Elektrotechnik & Informationstechnik (2020) 137/6: 309–315. https://doi.org/10.1007/s00502-020-00816-7

Combining decision making and dynamical systems for monitoring and executing manipulation tasks

M. Saveriano, J. Piater

In this paper, we propose a unified framework for online task scheduling, monitoring, and execution that integrates reconfigurable behavior trees, a decision-making framework with integrated low-level control functionalities, and reactive motion generation with stable dynamical systems. In this way, we realize a flexible and reactive system capable of coping with unexpected variations in the executive context without penalizing modularity, expressiveness, and readability of humans. The framework is evaluated in a simulated sorting task showing promising results in terms of flexibility regarding task scheduling and robustness to external disturbances.

Keywords: decision making; task execution monitoring; learning from demonstration; reactive motion planning

Überwachung und Ausführung von Roboter-Tätigkeiten durch kombinierte Entscheidungsfindung mit dynamischen Systemen.

Für die Online-Aufgabenplanung von Robotern sowie deren Ausführung und Überwachung haben wir eine vereinheitlichte Architektur entwickelt, die rekonfigurierbare Verhaltensbäume, Entscheidungsfindung mit integrierter Regelungstechnik und reaktive Bewegungserzeugung mit stabilen dynamischen Systemen integriert. Das Ergebnis ist ein flexibles und reaktives System, das unerwartete Schwankungen im Ausführungskontext bewältigen kann, ohne Modularität, Ausdruckskraft oder menschliche Lesbarkeit zu beeinträchtigen. Das System wird mittels einer simulierten Sortieraufgabe evaluiert, die vielversprechende Ergebnisse in Bezug auf Flexibilität bei der Aufgabenplanung und Robustheit gegenüber externen Störungen liefert.

Schlüsselwörter: Entscheidungsfindung; Ausführung und Überwachung von Tätigkeiten; Nachahmungslernen; reaktive Bewegungsplanung

Received May 31, 2020, accepted July 16, 2020, published online July 30, 2020

1. Introduction

Robots operating in everyday environments need increased flexibility to cope with sudden and unexpected variations in their executive context. In dynamic environments, the robot needs to perform sensing and planning operations at low cost, as well as to monitor its own actions in a goal-oriented fashion [1, 2]. High flexibility and adaptation capabilities are required at each level of the sense-plan-act loop to deploy robust and effective robotic solutions [3]. At the same time, the enhanced reasoning and acting capabilities of cognitive robots have to be implemented considering facets like modularity, reusability, design effectiveness, and human-readability.

In this work, we propose a unified framework that combines high-level decision making, continuous execution monitoring, and online motion generation. In our framework, task scheduling and execution monitoring are handled by Reconfigurable Behaviour Trees (RBTs) [4], while stable dynamical systems are exploited for online motion generation. RBTs extend the traditional behavior trees [2] with control layer features permitting the continuous monitoring of the task execution and the online replanning of the task to react to perceptual stimuli. This is particularly useful to rapidly react to external perturbations in the task execution like unexpected changes in the pose of the manipulated object. The control layer of RBT is implemented using only the basic components of a traditional BT, de facto keeping the same level of expressiveness, modularity, and human-readability.

Motion planning with dynamical systems (DS) has several interesting features, which explains why it is gaining interest in the learning community. A DS generates goal-oriented, converging trajectories that connect any two points in the robot's workspace. DS trajectories are generated at runtime, allowing for online motion replanning to handle unexpected perturbations [5–8]. A stable dynamical system can be learned from human demonstrations [9] in an incremental way [10, 11]. Finally, suitable control can be applied to constrain the motion within a certain region and adapt to changes in the workspace [12]. In this work, we use the Energy-based Stabilizer of Dynamical Systems (ESDS) [13] that offers a good compromise between training time and accuracy in motion reproduction. Overall, the combination of reconfigurable behaviour trees for task monitoring and dynamical systems for trajectory generation results in a flexible and efficient framework for robotic task learning and execution.

The rest of the paper is organized as follows. Section 2 reviews related work. The key aspects of the RBT and the ESDS approaches are presented in Sect. 3. Results obtained in a sorting task are shown in Sect. 4. Section 5 states the conclusion and proposes further exten-

Saveriano, Matteo, Department of Computer Science and Digital Science Center (DiSC), University of Innsbruck, Innsbruck, Austria (E-mail: Matteo.Saveriano@uibk.ac.at); **Piater, Justus,** Department of Computer Science and Digital Science Center (DiSC), University of Innsbruck, Innsbruck, Austria (E-mail: Justus.Piater@uibk.ac.at)

Table 1. List of abbreviations

BT	Behavior Tree
RBT	Reconfigurable Behavior Tree
DS	Dynamical System
ESDS	Energy-based Stabilizer of Dynamical Systems
WM	Working Memory
LTM	Long-Term Memory

sions. To improve readability, a list of abbreviations used throughout the paper is provided in Table 1.

