Reaction Templates: A Formal Approach to Realize Reactivity in Task and Motion Planning-Based Action Execution

Anne Köpken[1], Adrian S. Bauer[1], Nesrine Batti[1], Daniel Leidner[1]

***Abstract*— Recent advancements in artificial intelligence have broadened the spectrum of tasks that robots can effectively tackle. However, the seamless execution of prolonged action sequences continues to pose a considerable challenge, attributed to limitations in the abilities of today's planning-based robots to react to unforeseen situations or failures. In response, we introduce Reaction Templates (RTs), a formal approach for integrating reactivity into task and motion planning. Operating concurrently with the primary execution logic, RTs enable a clear differentiation between planned actions and the necessary recovery strategies for handling unexpected events. This design promotes scalability by establishing reusable building blocks and customizable parameters, thereby enhancing flexibility in application. We provide a thorough introduction to the RT concept, elucidating its principles, mechanisms, and the rationale behind its design decisions. The resulting benefits of the approach are demonstrated through experimental validation with the humanoid robot Rollin' Justin. A video highlighting the core concepts of the paper and depicting the experimental validation can be viewed at https://www.youtube.com/watch?v=aiLKXQCcMy4..**

***Index Terms*— Failure Detection and Recovery, Service Robotics, Reactive and Sensor-Based Planning**

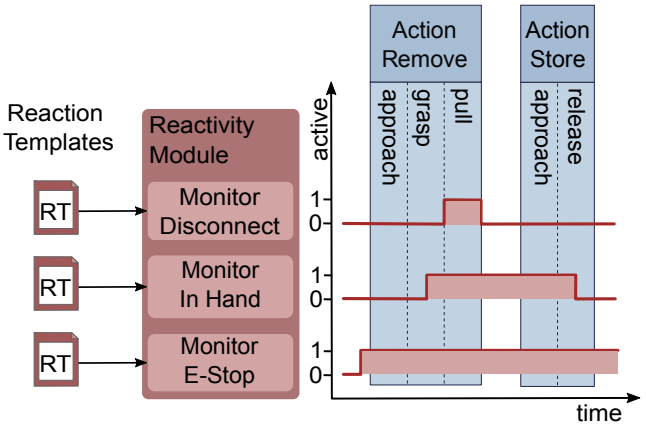


Fig. 1: We propose to decouple action definition (blue) from reactive behavior (red). Different Monitors are automatically activated dependent on their definition inside the corresponding Reaction Template (RT) and trigger reactive behaviors if an anomaly is being detected.

I. Introduction

While recent videos show impressive feature demos that are working in a controlled environment such as folding laundry[1], serving tea[2], or unpacking groceries[3], a standing challenge lies in chaining manipulation actions into longer, cohesive sequences that are executed autonomously in a robust fashion. This is partially due to limitations in error management, which significantly inhibits the ability to navigate unforeseen situations, thereby impeding the agent's overall adaptability and performance in diverse environments.

Depending on how robots are programmed – typically relying on methods such as *state machines*, *behavior trees*, or integrated *Task and Motion Planning (TAMP)* – there are different approaches to manage errors or achieve reactivity in general. Typical approaches for state machines often suffer from code repetition across multiple actions and deep intertwining of error recovery and execution code. This results in a complicated structure where error handling becomes an inseparable part of the overall action definition.

Behavior trees organize actions hierarchically using nodes to represent specific behaviors [1]. Built-in error-handling mechanisms empower robots to adapt and continue tasks. However, reactivity becomes part of the behavior tree, thus, increasing complexity and hindering scaling to complex scenarios.

Integrated TAMP is another approach used to achieve long-horizon autonomy in robotics [2]. It involves pre-planning action sequences and using backtracking to find a suitable plan towards the desired goal state. This method relies on assumptions of a static and deterministic world, which are frequently challenged in non-fenced environments. There is little work on reactivity in integrated TAMP, often times relying on probabilistic planning which leads to exploding computational costs and requires an adequate model of the probabilistic behavior of the environment [2].

In contrast to the aforementioned approaches, we propose a method that introduces reactivity via templatized reactions. This is a formalization of the approach demonstrated in a space-to-ground telerobotics experiment in [3]. Operating concurrently to the forward path, *Reaction Templates (RT)* separate the strategy for achieving the desired outcome from the recovery strategy. Fig. 1 demonstrates this separation as the reactivity module is running in parallel to the action execution. The advantage of our approach lies

The research reported in this paper has been (partially) supported by the German Research Foundation DFG, as part of Collaborative Research Center (Sonderforschungsbereich) 1320 Project-ID 329551904 "EASE - Everyday Activity Science and Engineering", University of Bremen (http://www.ease-crc.org/). The research was conducted in subproject R6. *(Corresponding author: Anne Köpken)*

[1]German Aerospace Center (DLR), Robotics and Mechatronics Center (RMC), Münchner Str. 20, 82234 Weßling, Germany (e-mail: {anne.koepken, adrian.bauer, nesrine.batti, daniel.leidner}@dlr.de)

[1]https://www.youtube.com/watch?v=HOoRnv3lA0k

[2]https://www.youtube.com/watch?v=uVcBa6NXAbk

[3]https://www.youtube.com/watch?v=Z3yQHYNXPws

©2026 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works. DOI: 10.1109/ICRA57385.2026.11697161

in its compatibility with classical programming methods like integrated TAMP, enabling effective responsiveness to unforeseen events and facilitating code reuse across tasks.

The contributions of this paper include: (i) a novel concept for reactivity as separate structure, establishing a paradigm to separate strategies for action and reaction, (ii) the introduction of RTs as representation of this formulation, and (iii) a proof of concept implementation of RTs and the evaluation of these concepts conducted through real world experiments on the humanoid robot Rollin' Justin.

