Chapter 21
Meta-reasoning in Assembly Robots

Priyam Parashar and Ashok K. Goel

Abstract As robots become increasingly pervasive in human society, there is a need for developing theoretical frameworks for "human–machine shared contexts." In this chapter, we develop a framework for endowing robots with a human-like capacity for meta-reasoning. We consider the case of an assembly robot that is given a task slightly different from the one for which it was preprogrammed. In this scenario, the assembly robot may fail to accomplish the novel task. We develop a conceptual framework for using meta-reasoning to recover and learn from the robot failure, including a specification of the problem, a taxonomy of failures, and an architecture for meta-reasoning. Our framework for robot learning from failure grounds meta-reasoning in action and perception.

21.1 Introduction and Background

From thermostats and toasters to self-driving cars and unmanned aerial vehicles, robots are entering the human world in large numbers. Soon, robots of many different kinds will be ubiquitous in almost all aspects of human life. This pervasiveness raises many interesting questions from the perspective of "human–machine shared contexts." How will humans and robots work, learn, and live together? How will they communicate and collaborate with one another? How will humans understand robots? How will robots explain themselves? How will robots learn from observing humans? How will robots learn from their failures?

In a chapter (Goel et al., 2020) in the previous volume in this series of books on Human–Machine Shared Contexts (Lawless et al., 2020), we had advocated the use of cognitive strategies to afford effective human–robot cooperation: "if we want

P. Parashar (✉)
University of California, San Diego, USA
e-mail: pparashar@eng.ucsd.edu

A. K. Goel
Georgia Institute of Technology, Atlanta, GA, USA
e-mail: ashok.goel@cc.gatech.edu

W. F. Lawless et al. (eds.), *Systems Engineering and Artificial Intelligence*,
https://doi.org/10.1007/978-3-030-77283-3_21

robots to live and work among humans, then we may start with human-like, human-level, *cognitive* strategies to addressing novel situations." In the previous chapter, we described analogy as a cognitive strategy for robot learning from human demonstrations and meta-reasoning as a cognitive strategy for a robot to learn from its own failures. In this chapter, while focusing on assembly robots, we investigate meta-reasoning in more depth.

Apart from the ubiquitous Roomba, assembly robots are among the most commonly used commercial robots in the world today (IFR (International Federation of Robotics), 2020). In fact, assembly robots are critical to industrial economies, especially in the manufacturing sector. Most assembly robots are preprogrammed for some specific routine tasks such as fastening a nut into a bolt to hold two surfaces together. In fact, the preprogramming is done carefully to avoid failures because a failure can have significant economic costs (1 min of downtime for a large automotive assembly line past might carry a cost of the order of $50,000). This is also reflected by the vast efforts to structure the manufacturing environment so that the assumptions underlying the programming are always respected.

However, agile manufacturing often requires more flexibility: an assembly robot may face a task slightly different from the one for which it was programmed. Given the reliance of the preprogrammed assembly robot on the rigid environmental structure may mean that it fails to accomplish the new task, even if the difference from the familiar task is very small. In this chapter, we explore the question: how may an assembly robot use meta-reasoning to recover and learn from a failure in such a context? Depending on the specification of the problem, the cost of recovering from a small failure may be less than the cost of re-programming the assembly robot for every new situation in agile manufacturing. This prospect raises a new question: what kind of failures may occur when an assembly robot preprogrammed for a specific task is exposed to a slightly different task?

Meta-reasoning—thinking about one's own thinking—is one strategy. Meta-reasoning has received significant attention in research on AI as reviewed and outlined in Anderson and Oates (2007), Cox (2005), Cox and Raja (2011), Russell and Wefald (1991). However, much of previous work on meta-reasoning has been on simulated robots; meta-reasoning in physical robots has, thus, far received relatively little attention. As one may expect, physical robots impose hard constraints arising out of action and perception. This raises another question: What do the constraints imposed by action and perception mean for meta-reasoning? In this chapter, we are interested in exploring the relationship between action, perception, planning, and meta-reasoning. We specify the nature of the problem and identify trade-offs in designs of possible solutions.

21.1.1 Related Work

Our research lies at the intersection of artificial intelligence and robotics. Our work on meta-reasoning builds on a long line of research: The Autognostic project (Stroulia &

Goel, 1995, 1999) developed a multilayered agent architecture including situated, deliberative, and meta-reasoning components. In analogy to the redesign of physical devices, the Autognostic system viewed an intelligent agent as an abstract device and used a functional model to describe and repair a deliberative navigation planner. The Reflective Evolutionary Mind (REM) project (Murdock & Goel, 2008) generalized the Autognostic agent architecture. It developed a knowledge representation language called Task-Method-Knowledge Language for encoding functional models of agent designs that are more expressive than hierarchical task networks (Nau et al., 2003) and enable explicit expectation descriptions. This depth of description allows the REM architecture to conduct both retrospective and proactive adaptations for an assembly agent.

Unlike the Autognostic and REM projects, the Augur project (Goel & Jones, 2011; Jones & Goel, 2012) focuses on the use of meta-reasoning to diagnose and repair domain knowledge grounded in perception. Given domain knowledge in the form of a classification hierarchy, Augur associates meta-knowledge in the form of empirically verified procedures that capture the expectations about world states with each node in the hierarchy. When the classifier makes an incorrect prediction, Augur systematically invokes its empirical verification procedures to diagnose the classification knowledge. Finally, the GAIA project (Goel & Rugaber, 2017) provides an interactive CAD-like environment for constructing game-playing agents (for playing Freeciv) in the REM architecture and the Task-Method-Knowledge Language (TMKL). Given the design of a game-playing agent, GAIA executes the agent in the game environment. If the agent fails, then GAIA enables interactive diagnosis and repair of the agents' design. Parashar and colleagues (2018) extend this architecture to a situated agent in Minecraft with a separate and explicit perception process supporting the information gathering and repair efforts. They use an occupancy grid to represent task expectations grounded in the ego-centric view of the agent that is then used as a heuristic for guiding ground-level actions to learn task-level repair.

Goal-driven autonomy (GDA) (Muñoz-Avila et al., 2010) is a goal-reasoning framework where the agent makes use of an expectation knowledge-base (KB) to keep track of plan execution and to find discrepancies if any. GDA is based on meta-reasoning concepts like expectation matching system, discrepancy detection, explanation generation, etc. Dannenhauer and Munoz-Avila (2015) use hierarchical plans with annotated expectations, called h-plans, along with a semantic web ontology, to make better assertions about game states and to reason at various levels of a plan hierarchy leading to better control over strategy reformulation.

There are several threads in robotics research supporting and informing our work on representation and reasoning. It is generally agreed in robotics that hybrid systems, which can do both deliberation and some kind of reactive revisions, pave the way for more complex robotic applications (Kortenkamp et al., 2016), but there does not exist a systematic theory of how to combine different levels of planning, reaction, and learning. Müller and colleagues (2007) present a system that encodes plan transformations to make actuation easy based on pre-compiled heuristics and to apply these transformations at run time to make plan execution more efficient. This way, the planner abstracts away the difficulty of manipulation because of geometric and other

specific requirements (e.g., the number of objects to move or the number of rooms to visit) onto an abstract space. Beetz and colleagues (2010) present a cohesive "robot abstract machine," which integrates task reasoning with motion planning, perception, and other lower level modules on real robots, and, in theory, also allows meta-reasoning. This architecture is supported by KnowRob (Tenorth & Beetz, 2013), a knowledge database, which collects facts and beliefs for reasoning and revising plans.