2. Related work

2.1 Motion planning with dynamical systems

A prominent approach in the field of DS-based motion generation are Dynamic Movement Primitives (DMP) [14]. In a DMP, a linear dynamics is summed to a nonlinear forcing term used to reproduce a given trajectory. The nonlinear term is suppressed by a time-dependent clock signal to guarantee convergence to a given target. DMPs have been extended in several ways. Among the others: [15] tests different approaches to chain multiple DMPs representing position and orientation trajectories, while [16] introduces task-dependent parameters to customize the execution. The time-dependency is the main drawback of DMPs, since it limits the generalization capabilities pf the DMP outside the demonstration area [17].

The stable estimator of dynamical systems (SEDS) [9] is probably the first example of stable and time-independent DS learned from demonstration. A known problem of SEDS is the lack of accuracy in reproducing nonlinear motions. This problem is alleviated in [18] by learning a Lyapunov function that complies with the demonstrations and stabilizes the DS with minimal intervention, and in [17] by learning a diffeomorphism that maps the training data into a space where the SEDS works accurately. These approaches permit an accurate reproduction of the demonstrated motion, but introduce extra training time and open parameters to fit also the Lyapunov function or the diffeomorphism.

Some approaches attempt to find a compromise between training time and accuracy [13, 19, 20]. Blocher et al. [19] applies only if Gaussian mixture regression is used to encode the nonlinear system, while the approach in [20] works with a single demonstration. In previous work [13], we propose a solution that works with any regression technique and multiple demonstrations, offering a good compromise between accuracy and training time. This approach, outlined in Sect. 3.1, is exploited here to generate the motion trajectories.

2.2 Task scheduling and execution monitoring

Behavior Trees (BT) [2, 21] are a popular approach for task scheduling that extend and generalize several other approaches including decision trees [22], the subsumption architecture [23], and sequential behavior composition [24]. BTs use a fixed number of node types to build a high-level representation of a robotic task as a rooted tree. A BT is expressive, human-readable, modular, and reusable; all features that make BTs a popular and attractive technique. However, BTs are a high-level task scheduling approach where the decision making is decoupled from the physical executive state and possible ambiguities in the execution are not considered.

During the operational life of a robot, unexpected events may occur and a certain degree of flexibility in the task scheduling is beneficial. Approaches like [25, 26] permit continuous monitoring of the task execution by integrating perceptual stimuli in the decision making. We have exploited this functionality to build a framework for learning, monitoring, and executing manipulation tasks [27–29]. The drawback of these approaches is that they may lose the modularity and reusability of BTs. In [4], we proposed Reconfigurable Behaviour Trees (RBTs) as an executive framework that combines high-level decision making and low-level control features. RBTs are described in Sect. 3.2.

3. Methods

In this section, we present the two components of our framework, namely the Energy-based Stabilizer of Dynamical Systems (ESDS) for online motion generation and the Reconfigurable Behavior Tree (RBT) for task scheduling and execution monitoring.

3.1 Energy-based stabilizer of dynamical systems

Dynamical system definition ESDS encodes the demonstrated trajectories into a nonlinear DS. Without loss of generality, we assume that such a DS has an equilibrium at $\hat{\boldsymbol{x}} = \mathbf{0}$ where also the velocity vanished. Under this assumption, the DS used by ESDS can be written as

$$\dot{\boldsymbol{x}} = -\boldsymbol{x} + \gamma(z,s)\kappa(\|\boldsymbol{x}\|)\boldsymbol{f}(\boldsymbol{x}), \tag{1}$$

where $\boldsymbol{f}(\boldsymbol{x})$ is a smooth and nonlinear vector field, $\kappa(\|\boldsymbol{x}\|) = 1 - e^{-0.1\|\boldsymbol{x}\|^2}$ ensures that (1) has equilibrium point at $\hat{\boldsymbol{x}} = \mathbf{0}$, and $\gamma(z,s)$ a stabilizing gain defined later in this section. ESDS makes no prior assumption on the stability of $\boldsymbol{f}(\boldsymbol{x})$ and uses $\gamma(z,s)$ to ensure global convergence to $\hat{\boldsymbol{x}}$ at runtime. This permits learning the nonlinear term $\boldsymbol{f}(\boldsymbol{x})$ from training data using any regression technique, using the approach described as follows.

Learning from demonstration The nonlinear term $\boldsymbol{f}(\boldsymbol{x})$ in (1) is learned in a supervised manner. In our setting, a demonstrated trajectory $\{\boldsymbol{x}_t, \dot{\boldsymbol{x}}_t\}_{t=1}^{T}$ consists of desired robot positions $\boldsymbol{x}_t$ and velocities $\dot{\boldsymbol{x}}_t$ sampled in T consecutive time instances. This has to be cast into input/output pairs for supervised learning. To this end, we assume $\gamma(z,s) = 1$ and rewrite (1) as

$$\kappa^{-1}(\|\boldsymbol{x}\|)\,(\dot{\boldsymbol{x}} + \boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x}), \tag{2}$$

where

$$\kappa^{-1}(\|\boldsymbol{x}\|) = \begin{cases} \frac{1}{\kappa(\|\boldsymbol{x}\|)} & \|\boldsymbol{x}\| \neq 0 \\ 1 & \text{otherwise} \end{cases}. \tag{3}$$

