Temporal Behavior Trees as a Multi-Purpose Specification Language for Robotics

Sebastian Schirmer
DLR German Aerospace Center
Braunschweig, Germany
sebastian.schirmer@dlr.de

Aniruddh G. Puranic
University of Maryland
Collage Park, USA
puranic@umd.edu

Bernd Finkbeiner
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
finkbeiner@cispa.de

Sriram Sankaranarayanan
University of Colorado Boulder
Boulder, USA
srirams@colorado.edu

***Abstract*—This paper presents a survey of tools and methods for Temporal Behavior Trees (TBTs). TBTs are a specification language that retrofits properties expressed in a formal language to Behavior Trees (BT). BTs are a prominent control structure for programming complex robotic tasks. Thus, TBTs bridge the gap between formal methods and robotics. We review prior work demonstrating how TBTs can be used to analyze and improve robotic executions. In particular, we discuss trace segmentation that identifies which parts of a BT execution requires further refinement, and trace repair that provides explanations of how a BT execution should have been proceeded in order to satisfy the TBT specification. Furthermore, we highlight recent work showing that TBTs serve as a multi-purpose specification for robotics. In particular, TBTs have been applied in reinforcement learning as a structured reward function and for denoising imperfect demonstrations.**

I. Introduction

Behavior Trees (BTs) have become a widely adopted framework for designing control in robotics and autonomous systems due to their modularity, readability, and scalability [1]–[5]. Originally popularized in the gaming industry [6], BTs provide a structured way to compose complex behaviors from operators such as sequences, fallbacks, parallel composition, and repetitions over leaf actions.

However, classical BTs lack an explicit representation of an objective or constraint associated with actions. While they describe what actions should be executed and in what order, they do not capture when actions should occur or within which time bounds they must be completed. This limitation becomes critical in robotic systems operating in real-world environments, where timing, coordination, and temporal dependencies are essential. Temporal Behavior Trees (TBTs) address this shortcoming by incorporating temporal logic properties, enabling the specification of time-bounded and temporally constrained behaviors. As a result, this extension transforms BTs from purely execution-oriented structures into rich specifications that can be used for multiple purposes. [1]

As an example consider Figure 1 that shows an excerpt of a TBT that specifies an automated landing maneuver of an unmanned aerial vehicle (UAV) on a ship. The TBT decomposes the maneuver into two parts before descending: first, the UAV moves to a position diagonally behind the ship and holds there for five seconds; second, the UAV transitions above the touchdown point while aligning its heading with the heading of the ship. Typically, the TBT would include other fallback maneuvers, such as approaching the ship from behind or from the side, which we omit here for brevity. The formulas at the TBT's leaf nodes specify the successful execution of the various actions. Note that the considered maneuver can alternatively be expressed as a single leaf node with $\Diamond\Box_{[0,5]}\, diagonalBehind \wedge \Diamond_{[6,6]}(alignedHeading\ \mathcal{U}(\ aboveTouchdown\ \wedge\ \bigcirc\Diamond\, descended))$. However, modifying temporal constraints, e.g., $\Box_{[0,5]}$, generally requires corresponding adjustments to other parts of the specification, e.g., $\Diamond_{[6,6]}$. Such dependencies can easily introduce inconsistencies, especially as behaviors become more complex and temporal correlations between sub-specifications increase. In essence, TBTs inherit key properties of BTs such as modularity, readability, and scalability.

