

Adaptive Manipulation using Behavior Trees

Jacques Cloete¹, Wolfgang Merkt¹, and Ioannis Havoutis¹

Abstract—Many manipulation tasks pose a challenge since they depend on non-visual environmental information that can only be determined after sustained physical interaction has already begun. This is particularly relevant for effort-sensitive, dynamics-dependent tasks such as tightening a valve. To perform these tasks safely and reliably, robots must be able to quickly adapt in response to unexpected changes during task execution, and should also learn from past experience to better inform future decisions. Humans can intuitively respond and adapt their manipulation strategy to suit such problems, but representing and implementing such behaviors for robots remains a challenge. In this work we show how this can be achieved within the framework of behavior trees. We present the adaptive behavior tree, a scalable and generalizable behavior tree design that enables a robot to quickly adapt to and learn from both visual and non-visual observations during task execution, preempting task failure or switching to a different manipulation strategy. The adaptive behavior tree selects the manipulation strategy that is predicted to optimize task performance, and learns from past experience to improve these predictions for future attempts. We test our approach on a variety of tasks commonly found in industry; the adaptive behavior tree demonstrates safety, robustness (100% success rate) and efficiency in task completion (up to 36% task speedup from the baseline).

I. INTRODUCTION

Service robots can revolutionize how entire service-oriented domains work, with potential applications including autonomous maintenance in – often remote – industrial facilities [1], [2], such as oil rigs [3], or care robots in domestic environments [4], [5]. To complete tasks effectively, robots must use observations from the environment to construct and execute a suitable plan; a number of frameworks have been developed for task and motion planning (TAMP) for manipulation [6]–[8].

This work is motivated by effort-sensitive, dynamics-dependent manipulation tasks for articulated devices commonly found in industrial settings, for example twisting a valve until tightened to a specified torque, pushing an e-stop button until pressed, or firmly picking and placing a fragile object. Robots generally require highly controlled environments with known task parameters for reliable planning and execution of manipulation tasks; however, the tasks we are interested in pose a significant challenge since they depend on information that can only be determined after sustained physical interaction has already begun. For example, a robot cannot determine the initial tightness of a valve or stiffness of a button from visual data alone, and can only gather this information by beginning to interact with it [9]. Even then, in the absence of an accurate model for the dynamic profile for the device, the robot cannot know for how long it must twist the valve, or how hard it must push the button, to

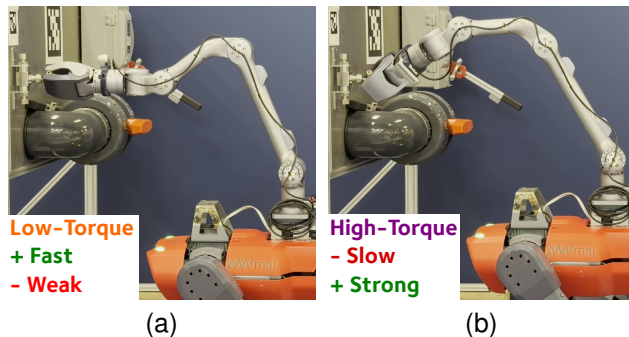


Fig. 1. Different strategies for tightening a needle valve. (1a) Quickly grasping head-on means that all applied torque is generated purely from the wrist joint effort, producing a fast but ‘low-torque’ strategy. (1b) Grasping from the side requires more complex motion but enables the robot to use the arm as a wrench, producing a slow but ‘high-torque’ strategy.

complete the task. Even if provided a model, stiction from corrosion or unclear servicing may lead to uncertainty and deviation from standard assumptions. To solve these kinds of tasks, we argue that the robot must be able to instead decide on an initial plan with which to start the task and adapt to observations it makes during task execution. Since the tasks we consider may require the robot to apply a significant amount of effort to manipulate the device, adaptability is not only critical for task completion but also for safe robot operation; failure to respond to a sudden unexpected spike in effort can cause catastrophic failure and even damage to the robot or environment.

Existing TAMP frameworks generate plans based on the current task environment, enabling robots to adapt to uncertain and changing environments; however, this adaptability is restricted to visually observable changes, such as for real-time collision avoidance. We require a solution that can handle unexpected deviations that are not visually observable.

We also consider the utility of a robot having access to different manipulation strategies for the same task. A *strategy* is the specific approach that a robot may take to carry out a task. For manipulation tasks, different strategies may involve grasping the object differently, or even using a tool to assist manipulation. Different strategies can have different trade-offs, for example speed, complexity, effort and power expenditure; Fig. 1 presents examples for tightening a valve. We seek a system that can smartly select and switch between known strategies, ideally optimizing for performance by learning from past manipulation attempts.

We tackle both problems at the task planning level within the framework of behavior trees, exploiting their reactivity and modularity. The key contribution of our work is the adaptive behavior tree, which enables a robot to handle uncertainty in the environment and quickly adapt to unexpected

¹The authors are with the Dynamic Robot Systems (DRS) group, Oxford Robotics Institute, University of Oxford, Oxford, UK {jacques, wolfgang, ioannis}@robots.ox.ac.uk

safety-critical changes during task execution, preempting task failure or switching to a more optimal strategy based on data gathered over previous attempts. It comprises two stages; a selector that uses data recorded over previous attempts to predict the optimal strategy to deploy, and a monitor that checks a set of conditions concurrently to strategy execution, halting the current strategy if the task is complete or a safety condition is violated and re-attempting the task if necessary. The design is highly modular and generalizable, allowing for easy scaling to new strategies and tasks. We test the adaptive behavior tree on a diverse range of case studies representing complex, effort-intensive manipulation tasks from industry.

II. RELATED WORK

A. Adaptive Control for Robotics

It may be argued that our problem of manipulating articulated devices with unknown and changing dynamics intrinsics can be solved with an appropriate adaptive controller for each task. Adaptive control has existed in the literature for many years, and generally focuses on online parameter estimation and learning for uncertain dynamic systems. There exist many good surveys on the literature for adaptive control [10], [11]. The approach has been extended to robotic manipulation tasks, for example estimating model parameters for an uncertain end-effector payload [12]–[14].