The relation (2) clearly shows that $\boldsymbol{f}(\boldsymbol{x})$ is a nonlinear mapping between the position ($\boldsymbol{x}$) and $\kappa^{-1}(\|\boldsymbol{x}\|)\,(\dot{\boldsymbol{x}} + \boldsymbol{x})$. Hence, we define the input data as $\mathcal{I} = \{\boldsymbol{x}_t\}_{t=1}^{T}$ and the output data as $\mathcal{O} = \left\{\kappa^{-1}(\|\boldsymbol{x}_t\|)\,(\dot{\boldsymbol{x}}_t + \boldsymbol{x}_t)\right\}_{t=1}^{T}$. The procedure is repeated in case multiple demonstrations are provided. Once the input/output pairs $\mathcal{I}/\mathcal{O}$ are computed, any regression technique can be used to retrieve an estimate of $\boldsymbol{f}(\boldsymbol{x})$ for each input state. It is worth mentioning that the self-defined inverse function $\kappa^{-1}(\cdot)$ in (3) does not generate discontinuities in the training data since $\dot{\boldsymbol{x}}_t$ and $\boldsymbol{x}_t$ vanishes while approaching the equilibrium point.

Online stabilization The dynamical system (1) has an equilibrium point at $\hat{\boldsymbol{x}} = 0$, but so far we have not discussed the stability of this equilibrium. Global stability of the equilibrium, implying convergence of the trajectory to a given target, is of fundamental importance to apply a DS to robot motion generation. In ESDS, the dynamics is stabilized at runtime by the stabilizing gain $\gamma(z,s)$, which

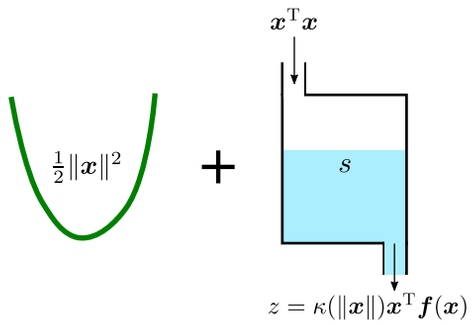


Fig. 1. The Lyapunov function used to prove the stability of (1)

Table 2. Definition of the function $\gamma(z,s)$

$h_1(x,\underline{x},\overline{x}) = \begin{cases} [\mathit{II}]1 & x \geq \overline{x} \\ 0 & x \leq \underline{x} \\ 0.5(1+\sin\left(\pi\left(\frac{x-\underline{x}}{\overline{x}-\underline{x}}-0.5\right)\right) & \text{otherwise} \end{cases}$
$h_2(x,\underline{x},\overline{x}) = 1 - h_1(x,\underline{x},\overline{x})$
$\alpha(s) = \min(0.99, h_1(s,0,0.1\kappa(\|\boldsymbol{x}\|)\overline{s}) \cdot h_2(s,0.9\kappa(\|\boldsymbol{x}\|)\overline{s},\kappa(\|\boldsymbol{x}\|)\overline{s}))$
$\beta(z,s) = 1 - h_1(z,-0.01,0) \cdot h_2(s,0,0.1\kappa(\|\boldsymbol{x}\|)\overline{s})$ $-h_1(s,0.9\kappa(\|\boldsymbol{x}\|)\overline{s},\kappa(\|\boldsymbol{x}\|)\overline{s}) \cdot h_2(z,0,0.01)$
$\gamma(z,s) = 1 - h_1(z,0,0.01) \cdot h_2(s,0,0.1\kappa(\|\boldsymbol{x}\|)\overline{s})$

is derived from energy considerations. We summarize the main aspects of this derivation and refer to [13] for further details.

Stability of nonlinear dynamical systems is usually analyzed using Lyapunov theory [30] by defining a positive definite and vanishing at the equilibrium (Lyapunov) function and showing that its time derivative is negative definite and vanishes at the equilibrium. For the DS (1), we consider the Lyapunov function depicted in Fig. 1, consisting of a quadratic potential $\frac{1}{2}\|\boldsymbol{x}\|^2$ and an energy tank. The level of the energy tank is represented by the additional state variable s. Taking the time derivative of this function, it is possible to find stability conditions that affects the value of s and its rate of change $\dot{s}$. In particular, one can prove that the energy level s is increased by the term $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x}$ and decreased when $z = \kappa(\|\boldsymbol{x}\|)\boldsymbol{x}^{\mathrm{T}}\boldsymbol{f}(\boldsymbol{x}) < 0$. Therefore, the value of s and $\dot{s}$ can be controlled to render the function in Fig. 1 a proper Lyapunov function. This is obtained by assigning an initial value $\overline{s}$ to s and by designing $0 \leq \gamma(z,s) \leq$ to prevent that s becomes negative. The function $\gamma(z,s)$ is computed as summarized in Table 2. Note that the dynamics $-\boldsymbol{x}$ in (1) vanishes only at the equilibrium and prevents the DS from stopping in a spurious attractor if $\gamma(z,s) = 0$. As detailed in [13], the initial value of s can be efficiently estimated from training data.