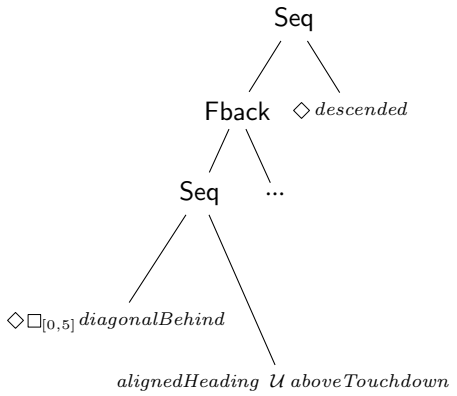

Fig. 1: Excerpt from a TBT specifying a UAV landing maneuver on a ship [8]

In this paper, we first present TBTs and then explore their multiple purposes: execution understanding via trace segmentation, error explanation based on trace repair, learning from demonstrations, and reward design. Trace segmentation assigns portions of the TBT specification to segments of a BT execution trace [7]. Trace repair minimally modifies a given violating trace such that the resulting trace satisfies the specification [8]. This technique has been used to denoise

[1]The content of this paper is based on prior work [7]–[10].

imperfect demonstrations prior to learning from them [10]. Finally, a TBT specification can be interpreted as a reward machine that continuously provides reward to an agent during reinforcement learning [9].

II. Temporal Behavior Trees

We define TBTs as in [7], [8]. A trace is a finite sequence of observable states of a system: $\sigma(1), \ldots, \sigma(N)$, wherein $\sigma(i)$ denotes the state at time $i \in [1, N]$. We write the length of the trace as $|\sigma|$. A trace with $|\sigma| = 0$ is the *empty trace*. Given trace σ, we denote $\sigma[l : u]$ as a slice of the trace from "offsets" l to u (inclusive), as follows: $\sigma[l : u] = \sigma(l+1), \cdots, \sigma(\min(u+1, N))$ if $l \leq u$ and $l < N$ and the empty trace otherwise. For convenience, let $\sigma[l :] = \sigma[l : N-1]$ and $\sigma[: u] = \sigma[0 : u]$. Note that offsets begin with 0 and end at $|\sigma| - 1$. For instance, $\sigma[0 : 3] = \sigma(1), \ldots, \sigma(4)$ and $\sigma[5 :] = \sigma(6), \ldots, \sigma(N)$.

A TBT borrows operators from behavior trees to control the order of actions.

Definition 1 (Syntax of Temporal Behavior Trees)**.** We construct a TBT $\mathcal{T}$ using the following syntax:

$$
\begin{array}{lll}
\mathcal{T} & ::= \mathsf{Fback}([\mathcal{T}, \ldots, \mathcal{T}]), & \leftarrow \text{Fallback node} \\
& \mid \mathsf{Par}_M([\mathcal{T}, \ldots, \mathcal{T}]), M \in \mathbb{N} & \leftarrow \text{Parallel node} \\
& \mid \mathsf{Seq}([\mathcal{T}, \mathcal{T}]), & \leftarrow \text{Sequence node} \\
& \mid \mathsf{Tout}_t(\mathcal{T}), t \in \mathbb{N} & \leftarrow \text{Timeout node} \\
& \mid \bigstar_n(\mathcal{T}), n \in \mathbb{N} \cup \{\infty\} & \leftarrow \text{Repeat node} \\
& \mid \mathsf{Leaf}(\varphi), & \leftarrow \text{Leaf node}
\end{array}
$$

where φ is a local property expressed in a temporal language.

Informally, $\mathsf{Fback}([\mathcal{T}_1, \ldots, \mathcal{T}_n])$ mimics the semantics of a "fallback" node in a behavior tree: at least one of the subtrees $\mathcal{T}_1, \ldots, \mathcal{T}_n$ must eventually be satisfied by the trace. $\mathsf{Par}_M([\mathcal{T}_1, \ldots, \mathcal{T}_n])$ denotes a parallel operator that specifies that at least M distinct subtrees must be satisfied simultaneously by the trace σ. $\mathsf{Seq}([\mathcal{T}_1, \mathcal{T}_2])$ is a sequential node that denotes that σ must be partitioned into two parts $\sigma_1; \sigma_2$ such that $\sigma_1 \models \mathcal{T}_1$ and $\sigma_2 \models \mathcal{T}_2$. For convenience, we rewrite $\mathsf{Seq}([\mathcal{T}_1, (\mathsf{Seq}([\mathcal{T}_2, \cdots, \mathsf{Seq}([\mathcal{T}_{n-1}, \mathcal{T}_n]))))$ for $n \geq 2$ as $\mathsf{Seq}([\mathcal{T}_1, \ldots, \mathcal{T}_n])$. $\mathsf{Tout}_t(\mathcal{T})$ is a timeout node denoting the subtree $\mathcal{T}$ must be satisfied by a prefix of the trace of size t. $\bigstar_n(\mathcal{T})$ is a repeat operator node denoting σ must be partitioned into $k \leq n$ parts $\sigma_1; \sigma_2; \sigma_3; \cdots; \sigma_k$ such that $\sigma_i \models \mathcal{T}$. Finally, a local formula at a leaf specifies that the corresponding trace segment must satisfy the formula. For brevity, we occasionally omit $\mathsf{Leaf}(\varphi)$ and write φ. Note that different temporal language can be combined within a single TBT.