However, we argue that adaptive behavior by parameter tuning is inherently limited in the diversity that it provides for adaptability, and does not readily allow for adaptability to the extent of reactively switching to a completely different manipulation strategy, such as grasping the device in a different way or using a tool, which may be necessary for task completion. Thus, the adaptive behavior we seek must go beyond the control level to the planning level; we opt to instead tackle the problem as a task planning problem. Doing so also allows us to define a more diverse range of conditions for task success and when to halt a strategy, and how to handle when these conditions are met. We will see that our approach can enable a robot to achieve impressive task performance despite only using out-the-box controllers.

B. Behavior Trees for Robotics

There is much work on task planning and autonomous decision-making for robotics, with the state of the art typically adopting a Finite State Machine (FSM), Behavior Tree (BT) or Partially Observable Markov Decision Process (POMDP) [15], [16]. The appropriate choice depends on the application and intended robot behavior.

A BT is a hierarchical directed tree that controls the order of execution of all the tasks that the robot may perform, each represented as a leaf node. Subtrees representing simpler behaviors can be used to build more complex ones, giving BTs a very composable and modular structure. During normal operation a ‘tick’ signal is propagated between the root of the tree and its leaves, with the active leaf node either returning *Success*, *Failure* or *Running*. This return signal determines which node is executed on the subsequent tick. It can be demonstrated that BTs offer far more modularity than FSMs [16]–[18], and BTs also offer superior reactivity by allowing for the concurrent execution of different tree

nodes that can interrupt each other if needed. BTs will be our planning framework of choice since the reactivity that they offer is critical for the manipulation tasks we consider, and our approach will also exploit their modularity.

There are many examples demonstrating how BTs can be used to coordinate manipulation tasks [19]–[21] and robot control [22], [23]; *Iovino et al.* provide a comprehensive survey of BTs for robotics [15]. BTs for robotics can be hand-coded [24]–[26], planned analytically [27]–[31], learned from demonstration [32]–[34] or even generated using Large Language Models (LLMs) [35], [36].

Reactive BTs are particularly relevant to our problem; they use a planning algorithm to automatically create and update a BT on the fly based on observations during task execution, achieving robot behavior that is adaptive to uncertain and changing environments [27], [37]. While this is very promising work, due to the safety-critical nature of the manipulation tasks we aim to tackle, we will opt for an approach that provides a more predictable method of handling failure preemption and recovery.

On the other hand, *Wu et al.* [21] propose an automated BT error recovery framework that uses a Bayesian Network (BN) as a decision model for error recovery upon task failure, alongside a human-made Failure Mode and Effects Analysis (FMEA) chart used to inform the network of possible sources of error as well as the corresponding solutions. In contrast, we desire a robot behavior that can preempt failure entirely by stopping and re-attempting the task once a safety condition is violated, potentially using a different strategy.

III. BEHAVIOR TREES – AN OVERVIEW

Without loss of generality, we follow the BT notation used by BT.CPP [38]; all BT nodes used in this work fall under one of a set of categories represented in Fig. 2.

A **ControlNode** controls the sequence of execution of its children; those used in this work are of type *Sequence*, *ReactiveSequence*, *Fallback* or *ReactiveFallback*.

A **Sequence** propagates the tick to its children from left to right, returning *Success* if and only if all their children return *Success*. When it receives *Running*, it returns *Running* and ticks the same child on the next tick, and when it receives *Failure*, it returns *Failure* and resets the sequence progress. A *ReactiveSequence* is similar, but when it receives *Running*, it returns *Running* and also ticks all children up to the current child on the next tick; thus all prior children are still checked for *Failure*, enabling reactive behavior.

A **Fallback** (known as a Selector in other works) propagates the tick to its children from left to right until one of them returns *Success*, at which point the Fallback stops and returns *Success*. When it receives *Running*, it returns *Running* and ticks the same child on the next tick. It returns *Failure* only if all children return *Failure*. A *ReactiveFallback* is similar, but when it receives *Running*, it returns *Running* and also ticks all children up to the current child on the next tick; thus all prior children are still checked for *Success*, enabling reactive behavior (much like the *ReactiveSequence*).

An **ExecutionNode** carries out a command, forming the leaves of the BT, and can be of type *ActionNode* or *ConditionNode*. An *ActionNode* performs a command, returning *Success* if the command is successfully completed, *Failure*

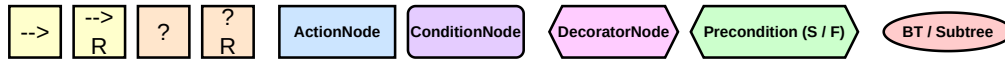


Fig. 2. Categories of BT tree nodes used in this work. From left to right; ControlNode (Sequence, ReactiveSequence, Fallback, ReactiveFallback), ExecutionNode (ActionNode, ConditionNode), DecoratorNode, Precondition, BT.

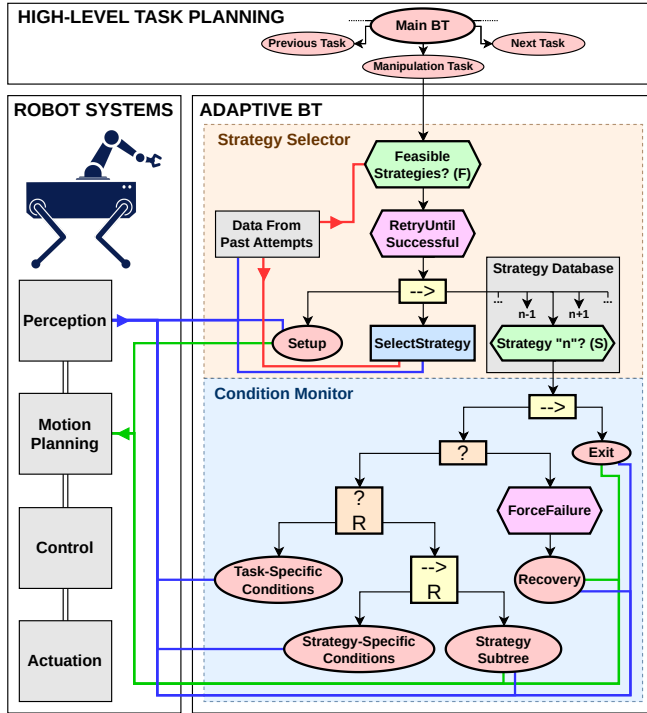


Fig. 3. The adaptive BT, integrated within a wider robot system representing the test platform in this work. The colored lines represent communication lines for perception (blue), motion planning (green) and strategy selection (red).

if failed, or *Running* if ongoing. A *ConditionNode* checks a condition, returning *Success* if true or *Failure* if false.