II. Related Work

Varying reactive behaviors of autonomous agents can be achieved on multiple levels by different approaches. In the following we will introduce approaches to reactivity that are related to ours.

A. Reactivity in Planning-Based Action Execution

Russell et al. [4, pp. 383-392] discuss a trade-off between online planning and contingency planning in uncertain or partially observable environments. Contingency planning branches out for multiple possible outcomes, which becomes impractical with increasing complexity. Alternatively, planning online is adaptable to unknown or dynamic environments, but it assumes easy recovery from actions, which may not always hold true. In the following we give a short overview of approaches that employ either contingency planning, online planning, or a combination of both. Our approach involves pre-planning the most probable action sequence and resorting to online replanning only when required.

TAMP dissects the problem of planning to a given goal state into planning in a symbolic and a geometric space. This allows the planner to search in the less dimensional and discrete space of symbolic planning to find a plan that leads to the goal state and then refine it in the continuous geometric domain, employing backracking to iterate between these stages. Many different approaches implement integrated TAMP [5], [6], [7], [8], [9], with one approach being the usage of *Action Templates (ATs)* [10]. ATs are object-centric action representations consisting of a symbolic description of the action alongside a robot-agnostic grounding to geometric operations. The listed approaches assume a static, deterministic world, risking failure if the environment changes between planning and execution or during execution. Our work addresses this by introducing reactivity.

There are different approaches for employing TAMP in partially observable and non-deterministic environments. For an overview of recent trends in TAMP please refer to [11]. Some approaches add checks on the symbolic predicates and change plans at run-time to achieve reactivity [12], [2], [13]. There are two major differences to our approach. We propose to monitor the assumptions in parallel, such that the robot can react instantaneously and not only at defined points between sub-tasks and our approach also allows to monitor assumptions that are not explicitly defined in the symbolic actions definition.

Shah et al. [12] generate branching plans that handle multiple contingencies based on probabilistic planning with a probabilistic complete algorithm. As waiting for a complete refinement of the policy tree takes long, the approach prioritizes paths that are likely to be encountered and have a low cost of refinement. The approach adapts the task plan between actions but neither incorporates monitoring in parallel to the action execution nor adaptations on the level of motion planning and the method requires an estimate of the probability distributions.

Garret et al. [2] propose online replanning on geometric and symbolic level. They use hybrid belief distributions which are adapted by observations and defer planning of tasks that are likely to succeed and computationally expensive to avoid unnecessary replanning. They replan the task after every action but do not react to uncertainty during a motion execution and also require an estimate of the probability distribution.

Lin et al [13] propose a modular multi-level replanning TAMP framework (MMRF), which allows reactions on three levels: replanning the motion, repeating parts of the symbolic plan or replanning on the symbolic level. MMRF performs feasibility checks after each sub-task on the preconditions of the next task by evaluating the logic state through a predicate parser on the numerical sensor state. The approach checks the conditions between sub-tasks, but does not monitor them in parallel to action execution, such that reactivity is only possible between sub-tasks. Furthermore, the approach checks only symbolic information. Other implicit assumptions in the task definition that are not part of the semantic header will, thus, not be checked by this approach.

Another approach to cope with uncertainties is the utilization of classical planning with *failure handling* when the execution deviates from the expected behavior. This is highly related to reactive TAMP as there is a big overlap in what is considered reactivity or failure handling.

Okada et al. [14] propose to generate state machines with a PDDL planner, by adding an extra *failed* outcome to plan for anomalies on the task level between actions.

In contrast to the aforementioned methods Ueda et al. [15] monitor and react to certain conditions in parallel to the action execution. They introduce *On-Conditions* in the symbolic action description to monitor relevant states via a dedicated module. Errors are categorized into *geometric* and *symbolic* errors and are resolved through path adjustment or symbolic replanning. While related to our work, their approach embeds reactive behavior in the action definitions. In contrast, we advocate for a concept which enables the robot to derive when to activate monitors based on the action context and a set of reusable reaction strategies.

Recent work leverages *Large Language Models (LLMs)* and *Vision-Language Models (VLMs)* for reaction strategies and failure handling in TAMP. For example, Ouyang et al. [16] utilize LLMs for high level reasoning in combination with *Reinforcement Learning (RL)* for generating motions of a quadruped robot. They implement reactivity by querying an LLM with current state information if a sub-task fails.

Duan et al. [17] employ a VLM to assess task success from a language specification and an image, and to provide an explanation upon failure. Both approaches do not implement monitoring in parallel, but check outcomes between subtasks.

B. Reactive Motion Planning

Reactivity on the level of motion planning has been widely studied, see [18] for an overview. For example, Lehner et al. [19] propose an incremental approach to generate motions in initially unknown environments with moving obstacles. In another domain, Bevilacqua et al. [20] propose a robotic walking assistant, that predicts human movements and modifies the robot's path to avoid them. To speed up motion planning, Orton et al. [21] reuse information from previous queries and adapt the initial plan online only if necessary. Similar to contingency planning, they cache alternative solutions to speed up online adaptation. Vasilopoulos et al. [22] combine hybrid planning with reactive motion planning in a 2D Warehouseman's Problem. However, their approach misses reactivity on the task planning level and it is not clear whether it can be transferred to motion planning in higher dimensions.

In contrast to the previous approaches, we aim to introduce reactivity at the symbolic level as well, treating it jointly with reactive motion planning because the two are tightly intertwined.