Comparison with existing approaches The effectiveness of ESDS is demonstrated on a public benchmark (the LASA Handwriting dataset[1]) containing 26 2D motions. For a quantitative comparison, we consider the reproduction accuracy and the training time. Accuracy is measured with the Swept Error Area (SEA) metric [18] that represents the distortion (area) between a generated trajectory and the relative demonstration. Comparative results from [13] are reported in Table 3, using a third-party implementation[2] for CLF-DM. The comparison shows that ESDS offers a good compromise between accuracy and training time, especially considering that it does not impose any restriction on the regression technique used to retrieve the motion.

Table 3. Reproduction error and training time (mean / range) of different approaches on the LASA dataset

Approach	*SEA* [mm^2]	Train. Time [s]
ESDS [13]	431.5 / [26.0-1307]	0.08 / [0.03-0.17]
CLF-DM [18]	460.7 / [16.6-1269]	2.3 / [0.09-21.5]
τ-SEDS [17]	537.0 / [26.4-1139]	25.3 / [7.6-55.4]
C-GMR [19]	496.7 / [20.3-1840]	0.1 / [0.03-0.28]

3.2 Task monitoring with reconfigurable behavior trees

Behavior trees A BT is a graphical language that models the behavior of an autonomous agent as rooted trees obtained by combing the finite set of primitive nodes types shown in Table 4. Condition and Action nodes are *execution* nodes, while Sequence, Fallback/Selector, Decorators, and Parallel are *control flow* nodes. During the execution, each node enters a *running* state that terminates with a *success* or a *failure*. Executing a BT means periodically traversing the tree top-down and left to right. The traversal is regulated by a clock signal called "tick". As shown in Table 4, each node in the BT responds to the tick depending on its own type and on the return state of the other nodes.

Reconfigurable behavior trees Intuitively, we can think of a Reconfigurable Behavior Tree (RBT) [4] as a BT with branches that can be dynamically allocated and deallocated. The dynamic allocation mechanism is triggered by environmental stimuli like object addition or removal. The continuous monitoring of such stimuli, as well as the dynamic allocation of the tree's branches, requires the following functionalities: *i)* a *Long-Term Memory (LTM)*, organized in JSON schemata, to conveniently store tree branches that are dynamically loaded into a *Working Memory (WM)*, *ii)* an *Emphasizer* that transforms logical pre- and postconditions and sensory data into a priority assigned to each branch in the LTM, and *iii)* an *Instantiator* process that queries the LTM and dynamically load the subtree in the WM. Distinctive features of a RBT are implemented using only the six nodes in Table 4, i.e. without increasing the design effort.

Task monitoring and execution The generic RBT, depicted in Fig. 2, combines static and dynamic nodes. It is a goal oriented task scheduler since it terminates as soon as the `goal reached` condition turns `True`. It exploits a *blackboard* to share variables across the nodes and store the logical pre- and postconditions and the priority list. The blackboard is a thread-safe mechanism that greatly simplifies the communication between nodes. The core of the RBT are the green and blue nodes that are executed in parallel, preserving the asynchronous nature of sensor readings and decision making. The Emphasizer (green node) transforms sensory input into subtree priorities and it never terminates (is always in the *Running* state). This, and the fact the Parallel node parent of the Emphasizer terminates only if its two children do, let the RBT run until the goal is reached. The blue nodes, which are dynamically allocated at each tick, load from the LTM the branch with higher priority and prepare a small BT ready for execution. The dynamic allocation of the blue nodes is required to prevent deadlocks, letting the RBT reach the goal.

[1] https://bitbucket.org/khansari/lasahandwritingdataset.

[2] https://bitbucket.org/khansari/clfdm/src/master/.