A *DecoratorNode* modifies the return status of a single child. A *Precondition* is a special type of *DecoratorNode* that ticks the child only if a precondition is met, or else returns either *Success* or *Failure* (marked by ‘S’ or ‘F’ respectively).

IV. ADAPTIVE BT

Fig. 3 presents the adaptive BT, as well as a demonstration of how it can be integrated within a wider robot system. The adaptive BT is composed of two core stages; the strategy selector first decides which strategy to use for the next attempt, based on current perception data as well as data recorded over past attempts (such as joint efforts or time taken). The condition monitor then wraps around the subtree for the selected strategy, checking a set of task- and strategy-specific conditions concurrently to strategy execution. The adaptive BT also admits subtrees to coordinate robot setup prior to strategy selection, and for safely exiting or recovering from the current attempt in preparation for the next.

The database of the subtrees for all $N \in \mathbb{N}$ known strategies are inserted as modules within a *Sequence*, but all are preconditioned to only be executed if that specific strategy has been selected. Thus, if strategy n is selected, only that strategy’s subtree will be ticked, and all the other

subtrees trivially return *Success*. If no strategies are considered feasible for the current task instance, *SelectStrategy* returns *Failure* and the adaptive BT exits as failed.

The condition checking is such that if the subtree of task-specific conditions returns *Success*, the task is considered to have been completed successfully, and if it returns *Failure*, the task is still ongoing. This follows the BT design principle of backward chaining [16], commanding the robot to keep executing the strategy until task completion conditions are met. Meanwhile, if the subtree of strategy-specific conditions returns *Failure*, a constraint has been violated during execution of the current strategy and the robot must halt the current attempt and try again, potentially selecting a different strategy. If it returns *Success*, no violations have occurred. This logic makes the condition monitor reactive and interpretable. If there are multiple task- or strategy-specific conditions to check, they should be placed in a *ReactiveSequence* to ensure proper operation of the adaptive BT. Each condition is ticked for every tick of the strategy subtree, allowing the robot to rapidly respond to new observations; thus the inherent reactivity of BTs is exploited. If a condition is not relevant to the current stage of the strategy subtree, it can be skipped by wrapping with an appropriate *Precondition*.

The subtrees for exit and recovery allow the robot to safely exit the current attempt, due to task success or preempting failure respectively. A *ForceFailure* is wrapped around the recovery subtree to signal that the task is not yet complete and a re-attempt is required.

Very little has been assumed concerning the algorithm behind how the next strategy is selected, or the implementation of any of the subtrees within the strategy selector or condition monitor. The adaptive BT is thus highly modular, and generalizes to a wide range of different problems and can satisfy all manner of tasks, specifications and constraints; all that is required are the subtrees for the task and strategy.

The adaptive BT can easily be added as a subtree within a greater BT structure, and is thus readily applicable to existing BT-based robot task planning systems without major changes. The only required knowledge for adding the proposed design into a larger BT is that the adaptive BT must be repeatedly ticked while *Running*, and will return *Success* if the task was completed successfully and *Failure* otherwise.

The adaptive BT is designed to coordinate efficient selection, monitoring and switching of strategies, not the synthesis of the strategies themselves. The adaptive BT would synergize well with methods for the automatic synthesis of BTs [27]–[37] to generate new strategies for it to use, but exploring this is beyond the scope of this work.

V. ADAPTIVE BT FOR MANIPULATION

We now apply the adaptive BT to manipulation tasks; Fig. 4 presents the general subtrees for all strategies used in this work. We stress that these designs are just examples used

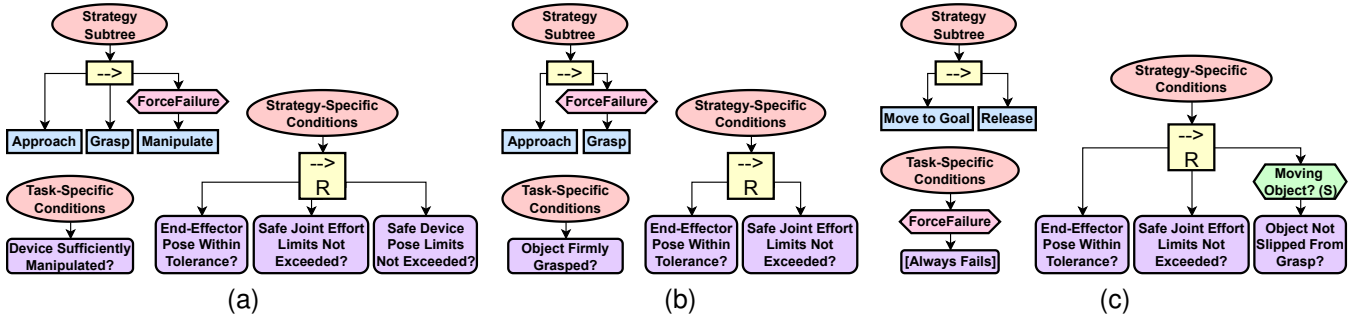


Fig. 4. Subtrees for strategies used in this work. (4a) Manipulating an articulated device. (4b) Grasping an object. (4c) Moving an object to a goal.

to test the adaptive BT, with an emphasis on simplicity; the adaptive BT readily accepts new strategies for new tasks.

Manipulating Articulated Devices. Fig. 4a presents general subtree structures for strategies that manipulate an articulated device, where task success occurs when the device is considered sufficiently manipulated (e.g. valve is tight). The strategies consist of approaching, grasping and manipulating the device (e.g. twist by one-half rotation), while checking safety conditions. Note that these same subtree structures are used for both the low-torque and high-torque strategies depicted in Fig. 1, and even the strategies for manipulating the different devices (e.g. tightening a valve vs pressing a button), though the action and condition nodes have different parameters (or even implementations) for each strategy.