III. Design Considerations

A robot task execution may fail if assumptions relied upon during planning are violated at run-time. Theses assumptions can either be *explicit* or *implicit*. Explicit assumptions are mentioned in the definition of an action, usually accompanied by how they are handled. Implicit assumptions, on the other hand, are assumed to hold but are not explicitly mentioned anywhere. One possibility to avoid failures is to make every assumption explicit in order to check it at execution time. This is similar to common approaches in state machines [1], where usually every step contains different contingency measures to be taken if certain assumptions are violated. This leads to action definitions that are dominated by failure handling routines which makes the action code harder to understand for task designers. Furthermore, some assumptions are related to general states and not to specific actions. Handling them explicitly leads to code repetition as they appear in many actions. This is why we propose to handle assumptions, that are not specific to an action, implicitly and separate the definition of how to detect and react to a potential violation from the action definitions. The underlying hypothesis stems from the intuition, that the majority of assumptions can be handled implicitly, whereas only a few assumptions require explicit attention inside the action definition.

We define *Actions* as the builing blocks of plans made to reach a goal in a fully observable deterministic world and *Reactions* as the adaptations that are necessary to make the plan work in the real wold if any assumption made at planning time is violated.

We introduce two running examples that will be used throughout the paper to illustrate the approach: Pouring from one container into another, and disconnecting a plug from a socket. Pouring liquid or granulate between containers requires precision to avoid spills, influenced by uncertainties in container poses. If a deviation between the expected positions at planning time and the observed positions at execution time is detected the robot must adjust the pouring trajectory or replan the action to prevent spillage.

The second example examines how a robot disconnects a plug from a socket, facing timing variability. Due to minor variations, the exact moment when the plug disconnects from the socket varies between executions. Without reactivity, the robot either continues the disconnection motion even after successful disconnection of the plug or does not manage to disconnect it before the motion ends.

A. Monitoring of Implicit Assumptions

Our approach uses rules that are separated from the action definition to monitor implicit assumptions at run time. This allows a task designer to define tasks without having to consider recovery strategies needed if the assumptions are violated.

Therefore, *monitors* are activated and deactivated automatically and run in parallel during task execution to check whether the implicit assumptions made during planning are fulfilled. Each monitor monitors one assumption and triggers a reaction if it is violated. We expect this to be beneficial for assumptions that are shared among different tasks, as it allows to reuse their monitors and reactions. Illustrations for implicit assumptions can be derived from both running examples. On grasping an object, it is assumed that the transformation from the object to the robot's end-effector equals the transformation used during planning. Thus, a monitor observes this transformation and triggers a reaction if the assumption is violated. In the following this particular monitor, shared between both examples, is called `In Hand`.

Fig. 1 shows an example of two actions *Remove* and *Store*, which each consist of multiple steps. The *Remove* Action consists of the steps *approach*, *grasp*, and *pull* and the *Store* action consists of the steps *approach* and *release*. The strategy of decoupling action and reaction definitions saves the task designer from having to specify explicitly in every manipulation task that the object's position in the robot's hand must be monitored. Instead, this assumption is automatically monitored based on the action semantics, i.e. between the *grasp* and *release* step. Similarly, the activation of the E-Stop is automatically monitored in every task through a dedicated monitor.

B. Reaction Templates

In order to allow for automatic monitoring of assumptions, a mechanism is required that orchestrates the monitors and triggers reactions based on the monitors outcome. We propose a novel representation called *Reaction Templates*

(RTs) that defines *when* a monitor is active, *which* state it monitors, and *when* it triggers *which* reaction. An RT combines this information in a form that is re-usable across a variety of tasks. Based on the templates, the robot activates and deactivates monitors during execution time and triggers reactions if necessary. RTs are evaluated at execution time based on the semantic state of the robot and its environment, thus, allowing to separate the definition of reactivity from the definition of actions. The anatomy of RTs is described in more detail in Section IV.

C. Parameter Definition

A complete separation of reactions from actions is not feasible, as reactions depend on the specific action being executed, including the objects involved in the action.

In the `In Hand` example, the tolerable difference to the expected position is action-dependent: For a simple pick-and-place task, the position in hand might not matter as much as for pouring from one container into another. But the required accuracy also depends on the containers in use. If the robot is tasked to pour into a huge bowl, the position does not have to be as precise as if it pours into a narrow bottle [23].

In order to foster re-usability of reaction strategies, we supplement RTs with *Parameters*, which allow to tune the reactive behavior to specific actions and their parameterization. While we aim to decouple action and reaction *definitions*, it can be argued that the parameterization serves as the key link between actions and reactions. The parameterization of RTs allows to reuse the same RT in different contexts with custom parameters. Therefore, an RT provides a default parameter set which can be overwritten hierarchically on multiple levels. The first hierarchy level for customizing parameters are the actions (e.g. pouring vs. pick and place). On the next level, RT parameters can be customized to the action parameters (e.g. narrow bottle vs. big bowl). This allows to fine tune an RT not only to a certain type of action, but also, for example, to the objects that are involved in the action. When an RT is activated, the most specialized parameter set that suits the current context is selected, falling back to the default set if no specialized parameter set is available.

D. Monitoring of Explicit Assumptions

While we argue that it is beneficial, to allow that most assumptions remain implicit in the action definition and are monitored automatically, there are use cases where explicit assumptions can be beneficial especially for action specific assumptions. *Explicit assumptions*, in contrast to the aforementioned *implicit assumptions*, allow a task designer to define an action explicitly dependent on feedback. In this case reactivity is used to control the action flow by reacting to expected events. While this is not the focus of our approach, we also support starting and stopping specific monitors explicitly from within an action definition to complement the default rule-based activation.