Local properties are specified using Signal Temporal Logic (STL), a prominent specification language for cyber-physical systems. STL is defined recursively as follows:

$$\varphi ::= \text{AP} \mid \neg\varphi \mid \varphi \vee \varphi \mid \Diamond_{[l,u]}(\varphi) \mid \Box_{[l,u]}(\varphi) \mid \varphi \ \mathcal{U}_{[l,u]} \ \varphi$$

with an interval $[l, u]$ wherein $0 \leq l \leq u \leq \infty$. Note that $\bigcirc(\varphi)$ is a shorthand for $\Diamond_{[1,1]}$ as is $\Diamond(\varphi)$ for $\Diamond_{[0,\infty)}$. Similar conventions apply for $\Box(\varphi)$ and $\varphi_1 \mathcal{U} \varphi_2$. The set of *atomic propositions* is AP $\in \{p_1, \ldots, p_m\}$, with each p_i associated with a function f_i that maps states to a real number. In this work, f_i is restricted to be a linear function A trace σ satisfies p_i if and only if $f_i(\sigma(1)) \geq 0$. Informally, STL extends propositional logic by temporal operators, where $\Diamond_{[l,u]}(\varphi)$ requires φ to be satisfied eventually at some point between l and u steps, $\Box_{[l,u]}(\varphi)$ requires φ to be satisfied at every step between l and u, and $\varphi_1 \ \mathcal{U}_{[l,u]} \ \varphi_2$ requires φ_1 to hold *until* φ_2 is eventually satisfied at some point between l and u steps.

Example 1. Let $\mathcal{T}$ be the TBT given in Figure 1. We can add requirements "on top", e.g., $\mathsf{Par}_2([\mathsf{Leaf}(\Box \neg obstacle), \mathcal{T}])$ to specify that obstacles are avoided during landing, and $\mathsf{Fback}([\mathsf{Tout}_t(\mathcal{T}), \mathsf{Leaf}(contingency)])$ to force a contingency once the timeout has been reached. This shows the modular nature of tree specification languages.

Next, we formally define the satisfaction of a TBT specification that uses STL as specification language for its leaf nodes.

Definition 2 (Boolean Semantics of Temporal Behavior Trees)**.** We write $\sigma \models \mathcal{T}$ to denote that trace σ satisfies the TBT $\mathcal{T}$, defined as follows:

$$
\begin{array}{lcl}
\sigma \models \mathsf{Fback}([\mathcal{T}_1, \ldots, \mathcal{T}_n]) & \iff & \exists j \in [1, n], \exists i \in [0, |\sigma| - 1], \sigma[i :] \models \mathcal{T}_j \\
\sigma \models \mathsf{Par}_M([\mathcal{T}_1, \ldots, \mathcal{T}_n]) & \iff & \exists I \subseteq [1, n], |I| \geq M \wedge \forall i \in I, \sigma \models \mathcal{T}_i \\
\sigma \models \mathsf{Seq}([\mathcal{T}_1, \mathcal{T}_2]) & \iff & \exists i \in [0, |\sigma| - 1], \sigma[: i] \models T_1 \wedge \sigma[i + 1 :] \models T_2 \\
\sigma \models \mathsf{Tout}_t(\mathcal{T}) & \iff & \sigma[: t - 1] \models \mathcal{T} \text{ if } t \leq |\sigma| \text{ else } \sigma \models \mathcal{T} \\
\sigma \models \bigstar_n(\mathcal{T}) & \iff & \sigma \models \mathsf{Seq}([\mathcal{T}, \bigstar_{n-1}(\mathcal{T})] \text{ if } |\sigma| > 0 \wedge n > 0 \text{ else } \mathit{true} \\
\sigma \models \mathsf{Leaf}(\varphi) & \iff & \sigma \models \varphi \\
\sigma \models p_i & \iff & f_i(\sigma(1)) \geq 0 \text{ if } |\sigma| > 0 \text{ else } \mathit{false} \\
\sigma \models \neg\varphi & \iff & \sigma \not\models \varphi \\
\sigma \models \varphi_1 \wedge \varphi_2 & \iff & \sigma \models \varphi_1 \wedge \sigma \models \varphi_2 \\
\sigma \models \varphi_1 \vee \varphi_2 & \iff & \sigma \models \varphi_1 \vee \sigma \models \varphi_2 \\
\sigma \models \Diamond_{[l,u]}(\varphi) & \iff & \exists \ i \in [l, u], \ \sigma[i :] \models \varphi \\
\sigma \models \Box_{[l,u]}(\varphi) & \iff & \forall \ i \in [l, u], \ \sigma[i :] \models \varphi \\
\sigma \models \varphi_1 \ \mathcal{U}_{[l,u]} \ \varphi_2 & \iff & \exists i \in [l, u], \quad (\forall j \in [0, i - 1], \ \sigma[j :] \models \ \varphi_1) \ \wedge \ \sigma[i :] \models \varphi_2
\end{array}
$$

Note that by Definition 2, the formula $\neg \bigcirc true$ is true only in the last state of a finite trace, abbreviated as $\bullet$.

TBTs also provide a robust semantics yielding a numerical value [7]. Intuitively, a positive value indicates that the specification is satisfied, while a negative value corresponds to a violation. Additionally, the magnitude of the numerical value reflects the degree to which the specification is satisfied or violated. In essence, we obtain the robust semantics by replacing all instances of $\exists$ and $\vee$ by $\max$ and all instances of $\forall$ and $\wedge$ by $\min$ in Definition 2. For instance, given a signal a that eventually must be greater than two (i.e., $\Diamond(a > 2)$), the robustness computes $\max_{i \in [l,u]}(a_i - 2)$.

III. Multiple Purposes

In this section, we revisit existing algorithms for TBTs. Although TBTs are based on BTs, the presented algorithms work with any traces and do not require executions to be based on BT. We begin with offline monitoring that helps engineers to develop autonomous systems executing complex behaviors.

A. Trace Segmentation

We define the segmentation problem [7] as follows: given a trace and a TBT specification, the goal is to assign portions

of the trace, i.e., $\sigma[i:j]$, to nodes of the TBT specification in a way that maximizes the overall robustness of the specification. Such an assignment is called a segmentation and is represented as a graph that consists of a set of vertices $V = \{(\hat{\mathcal{T}}, i, j) \mid \hat{\mathcal{T}} \text{ is a subtree of } \mathcal{T},\ 0 \leq i \leq j \leq |\sigma| - 1\}$. We compute an optimal assignment using dynamic programming. The resulting dynamic programming table not only yields the optimal assignment but also provides insights into which parts of the specification are violated.

For example, consider the TBT shown in Figure 1. If $(\mathsf{Leaf}(\Diamond\, descended), i, j) \in V$, then the corresponding table entry stores the computed robustness of this node with respect to the segment $\sigma[i, j]$. A positive value indicates that the leaf node $\Diamond\, descended$ is satisfied, whereas a negative value indicates a violation. Hence, if a segment with a violated leaf node exists, an engineer should focus on improving this part of the behavior execution.