Object Grasping. Fig. 4b presents general subtree structures for strategies that grasp an object, where task success occurs when the object is considered firmly grasped (e.g. by checking gripper effort). The strategies consist of approaching and grasping, while checking safety conditions.

Object Placing. Fig. 4c presents general subtree structures for strategies that move an (already grasped) object to a goal pose, where task success occurs if the strategy completes without preemption (thus no task-specific conditions). The strategies consist of moving and then releasing the object, while checking safety conditions.

Setup/Recovery/Exit. In this work, these are simply action nodes that position the robot to view the scene for strategy selection (for setup) or reset the robot pose (otherwise).

Strategy Selection. The SelectStrategy node estimates the optimal manipulation strategy to use; for this work, the optimal strategy a^* for task state s minimizes task duration t_{est} while not violating strategy-specific constraints (often corresponding to the safety conditions, e.g. maximum safe joint efforts). This is equivalent to the following problem:

$$a^*(s) = \min_{a \in \mathcal{A}_{\text{task}}} t_{\text{est}}(a, s) \quad \text{subject to} \quad a \in \mathcal{A}_{\text{safe}}(s),$$

where strategies a are selected from the set of available task-relevant strategies $\mathcal{A}_{\text{task}}$, and $\mathcal{A}_{\text{safe}} \subseteq \mathcal{A}_{\text{task}}$ is the set of feasible strategies given the constraints. Both t_{est} and $\mathcal{A}_{\text{safe}}$ are estimated from data recorded from previous attempts; if no data exists at s , we let $\mathcal{A}_{\text{safe}} = \mathcal{A}_{\text{task}}$, and calculate t_{est} from nominal durations of the strategy’s motion trajectories.

VI. IMPLEMENTATION

Test Platform. We used an ANYbotics ANYmal quadruped with mounted Unitree Z1 robotic arm, equipped

with a wrist-mounted Intel Realsense Depth Camera D345 for perception. ROS, the Robot Operating System, was used for the implementation of the test setup as well as communication with hardware.

Behavior Tree. BT.CPP 4.0 [38] was used to construct the adaptive BT. The ActionNode types specific to this work take the form of ROS action and service clients, implemented as derived classes of the *StatefulActionNode* from the BT.CPP core library; this node type supports asynchronous actions and is recommended for this use case.

Motion Planning. This work uses a whole-body motion planner made using EXOTica, the EXtensible Optimisation Toolset [39]. The planner solves an EXOTica *DynamicTimeIndexedShootingProblem*, tracking pre-defined waypointed end-effector pose trajectories for the approach, grasp, manipulate and retract stages of the manipulation task for each strategy. The pose trajectories are defined relative to the pose of the object or device handle; in this way, the planner generalizes to arbitrary poses and handle angles (so long as robot joint limits permit the motion). In the case of planning failure, the planner action nodes return *Failure*, prompting the adaptive BT to retry the task.

Control. This work uses the standard controllers provided with the ANYmal and Unitree Z1. The combined controller is by no means perfect, since the ANYmal controller has no awareness of the arm and cannot compensate for the induced wrench on the quadruped when the arm is outstretched. In practice, there was often non-negligible deviation between the true end-effector pose and that in the motion plan, especially when the arm was outstretched; to ensure safe operation, this deviation is checked by the condition monitor during manipulation. Using a bespoke whole-body controller that compensates for both the motion of the quadruped and arm would improve performance, but this is beyond the scope of this work; instead, we demonstrate how our adaptive BT can be used to overcome such problems at the task planning level by autonomously preempting failure and retrying.

Perception. Apriltags [40] were used to identify the pose of objects as well as device handles in an ‘unrotated’ state, and invariant template matching [41] was used to identify handle angles and the pose of the rotated handles.

VII. CASE STUDIES

All device manipulation tasks use the general subtree structures depicted by Fig. 4a, albeit with task- and strategy-specific implementations of the action and condition nodes. Appendix I describes estimation of s and $\mathcal{A}_{\text{safe}}$ for all tasks.

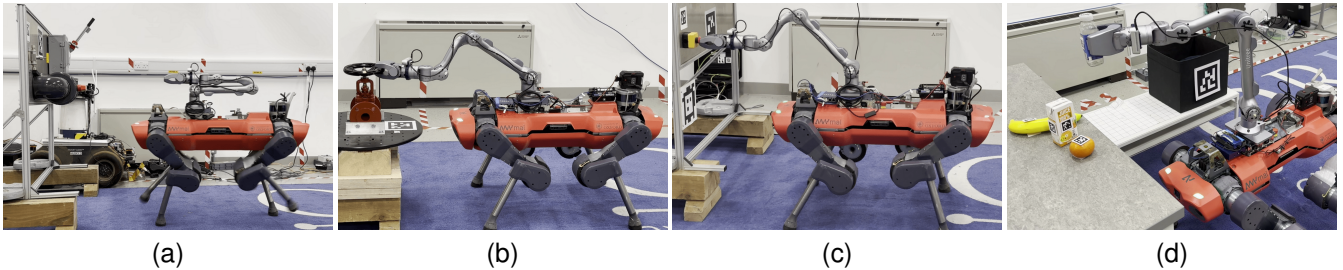


Fig. 5. Snapshots from experiments. (5a) Walking to the needle valve (as part of the walk-and-manipulate task). (5b) Tightening the globe valve. (5c) Pushing the e-stop button until fully pressed. (5d) Groceries packing. Accompanying video with experiments: <https://youtu.be/im70CkEiQsU>.

Needle Valve. The robot twists a Tameson NLS-012 needle valve from unknown starting tightness until tight, and loosens it to a user-specified angle, with access to the two strategies depicted by Fig. 1. To demonstrate how the adaptive BT can be readily placed within larger BT structures, we also consider a walk-and-manipulate task where the robot navigates to, tightens, and navigates away from the needle valve (with access to both manipulation strategies, but the required applied torque for tightness reduced so as to be achievable by the ‘low-torque’ strategy). The adaptive BT is inserted as a subtree within a larger BT that also commands navigation to and from the device (using the standard ANYmal navigation and locomotion software).

Globe Valve. The robot twists a Dikkan DN40 PN16 globe valve from unknown starting tightness until tight, with access to a single strategy depicted by Fig. 5b.