The plug disconnection example can serve to illustrate the use of explicit assumptions. As the exact time of disconnection is hard to predict, it makes sense to explicitly specify that the robot should pull the plug backwards until its monitor signals that the plug is disconnected successfully and then proceed to the next step. Fig. 1 shows that the specific *Disconnect* monitor is only active during the pull operation.

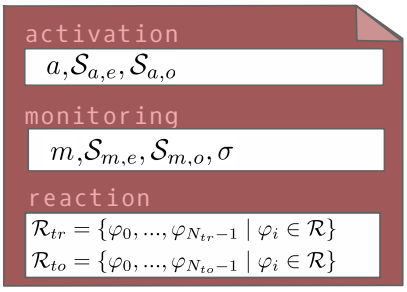


Fig. 2: An RT consists of three sections: `activation`, `monitoring` and `reaction`. The `activation` section defines when the corresponding monitor is active, the `monitoring` section defines when a reaction must be triggered, and the `reaction` section defines how to react.

E. Reaction Selection

If a problem during task execution occurs, it is likely that multiple assumptions are violated concurrently. Thus, multiple monitors trigger a reaction simultaneously, making it necessary to define how these reactions are handled. Our proposed solution deals with multiple concurrent reactions by sorting the reactions in a hierarchical manner: In a first step, only the reactions with the highest severity are considered, making sure to react as conservatively as necessary. For example, if a monitor triggers a geometric replanning while another triggers a symbolic replanning the symbolic replanning response is prioritized. This is due to the fact that the symbolic replanning entails a new symbolic plan which forces a geometric replanning anyways. In another example we assume that the object in the robot's hand (e.g. a bottle) and another object (e.g. a glass to pour into) both deviate from their expected position. If both deviations trigger a geometric replan, the replan only needs to be triggered once.

IV. The Anatomy of Reaction Templates

RTs enable reusable reaction strategies by providing flexible templates that can be customized with parameter sets. An RT consists of three sections: *Activation*, *Monitoring*, and *Reaction* (see Fig. 2) which we describe in the following.

A. Activation

The *Activation* section defines when the monitor corresponding to the RT is active. It, therefore, evaluates whether the expected and observed states $\mathcal{S}_{a,e}$ and $\mathcal{S}_{a,o}$ of the robot and its environment fulfill a certain activation condition a. If $b_a = a(\mathcal{S}_{a,e}, \mathcal{S}_{a,o})$ is $True$, the corresponding monitor is active. The RT of the `In Hand` example activates the corresponding monitor as long as an object is bound to the robot's manipulator. In this example, the state $\mathcal{S}_a$ is the

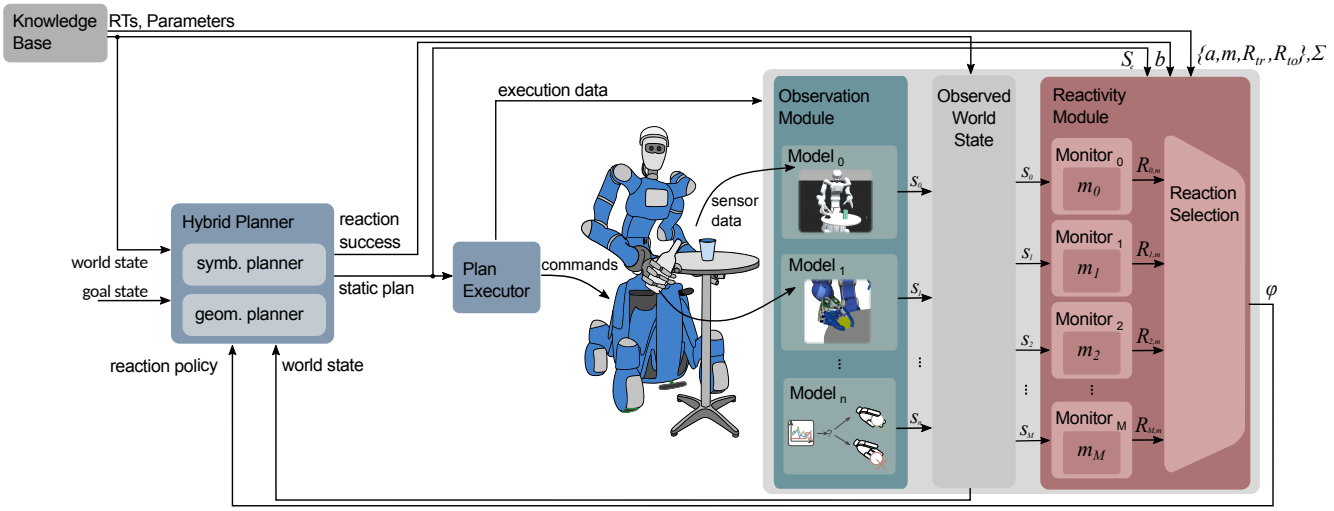


Fig. 3: Proof-of-Concept Implementation: The Hybrid Planner generates a static plan which is executed by the Plan Executor. In parallel, the Observation Module estimates the world state. The Reactivity Module activates monitors dependent on RTs, comparing the observed states $\mathcal{S}_o$ to the expected states $\mathcal{S}_e$. The monitoring function m decides if a reaction is necessary. The Reaction Selection selects one reaction policy from the reactions $\mathcal{R}_{i,m}$.

semantic attribute *bound* of the robot's manipulator and the activation condition a is true when an object is bound.