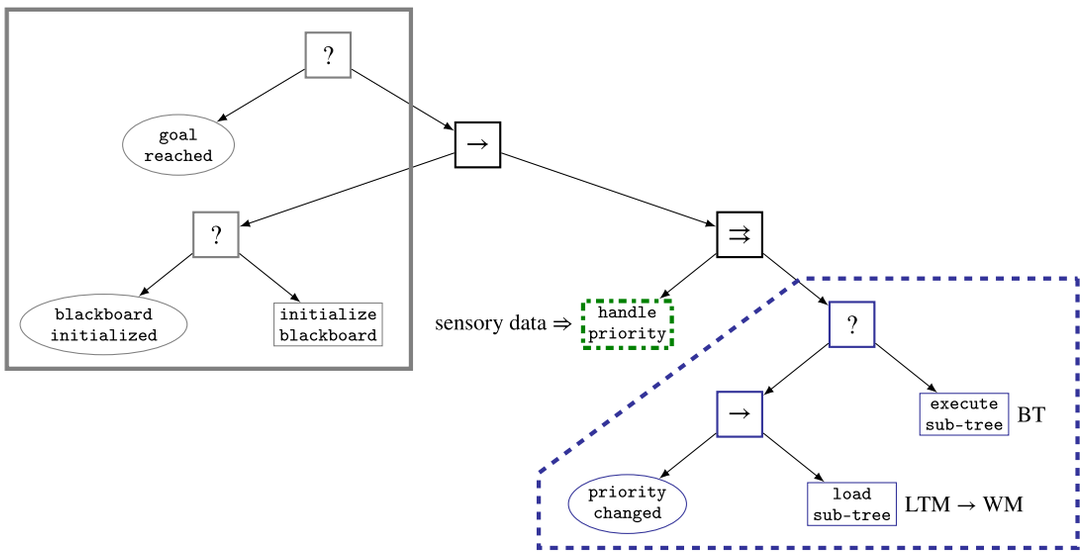


Fig. 2. The generic RBT combines static and dynamic nodes. The nodes surrounded by the gray thick square permits terminating the task once a global goal is reached. The *Emphasizer* (green dot-dashed node) changes the priority level of each box using perceptual information (robot-box distances). The *Instantiator* is responsible for allocating and deallocating the nodes surrounded by the blue dashed polygon when the box priorities change. The action node `change box` online modifies the box to sort based on the priority, while action node `sort box` generates the pick-and-place subtask (Color figure online)

Table 4. BT nodes and their return status

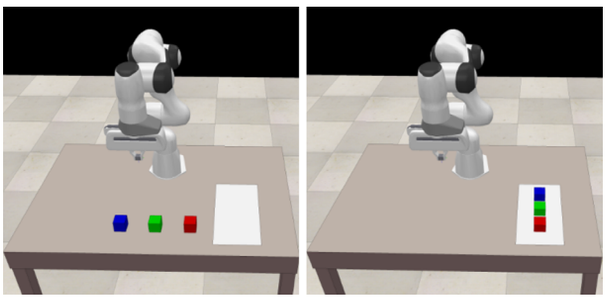
Type	Symbol	Success	Failure
Fallback/Selector	?	One child succeeds	All children fail
Sequence	→	All children succeed	One child fails
Parallel	⇉	>M children succeed	>$N - M$ children fail
Decorator	◊	Custom	Custom
Action	▭	Upon completion	Impossible to complete
Condition	⬭	`True`	`False`

Instantiator The procedure presented in Algorithm 1 is used by the *Instantiator* to query a tree branch from the LTM and instantiate an executable BT. The branch to load is identified by a unique label that serves as root of the BT (line 2). Once the JSON schemata are queried, the tree is built top-down by iteratively casting JSON schemata into the corresponding BT nodes and adding them to the tree (lines 4 to 19). Similarly to [2], each postcondition is represented by a Condition node (line 7) and attached to the existing tree with a Fallback $\mathcal{T}_{fal}$ (lines 8–9). This permits terminating the execution when the postcondition is `True`. Multiple postconditions are attached to a Sequence node (line 11) and therefore sequentially checked. After these steps, the Sequence node is connected to $\mathcal{T}_{fal}$ (line 12) that now contains all the postconditions and can be connected the BT (line 18). Preconditions are also cast into Condition nodes (line 16), while Action nodes are used to represent robot motions represented as stable dynamical systems (see Sect. 3.1). An Action can be executed only if all its preconditions are `True`. This is achieved by connecting Action and Conditions to a Sequence node that is then attached to the BT (lines 17–18).

Emphasizer In contrast to traditional BTs, RBTs allows a dynamic reconfiguration of the execution order and a continuous monitoring of the task execution. This is obtained by exploiting logical pre- and postconditions and continuous environmental stimuli. Pre- and

Algorithm 1 Load and instantiate a BT

```
1:  function INSTANTIATESUBTREE(l)                    ▷ l: subtask name
2:      schemaList ← GETTASKFROMLTM(l)
3:      T ← {}                                        ▷ empty BT
4:      for schema in schemaList do
5:          T ← SCHEMATONODE(T, schema)
6:          postC ← GETPOSTCONDITIONS(schema)
7:          C ← CONDITIONNODES(postC)
8:          if C.length == 1 then
9:              T_fal ← FALLBACKNODE(C)
10:         else
11:             T_seq ← SEQUENCENODE(C)
12:             T_fal ← ATTACHSUBTREE(T_seq)
13:         end if
14:         a, preC ← GETACTIONS(schema)
15:         A ← ACTIONNODES(a)
16:         C ← CONDITIONNODES(preC)
17:         T_seq ← SEQUENCENODE(C, A)
18:         T ← ATTACHSUBTREE(T_fal, T_seq)
19:     end for
20:     return T
21: end function
```



(a) Initial state. (b) Goal state.

Fig. 3. An illustration of the box sorting task. The manipulator has to pick one by one the boxes from the table and place them in the white (storage) area