Wolfe and colleagues (2010) and Kaelbling and Lozano-Perez (2011) provide two important insights that inform our architecture for meta-reasoning in assembly robots: (a) robots do not have all of the needed information before plan execution and thus the architecture needs to account for plan refinement during execution and (b) motion actions can be thought of as functional entities, which helps in reasoning about their effects in an abstract space. Christensen and colleagues (2010, Chap. 6) conceptualize replanning as a way of refining plans and present an elegant formulation of integrating facts from object perception as "agent now *knows* X" into the refinement-by-replanning framework. Dantam and Stilman (2013) conceptualize task language as a motion grammar relating high-level actions with semantic rules explicitly describing how the position, forces, and velocity of a robot arm are affected when a specific action is applied. Together, these various lines of previous work on robotics help in filling the blanks on how to view the physical processes of a robot in the same functional scope as task-planning. Finally, Parashar and colleagues (2019) present an exhaustive survey of how ontologies of tasks and actions enable and facilitate better human–machine understanding, motivating the design if our architecture for meta-reasoning in assembly robots that connects higher level ontologies to lower level constraints.

21.2 Illustrative Examples

Let us consider an example in which the robot needs to plan a sequence of actions so that it may pick up an assembly part, a pully in this case, lying on a table. Imagine that the robot has a camera mounted onto the gripper allowing the robot to plan motion using the eye-in-hand configuration. As an assembly agent, the robot has a library of action sequences for frequently used assembly tasks like "picking up a part," so it retrieves a *task method* that specifies such a sequence from its memory and applies it on the pulley. This method is shown in the top half of Fig. 21.1 along with the expected progression of the plan in the current context. However, suppose that this method previously has only been used for parts far smaller than the pulley and so the agent does not realize that the ***Align*** action (Fig. 21.1 (top)-iii) will not move the gripper all the way to the immediate top of the pulley as expected. This is because ***Align*** uses an algorithm that moves the gripper toward the object until the object is centered on the camera image and occupies a certain amount of area in that image. The distance at which this condition becomes true is different for objects of

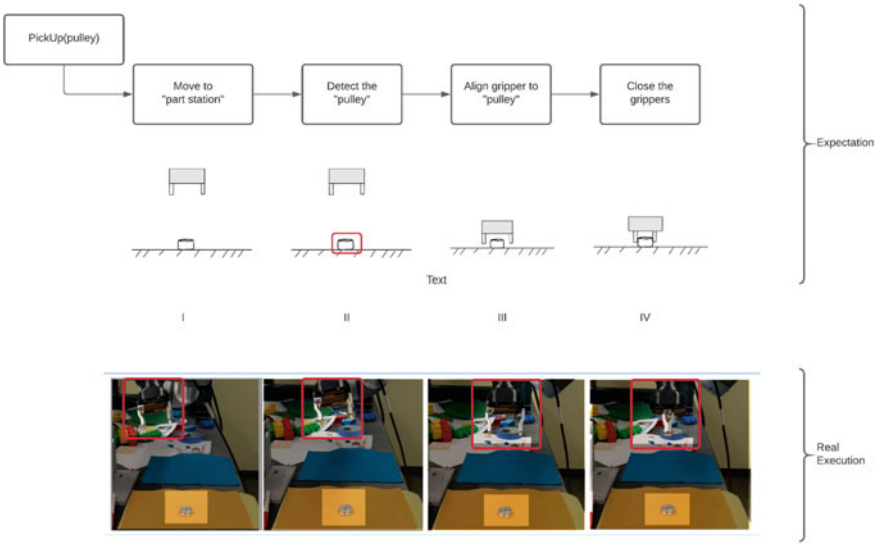


Fig. 21.1 The top half of this picture depicts what the agent expects will happen when it executes the stored plan for **PickUp** assembly action. The agent will (I) go to "part station," which is a fixed location in the environment, (II) agent will detect the pulley object at the part station, (III) the agent will align its gripper with pulley, and (iv) the agent will grasp the pulley. This plan was initially made for much smaller parts that the **Align** action could go close to. However, due to a larger size of the pulley and field-of-view restrictions of the agent's camera, the **Align** action is not able to go all the way to the immediate top of the pulley. This results in a failure as the grippers close far from the pulley as shown in the bottom half of the image

different sizes. Consequently, when the agent executes this method, the results look like Fig. 21.1 (bottom), which is an obvious task failure.

Let us analyze this failure and its possible repairs. This problem can be viewed in two ways: an incorrect parameterization of the ***Align*** action or a knowledge gap in the task procedure due to a mismatch between the expectation of executing the ***Align*** action and the actual state of the world. The first view considers that if the robot can just fix the parameterization of ***Align*** action to enforce the gripper to move closer then we can solve this problem. If ***Align*** is considered a function with the amount of area covered by object as a thresholding parameter, we can increase the value of this parameter and move closer. However, this repair has strict bounds! The physical shape and size of the part is an immutable property so while a re-parameterization of ***Align*** may bring the gripper closer, there will always be a minimum distance beyond which the camera will lose focus of the object due to the laws of optics. The second view considers this failure as a knowledge gap in the task specification that manifests as a physical failure. While ***Align*** falls short of matching the task-level expectations of the agent, the robot's method library may have more assembly skills (like a ***MoveTo***(x, y) action) that can be tried to compensate for this gap and repair the plan. This kind of repair is only limited by the completeness and soundness of

the method and skill library, which are far more malleable concepts than the laws of optics. We also cannot replace the ***Align*** action to avoid using it altogether because perception is necessary to situate the objects in the environment, which affects the motion and actions of the robot that manipulates the object under consideration.

The geometry of an object places objective constraints on *what configuration* of the object is compatible with a given assembly operation, while perception helps in grounding the geometry in a *particular situation.* A robot uses both kinds of information to understand *how to place the object* in the current situation (perception) and *how to move* to actuate the desired object state (geometry). Thus, geometry and perception both play an important role in transferring the plan from the familiar problem to a new problem.

To further extend this example, consider that the illustrated subtask was part of a larger assembly task where the robot is required to pick up the pulley and then ***Insert*** it onto a shaft (Fig. 21.2a). Assume that this time around the robot has a complete plan that works for the given scenario. Now suppose that the robot is given a complementary task in which it needs to pick up the shaft instead and insert it into the pulley affixed to the station Fig. 21.3b-III). At a high level, the agent may ask if the same general plan would still work for the new task? At a finer level, the agent may ask whether there are implicit assumptions in the previous plan based on the geometry of the pulley? If so, how may the agent transform the plan for the shaft for an insertion into the pulley? If the agent knows how to grasp the shaft, a direct transfer of the plan would result in the situation depicted in Fig. 21.2b, where the robot grasps the shaft and moves in the same relative way to insert it as it would have

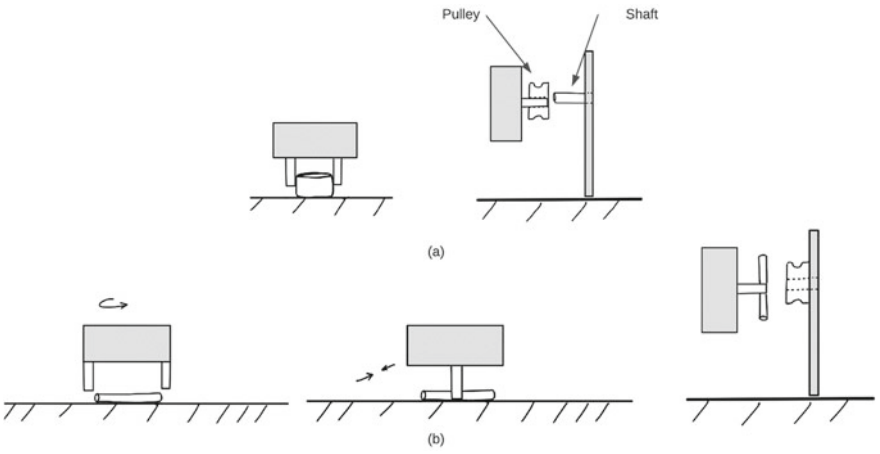


Fig. 21.2 **a** Depicts the larger assembly task whose subtask was analyzed in Fig. 21.1. The overall assembly task was to pick up the pulley and then insert it onto a shaft affixed to a station. **b** An example of failure due to direct transfer of the plan for the task "pulley-on-shaft" to the "shaft-into-pulley" task, where the geometry of the shaft is not accounted for. In the third diagram, the shaft is aligned with respect to the stationary station in the same way as the pulley, which may make sense causally, but does not make sense geometrically

with the pulley. This is obviously wrong since the shaft's length is compatible with insertion into the pulley and not it's surface. This is an example of task failure due to a knowledge gap in the agent's task model relating the geometry of an object to the assembly attachment.