However, dynamic programming suffers from scalability issues. To address this, approximation techniques such as subsampling and lazy evaluation have been introduced. Using these techniques, segmentations were computed for several autonomous ship landing maneuvers similar to Figure 1 based on traces with mission durations between 115 and 127 seconds and containing between 22,046 and 25,313 entries. The tool[2] was able to uncover a heading issue for one of the maneuvers that would have been difficult to detect through visual inspection alone.

B. Trace Repair

We define the trace repair problem [8] as follows: given a violating trace and a specification, compute a trace that satisfies the specification while minimally modifying the original trace. The repaired trace can be used as a modified plan that corrects the output of an untrusted planner based on a correctness specification. Furthermore, it provides valuable information about what went wrong in the violating trace. Repairing traces is motivated by the fact that sometimes understanding why a segmentation is violating is hard and an explanation is needed. The repaired trace provides such an explanation.

As an example, consider Figure 2 which shows a plot of a landing maneuver when executing a landing described by the TBT in Figure 1. While the UAV seems to reach the dotted line representing the correct diagonal position, it is actually slightly off and therefore violates the STL specification. Finding such minor deviations is challenging but correcting them is even more difficult with system dynamics involved. Here, the minimal trace repair highlights this by adjusting the UAV's position closer to the dotted line while keeping the rest of the trace unchanged.

Repairing traces in an optimal manner, such that they satisfy a TBT specification, can be encoded as a mixed-integer linear program (MILP) [8]. Given a linear system model $\mathcal{M}$, a TBT specification $\mathcal{T}$, and a trajectory τ with $\tau \not\models \mathcal{T}$, we construct a

[2]https://github.com/DLR-FT/TBT-Segmentation

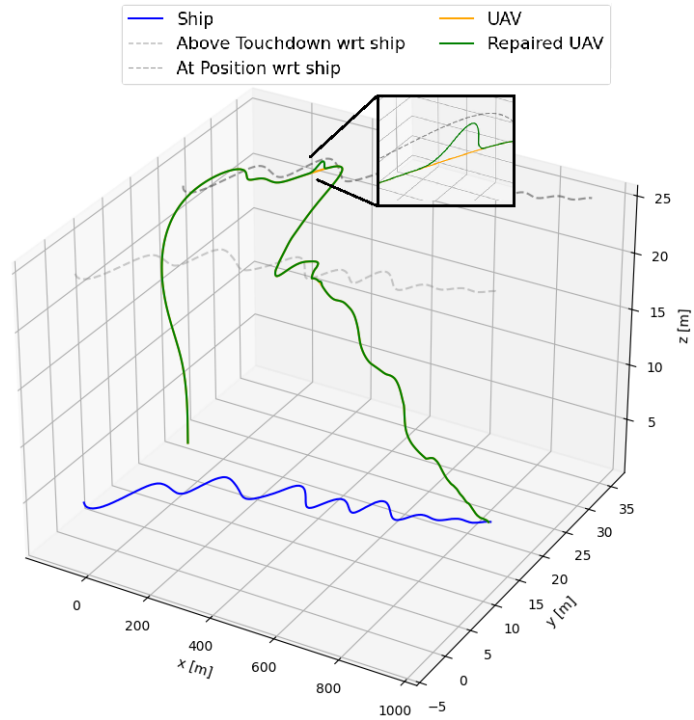

Fig. 2: Repair of a UAV ship landing maneuver against a TBT specification [8].

repaired trajectory τ_R that satisfies $\mathcal{T}$ with $|\tau_R| = |\tau|$, denoted by $\mathit{repair}_{\tau_R}(\mathcal{T}, \mathcal{M}, \tau)$, as follows:

$$\begin{array}{rll} \arg\min_{\tau_R} & J(\tau, \tau_R) & \leftarrow \text{Cost function} \\ s.t. & \tau_R \models \mathit{encode}_{\tau_R}(\mathcal{M}) & \leftarrow \text{Trace follows model} \\ & \tau_R \models \mathit{encode}_{\tau_R}(\mathcal{T}) & \leftarrow \text{Trace satisfies TBT} \end{array}$$

The cost function ensures that the repaired trace remains as close as possible to the original trace.