E-Stop Button. The robot manipulates an IDEC HW1X-BV411-R emergency stop; it pushes the button until fully pressed, with access to a single strategy depicted by Fig. 5c, and twists it until released, with access to a single strategy very similar to that depicted by Fig. 1a.

Groceries Packing. The robot packs a plastic bottle, (full) juice box, (real) orange and (plastic) banana into a basket, in a user-specified order, as depicted by Fig. 5d. The grasping stage uses a single strategy, with subtree structures depicted by Fig. 4b, determining firm grasping by checking gripper joint effort. The packing stage has access to a continuous set of strategies; all strategies have the same subtree structures depicted by Fig. 4c, but the speed of the motion trajectories is parameterized by a speed factor (i.e. a strategy) $a \in \mathcal{A}_{\text{safe}}$, where $\mathcal{A}_{\text{safe}}$ is determined by the weight of the object (lower maximum safe speeds for heavier objects, which may slip out of grasp when the motions are too fast). The adaptive BTs for the two stages are placed in sequence such that the robot returns to the first stage if the second stage is preempted.

Baseline. For all tasks, we use backward chaining with stacked Fallbacks as the baseline BT structure [16]. This is the standard BT design used when the robot must choose between different actions with robustness to failure. For fair comparison to the adaptive BT, we manually arranged the (nominal) ‘faster’ strategies before ‘slower’ ones in the baseline’s Fallbacks such that the robot tries them first, with a reminder that the adaptive BT automates (and optimizes) the strategy selection process. Since the baseline cannot select from a continuous set of strategies, it can only (safely) use the lowest speed factor when grocery packing.

VIII. RESULTS AND DISCUSSION

To evaluate the adaptive BT, we performed a series of ablations to assess how each component impacts performance, with comparison to the baseline where relevant. For our discussion, we refer to the results in Table I, which records, for 10 trials of each experiment, the cumulative success rate after each attempt (ignoring re-attempts triggered to range strategy and re-grasping when device handle pose limits were exceeded) and time taken for task completion. A maximum of 5 attempts was permitted for each trial, task success was verified by checking the scene after the robot had exited the task, and the initial device states were reset to be the same for each trial. Strategy effort limits were set to the Z1’s maximum safe operating efforts scaled down by a safety factor, except for the needle valve low-torque strategy, which used a larger safety factor to avoid the quadruped becoming unstable from induced wrench during heavy twisting.

Reactive Failure Preemption (RFP). For Ablation A, the robot only has access to one strategy (preventing strategy adaptation) and manipulation data is reset between trials (preventing learning from past experience). We see a 100% success rate within the permitted number of attempts for all but one experiment when using the adaptive BT, despite having no knowledge of the valves’ initial tightness, or how much they needed to be twisted to be tightened. The robot never suffered catastrophic failure since the condition monitor preempted failure before it occurred. When using the low-torque strategy, the robot aborted the needle valve tasks after exceeding the safe effort limit for the arm’s wrist joint. This demonstrates the safety enabled by the adaptive BT, since the robot was able to react and safely exit the task before overexerting itself. The robot was also able to safely twist the valves for an arbitrary number of rotations without the danger of overextending itself, by periodically re-grasping the valves using the condition of exceeding safe device pose limits; once triggered, the robot re-attempted the task and re-grasped the device in a more favorable joint configuration for further twisting. The robot was also able to firmly grasp the different-shaped fragile objects without damaging them by reacting to spikes in gripper effort.

Under this ablation, the adaptive BT produces equivalent behaviors to the baseline. However, the adaptive BT enables easier scalability of task composition over the baseline, since the adaptive BT provides an abstracted, modular interface for appending and removing subtrees for strategies and conditions to the strategy database (see Fig. 3), while editing the baseline backward chained BT structure can become

TABLE I
EXPERIMENT RESULTS FOR ALL TASKS

Task	Experiment	Cumulative Success Rate (%) After Attempt Number					Time Taken (s)	
		1	2	3	4	5	Minimum	Mean
<i>Ablation A: RFP Enabled (SAT + LPE Disabled, i.e. Equivalent to Baseline)</i>								
Needle Valve (Tighten)	Low-Torque Strategy Only	0	Aborted	Aborted	Aborted	Aborted	135 (Fail)	136 (Fail)
	High-Torque Strategy Only	50	70	90	100	Done	279	336
	Walk-and-Manipulate	60	80	90	100	Done	212	230
Needle Valve (Loosen)	Low-Torque Strategy Only	0	Aborted	Aborted	Aborted	Aborted	58 (Fail)	58 (Fail)
	High-Torque Strategy Only	70	90	100	Done	Done	228	247
Globe Valve	Tighten	70	100	Done	Done	Done	95	107
E-Stop Button	Press	80	100	Done	Done	Done	28	32
	Release	70	80	100	Done	Done	45	63
Groceries Packing	Lowest Speed Only	60	70	90	100	Done	261	269
<i>Ablation B: RFP + SAT Enabled (LPE Disabled)</i>								
Needle Valve (Tighten)	Both Strategies (Baseline)	70	100	Done	Done	Done	175	186
	Both Strategies (Adaptive BT)	80	100	Done	Done	Done	175	182
Needle Valve (Loosen)	Both Strategies (Baseline)	60	90	100	Done	Done	259	283
	Both Strategies (Adaptive BT)	80	100	Done	Done	Done	204	212
Groceries Packing	Lowest Speed Only (Baseline)	60	70	90	100	Done	261	269
	Weight-Based Speed (Adaptive BT)	50	70	80	100	Done	196	206
<i>Ablation C: RFP + SAT + LPE Enabled (Complete Adaptive BT)</i>								
Needle Valve (Tighten)	Both Strategies (Baseline)	80	100	Done	Done	Done	94	126
	Both Strategies (Adaptive BT)	90	90	100	Done	Done	58	118
Needle Valve (Loosen)	Both Strategies (Baseline)	60	90	100	Done	Done	259	283
	Both Strategies (Adaptive BT)	80	100	Done	Done	Done	172	182

unwieldy as the number of strategies and conditions grows.