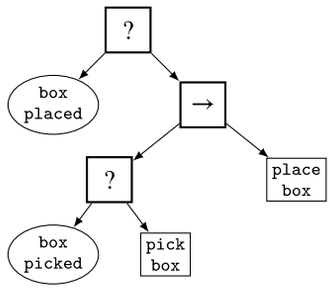
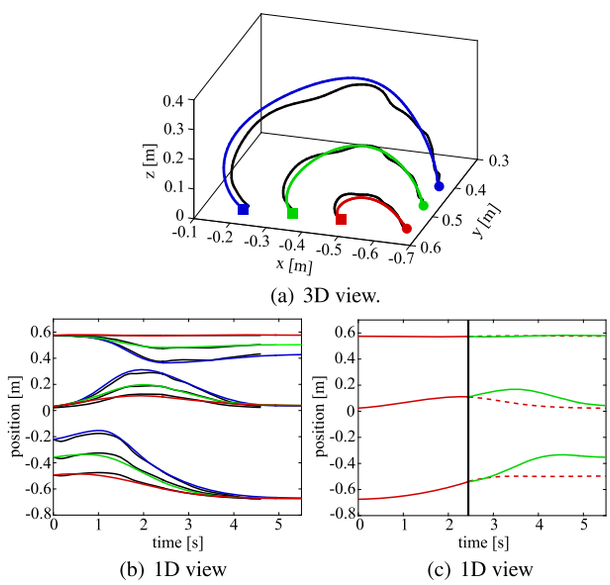


Fig. 4. The BT used to sort (pick from the table and place in the storage area) a generic box. This BT replaces the generic `execute subtree` node in Fig. 2



(a) 3D view.

(b) 1D view (c) 1D view

Fig. 5. (a)-(b) Placement trajectories generated with ESDS. The black solid lines indicate the demonstrations. Colored lines are the trajectories generated for each box. (c) Picking trajectories generated with ESDS. The vertical black dashed line indicates the time when the red box is removed from the scene. Starting from this position a new motion is generated to pick the green box (green solid line). The red dashed line shows the originally-planned trajectory (`pick r_box`)

postconditions are used to identify *active* branches in the tree, i.e. subtrees with `True` preconditions and at least a `False` postconditions. The Emphasizer periodically looks for active subtrees and determines if there are execution conflicts, i.e. multiple branches that are concurrently active. This ambiguity in the decision process is resolved using a priority-based mechanism. We introduce a priority for each active branch, a real value normalized between 0 and 1, and use it to determine which subtree has to be loaded and executed. Following [4], we define the priority e as

$$e(\omega) = \begin{cases} 1 & \text{if } \omega \leq \omega_{\min} \\ \frac{\omega - \omega_{\max}}{\omega_{\min} - \omega_{\max}} & \text{if } \omega_{\min} < \omega < \omega_{\max} \\ 0 & \text{if } \omega \geq \omega_{\max} \end{cases} \tag{4}$$

where ω is the distance between the robot end-effector and the object to manipulate, and the thresholds $\omega_{\min}$ and $\omega_{\max}$ represent the minimum and maximum distance to the object. A typical choice for $\omega_{\min}$ is the size of the object, while $\omega_{\max}$ represent the maximum distance at which the object can be successfully grasped.

4. Evaluation

We evaluate our framework in the box sorting task depicted in Fig. 3. The robotic has to sort colored boxes (`r_box`, `b_box`, and `g_box`) by picking them from the table (Fig. 3(a)) and placing them in the "storage" area indicated by a white patch (Fig. 3(b)). The Panda robot and the operational space use CoppeliaSim [31] and the robot model identified in [32]. Each box can be sorted by executing the BT shown in Fig. 4, where the generic `box` reads as `r_box`, `b_box`, or `g_box`. The presented scenario is simple, but it is sufficient to show the modularity and reusability of the proposed solution. Indeed, the nodes in Fig. 4 can be abstracted into the higher-level action node `execute subtree` in Fig. 2 (modularity). Moreover, the subtree in Fig. 4 can be exploited to pick and place similar objects (reusability), like the 3 colored boxes in Fig. 3. The switching between the 3 sorting subtasks is regulated by a RBT like the one in Fig. 2. At runtime, the sorting BT of the closest (highest priority) box is loaded and connected to the BT using Algorithm 1, replacing the block `execute subtree`. To compute the subtree priority (4), we choose $\omega_{\min}$ as the length of the box side ($\omega_{\min} = 0.05$ m) and we estimate the maximum distance that still allows grasping a box to be $\theta_{\max} = 1$ m. The RBT successfully terminates if the 3 boxes are sorted in the storage area. This is obtained by defining the RBT goal as `r_box placed` $\wedge$ `b_box placed` $\wedge$ `g_box placed`. RBTs are implemented in Python using the basic BT nodes provided by `py_trees`.[3] In our implementation, the RBT has 19 nodes, obtained by merging the trees in Fig. 2 and Fig. 4. Tree traversals (ticks) are periodically performed every 38 ms. For comparison, a standard BT requires 151 nodes to schedule the same task [4]. This result in an increase in tick time of ≈ 38 %.

The `pick box` and `place box` action nodes in Fig. 4 are mapped to stable dynamical systems using the ESDS approach presented in Sect. 3.1. The DS representing each motion is learned from a demonstrated trajectory using a Gaussian processes [33]. The training data and the retrieved trajectories are shown in Fig. 5. At runtime, the system generates a smooth trajectory connecting the current end-effector position with a given target position. The target for picking actions is the box position, while for placement actions it is the desired position in the storage area. Trajectories generated by ESDS (implemented in Matlab®) are depicted in Fig. 5.

[3] https://py-trees.readthedocs.io.