However, computing optimal repairs is often infeasible in practice, as the underlying MILP formulation is NP-hard. This problem was addressed with two complementary repair strategies. The first strategy, *incremental repair*, uses the results of a trace segmentation to avoid encoding the entire trace. Instead, it locally repairs segments, starting with leaf nodes and incrementally moves up the TBT only if necessary. In experiments [8], for a landing depicted in Figure 2 with a trace length of 1014, this approach successfully repairs within 30 seconds, whereas the straightforward MILP encoding takes over 2000 seconds. The second strategy, *landmark-based repair*, is an iterative approach that identifies candidate landmarks within the trace for the repair. If a repaired trace is found that satisfies the landmark, then its satisfaction guarantees the satisfaction of the TBT specification. Given the abundance of candidates, we use the robust semantics of TBTs as a heuristic to find good candidates. This approach entirely avoids the integer and binary variables introduced by the TBT encoding, enabling the problem to be solved as a linear program instead. As a result, landmark-based repair successfully finds a solution for

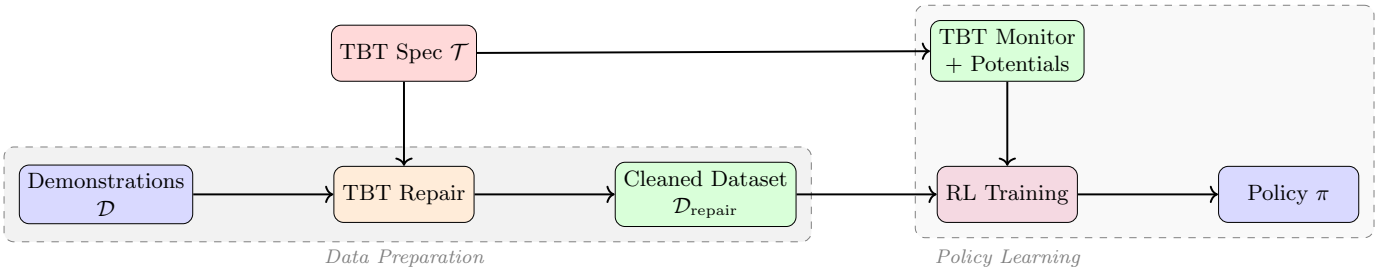

Fig. 3: Overview of the proposed TBT-guided reinforcement learning framework [10]. The TBT specification $\mathcal{T}$ governs both the repair of suboptimal demonstrations and the construction of the TBT monitor and reward potentials used during RL training.

a trace with over $25,000$ entries in under ten minutes, where approaches that fully encode the TBT run out of memory.

C. TBT-Guided Reinforcement Learning

With a formal repair mechanism in place, we apply TBT-based trace repair as a preprocessing step for robot policy learning [10]. Given a dataset of imperfect demonstrations that may violate the TBT specification, the repair procedure corrects offending trace segments to produce a cleaned dataset that is guaranteed to satisfy the specification. From these repaired traces, two complementary potential functions are extracted: a proximity potential that rewards the agent for staying spatially close to repaired poses, and a safety potential that penalizes large deviations from them. Together with the TBT specification acting as an online monitor [9], these are combined into a shaped reward signal that guides a reinforcement learning agent toward specification-satisfying behavior, without requiring any manual reward engineering or knowledge of the agent's kinematic model. Figure 3 shows how the TBT specification plays a pivotal role in learning-based control synthesis.