There are several types of which can occur in a complex assembly system. For example, perception might be imperfect, or the actuators might not work due to network error, or new objects have different physical properties, which breaks the dynamic modeling of the ***Screw*** action. Thus, to systematically situate the use of meta-reasoning for recovering and learning from failures, it is important to analyze the failures as to whether they are recoverable and the recovery is affordable, as well as to design the architecture of the assembly agent to make the recovery process more efficient and affordable.

Problem statement. Our conceptual framework for meta-reasoning for robotics, meta-reasoning, action, and perception form three vertices of a triangle with deliberative planning and the skill library lying inside the triangle. In a simulated world, we can make assumptions about action and perception and focus on the interactions between the meta-reasoning, deliberative, and skill layers. However, for physical robots, action and perception impose strong constraints. The general goal of our research is to understand the interplay among meta-reasoning, action, and perception. In particular, we seek to ground meta-reasoning in action and perception so that an agent can exploit the task structure and domain knowledge to guide adaptation routines when faced with small variations in the task environment. This capability will help in creating robot agents that are more autonomous and capable of self-adaptation in dealing with small degrees of novelty. Our goal in this chapter is to specify the structure of the problem and thereby characterize the space of feasible solutions.

21.3 Assembly as a Reasoning Problem

Assembly planning and execution is a hard problem requiring solutions to a varied class of subproblems, such as part sequencing as well as metric-level precision planning, to succeed. We choose to model this problem using a three-level problem hierarchy with each subsequent level planning for a smaller scope of the overall assembly problem informed by the solutions of the level above. It is a cascade of mission-level (product planning, longer time-horizon), task-level (action sequencing, medium time-horizon), and skill-level planning (metric-level routines, shortest time-horizon) with each informed of the planning context from the previous component. The assembly mission planner generates a partial-order plan for sequencing part attachments for assembling the whole product. This sequence is used by the assembly task planner to come up with totally ordered task plan actions or *assembly skills* to do the part attachments. Finally, the assembly skills invoke lower level specialized planners and routines to plan for motion or grasping and gain perceptual information from the environment. This decomposition is illustrated in Fig. 21.3.

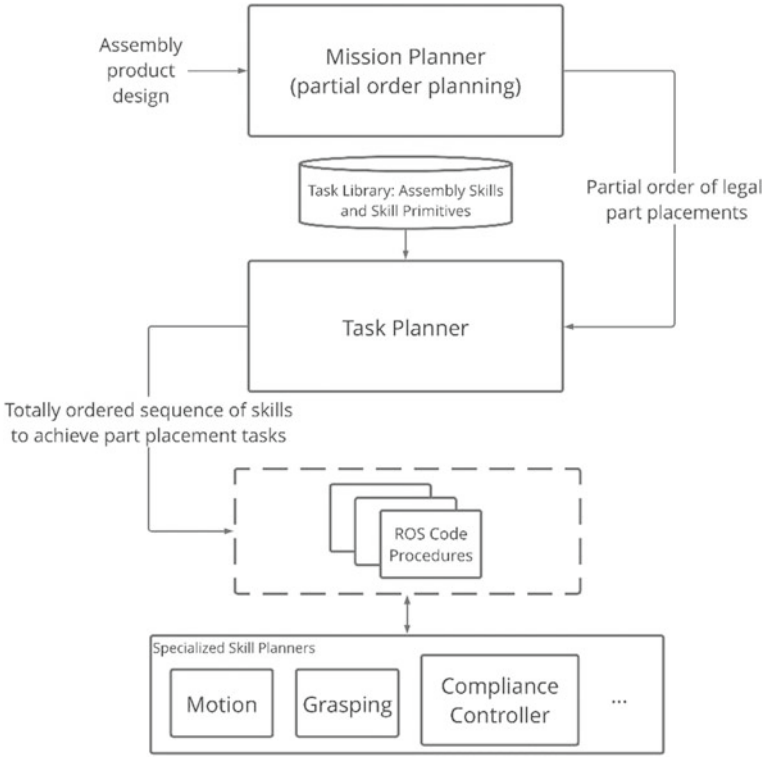
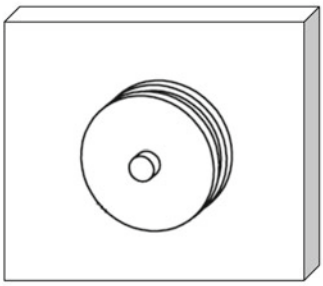


Fig. 21.3 The decomposition and structure of an end-to-end assembly problem. At the top level, a mission planner plans for part placements based on the geometry and specifics of the assembly product. This context is sent to an assembly task planner in the form of partial ordering of part attachments. The task planner uses a skill library made of assembly skills to generate an action sequence, which, in turn, invokes the lower level code procedures to perceive and manipulate the environment

Mission Planning: To represent the assembly product as a mission planning problem, we adapt and refine the description of the *assembly mission* introduced by Mello and Sanderson (1991) for our needs. As shown in Fig. 21.4, the mission M is the tuple <P, A, R, f> , where P is the set of all the parts, A is the set of attachments required between the parts, R is a set of relations linking parts with appropriate attachments, and f is the set of functions relating the parts and attachments to their geometric properties. This mission is supported via a knowledge base of parts and their properties including the type of the part, the relevant attachments for each part type, and the part relations enabled by the different attachments. To clarify, R in Fig. 21.4 is the configuration of parts to be achieved ($attached$(p0, p1) V $shaft$(p1) V $task\text{-}board$(p0)), while A is the set of specific assembly skills required to achieve the relation (***Insert***(p2, p1) → $attached$(p2, p1)). R also stores a crucial piece of information, which we assume the user will provide, about the relative part configurations; for example, ($on\text{-}top$(p0, p1) ∩ $on\text{-}top$(p1, p2)). This knowledge



```
P = { p0: taskboard,
      p1: shaft,
      p2: pulley}
A = { a0: fixed,
      a1: insert}
R = { "a-to-p":
        { r0: (a0, p0, p1),
          r1: (a1, p1, p2)},
      "p-to-p":
        { r2: on-top(p1, p0),
          r3: on-top(p2, p1)}}
f = { "final-pose":     { ... },
      "approach-pose":  { ... }}
```

Fig. 21.4 The code snippet on the right-hand side describes the assembly product on the left using our formal mission description based on Mello and Sanderson (1991). P collects all the parts while A collects all the attachments used in the product. R relates attachments to parts and the relative configuration of parts with respect to each other. F collects the geometric information associated with the attachments in any arbitrary coordinate frame

helps the mission planner come up with a partial order for progressing through the attachments and send it to the task planner. For a completely pre-defined mission, f stores function mapping each part and attachment to its goal-state in the metric space in the form of its six-dimensional pose. This information is utilized by the task planner to ground the tasks and consumed by the motion planner to plan in metric space. Thus, the input to the mission planner is the assembly product and the output is a partial-order plan of part sequencing with associated pre-defined attachments.