Nominal execution In this experiment, the robot performs the sorting task in an "ideal" scenario where the boxes are placed on the table like in Fig. 3(a) and no external perturbation occurs. The robot performs the sorting starting with the blue box (closest), then switches to the red box, and finally to the green one. As already mentioned, the task execution order is regulated by the distance between the robot gripper and the boxes. After placing the `b_box` in the storage area, the RBT updates the priority of `r_box` and `g_box`. Since `r_box` is the closest to the robot, the subtree `sort r_box` (Fig. 4) is loaded and executed. The `sort g_box` subtask is executed at the end and the RBT successfully terminates.

External perturbations In this experiment, we introduce a perturbation during the execution of the sorting task. The task is the same as depicted in Fig. 4 and described in the previous paragraph. As before, the robot performs the sorting starting with the blue box (closest) and then switches to the `r_box`. However, during the execution of the `pick r_box` action, we remove the `r_box` from the scene. The system detects this incident and promptly reacts by loading the `sort g_box` subtask. ESDS replans the pick trajectory on the fly without discontinuities (Fig. 5(c)). Once the green box is sorted, the RBT does not terminate since the goal `r_box placed` $\wedge$ `b_box placed` $\wedge$ `g_box placed` is `False` and keeps monitoring the scene to detect eventual changes. At this point, one can place the red box in the storage area or back on the table. If `r_box` is placed in the storage area then `r_box placed` becomes `True` and the task successfully terminates. In case `r_box` is placed back to the table, the Instantiator loads the `sort r_box` subtree and the sorting task successfully terminates.

5. Conclusion and future work

In this work, we presented a framework for monitoring and executing robotic tasks. The framework has two key components, namely the Reconfigurable Behavior Trees and the Energy-based Stabilizer of Dynamical Systems. RBTs are a novel executive framework that features high-level decision making and low-level control capabilities, enabling continuous monitoring and task switching. ESDS learns a flexible and robust motion representation from a handful of demonstrations that permits generating motion trajectories with proved convergence. The proposed framework is tested in a sorting scenario, showing its capabilities of handling perturbations during task execution.

In future work, we plan to test our framework on real robots and to provide a more comprehensive evaluation in challenging human-robot interaction scenarios. We will also investigate the possibility of learning also the RBT from human demonstrations.

Acknowledgements

Open access funding provided by University of Innsbruck and Medical University of Innsbruck. This research has received funding from the European Union's Horizon 2020 research and innovation programme (grant agreement no. 731761, IMAGINE) and from the Austrian Research Foundation (Euregio IPN 86-N30, OLIVER).

References

1. López, J., Santana-Alonso, A., Díaz-Cacho Medina, M. (2019): Formal verification for task description languages. A Petri net approach. Sensors, 19(22), 4965.
2. Colledanchise, M., Ögren, P. (2018): Behavior trees in robotics and AI: an introduction. Boca Raton: CRC Press.
3. Ghallab, M., Nau, D., Traverso, P. (2014): The actor's view of automated planning and acting: a position paper. Artif. Intell., 208, 1–17.
4. de la Cruz, P., Piater, J., Saveriano, M. (2020): Reconfigurable behavior trees: towards an executive framework meeting high-level decision making and control layer features. In International conference on systems, man, and cybernetics. In press. arXiv:2007.10663.
5. Saveriano, M., Lee, D. (2013): Point cloud based dynamical system modulation for reactive avoidance of convex and concave obstacles. In International conference on intelligent robots and systems (pp. 5380–5387).
6. Khansari-Zadeh, S. M., Billard, A. (2012): A dynamical system approach to realtime obstacle avoidance. Auton. Robots, 32(4), 433–454.
7. Saveriano, M., Lee, D. (2014): Distance based dynamical system modulation for reactive avoidance of moving obstacles. In International conference on robotics and automation (pp. 5618–5623).
8. Saveriano, M., Hirt, F., Lee, D. (2017): Human-aware motion reshaping using dynamical systems. Pattern Recognit. Lett., 96, 96–104.
9. Khansari-Zadeh, S. M., Billard, A. (2011): Learning stable non-linear dynamical systems with Gaussian mixture models. IEEE Trans. Robot., 27(5), 943–957.
10. Kronander, K., Khansari-Zadeh, S. M., Billard, A. (2015): Incremental motion learning with locally modulated dynamical systems. Robot. Auton. Syst., 70, 52–62.
11. Saveriano, M., Lee, D. (2018): Incremental skill learning of stable dynamical systems. In International conference on intelligent robots and systems (pp. 6574–6581).
12. Saveriano, M., Lee, D. (2019): Learning barrier functions for constrained motion planning with dynamical systems. In International conference on intelligent robots and systems (pp. 112–119).
13. Saveriano, M. (2020): An energy-based approach to ensure the stability of learned dynamical systems. In International conference on robotics and automation (pp. 4407–4413). In press.
14. Ijspeert, A., Nakanishi, J., Pastor, P., Hoffmann, H., Schaal, S. (2013): Dynamical movement primitives: learning attractor models for motor behaviors. Neural Comput., 25(2), 328–373.
15. Saveriano, M., Franzel, F., Lee, D. (2019): Merging position and orientation motion primitives. In International conference on robotics and automation (pp. 7041–7047).
16. Pervez, A., Lee, D. (2018): Learning task-parameterized dynamic movement primitives using mixture of GMMs. Intell. Serv. Robot., 11(1), 61–78.
17. Neumann, K., Steil, J. J. (2015): Learning robot motions with stable dynamical systems under diffeomorphic transformations. Robot. Auton. Syst., 70, 1–15.
18. Khansari-Zadeh, S. M., Billard, A. (2014): Learning control Lyapunov function to ensure stability of dynamical system-based robot reaching motions. Robot. Auton. Syst., 62(6), 752–765.
19. Blocher, C., Saveriano, M., Lee, D. (2017): Learning stable dynamical systems using contraction theory. In International conference on ubiquitous robots and ambient intelligence (pp. 124–129).
20. Perrin, N., Schlehuber-Caissier, P. (2016): Fast diffeomorphic matching to learn globally asymptotically stable nonlinear dynamical systems. Syst. Control Lett., 96, 51–59.
21. Colledanchise, M., Ögren, P. (2016): How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees. IEEE Trans. Robot., 33(2), 372–389.
22. Breiman, L., Friedman, J., Stone, C. J., Olshen, R. A. (1984): Classification and regression trees. Boca Raton: CRC Press.
23. Brooks, R. (1986): A robust layered control system for a mobile robot. IEEE J. Robot. Autom., 2(1), 14–23.
24. Burridge, R. R., Rizzi, A. A., Koditschek, D. E. (1999): Sequential composition of dynamically dexterous robot behaviors. Int. J. Robot. Res., 18(6), 534–555.
25. Borji, A., Ahmadabadi, M. N., Araabi, B. N., Hamidi, M. (2010): Online learning of task-driven object-based visual attention control. Image Vis. Comput., 28(7), 1130–1145.
26. Caccavale, R., Finzi, A. (2016): Flexible task execution and attentional regulations in human-robot interaction. Trans. Cognitive Develop. Syst., 9(1), 68–79.