B. Monitoring

The *Monitoring* function m defines when the corresponding monitor triggers a reaction. It compares the observed state $\mathcal{S}_o$ to the expected state $\mathcal{S}_e$ and takes the parameters σ under consideration to decide whether a reaction is needed. If the result $b_m = m(\mathcal{S}_{m,e}, \mathcal{S}_{m,o}, \sigma)$ is $True$, a reaction is triggered. For the `In Hand` example, the monitored state $\mathcal{S}_m$ is the position of the object in the robot's manipulator. The monitoring function m checks whether the deviation between the expected and observed position exceeds a certain threshold defined in the RT's parameter set σ.

C. Reaction

The *Reaction* specified in an RT can include *trigger reactions* $\mathcal{R}_{tr}$ and *timeout reactions* $\mathcal{R}_{to}$, which serve different purposes and are described in the following.

1) Trigger Reactions: Trigger reactions are reactions that are activated if a monitored assumption is violated. The trigger reactions $\mathcal{R}_{tr} = \{\varphi_0, ..., \varphi_{N_{tr}-1} \mid \varphi_i \in \mathcal{R}\}$ describe a set of reaction policies φ_i that are triggered in escalating order, where $\mathcal{R}$ describes the set of possible reaction policies φ_i. The reaction policies $\mathcal{R}$ are specified in a robot agnostic library and are re-used in different RTs. Trigger reactions are triggered if the respective monitor detects an error, i.e. the classification b_m evaluates to $True$:

$$\mathcal{R}_{m,tr} = s(\mathcal{R}_{tr}, b_m) = \begin{cases} \emptyset & \text{for } b_m = False \\ \mathcal{R}_{tr} & \text{for } b_m = True \end{cases} \quad (1)$$

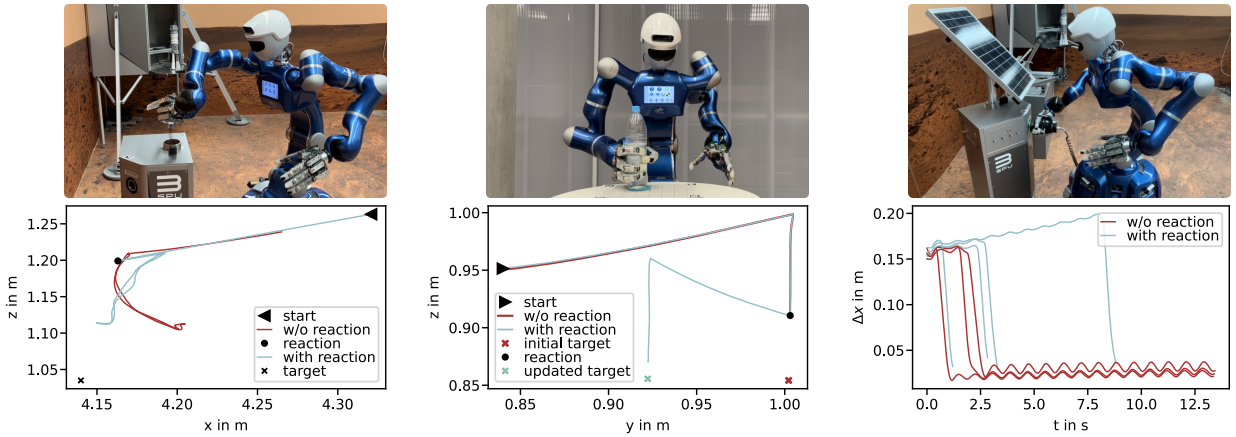
where $\mathcal{R}_{m,tr}$ are the reaction policies that the monitor outputs. Robot-agnostic policies $\varphi_i \in \mathcal{R}_{m,tr}$ are transformed into robot-specific reactions. For example, a robot-agnostic policy might consist of shifting an end-effector trajectory by a fixed offset. The robot's motion planner then generates a joint trajectory that respects the robot's kinematic constraints to realize this shift, producing a robot-specific reaction. If a particular policy φ_i cannot be applied (e.g. the robot-agnostic policy φ_i cannot be transformed into a valid robot-specific reaction) the system selects the next policy φ_{i+1}.

The `In Hand` RT utilizes trigger reactions to trigger a reaction in case of a discrepancy between the observed and expected position of an object in the robot's hand. A typical set of trigger reactions is for example $\mathcal{R}_{tr} = \{\texttt{geom_adapt}, \texttt{geom_replan}, \texttt{symb_replan}, \texttt{teleop_error}\}$. The first reaction checks whether a local geometric adaptation of the planned trajectory is possible. If that fails it tries to plan a new path, followed by a symbolic replan, and finally resorts to asking a teleoperator for help.

2) Timeout Reactions: Unlike trigger reactions, *Timeout Reactions* $\mathcal{R}_{to} = \{\varphi_0, ..., \varphi_{N_{to}-1} \mid \varphi_i \in \mathcal{R}\}$ provide a default response if a monitor deactivates without triggering a reaction. They're often combined with explicit reactions. The `Disconnect` RT in the running example demonstrates one use case. The monitor stops the execution of the pull-motion when the plug is disconnected. If that does not happen in the specified time, a timeout reaction is triggered to handle the failed disconnection.

V. Proof-of-Concept Implementation

This section introduces a proof-of-concept implementation to demonstrate our approach with a so called *Reactivity Module*, an extension for reactivity in planning based action execution frameworks. Fig. 3 shows an overview of the system. An integrated task and motion planner (here *Hybrid Planner* [24], in blue) generates a static plan based on world knowledge, a goal state, and the current world state. This plan leads to the desired goal state in a deterministic world and is open-loop executed by the *Plan Executor*. An



(a) Pouring (b) Stacking (c) Disconnecting a Plug

Fig. 4: We demonstrate our architecture with three different experiments on the humanoid robot Rollin' Justin. The trajectories without reactions are shown in red, the ones with reactions in blue. In the pouring example and the stacking, the robot uses reactivity to adapt its trajectory to successfully perform the task. During plug disconnection, the robot detects the moment of disconnection and stops pulling.