The framework is validated across grid-world navigation and continuous reach-avoid tasks with single and multiple agents. In all settings, the approach matches or outperforms expert-designed reward baselines, and in the multi-agent case achieves a 47% reduction in obstacle cost rate — despite the potential functions being inferred from a small, configuration-agnostic set of repaired demonstrations. Crucially, because the potential functions operate purely on 2D pose sequences, they transfer across kinematically distinct robot platforms without modification, demonstrating that trajectory repair can serve as a practical and model-agnostic bridge between specification-based formal methods and data-driven robot learning.

IV. Related Work

The key benefit of TBTs is that they preserve the structure of BTs while adding monitoring capabilities. Yet, they are by no means the first formal specification language used to analyze or synthesize robotic behaviors. Trace segmentation is closely related to timed pattern matching [11], but extends it by not only identifying matching segments, but also computing an optimal assignment of trace segments to the specification, which may include segments that *violate* it. Falsification approaches [12] aim to find traces that violate a given specification by minimizing its robustness using stochastic optimization, while trace synthesis methods [13] generate traces that satisfy the specification. Trace repair does not use stochastic optimization but a MILP encoding instead. Signal Temporal Logic (STL) is widely used as a specification language for reasoning about and controlling system behaviors. For instance, Raman et al. [14] encode STL specifications as mixed-integer linear programs (MILPs) for model predictive control. So far, no direct TBT synthesis approach exists.

Several works use STL semantics to shape rewards for RL and imitation learning [15]–[17]. Formal methods also support learning from demonstrations, including logic-augmented automata for RL initialization [18] and safety-aware apprenticeship learning [19]. Other approaches integrate specifications via loss functions [20] or model predictive control [21]. Closely related works [22], [23] use STL semantics to evaluate suboptimal demonstrations and infer rewards for RL. We show that TBTs can be used for learning by exploiting existing TBT analysis techniques.

V. Conclusion

Temporal Behavior Trees (TBTs) provide a unifying specification framework that combines the modular structure of Behavior Trees with the expressive power of temporal logic. In this paper, we surveyed recent advances in TBTs and highlighted their versatility across multiple tasks in robotics, including trace segmentation, trace repair, and learning from demonstrations. These applications demonstrate that TBTs go beyond traditional monitoring by enabling detailed analysis, interpretable debugging, and integration with learning-based methods. For practitioners using Behavior Trees to execute behaviors, the key advantage of TBTs is that monitoring properties can be retrofitted without altering the underlying tree structure. For those not using BTs, TBTs still offer desirable properties such as modularity, readability, and scalability, inherited from the BT formalism. Moreover, TBTs support the use of different temporal specification languages within their leaf nodes, provided these languages support both Boolean and robust semantics. Finally, by modifying the TBT structure itself, one can flexibly control segmentation and trace repair, enabling deeper analysis and improved understanding of the system under scrutiny.

REFERENCES

[1] K. Y. W. Scheper, S. Tijmons, C. C. de Visser, and G. C. H. E. de Croon, "Behavior Trees for Evolutionary Robotics†," *Artificial Life*, vol. 22, no. 1, pp. 23–48, 02 2016.

[2] R. Ghzouli, T. Berger, E. B. Johnsen, S. Dragule, and A. Wkasowski, "Behavior trees in action: A study of robotics applications," in *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering*, ser. SLE 2020. ACM, 2020, p. 196–209.

[3] Z. He, X. Zhang, S. Jones, S. Hauert, D. Zhang, and N. F. Lepora, "Tacmms: Tactile mobile manipulators for warehouse automation," *IEEE Robotics and Automation Letters*, vol. 8, no. 8, pp. 4729–4736, 2023.

[4] A. Sidorenko, J. Hermann, and M. Ruskowski, "Using behavior trees for coordination of skills in modular reconfigurable cppms," in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2022, pp. 1–8.

[5] H. Hu, X. Jia, K. Liu, and B. Sun, "Self-adaptive traffic control model with behavior trees and reinforcement learning for agv in industry 4.0," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 12, pp. 7968–7979, 2021.