Task Planning: Given that the mission description includes the specific attachment to be used for a part's placement, *assembly task* planning addresses the problem of how may the robot actuate the given attachment for the assembly part? A special top-level method is responsible for sequencing the attachments and does so by choosing the next unplaced part and associated attachment from the partially ordered output of mission planner and decomposing the assembly task associated with it. We use HTN planning (Nau et al., 1999) for assembly task planning. There is a central task library that consists of compound and primitive tasks: each compound task is hierarchically decomposed into a totally ordered sequence of primitive tasks. The primitive tasks directly invoke the metric-level planners and routines to (a) actuate the robot or (b) gather perceptual information for grounding the task decomposition: by grounding we mean parameterizing the compound and primitive tasks with the correct environmental coordinates based on perception.

In this chapter, we will call the primitive tasks as *assembly skill primitives* based on the taxonomy proposed by Huckaby (2014) (see Fig. 21.5). Thus, the mission is made up of tasks and each task is made up of compound or primitive assembly skills. To associate compound assembly skills with attachments, the knowledge base consists of *{attachment-name, goal-configuration, list-of-relevant-parts, list-of-tools} tuples,* where *attachment-name* maps to each unique attachment-type possible in A and associates it with an assembly skill in the task library. The symbols in *list-of-relevant-parts* and *list-of-tools* are used to parameterize the decomposition. Each assembly

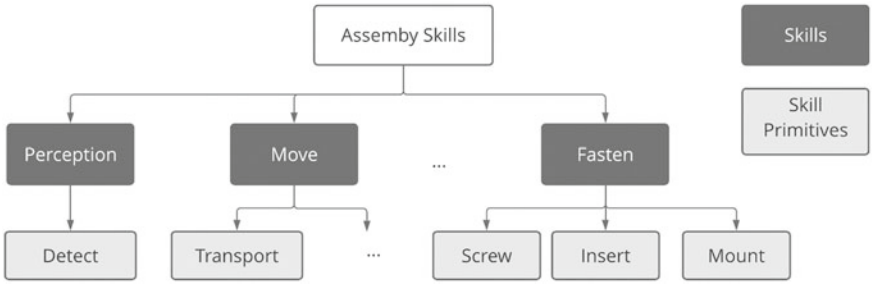


Fig. 21.5 An excerpt from assembly skill taxonomy proposed by Huckaby (2014). The taxonomy defines and organizes all the physical capabilities required to do automatic assembly in a hierarchical manner. At the top are general skills (dark grey boxes), which define high-level capabilities a robot should have. Some general skills are then decomposed further into leaf nodes, which are termed skill primitives (light grey boxes). These are the specific but different kinds of instantiations of the general skill that a robot can have. For example, a robot can **Fasten** two objects by **Screwing** or **Inserting** or **Mounting**

skill is decomposed into a totally ordered sequence of more skills or skill primitives. As shown in Fig. 21.5, some skills act as a high-level guard for contextualizing which skill primitive to invoke, for example, a ***Move*** skill has a list of {pre-conditions, skill-primitive} tuples; viz, if pre-condition(i) is true then primitive(i) is executed. Other skills can have more complex decompositions. The preconditions of skills encode two important kinds of knowledge: applicability of different primitives under different conditions and different primitive sequences for manipulating objects for the attachment depending upon different initial states.

To make the subsequent discussions about failure analysis in assembly more precise, we introduce minimal terminology here. We define an attachment skill primitive as a special kind of assembly skill primitive that actuates the physical contact-based manipulation, which *attaches* objects in desired configurations (e.g. ***Screw*** and ***Insert***). Preparatory assembly skill primitives are the rest of the domain-relevant object manipulations required to get to a state where an attachment action can be executed.

Definition 0 An attachment primitive, $\mathrm{aa}(\mathrm{a_i})$, is an assembly skill primitive that actuates the main contact relation achieved by completion of an attachment $a_i \in A$.

Definition 1 Plan segment $p' = prep(p(a_i)) : last\left(p'\right) \prec aa(a_i)$ manipulates the world to bring about the preconditions required by an attachment primitive; it is the *preparation method* for that assembly attachment.

Skill Planning: Specialized planners, both off-the-shelf and specifically implemented for the assembly robot, are used at this level to actuate the assembly primitives. We consider the underlying algorithm of the primitives as a black-box and are only concerned with *how* each primitive's execution changes the robot state. We only assume each primitive is completely parameterized by the task, object, and other physical parameters sent by the task planner. Each primitive skill execution is like

a blocking function call, with the agent executing the next ordered primitive skill when the control returns.

$$S_{prim} = f_{s\_prim}(\text{skill} - \text{parameters})$$

21.4 Failure Mode, Effect, and Repair Analysis

This section presents our analysis of the failures that we observed while developing a dual-arm assembly system for the World Robot Summit assembly challenge 2020 (aka WRC, 2020; WRC (World Robotic Challenge), 2020) based on the reasoning structure described in the previous Sect. 21.4 (more details can be found in Parashar et al. (2021)). We cannot eliminate all failures in an assembly system that is supposed to operate for different variations of the same product, but we can provide an informed opinion using a qualitative Failure Mode and Effect Analysis (Liu et al., 2013) in an effort to identify which slice of the failure-space is more amenable to an automatic recovery versus meticulous preprogramming. Thus, our research goal is not to automatically recover from all possible failures in the assembly task but to allow the robot to only fail in ways that are non-fatal and then enable it to learn from them. We want to understand the nature of failures, their cost, and the tradeoffs against the cost of meta-reasoning for repairing the failures. We want to answer the question: which class of failures can an assembly agent afford to repair? What kind of repairs are available for such failures? And most importantly, which failures manifest when the task goal or the task environment changes incrementally? We chose to categorize failures by their origin in the planning stack, type of knowledge required for repair, and the practical implications (including severity and detectability). We first present an origin-based analysis of failures at every level followed by their classification into a preliminary taxonomy.

Interestingly, failures can only happen during reasoning or during the run-time execution of a task. However, repairs for potential failures can be identified even before the run time through a meticulous examination and analysis by domain experts. In fact, we observed the need for the pre-meta-reasoning structure (Pourazin & Barforoush, 2006) arising in our work specifically for the failures, which were too costly for our run-time budget defined by the constraints of the competition: speed, repeatability (accuracy of actuation when a motion is repeated), and accuracy of operations.

21.4.1 Skill-level Failures

In general, the function of a skill in an assembly task is either to set up objects in a state where an attachment action can occur (picking, aligning, or placing) or

to execute the attachment itself (inserting, screwing, or mounting). Thus, we can distinguish between the skill failures, which occur during the preparation phase versus those which occur during the attachment phase. Based on the taxonomy from Huckaby (2014), we can infer that failures during the preparation phase mainly relate to aligning, grasping, transporting, and detecting objects. These actions typically do not have fatal consequences; most of these skills use heuristic-based algorithms to conduct a search in the physical space until a threshold is passed by the relevant physical measurements. This activity is a classic example of approximate reasoning that provides "adequate" repairs, i.e., near-optimal repairs with defined lower bounds (Russell & Wefald, 1991). If the task-level parameters situate an action in a "good-enough" region of a search space, then reasoning at the lower level can handle recovery from a failure. Therefore, preparation-phase failures can be repaired using behavioral heuristics about how the parameters affect the expansion or contraction of the operational state-space of an action. Note that handling such failures should be within the bounds set by our assumption of skills being a black-box function with the task planner grounding it in parameter values. However, we will need additional meta-knowledge at this level about how each black-box's parameter affects its function.