Strategy Adaptation (SAT). For Ablation B, the robot has access to all strategies but manipulation data is still reset between trials (preventing learning from past experience). We see substantial average task speedups for needle valve tightening (46%) and loosening (14%) as well as groceries packing (23%) when using the adaptive BT compared to Ablation A; the cause for this speedup is that the adaptive BT selects faster strategies when feasible and switches to slower, ‘safer’ strategies when necessary.

For needle valve tightening, since the low-torque strategy is approximately 5 times faster while twisting, it is optimal for the robot to use it for most of the manipulation sequence, but when the robot experiences a sudden effort spike as the valve begins to tighten, it quickly adapts and uses the high-torque strategy to fully tighten the valve; Fig. 6 provides an annotated timeline. We note that this is similar to behavior displayed by humans, twisting quickly when the valve is loose but changing grip near the end to be able to apply more wrench without hurting oneself. For this particular task the baseline produces the same behavior, leading to roughly the same average time taken; this is because the optimal behavior for this task (start low-torque, switch to high-torque at the end) can also be achieved using a simple Fallback. However in general this is not the case, as demonstrated by the needle valve loosening task, wherein the adaptive BT achieves a 25% average speedup over the baseline. For this task, both the adaptive BT and baseline first (unsuccessfully) try the low-torque strategy (which cannot safely apply enough torque to loosen the valve) and then switch to the high-torque strategy for the first successful twist, after which

the valve is loose enough to be loosened further using the low-torque strategy. While the logic of the Fallback means that the baseline will keep using the high-torque strategy for the remainder of the task, the adaptive BT optimizes over feasible strategies at each re-attempt, and thus chooses the (faster) low-torque strategy for the remainder of the task.

Learning from Past Experience (LPE). For Ablation C, the robot has access to all strategies as well as the manipulation data recorded from Ablations A and B; note that the baseline has no mechanism for learning from past manipulation data. For needle valve loosening, the adaptive BT achieves an even greater average speedup (36%) over the baseline; this is because past manipulation data suggests that the valve will initially be too tight to be loosened with the low-torque strategy, so the robot has learned to start with the high-torque strategy (while in Ablation B, due to lack of knowledge of the valve’s initial tightness, it would initially try low-torque then immediately switch to high-torque).

For needle valve tightening, we see no change from Ablation B if the robot has no initial estimate of the required rotation until tight, i.e. task state s . Note that as we vary initial s , in some cases it is faster to start with the high-torque strategy, since it avoids the overhead of switching from low-torque to high-torque to finish tightening. If we now provide the adaptive BT with an accurate initial estimate for s , it always starts with the strategy that minimizes time taken to tighten the valve according to past experience (unlike Ablation B, which always starts with low-torque). This is illustrated by Fig. 7, which plots minimum time taken to tighten the needle valve for 5 different known required rotations, with the adaptive BT achieving up to 34% speedup

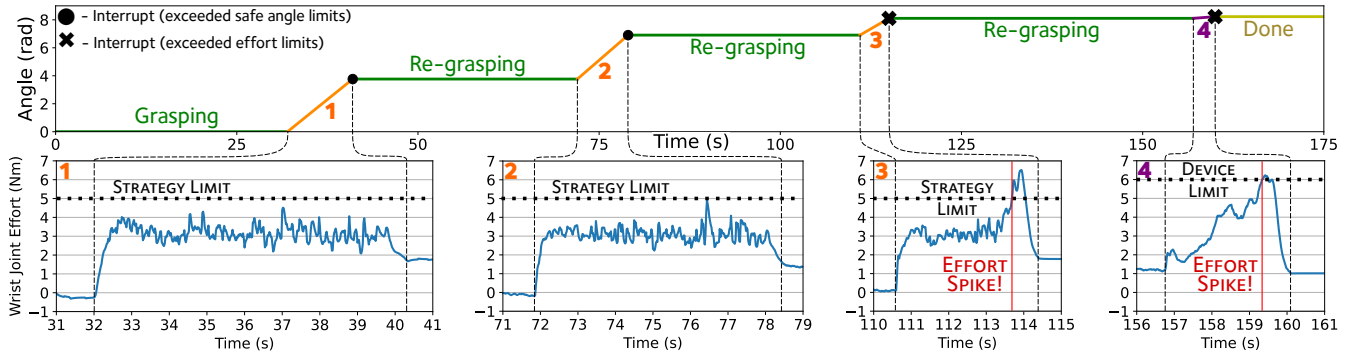


Fig. 6. An annotated timeline of tightening the needle valve using the adaptive BT with both available strategies, with plots of the effort in the wrist joint during each manipulation attempt A, B, C (which all use the ‘low-torque’ strategy) and D (which uses the ‘high-torque’ strategy). The adaptive BT initially selects the low-torque strategy to twist quickly while the valve is loose but switches to the high-torque strategy near the end to apply more wrench and finish tightening. Note the effort spikes during attempts C and D; the first indicates that the low-torque strategy is no longer feasible, while the second indicates that the device is sufficiently tightened. For clarity, only the lowest active effort limit is shown in each plot.

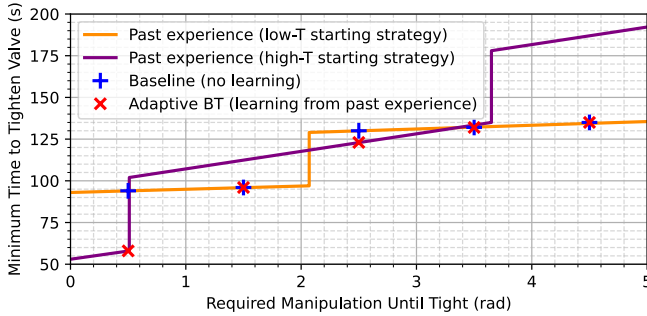


Fig. 7. Minimum time to tighten the needle valve for different (known) required rotations until tight. The adaptive BT starts with the strategy that minimizes time taken to tighten the valve according to past experience.

over the baseline (at 0.5 radians). For Ablation C’s results for this task in Table I, the 10 trials are instead 2 trials each over these 5 different known required rotations; here the adaptive BT achieves 6.3% average speedup over the baseline.

In practice, initial s could be a random variable with a known distribution; for this case, the mean time taken over this distribution would be the appropriate t_{est} to use for strategy selection. If the distribution is unknown, it could be estimated empirically over many manipulations.