Observation Module (here *Digital Twin Framework* [25], in green) estimates the current geometric and symbolic states $\mathcal{S}_m = s_0, ..., s_n$ based on sensor data and updates the current *Observed World State*.

The *Reactivity Module* (in red) closes the feedback loop on the level of TAMP. It uses measured and estimated states to generate plan adjustments which are relayed back to the *Hybrid Planner*.

The monitors (light red blocks in Fig. 3) compare a particular set of observed states $\mathcal{S}_o$ to the corresponding expected states $\mathcal{S}_e$. These states can either be sub-symbolic, such as sensor readings or the measured position of an object, or symbolic states, such as whether an object is bound to the robot's manipulator or not. The expected states $\mathcal{S}_e$ are generated by the *Hybrid Planner*. The static plan includes trajectories and object positions but also symbolic information, such as the expected effect of an action. If a monitor triggers a reaction, the corresponding reaction policies $\mathcal{R}_m$ are fed to the *Reaction Selection* (light red block in Fig. 3) which selects one reaction policy φ from all reaction policies (see Section III-E) and feeds it back to the *Hybrid Planner*. Depending on the type of reaction policy, the *Plan Executor* either adjusts the current plan or executes an updated plan from the *Hybrid Planner*.

With this proof-of-concept implementation we show that it is possible to decouple the definition of actions and reactions for conditions that can be handled implicitly. Assumptions that require explicit attention are handled inside the action definition. The action sequence actively starts and stops the monitor. Nevertheless, the corresponding monitor runs inside the *Reactivity Module*.

VI. Evaluation with Humanoid Robot

We demonstrate our implementation of the concepts proposed in this paper in three different robot scenarios with manually induced deviations executed on the humanoid robot Rollin' Justin [26]. We use the running examples of pouring and disconnecting a plug, and a task of placing a bottle on a coaster. Fig. 4 shows Justin performing these tasks.

A. Example Reaction Templates

In these scenarios we use four different RTs: The `In Hand`, `Object Localization`, `E-Stop`, and `Disconnect` RT. `In Hand` triggers a recovery if the position of an object in the robot's manipulator deviates from the expected position. `Object Localization` monitors if the position of an object that the robot interacts with (except if it is bound) deviates. `E-Stop` is triggered if the emergency stop of the robot is pressed and interrupts the robot's plan executor. `Disconnect` is a very specific RT used to react to the disconnection of a plug. It allows to preemptively end the pulling motion for disconnection. The `Object Localization` RT uses the same monitoring function and the same reaction policies as `In Hand`. This highlights the code re-usability in RTs.

B. Task 1: Pouring Granulate

In the first task Justin pours granulate from a container into a hole inside a *Smart Payload Unit (SPU)*. A container is standing in front of Justin on an experiment tray and is filled with granulate while the SPU is empty. Justin's goal is to fill the SPU with granulate. It plans to (1) *pick up* the container and (2) *pour* the granulate from the container into the SPU. In order to provoke a deviation between expected and observed state, we purposely add an offset in the expected position of the container in hand. We perform the experiment twice, first without reactivity and then with reactivity.

Fig. 4a shows the trajectories of the tip of the container. The trial without reactivity where Justin spills the granulate is shown in red. The second trial with reactivity is shown

in blue. Here, the *Reactivity Module* is active with the RTs described above. After successfully picking up the container, the `In Hand` RT and `Object Localization` RT each start a corresponding monitor monitoring the position of container and SPU respectively. The `In Hand` Monitor detects the misalignment of the container in the robot's manipulator and triggers a reaction to replan geometrically. Justin replans the trajectory and successfully pours the granulate from the container into the SPU.

C. Task 2: Placing a Bottle on a Coaster

In the second task Justin places a bottle on top of a coaster. We provide the robot with a wrong initial position of the bottle *and* move the coaster to the side while the robot is grasping the bottle. This task uses the same monitors as Task 1, but this time both the `In Hand` and `Object Localization` monitors trigger reactions concurrently, and their reactions are merged into a single response.

Fig. 4b shows the trajectory of the point in the bottom center of the bottle. The unsuccessful trial without reactivity and the initial target are shown in red. The trajectory that includes the recovery strategy and the updated target are shown in blue. Without reactivity the robot would drop the bottle on the table next to the coaster, while it successfully places the bottle on top of the coaster with reactivity. In the final trial, we move the coaster further to the side, such that the robot cannot reach it with its right arm. As the planner fails to find a solution to the geometric replan reaction, the reactivity module triggers a symbolic replanning. As a result of the new plan, the robot hands the bottle over to the left arm before successfully placing it on the updated target position.

D. Task 3: DIP Disconnection

The third task is the running example of disconnecting a plug. This problem stems from the ISS space-to-ground mission Surface Avatar [27], where a plug is disconnected from an SPU. In this task we do not show how the robot is able to recover from an error, but demonstrate the explicit reactions of skipping to the next operation after a successful disconnection. Analogously to Experiment 1 and 2, the robot is, however, also able to use the `Object Localization` and `In Hand` RT in this task to recover from a deviating SPU location or plug location.

Fig. 4c shows an image of the disconnection task and a plot of Δx, the difference between the commanded and executed x-coordinate of the robot's end-effector in world coordinates. The drop in Δx indicates the moment of disconnection. Without reactivity the robot keeps executing the pulling motion while with reactivity the pulling motion stops after the successful disconnection.