On the other hand, failures arising during the execution of an attachment action may have fatal consequences. Here by "fatal" we mean rendering a state where either the assembly material is damaged beyond repair or one that requires manual intervention to reset the agent. Consider when an agent applies too much force in the wrong direction while screwing the bolt into the nut and jams the bolt (Fig. 21.6). This is a hard-to-detect failure with a high probability of ruining the parts based on the material. Additionally, to prevent further ruining of the material, the repair requires a human to manually reset the system and assembly line before restarting the process. Generally, such actions require either specialized hardware or implementing sophisticated closed-loop control dynamics (Jia et al., 2018). As a rule, complex attachments require much more sophisticated knowledge for execution and recovery while simpler attachments can be repaired with general heuristics. However, if the attachments have more complex control flows than a linear operation, then such generic repairs like reversals of action (Laursen et al., 2015) might not work at all. Thus, it is reasonable to surmise that there exists a small subset of simple attachment actions for which errors are recoverable like block stacking.

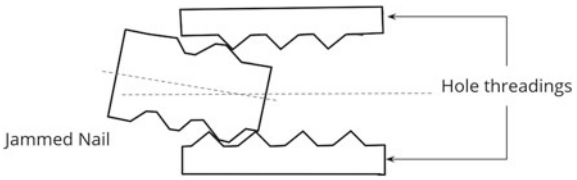


Fig. 21.6 Visual depiction of a nail jamming in a threaded hole due to excess force and/or wrong insertion angle. This failure is not trivially repairable and requires deep modeling of the dynamics between surfaces as well as the helical geometry. Such complex failures are beyond the scope of repairs considering our formal definition of skill as black-box

To summarize this section, we noted two failure modes arising at the skill level, one recoverable and the other fatal. The recoverable failures are repaired by heuristic of "retry with a perturbation." However, each action has a specific meaning for each perturbation associated with it; for example, if a camera cannot detect an object due to glare, then the perturbation is physically achieved by displacing the camera by an arbitrary amount around its focus point. On the other hand, to perturb an alignment action (which is basically a spiral search with some forward force to align and latch the features of bolt and nut with each other), the perturbation is an increase of the search radius or a slight bump in the forward force to make it more detectable. Thus, any repair here would also need specific meta-knowledge about the portable parameters and the desired perturbation function to be applied for each assembly skill.

21.4.2 Task-level Failures

The task-level planner is responsible for ordering skills and figuring out the task and object parameters consumed by the lower level procedures for execution. The former ensures the agent progresses in a causally correct manner in the abstract task space while the latter ensures that the progress is grounded in correct physical positions, forces, and artifacts. Thus, we can classify two kinds of major failures right away: ordering failures and grounding failures.

Failures due to a knowledge gap in causal ordering usually arise when an assembly task is changed in some way that violates the assumptions of the coded recipes/methods. Since a task is scoped by the attachment it achieves, we can ignore a functional change in the goal configuration to be a reason for failure. However, an assembly task can have different resources available in the environment than those assumed by preprogrammed knowledge. In this case, a new task model may need to be inferred. An example is to attach two tiles: the domain designers assume the availability of a dowel and defined *insert* attachment task, but the run-time environment only has a glue-gun available. Also, sometimes a conceptual repair can lead to a grounding repair, for example, if the glue-gun's action model is not known, it would need to be learned or inferred from data. Fitzgerald and colleagues (2019) use demonstrations to ground such constraints for a task model. However, it is difficult to collect demonstrations for learning these constraints for an assembly task simply because the details lie in the force-based maneuvering and jiggling when doing any contact-based action like screwing or insertion. These observations are hard to collect data from and require specialized data collection rigs. Instead, a more applicable repair is to infer the constraints using geometric knowledge and models of the parts since assembly products usually have highly meticulous CAD representations available. Our example in Sect. 21.3 is of a complex failure where there is a grounding failure such that the physical position of the gripper does not match its expected position, and the solution does not just change the grounding information, but rather affects the ordering of the actions themselves by adding a new transport/move action.

Grounding failures can easily lead to fatal behavioral failures if not scoped within the stable region of an agent's motion space: for example, the object position is correctly detected, but the object itself is outside of an agent's reach (assuming a stationary robot). Thus, it is a reasonable assumption that we only consider grounding failures, which respect the motion limits of a robot, i.e., if a task is given, it is achievable for the given physical configuration of the agent. Even with this assumption, a traditional assembly system relies on a perfect encoding of geometric information to operate efficiently. Therefore, repairing grounding problems has a clear metric of overall task and motion efficiency, which can be used to iteratively improve their solutions.

Ordering a repair is based on learning or inferring a new task model for using a new object's features for its intended function, while grounding the repair is based on lifting the constraints added/modified in world geometry to a higher level plane and adapting tasks to account for it. Ordering repairs can range from inferring reversal actions (Murdock & Goel, 2008), or learning a new policy guided by the known goal configuration constraints when new objects are introduced in the task space (Parashar et al., 2018). Grounding repairs usually require integration with a finer-grained representation of the environment to enable the analysis of relations between task symbols and physical configurations (Bullard et al., 2016).

21.4.3 Mission-level Failures

The mission knowledge is necessary for understanding the sequence of part placement that leads to legal product configurations. While some attachments can be reversed (e.g., insertion), others are irreversible by nature (e.g., gluing). Thus, the lower level physical properties of the attachments can make a mission sequencing failure irreversible as well. At this point, the mission planner needs to find new sequences of possible object ordering and check the legality of those attachments by asking the task planner about available procedures. This step is another example of non-fatal failure; however, the repair costs can vary over a large spectrum based on the number of parts and possible attachments between them. A better way would be to ask the task planner for proposals of object combinations since it has access to information about the features of parts, enabling the mission planner to infer the more correct orderings as opposed to random combinations. Better yet, if the mission planner can figure out which attachments are irreversible and order those correctly, then the agent might be able to avoid all major failures at this level. This suggests that knowledge about critical mission landmarks can significantly reduce the overall failure and repair costs of an assembly system. One way of inferring such knowledge is by a geometrical analysis of parts in a simulated environment as shown in De Mello and Sanderson (1989) and by inferring relations between these part orders and the available parts in the current mission (Fitzgerald et al., 2018).

21.4.4 A Preliminary Taxonomy of Failure Modes

Table 21.1 summarizes the failures discussed above and organizes them in a taxonomy of origin and mode, along with whether such failures are recoverable or not, and the knowledge base that recovery depends on. When we say that a failure is recoverable, we assume the availability of the required recovery knowledge.

21.5 Assembly Plan Repair as a Meta-reasoning Problem

The meta-reasoning space of the plan repair domain is composed of a Meta-KB, Trace-KB, and Expectation-KB. Meta-KB compiles facts and beliefs about objects, properties, and their association with task/skill goals. Trace-KB is a time series of environmental observations and the logical progression of tasks that bookend each action. Expectation-KB is a baseline trace generated by the meta-reasoner based on optimistic assumptions. The grounding of Meta-KB and Expectation-KB largely depends upon the context of the failure and repair to be undertaken. While the framework and formulation of meta-reasoning below are general, the choice of representations and the corresponding methods of reasoning and learning cannot be isolated from the specifics of the domain. The meta-reasoning problem could be one of learning, and the Meta-KB and Expectation-KB then may guide the learner to *learn more efficiently*; similarly, if the meta-reasoning problem is one of knowledge modification, then Meta-KB and Expectation-KB should guide modification routines to *modify efficiently*. Meta-KB is especially dependent on the choice of a failure repair and its parametrization. Therefore, it is important to understand the nature of failures in assembly and their corresponding repair mechanisms to propose a class of meta-reasoning strategies, which would allow the agent to recover from unexpected task variations.