A limitation of the test setup is that the effort conditions for task success were found empirically from preliminary testing; however, we stress that this is unrelated to the adaptive BT itself, which assumes very little about each condition and would readily accept more sophisticated conditions based on task-space wrench estimation.

IX. CONCLUSION AND FUTURE WORK

In this work we proposed the adaptive BT, a flexible and highly modular BT design that enables a robot to adapt to environment uncertainties by pairing reactive condition monitoring with adaptive strategy selection, and learning from past experience to predict the optimal strategy given the task state. Our test robot used the adaptive BT to safely and robustly complete a diverse range of complex effort-intensive manipulation tasks commonly found in industrial and domestic settings, despite using out-the-box controllers, and with reduced time taken over the baseline BT design for all experiments where multiple strategies were available.

The adaptive BT can be readily added to existing BT implementations without major changes, enhancing performance while remaining interpretable and without significantly changing the system architecture. Future work will investigate algorithmic and learning-based synthesis of new strategies and conditions for the adaptive BT to enable a robot to generalize to new tasks, including synthesis from more semantic, conceptual descriptions of each strategy, or autonomous discovery of new strategies.

ACKNOWLEDGMENTS

This work was supported by the EPSRC CDT in Autonomous Intelligent Machines and Systems [EP/S024050/1] and the Mobile Robotic Inspector project [EP/Z531212/1].

APPENDIX I

COMPUTING THE FEASIBLE SET OF STRATEGIES

When manipulating articulated devices, feasible strategies at task state s satisfy joint effort limits across a state window $\{s' : s_L \leq s' - s \leq s_U\}$ around s . Thus, $\mathcal{A}_{\text{safe}}$ is given by

$$\mathcal{A}_{\text{safe}}(s) = \{a \in \mathcal{A}_{\text{task}} : |T_{\text{est},i}(a, s')| \leq |T_{\text{lim},i}(a)| \forall i \in [1..N_{\text{DOF}}], s_L \leq s' - s \leq s_U\},$$

where $T_{\text{est},i}(a, s)$ are the estimated efforts for each arm joint i at s using strategy a , recorded from previous attempts, and $T_{\text{lim},i}(a)$ are the maximum safe efforts under strategy a . For valve tasks, s is the total rotation angle θ away from being tight, with initial estimate 0 radians when loosening. When tightening, initial θ is predicted from a distribution if available or else estimated to be a large default value, and once the valve is tight, all θ values in the new batch of manipulation data are shifted such that the final entry is 0 radians. For button tasks, s is a (known) binary on/off state.

When grocery packing, a is a real-valued speed factor limited by the efforts induced in the arm joints by the object:

$$\mathcal{A}_{\text{safe}}(s) = \{a \in \mathbb{R}^+ : a_L \leq a \leq a_H(s)\}$$

$$\text{where } a_H(s) = \min\{a_L, 1 - \sum_i \alpha_i |T_{\text{obj},i}(s) - T_{\text{pre},i}|\},$$

where $T_{\text{pre},i}$ and $T_{\text{obj},i}(s)$ are the efforts recorded immediately before and after lifting the object respectively, a_L is a minimum speed factor and $\alpha_i \geq 0$ are pre-tuned parameters. State s is the object weight, but is not directly measured.

