.11.002>
- [16] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning and Acting*. Cambridge University Press, 2016.
- [17] D. Isla, "Handling Complexity in the Halo 2 AI," in *Game Developers Conference*, 2005.
- [18] I. Millington and J. Funge, *Artificial intelligence for games*. CRC Press, 2009.
- [19] S. Rabin, *Game AI Pro*. CRC Press, 2014, ch. 6. The Behavior Tree Starter Kit.
- [20] M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and AI: An Introduction*, ser. Chapman and Hall/CRC Artificial Intelligence and Robotics Series. CRC Press, Taylor & Francis Group, 2018.
- [21] P. Ögren, "Increasing Modularity of UAV Control Systems using Computer Game Behavior Trees," in *AIAA Guidance, Navigation and Control Conference*, Minneapolis, MN, 2012.
- [22] A. Klöckner, "Behavior trees with stateful tasks," in *Advances in Aerospace Guidance, Navigation and Control*. Springer, 2015, pp. 509–519.
- [23] L. Pena, S. Ossowski, J. M. Pena, and S. M. Lucas, "Learning and Evolving Combat Game Controllers," in *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*. IEEE, 2012, pp. 195–202.
- [24] 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.
- [25] M. Colledanchise, R. Parasuraman, and P. Ögren, "Learning of behavior trees for autonomous agents," *IEEE Transactions on Games*, 2018.
- [26] J.-C. Latombe, *Robot motion planning*. Springer Science & Business Media, 2012, vol. 124.
- [27] K. Hauser and V. Ng-Thow-Hing, "Randomized multi-modal motion planning for a humanoid robot manipulation task," *The International Journal of Robotics Research*, vol. 30, no. 6, pp. 678–698, 2011.
- [28] L. P. Kaelbling and T. Lozano-Pérez, "Integrated task and motion planning in belief space," *Int. J. Rob. Res.*, vol. 32, no. 9-10, pp. 1194–1227, Aug. 2013. [Online]. Available: <http://dx.doi.org/10.1177/0278364913484072>
- [29] L. Fraser, B. Reikabdar, M. Nicolescu, M. Nicolescu, D. Feil-Seifer, and G. Bebis, "A compact task representation for hierarchical robot control," in *Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference on*. IEEE, 2016, pp. 697–704.
- [30] S. Jiménez, T. De La Rosa, S. Fernández, F. Fernández, and D. Borrajo, "A review of machine learning for automated planning," *The Knowledge Engineering Review*, vol. 27, no. 04, pp. 433–467, 2012.
- [31] D. S. Nau, M. Ghallab, and P. Traverso, "Blended planning and acting: Preliminary approach, research challenges," in *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, ser. AAAI'15. AAAI Press, 2015, pp. 4047–4051. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2888116.2888281>
- [32] T. Lozano-Perez, "Automatic planning of manipulator transfer movements," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 11, no. 10, pp. 681–698, 1981.
- [33] T. Lozano-Perez, J. Jones, E. Mazer, P. O'Donnell, W. Grimson, P. Tournassoud, and A. Lanusse, "Handey: A robot system that recognizes, plans, and manipulates," in *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, vol. 4. IEEE, 1987, pp. 843–849.
- [34] F. Lagriffoul, D. Dimitrov, A. Saffiotti, and L. Karlsson, "Constraint propagation on interval bounds for dealing with geometric backtracking," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 957–964.
- [35] E. Erdem, K. Haspalamutgil, C. Palaz, V. Patoglu, and T. Uras, "Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 4575–4581.
- [36] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 639–646.
- [37] M. Levihn, L. P. Kaelbling, T. Lozano-Perez, and M. Stilman, "Fore-sight and reconsideration in hierarchical planning and execution," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 224–231.
- [38] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "Backward-forward search for manipulation planning," in *Intelligent Robots and Systems (IROS)*, 2015.
- [39] M. Mateas and A. Stern, "A behavior language for story-based believable agents," *IEEE Intelligent Systems*, vol. 17, no. 4, pp. 39–47, 2002.
- [40] J. Barry, L. P. Kaelbling, and T. Lozano-Pérez, "A hierarchical approach to manipulation with diverse actions," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.
- [41] D. Hadfield-Menell, E. Groshev, R. Chitnis, and P. Abbeel, "Modular task and motion planning in belief space," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 4991–4998.
- [42] P. Haslum *et al.*, "Optimal delete-relaxed (and semi-relaxed) planning with conditional effects," in *IJCAI*, 2013, pp. 2291–2297.