Table 21.1 Summary of failure modes, recoverability, and recovery knowledge required

Origin	Mode	Recoverability	Recovery KB
Skill	Preparation	Yes	Parameters and perturbation function
	Attachment	No except for simple ones	Reversibility knowledge
Task	Grounding	Yes	Physical positions and orientations of task-relevant objects
	Action ordering	Yes	Knowledge about robot and object features; interactions between different features
Mission	Part ordering	Depends on the parts and resources available	Reversibility knowledge, knowledge about legal part pairings for tasks

Given our analysis in Sect. 21.5, we are much better situated to propose a class of repairs for the assembly failures discussed. Now we want to answer two questions:

1. What kind of repairs are applicable to the non-fatal assembly failures in Table 21.1?
2. What new components, frameworks, and conceptual extensions are required to the base reasoning model in Sect. 21.4 to support the repairs?

21.5.1 Meta-reasoning Architecture for Robots

To do knowledge repairs, it is important to first be able to detect failures as they happen. Furthermore, as we mentioned, there can be many different locations of causes that could relate to this failure. Thus, to apply appropriate repair the knowledge, it is also important to localize the cause for the failure. Let us briefly discuss an architecture that will enable us to conduct this failure detection and localization. Robotic architectures are special in that they must planning or replanning while situated in a physical environment. The prototypical three-tiered architecture (aka 3 T) (Firby, 1994; Kortenkamp et al., 2016) seems well suited for our purposes. It has a deliberative layer at the top, which takes care of long-term planning for the domain. Next, an executive layer in the middle translates deliberative decisions to real-time process invocations and monitors real-time processes for feeding back into planners if needed. Finally, it has a behavioral layer at the bottom, which houses all the real-time processes (specialized planners and routines in our case), which run in parallel to manipulate and perceive the environment. In fact, our system for WRC 2020 used this exact architecture to detect and repair preparatory skill failures and for simple detection of task failures. In the rest of this section, we use this 3 T meta-reasoning architecture (Fig. 21.7) as base to conceptualize a meta-reasoning architecture for assembly robots. We systematically add feedback for failure detection (input to skill monitor), heuristics for cause localization (directing output from skill monitor), and the relevant meta-knowledge to conduct repairs (resolution and action/perception commands back to skill level). Figure 21.8 shows an example of how high-level tasks are decomposed into skills and executions at the executive level of this 3 T architecture.

21.5.2 Skill Failure Detection and Repair

Feedback: A skill in our formalization is a black-box function parameterized by the task planner. Once a skill is executed, the agent moves to the next planned skill to actuate. We can extend this notion of skill to a function by adding provisions for returning values. In the WRC 2020 system, we extended the skill primitives to be able to return three values: *success*, *failure,* and *running*. We used the behavior tree (BT) framework to instantiate plans using nodes and arbitrary control flows based

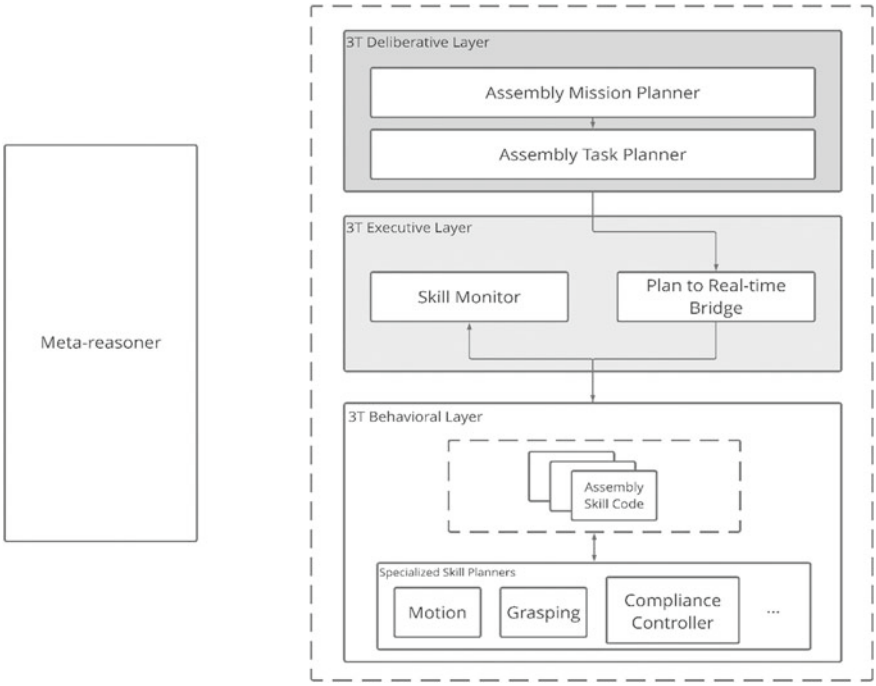


Fig. 21.7 The meta-reasoning architecture for assembly robots based on the assembly planning and execution architecture used in WRC 2020. Note that the meta-reasoner and assembly system are not yet connected; the subsequent sections illustrate the connections between them

on the planner logic (Colledanchise & Ögren, 2018). These three values are the prototypical signals returned by a BT as each node in the tree executes. We know that each skill node (action node in BT terminology) invokes a planner underneath, so a *failure* means the black-box algorithm could not succeed.

Repair: For every preparatory assembly skill, we can come up with a reparative description consisting of repair parameters, a perturbation function, and an expected number of retries. Given access to the transition function of an assembly action, we used a simple algorithm to apply a perturbation function to the repair parameters and repeated for the expected number of tries until the action succeeded. The Expectation-KB consisted of the expected number of retries for *success*, the Trace-KB consisted of the current iteration, and the Meta-KB consisted of a list of tuples {parameter-name, perturbation-function} (Algorithm 21.1). The Meta-KB could have also been empty, which defined the simplest case where the action just needed to be repeated; for example, when the gripper did not actuate due to drivers dropping out but could be retried. This worked because our system had an underlying process supervisor monitoring these processes and restarting them when they went down; describing this base-level recovery framework is outside the scope of the current chapter.

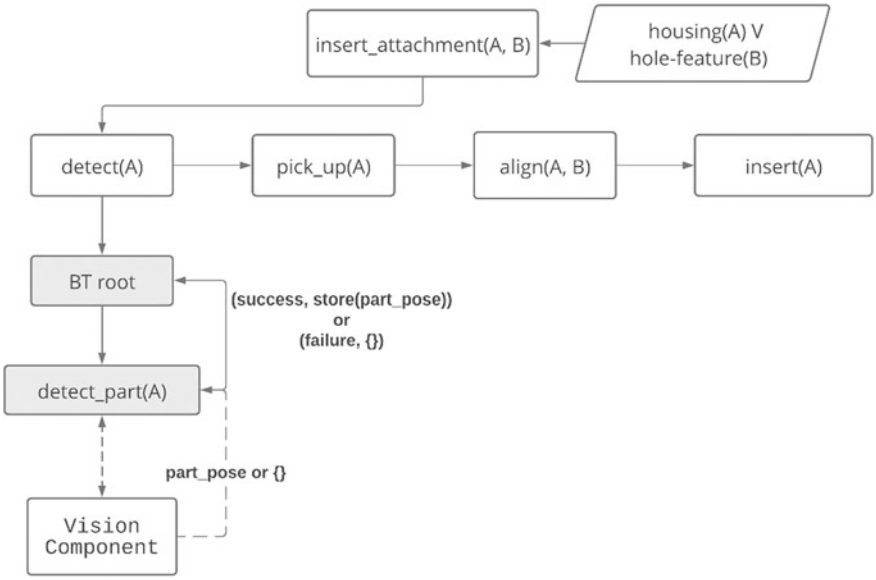


Fig. 21.8 A diagram depicting how high-level plans are translated into a behavior tree to communicate to lower level processes. Depending upon the results of the lower level process, the behavior tree returns success or failure to the tree above it. In this figure, assembly skill primitive detect (white boxes on top) is translated to a simple behavior tree (grey boxes), which relate to the vision component underneath. The "detect_part" behavior tree node is an "action node," which is directly executed. BT also has provision for other kinds of nodes, which assess a condition, or implement conditional control-flow changes