VII. Discussion

Our evaluation reveals that our approach enables the reuse of RTs and recovery strategies across different tasks in plan-based action execution. This is achieved by decoupling the definition of actions and reactions and parameterizing the RTs based on specific tasks and objects. Some RTs, like emergency stop status verification, are universally applicable, others, such as confirming object position relative to the end-effector, have broad utility in certain domains like in object manipulation tasks. We separate task specific properties from RTs in terms of parameter sets that allow fine tuning of RTs to specific actions. Thus, only parameters of RTs need to be overwritten if the RT is to be tuned to a certain task.

The separation of monitoring and reaction definition allows to re-use monitoring strategies and escalate recovery strategies utilizing the planning infrastructure. However, it does not support the definition of reactions directly depending on the severity of the error or deviation observed, like directly triggering a symbolic replan if the deviation is above a certain threshold instead of trying a geometric replan. We argue that this does not restrict the applicability of our approach as an escalating strategy of reactions policies is usually preferable.

The three different experiments showcase the flexibility and scalability of our approach as the system is able to leverage existing planning capabilities to generate recovery strategies. The stacking task clearly demonstrates this, when the robot uses a handover to recover from a misplaced coaster. The handover of the bottle is a direct result of leveraging the planning capabilities of the robot without having to define a handover manually as a solution for a reachability problem inside the action definition of the stack action. This reduces effort and code repetition during action implementation.

Our approach is best suited for *mildly probabilistic* environments, i.e. environments in which reactivity is not required constantly. If replanning happens very frequently, many previously made plans are discarded during replanning which can hinder seamless task execution, particularly in scenarios where geometric adaptation is impractical.

We demonstrated our approach in three example use cases with short task sequences. Due to the re-usability of RTs and recovery strategies and the leveraging of existing planning capabilities to generate recovery strategies, we expect our approach to perform well also for long-horizon plans. Future applications with more actions and longer-horizon action sequences will show how well the approach scales.

Our experiments highlight that our approach enhances task execution robustness, even though it does not replace reactive motion planning or visual servoing for highly dynamic environments. Instead, it should be viewed as a supplementary tool that enables the integration of reactivity at both task planning and motion planning levels. While the computational overhead is negligible compared to TAMP planning times in our experiments, its impact must be studied in more complex scenarios.

VIII. Conclusion and Outlook

In this work we present and demonstrate a novel method to incorporate reactivity into plan-based action execution by means of *Reaction Templates (RTs)*. RTs serve as a structured formalism that employ rule-based reactivity patterns, providing a systematic approach to handle unforeseen events during

task execution. This approach notably alleviates the manual effort typically required for designing reactive behavior as observed in conventional methods within the field.

While the ability to parameterize RTs is key to enable reusability and scalability, we identify the search for suitable parameters as one of the main challenges. We intend to explore automating the parameter identification in the future to further decrease manual labor. To achieve this, we will investigate learning-based approaches in physics simulation.

To accelerate task execution and reduce waiting time, we aim to cut down planning duration through two strategies. Firstly, we seek to minimize the need for replanning during task execution by pre-planning the most probable alternatives. Secondly, while currently planning the entire action sequence in detail, we recognize that with longer planning horizons, the certainty of the environment state diminishes. Therefore, we intend to refrain from planning actions too far ahead and instead employ more online replanning. By relaxing the constraints on how predictable the future must be for the robot in order to act successfully, we enable it to execute longer task sequences autonomously.

REFERENCES

[1] R. Ghzouli, T. Berger, E. B. Johnsen, A. Wasowski, and S. Dragule, "Behavior Trees and State Machines in Robotics Applications," pp. 4243–4267, Sep. 2023.

[2] C. R. Garrett, C. Paxton, T. Lozano-Perez, L. P. Kaelbling, and D. Fox, "Online Replanning in Belief Space for Partially Observable Task and Motion Problems," in *Proc. 2020 IEEE Int. Conf. Robot. and Automat.* Paris, France: IEEE, May 2020, pp. 5678–5684. doi: 10.1109/ICRA40945.2020.9196681

[3] A. Kopken *et al.*, "Toward robust task execution through telerobotic failure recovery in space operations," in *2025 IEEE Aerospace Conf.*, 2025, pp. 1–13. doi: 10.1109/AERO63441.2025.11068192

[4] S. J. Russell *et al.*, *Artificial Intelligence: A Modern Approach*, fourth edition, global edition ed., ser. Pearson Series in Artificial Intelligence. Harlow: Pearson, 2022.

[5] F. Gravot, S. Cambon, and R. Alami, "aSyMov: A Planner That Deals with Intricate Symbolic and Geometric Problems," in *Robotics Research. 11th Int. Symp.*, B. Siciliano, O. Khatib, P. Dario, and R. Chatila, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, vol. 15, pp. 100–110.

[6] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *Proc. 2011 IEEE Int. Conf. Robot. and Automat.*, May 2011, pp. 1470–1477. doi: 10.1109/ICRA.2011.5980391

[7] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *Proc. 2014 IEEE Int. Conf. Robot. and Automat.* Hong Kong, China: IEEE, May 2014, pp. 639–646. doi: 10.1109/ICRA.2014.6906922

[8] F. Lagriffoul, D. Dimitrov, J. Bidot, A. Saffiotti, and L. Karlsson, "Efficiently combining task and motion planning using geometric constraints," *Int. J. Robot. Res.*, vol. 33, no. 14, pp. 1726–1747, Dec. 2014. doi: 10.1177/0278364914545811

[9] C. R. Garrett, T. Lozano-Perez, and L. P. Kaelbling, "FFRob: Leveraging symbolic planning for efficient task and motion planning," *Int. J. Robot. Res.*, vol. 37, no. 1, pp. 104–136, Jan. 2018. doi: 10.1177/0278364917739114

[10] D. S. Leidner, *Cognitive Reasoning for Compliant Robot Manipulation*, ser. Springer Tracts in Advanced Robotics. Cham: Springer International Publishing, 2019, vol. 127.