[6] M. Iovino, E. Scukins, J. Styrud, P. Ögren, and C. Smith, "A survey of behavior trees in robotics and ai," *Robotics and Autonomous Systems*, vol. 154, p. 104096, 2022.

[7] S. Schirmer, J. Singh, E. Jensen, J. Dauer, B. Finkbeiner, and S. Sankaranarayanan, "Temporal behavior trees: Robustness and segmentation," in *Proceedings of the 27th ACM International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '24. ACM, 2024.

[8] S. Schirmer, P. Schitz, J. C. Dauer, B. Finkbeiner, and S. Sankaranarayanan, "Trace repair for temporal behavior trees," in *Tools and Algorithms for the Construction and Analysis of Systems*. Cham: Springer Nature Switzerland, 2026, pp. 660–679.

[9] R. Matheu, A. G. Puranic, J. S. Baras, and C. Belta, "Omtbt: Online monitoring of temporal behavior trees with applications to closed-loop learning," in *2025 European Control Conference (ECC)*, 2025, pp. 2129–2135.

[10] A. G. Puranic, S. Schirmer, J. S. Baras, and C. Belta, "Learning from imperfect demonstrations via temporal behavior tree-guided trajectory repair," 2026.

[11] D. Ulus, T. Ferrère, E. Asarin, and O. Maler, "Timed pattern matching," in *Formal Modeling and Analysis of Timed Systems*. Cham: Springer International Publishing, 2014, pp. 222–236.

[12] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, "S-taliro: A tool for temporal logic falsification for hybrid systems," in *Tools and Algorithms for the Construction and Analysis of Systems TACAS*, ser. Lecture Notes in Computer Science, vol. 6605. Springer, 2011, pp. 254–257.

[13] S. Sato, J. An, Z. Zhang, and I. Hasuo, "Optimization-based model checking and trace synthesis for complex STL specifications," in *Computer Aided Verification CAV*, ser. Lecture Notes in Computer Science, vol. 14683. Springer, 2024, pp. 282–306.

[14] V. Raman, M. Maasoumy, and A. Donzé, "Model predictive control from signal temporal logic specifications: A case study," in *Proceedings of the 4th ACM SIGBED International Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems*, 2014, pp. 52–55.

[15] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta, "Q-learning for robust satisfaction of signal temporal logic specifications," in *55th IEEE Conference on Decision and Control, CDC*. IEEE, 2016, pp. 6565–6570.

[16] X. Li, C. I. Vasile, and C. Belta, "Reinforcement learning with temporal logic rewards," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS*. IEEE, 2017, pp. 3834–3839.

[17] Y. Jiang, S. Bharadwaj, B. Wu, R. Shah, U. Topcu, and P. Stone, "Temporal-logic-based reward shaping for continuing reinforcement learning tasks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 9, pp. 7995–8003, May 2021.

[18] X. Li, Y. Ma, and C. Belta, "Automata guided reinforcement learning with demonstrations," *CoRR*, vol. abs/1809.06305, 2018.

[19] W. Zhou and W. Li, "Safety-aware apprenticeship learning," in *Computer Aided Verification - 30th International Conference, CAV*, ser. Lecture Notes in Computer Science, vol. 10981, 2018, pp. 662–680.

[20] C. Innes and S. Ramamoorthy, "Elaborating on learned demonstrations with temporal logic specifications," in *Robotics: Science and Systems XVI*, 2020.

[21] K. Cho and S. Oh, "Learning-based model predictive control under signal temporal logic specifications," in *2018 IEEE International Conference on Robotics and Automation, ICRA*. IEEE, 2018, pp. 7322–7329.

[22] A. G. Puranic, J. V. Deshmukh, and S. Nikolaidis, "Signal temporal logic-guided apprenticeship learning," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024, pp. 11 147–11 154.

[23] ——, "Learning from demonstrations using signal temporal logic in stochastic and continuous domains," *IEEE Robotics and Automation Letters (RA-L). Presented at IROS*, vol. 6, no. 4, pp. 6250–6257, 2021.