Algorithm 21.1. Repair of preparatory assembly skills given perturbations for each parameter

Algorithm 1 Reactive Repair for Preparatory Assembly Skills

```
Input: skill - reference to procedure, parameters - dictionary of names and values for arg-list of skill
i = 0
while skill(parameters[all].values)[0] ≠ SUCCESS or i < Exp-KB do
  for all (para-name, pert-func) tuples in Meta-KB do
    parameters ← pert-func(parameters[para-name].value)
  end for
  i++
end while
```

The important thing to note here is that the application of these reparative algorithms did not necessarily need the overhead of being sent up to a meta-reasoner and back. First, such a procedure can add unnecessary lag to the signal, which can hurt the run-time stability of the agent. Second, since our architecture is already using behavior trees to instantiate the decision-making process of the planner as a tree structure, and we can easily extend this to add the meta-reasoning procedural

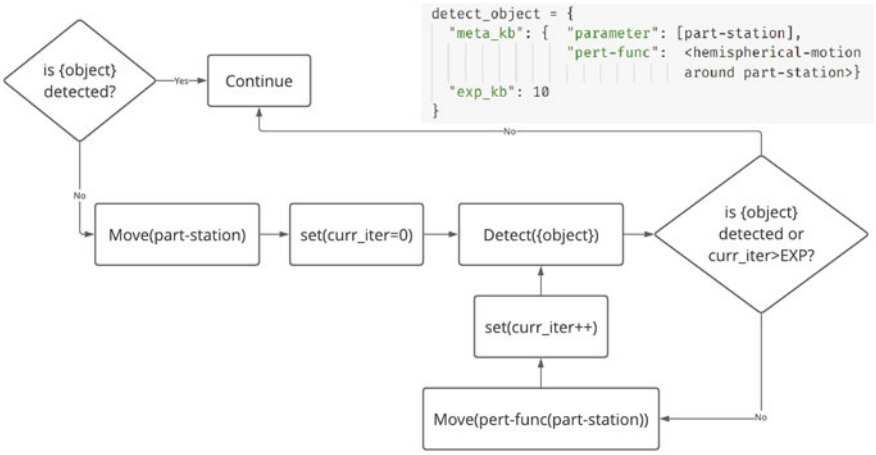


Fig. 21.9 The snippet on the top describes the meta-knowledge associated with the repair of **Detect** action. The tree structure encodes Algorithm 21.1, allowing efficient repair during execution. This tree checks if the parameter {object} is already detected, if not it **Moves** the agent to the part-station and invokes the **Detect** action over the object. The agent repeats this process if the object is not detected and moves the camera with small perturbation to account for any variable lighting, which might be preventing object detection. If the object is not detected even after expected number of retries, then it propagates a failure up to the rest of the tree

structure to it as well. Thus, this repair algorithm is pre-compiled within the reactive structure of the behavior tree allowing for quick catch-and-repair during run time. Figure 21.9 shows such a behavioral tree structure for the ***Detect*** action.

This form of applying pre-metareasoning (i.e., meta-reasoning based on heuristics and information known before undertaking the reasoning process) gives rise to an updated architecture for meta-reasoning for assembly robots as shown in Fig. 21.10. Our experience with this inquiry differs from the traditional meta-reasoning architectures proposed in Cox and Raja (2011) and suggests that multiple meta-reasoning structures can exist in one architecture supporting meta-reasoning to different levels. In our WRC system, we observed a significant drop in the fatal crashes of the system by implementing the described pre-meta-reasoning behavior.

21.5.3 Task Failures and Repairs

Feedback: The align-failure with pully and insertion of shaft-in-pulley instances in Sect. 21.3 are examples of knowledge gap failures stemming from different causes. The symptom of both these failures is that the preconditions for executing the next action (grasp and insert, respectively) are not satisfied. We, thus, propose the assembly skill taxonomy to be extended by adding the provision for *verification skills*, which can have arbitrary logic for asserting the truth or falsehood of pre/postconditions

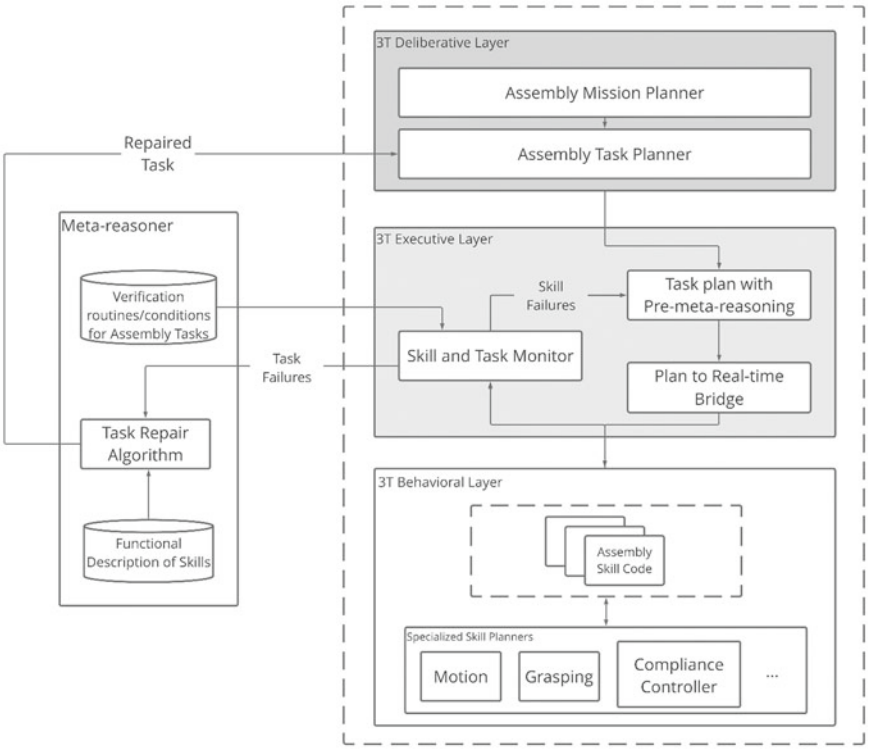


Fig. 21.10 The proposed meta-reasoning architecture integrates two kinds of action-perception repair loops for assembly robots. The inner loop (or the fast loop) is inside the Executive Layer where the plans are precompiled with repair heuristics and do not need to be propagated to the meta-reasoner for repair. The outer loop (or the slow loop) propagates task-level failures to the meta-reasoner, which uses additional knowledge about the functions of the assembly skills to propose repairs to the broken task

based on the output of ***Detect*** skill. The BT framework has *condition nodes*, which test a given logic returning *success* or *failure* based on the logic's assertion, thus incorporating this in our system should be relatively easy. The harder and more important part is grounding the pre and postconditions of skill primitives in states that are *perceivable by the agent*, for example, a task to turn on a switch inside a closed box without any light attached to that switch will not qualify as the kind of task, which can be verified or repaired.