REFERENCES

- [1] W. Yuan and W. Lu, "Research on the impact of industrial robot application on the status of countries in manufacturing global value chains," *PLOS ONE*, vol. 18, no. 6, p. e0286842, Jun. 2023.
- [2] K. T. Ly, M. Munks, W. Merkt, and I. Havoutis, "Asymptotically optimized multi-surface coverage path planning for loco-manipulation in inspection and monitoring," in *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*, 2023, pp. 1–7.
- [3] C. Gehring, P. Fankhauser, L. Isler, R. Diethelm, S. Bachmann, M. Potz, L. Gerstenberg, and M. Hutter, "Anymal in the field: Solving industrial inspection of an offshore hvdc platform with a quadrupedal robot," in *Field and Service Robotics*, ser. Springer Proceedings in Advanced Robotics, G. Ishigami and K. Yoshida, Eds. (Springer, Singapore), 2021, vol. 16.
- [4] C. Kenny and R. Wilson, *Robotics in Social Care*. UK Parliamentary Office of Science and Technology, Dec. 2018.
- [5] T. Yamamoto, K. Terada, A. Ochiai, F. Saito, Y. Asahara, and K. Murase, "Development of Human Support Robot as the research platform of a domestic mobile manipulator," *Robomech J*, vol. 6, 2019.
- [6] H. Guo, F. Wu, Y. Qin, R. Li, K. Li, and K. Li, "Recent trends in task and motion planning for robotics: A survey," *ACM Computing Surveys*, vol. 55, no. 13s, p. 1–36, Dec. 2023.
- [7] S. Saoji and J. Rosell, "Flexibly configuring task and motion planning problems for mobile manipulators," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2020, pp. 1285–1288.
- [8] K. T. Ly, V. Semenov, M. Risiglione, W. Merkt, and I. Havoutis, "Rlqp: A reachability-guided logic-geometric programming framework for optimal task and motion planning on mobile manipulators," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 14917–14923.
- [9] T. Corbères, C. Mastalli, W. Merkt, J. Shim, I. Havoutis, M. Fallon, N. Mansard, T. Flayols, S. Vijayakumar, and S. Tonneau, "Perceptive locomotion through whole-body mpc and optimal region selection," *IEEE Access*, vol. 13, pp. 69 062–69 080, 2025.
- [10] A. M. Annaswamy and A. L. Fradkov, "A historical perspective of adaptive control and learning," *Annual Reviews in Control*, vol. 52, p. 18–41, 2021.
- [11] G. Tao, "Multivariable adaptive control: A survey," *Automatica*, vol. 50, no. 11, p. 2737–2764, Nov. 2014.
- [12] J.-J. E. Slotine and W. Li, "On the adaptive control of robot manipulators," *The International Journal of Robotics Research*, vol. 6, no. 3, pp. 49–59, 1987.
- [13] P. M. Moubarak and P. Ben-Tzvi, "Adaptive manipulation of a hybrid mechanism mobile robot," in *2011 IEEE International Symposium on Robot and Sensors Environments (ROSE)*, 2011, pp. 113–118.
- [14] K. Lu, K. T. Ly, W. Heberd, K. Zhou, I. Havoutis, and A. Markham, "Learning generalizable manipulation policy with adapter-based parameter fine-tuning," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024, pp. 13 510–13 517.
- [15] M. Iovino, E. Scutkins, J. Styrud, P. Ögren, and C. Smith, "A survey of behavior trees in robotics and ai," *Robotics and Autonomous Systems*, vol. 154, p. 104096, Aug. 2022.
- [16] M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and AI*, 0th ed. CRC Press, Jul. 2018.
- [17] R. Ghzouli, T. Berger, E. B. Johnsen, A. Wasowski, and S. Dragule, "Behavior trees and state machines in robotics applications," *IEEE Transactions on Software Engineering*, vol. 49, no. 9, p. 4243–4267, Sep. 2023.
- [18] M. Iovino, J. Förster, P. Falco, J. J. Chung, R. Siegart, and C. Smith, "On the programming effort required to generate behavior trees and finite state machines for robotic applications," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. London, United Kingdom: IEEE, May 2023, p. 5807–5813.
- [19] A. Csiszar, M. Hoppe, S. A. Khader, and A. Verl, *Behavior Trees for Task-Level Programming of Industrial Robots*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2017, p. 175–186.
- [20] M. Iovino, F. I. Dogan, I. Leite, and C. Smith, "Interactive disambiguation for behavior tree execution," in *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*. Ginowan, Japan: IEEE, Nov. 2022, p. 82–89.
- [21] R. Wu, S. Kortik, and C. Santos, "Automated behavior tree error recovery framework for robotic systems," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 6898–6904.
- [22] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ögren, "Towards a unified behavior trees framework for robot control," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. Hong Kong, China: IEEE, May 2014, p. 5420–5427.
- [23] D. Caceres Dominguez, M. Iannotta, J. A. Stork, E. Schaffernicht, and T. Stoyanov, "A stack-of-tasks approach combined with behavior trees: A new framework for robot control," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, p. 12110–12117, Oct. 2022.
- [24] F. Rovida, D. Wuthier, B. Grossmann, M. Fumagalli, and V. Kruger, "Motion generators combined with behavior trees: A novel approach to skill modelling," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Madrid: IEEE, Oct. 2018, p. 5964–5971.
- [25] K. R. Guerin, C. Lea, C. Paxton, and G. D. Hager, "A framework for end-user instruction of a robot assistant for manufacturing," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. Seattle, WA, USA: IEEE, May 2015, p. 6167–6174.
- [26] C. Paxton, A. Hundt, F. Jonathan, K. Guerin, and G. D. Hager, "Costar: Instructing collaborative robots with behavior trees and vision," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. Singapore, Singapore: IEEE, May 2017, p. 564–571.
- [27] M. Colledanchise, D. Almeida, and P. Ögren, "Towards blended reactive planning and acting using behavior trees," in *2019 International Conference on Robotics and Automation (ICRA)*. Montreal, QC, Canada: IEEE, May 2019, p. 8839–8845.
- [28] J. Tumova, A. Marzinotto, D. V. Dimarogonas, and D. Kragic, "Maximally satisfying ltl action planning," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Chicago, IL, USA: IEEE, Sep. 2014, p. 1503–1510.
- [29] F. Rovida, B. Grossmann, and V. Kruger, "Extended behavior trees for quick definition of flexible robotic tasks," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vancouver, BC: IEEE, Sep. 2017, p. 6793–6800.
- [30] C. Paxton, N. Ratliff, C. Eppner, and D. Fox, "Representing robot task plans as robust logical-dynamical systems," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Macau, China: IEEE, Nov. 2019, p. 5588–5595.
- [31] P. Verma, M. Diab, and J. Rosell, "Automatic generation of behavior trees for the execution of robotic manipulation tasks," in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2021, pp. 1–4.
- [32] K. French, S. Wu, T. Pan, Z. Zhou, and O. C. Jenkins, "Learning behavior trees from demonstration," in *2019 International Conference on Robotics and Automation (ICRA)*. Montreal, QC, Canada: IEEE, May 2019, p. 7791–7797.
- [33] X. Cheng, A. Kumar, and D. Pathak, "Legs as manipulator: Pushing quadrupedal agility beyond locomotion," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. London, United Kingdom: IEEE, May 2023, p. 5106–5112.
- [34] A. Jain, P. Long, V. Villani, J. D. Kelleher, and M. Chiara Leva, "Cobt: Collaborative programming of behaviour trees from one demonstration for robot manipulation," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. Yokohama, Japan: IEEE, May 2024, p. 12993–12999.
- [35] H. Zhou, Y. Lin, L. Yan, J. Zhu, and H. Min, "Llm-bt: Performing robotic adaptive tasks based on large language models and behavior trees," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. Yokohama, Japan: IEEE, May 2024, p. 16655–16661.
- [36] J. Ao, F. Wu, Y. Wu, A. Swikir, and S. Haddadin, "Llm as bt-planner: Leveraging llms for behavior tree generation in robot task planning," *arXiv preprint arXiv:2409.10444*, Sep. 2024.
- [37] N. Li, H. Jiang, C. Li, and Z. Wang, "Towards adaptive behavior trees for robot task planning," in *2022 China Automation Congress (CAC)*. Xiamen, China: IEEE, Nov. 2022, p. 6720–6725.
- [38] D. Faconti, "Behaviortree.cpp," <https://www.behaviortree.dev/>, Last accessed on 9/8/2024.
- [39] V. Ivan, Y. Yang, W. Merkt, M. Camilleri, and S. Vijayakumar, "Exotica: An extensible optimization toolset for prototyping and benchmarking motion planning and control," in *Robot Operating System (ROS)*, ser. Studies in Computational Intelligence, A. Koubaa, Ed. (Springer, Cham), 2018, vol. 778.
- [40] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 3400–3407.
- [41] Z. Zhang and H. Shang, "Low-cost solution for vision-based robotic grasping," in *2021 International Conference on Networking Systems of AI (INSAI)*, 2021, pp. 54–61.