[11] 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 Comput. Surv.*, vol. 55, no. 13s, Jul. 2023. [Online]. Available: https://doi.org/10.1145/3583136. doi: 10.1145/3583136

[12] N. Shah, D. Kala Vasudevan, K. Kumar, P. Kamojjhala, and S. Srivastava, "Anytime Integrated Task and Motion Policies for Stochastic Environments," in *2020 IEEE Int. Conf.Robot. and Automat. (ICRA)*. Paris, France: IEEE, May 2020, pp. 9285–9291. doi: 10.1109/ICRA40945.2020.9197574

[13] T. Lin, C. Yue, Z. Liu, and X. Cao, "Modular Multi-Level Replanning TAMP Framework for Dynamic Environment," *IEEE Robotics and Automation Letters*, vol. 9, no. 5, pp. 4234–4241, May 2024. doi: 10.1109/LRA.2024.3377556

[14] K. Okada, Y. Kakiuchi, H. Azuma, H. Mikita, K. Murase, and M. Inaba, "Task Compiler : Transferring High-level Task Description to Behavior State Machine with Failure Recovery Mechanism," in *ICRA Workshop on Combining Task and Motion Planning*, 2013.

[15] R. Ueda, Y. Kakiuchi, S. Nozawa, K. Okada, and M. Inaba, "Anytime error recovery by integrating local and global feedback with monitoring task states," in *Proc. 2011 15th Int. Conf. Adv. Robot.*, Jun. 2011, pp. 298–303. doi: 10.1109/ICAR.2011.6088616

[16] Y. Ouyang *et al.*, "Long-horizon locomotion and manipulation on a quadrupedal robot with large language models," 2025. [Online]. Available: https://arxiv.org/abs/2404.05291

[17] J. Duan *et al.*, "Aha: A vision-language-model for detecting and reasoning over failures in robotic manipulation," 2024. [Online]. Available: https://arxiv.org/abs/2410.00371

[18] M. G. Mohanan and A. Salgoankar, "A survey of robotic motion planning in dynamic environments," *Robot. and Auton. Syst.*, vol. 100, pp. 171–185, Feb. 2018. doi: 10.1016/j.robot.2017.10.011

[19] P. Lehner, A. Sieverling, and O. Brock, "Incremental, sensor-based motion generation for mobile manipulators in unknown, dynamic environments," in *Proc. 2015 IEEE Int. Conf. Robot. and Automat.*, May 2015, pp. 4761–4767. doi: 10.1109/ICRA.2015.7139861

[20] P. Bevilacqua, M. Frego, D. Fontanelli, and L. Palopoli, "Reactive Planning for Assistive Robots," *IEEE Robot. and Automat. Lett.*, vol. 3, no. 2, pp. 1276–1283, Apr. 2018. doi: 10.1109/LRA.2018.2795642

[21] M. Orton, S. Dai, S. Schaffert, A. Hofmann, and B. Williams, "Improving Incremental Planning Performance through Overlapping Replanning and Execution," in *Proc. 2019 Int. Conf. Robot. and Automat.*, May 2019, pp. 2426–2432. doi: 10.1109/ICRA.2019.8793642

[22] V. Vasilopoulos, W. Vega-Brown, O. Arslan, N. Roy, and D. E. Koditschek, "Sensor-Based Reactive Symbolic Planning in Partially Known Environments," in *Proc. 2018 IEEE Int. Conf. Robot. and Automat.*, May 2018, pp. 5683–5690. doi: 10.1109/ICRA.2018.8460861

[23] A. S. Bauer, P. Schmaus, F. Stulp, and D. Leidner, "Probabilistic Effect Prediction through Semantic Augmentation and Physical Simulation," in *Proc. 2020 IEEE Int. Conf. Robot. and Automat.* Paris, France: IEEE, May 2020, pp. 9278–9284. doi: 10.1109/ICRA40945.2020.9197477

[24] D. Leidner, C. Borst, and G. Hirzinger, "Things are made for what they are: Solving manipulation tasks by using functional object classes," in *Proc. 2012 12th IEEE-RAS Int. Conf. Humanoid Robots*. Osaka, Japan: IEEE, Nov. 2012, pp. 429–435. doi: 10.1109/HUMANOIDS.2012.6651555

[25] A. S. Bauer, A. Köpken, and D. Leidner, "Multi-agent heterogeneous digital twin framework with dynamic responsibility allocation for complex task simulation," in *Proc. 21st Int. Conf. Auton. Agents and Multiagent Syst.*, ser. AAMAS '22. Richland, SC: Int. Found. for Auton. Agents and Multiagent Syst., 2022, pp. 53–61.

[26] C. Borst *et al.*, "Rollin' Justin - Mobile platform with variable base," in *Proc. 2009 IEEE Int. Conf. Robot. and Automat.*, 2009, pp. 1597–1598. doi: 10.1109/ROBOT.2009.5152586

[27] N. Y.-S. Lii *et al.*, "Introduction to Surface Avatar: The First Heterogeneous Robotic Team to be Commanded with Scalable Autonomy from the ISS," in *Proc. Int. Astronautical Congr.*, vol. IAC-22. Paris, France: International Astronautical Federation, IAF, Sep. 2022.