27. Caccavale, R., Saveriano, M., Fontanelli, G. A., Ficuciello, F., Lee, D., Finzi, A. (2017): Imitation learning and attentional supervision of dual-arm structured tasks. In Joint international conference on development and learning and epigenetic robotics (pp. 66–71).
28. Caccavale, R., Saveriano, M., Finzi, A., Lee, D. (2019): Kinesthetic teaching and attentional supervision of structured tasks in human–robot interaction. Auton. Robots, 43(6), 1291–1307.
29. Saveriano, M., Seegerer, M., Caccavale, R., Finzi, A., Lee, D. (2019): Symbolic task compression in structured task learning. In International conference on robotic computing (pp. 171–176).
30. Slotine, J. J. E., Li, W. (1991): Applied nonlinear control. Englewood Cliffs: Prentice-Hall.
31. Rohmer, E., Singh, S. P. N., Freese, M. (2013): CoppeliaSim (formerly V-REP): a versatile and scalable robot simulation framework. In International conference on intelligent robots and systems (pp. 1321–1326).
32. Gaz, C., Cognetti, M., Oliva, A., Robuffo Giordano, P., De Luca, A. (2019): Dynamic identification of the franka emika panda robot with retrieval of feasible parameters using penalty-based optimization. IEEE Robot. Autom. Lett., 4(4), 4147–4154.
33. Rasmussen, C. E., Williams, C. K. I. (2006): Gaussian processes for machine learning. Cambridge: MIT Press.

Authors

Matteo Saveriano
is a tenure-track assistant professor at the Intelligent and Interactive Systems lab and at the Digital Science Center (DiSC) at University of Innsbruck. He received his B.Sc. (2008) and M.Sc. (2011) in Automatic Control Engineering from University of Neaples "Federico II", and a Ph.D. in Robotics from the Technical University of Munich (2017). After the Ph.D., he was a postdoctoral researcher at the German Aerospace Center (DLR) until May 2019. His research interests include safe human-robot interaction, reactive collision avoidance, representation and classification of human activities, imitation learning, and intuitive skill transfer. Matteo has (co)-authored more than 30 articles in international, peer-reviewed journals, conferences and workshops. He has participated in the EU FP7 research projects AIRobots and SAPHARI, and in the German DFG project ROLITOS.

Justus Piater
professor of computer science at the University of Innsbruck, is the founder and leader of the IIS group. He earned his Ph.D. in computer science at the University of Massachusetts Amherst in 2001 after spending a year as a Fulbright graduate student, followed by two years of postdoctoral research at GRAVIR-IMAG, INRIA Rhone-Alpes, on a Marie-Curie Individual fellowship. After eight years as a professor at the University of Liege, Belgium, including one year as a visiting scientist at the Max Planck Institute for Biological Cybernetics in Tübingen, Germany, Prof. Piater moved to Innsbruck in 2010. There he founded the IIS group at the Institute of Computer Science, which currently counts 5 postdoctoral researchers and 6 doctoral students. He has written more than 180 articles in international, peer-reviewed journals, conferences and workshops several of which have received best-paper awards, and has been a principal investigator in 1 EU-FP6, 6 EU-FP7 and 1 H2020 projects. He is the coordinator of the currently active H2020 project IMAGINE.