Jones and Goel (2012) present an approach where meta-knowledge of verifying ground-level decisions is explicitly mapped in percepts thus enabling result-oriented guidance for higher levels. Parashar and colleagues (2018) present an instance of using occupancy grids to represent the environment within the configuration space of a robotic arm and to define a similarity computation over them allowing more detailed perspectives of conceptual plans on ground-level. Thus, these heuristics can be based on memory or explicitly engineered knowledge. In either case, we can now

use this as the baseline of agent's expectations as to how the world should evolve. We hypothesize that by grounding meta-reasoning in physical percepts, we can create a multi-modal understanding of actions over ground-level information.

Repair: The verification procedures add *heuristics* in abstract task space about monitoring the progression of the plan in the physical space. By analogy, we propose that the skill primitive descriptions need to be supplemented with heuristics for informing their functions in conceptual and physical spaces. In the context of hierarchical task networks, the extension into hierarchical goal networks (HGNs) makes a similar point; the nice thing about having a goal description is that it can be translated with more clarity into specialized planners down the line. Shivashankar and colleagues (2014) illustrate a process where such an integration is established between task planning and motion planning by using goal definitions as the bridge. Dantam and Stilman (2013) describe the function of actions in a motion grammar by ascribing explicit semantic rules to them while Wolfe and colleagues (2010) introduce the same concept as *transition models*, which have been adopted and refined to suit our needs in our formulation (Sect. 21.4). Stroulia and Goel (1995) and Murdock and Goel (2008) impose a similar view over conceptual transformations; these authors functionally index them for use in generic problem-solving methods, which hits close to our aims of proposing generic repair methods within domain boundaries. Using these functional representations of actions, we expect to be able to conduct repair in a more surgical fashion, with less supervision than an uninformed system.

An important thing to consider here is that an action is not the only cause for a task-level failure. A task failure can also arise due to imperfect perception: as an example, consider plugging a USB stick into the computer port. It is not easy for perception to recognize the correct configuration from an image alone that may be in variable office/home lighting. Furthermore, it is beneficial to figure out approaches, which can resolve actions conditioned on imperfect perception rather than rely on the perfect perception of bespoke features for general robotics. A heuristic that works for every human is to try and plug the USB in one configuration; if it works great, otherwise flip and try again. Thus, actions can provide definitive answers to possible questions if the actions are not too costly (like the WRS 2020 attachments). How may meta-reasoning help with these situations where high-level knowledge about a task and knowledge about an action can resolve perception ambiguities? Along the lines of the work by Christensen and colleagues (2010, Chap. 6), we propose to functionally index not only motion actions but also cognitive actions like perception. One way of implementing this is by explicitly linking the *acquiring* of environmental knowledge with the ***Detect*** action. This way if the plan leads to a failure, we may be able to backtrack wrong object state estimation to ***Detect*** action and revise the assignment, especially under uncertainties. We expect the resulting system to better prepare us for localizing errors and conducting repairs of the right kind.

21.5.4 Toward Mission Repair

Since we are specifically interested in the interplay between action and perception, let us briefly consider a specific kind of mission failure relating to this. A mission formulation has information about which orderings are impossible but not about which orderings are easier or harder for an agent to do with its specific physical and skill-level abilities. Thus, a suboptimal ordering could make for harder to plan tasks for an agent. For example, consider building a house made of lego bricks. The agent could build the house ground-up and then assemble the roof on top of the walls or can assemble the roof separately and place it on top of the walls later. The former is significantly harder for a robot than the latter due to multiple layers of force dynamics involved (robot presses roof, roof presses walls, walls may or may not fall down over a long period of the same). Thus, we end this section with another interesting question: how may we extend reasoning formulations of robot problems to not only consider problem structure but also the feasibility with respect to the acting agent's abilities?

21.6 Discussion and Conclusions

There is a growing need for developing theoretical frameworks for human–robot shared contexts. There is also a growing interest in endowing robots with human-level, human-like capabilities to enhance human–robot collaboration; for example, meta-reasoning. However, much of previous AI research on meta-reasoning has focused on simulated robots, thereby abstracting away from the hard issues of action and perception. In this chapter, we have described a preliminary theoretical framework for grounding meta-reasoning in action and perception.

We considered the case of assembly robots that are preprogrammed for repetitively accomplishing routine tasks without failure. We examined scenarios in which an assembly robot may fail if given a new task that is even slightly different from the task for which it was preprogrammed. In some contexts, the cost of recovering from a failure may be lower than that of reprogramming the robot. However, in such scenarios, we want the robot to not only recover from the failure but also to learn from it. We described a robot architecture in which meta-reasoning helps the robot to localize and identify the cause for the failure and then to repair the knowledge that caused the failure. We found that our robot architecture for meta-reasoning grounded in action and perception in physical robots is significantly different than the idealized architectures proposed in earlier AI research. In summary, we expect a meta-reasoning system to have knowledge about the following concepts:

1. *Visuospatial representations and operations*: In order to ground concepts in their geometric instantiations, it is necessary to have a qualitative mid-level visuospatial representation that can present task-relevant features and operate on them without the need for precise locations and positions.

2. *Specification of functions of actions and tasks*: When a failure is localized and a better recovery state is obtained, or a heuristic toward one is ascertained, the agent would need some form of a state transition model to infer which actions can take it to the desired state.
3. *Explicit model of the effects of perception*: To revise perceptual beliefs of the agent, it is necessary to have an explicit model that links decisions made on those beliefs to the perception action (like ***Detect***) and the perceptual states that led to them. As far we know, such an analysis of using the perception process as a functional entity is missing from the meta-reasoning literature. We believe by modeling perception and its effects in a similar way to actions, for example, by using a transition function that updates the cognitive belief of an object state, we can build a meta-reasoning theory that interweaves action and perception as equals.

Acknowledgements We thank Aayush Naik and Jiaming Hu for their help in implementing the end-to-end assembly robot system described in Parashar et al. (2021). We would also thank the Research Products Development Company for their support in developing the robot system for the World Robot Summit 2020 that enabled this analysis.

References

Anderson, M. L., & Oates, T. (2007). A review of recent research in metareasoning and metalearning. *AI Magazine, 28*, 12–12.

Beetz, M., Mösenlechner, L., & Tenorth, M. (2010). CRAM—A cognitive robot abstract machine for everyday manipulation in human environments. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 1012–1017).

Bullard, K., Akgun, B., Chernova, S., & Thomaz, A. L. (2016). Grounding action parameters from demonstration. In: *2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)* (pp. 253–260).

Christensen, H. I., Kruijff, G.-J. M., & Wyatt, J. L. (2010). *Cognitive systems*. Springer Science & Business Media.

Colledanchise, M., & Ögren, P. (2018). *Behavior trees in robotics and AI: An introduction.*

Cox, M. T. (2005). Metacognition in computation: A selected research review. *Artificial Intelligence, 169*, 104–141.

Cox, M. T., & Raja, A. (2011). *Metareasoning: Thinking about thinking*. MIT Press.

Dannenhauer, D., & Muñoz-Avila, H. (2015). Goal-driven autonomy with semantically-annotated hierarchical cases E. In Hüllermeier & M. Minor (Eds.), *Case-Based reasoning research and development* (pp. 88–103). Springer International Publishing.

Dantam, N., & Stilman, M. (2013). The motion grammar: Analysis of a linguistic method for robot control. *Trans. Rob., 29*, 704–718. https://doi.org/10.1109/TRO.2013.2239553

De Mello, L. H., & Sanderson, A. C. (1989). A correct and complete algorithm for the generation of mechanical assembly sequences. In *1989 IEEE International Conference on Robotics and Automation* (pp. 56–57).

Firby, R. J. (1994). Task networks for controlling continuous processes. In *Proceedings of the Second International Conference on AI Planning Systems* (pp. 49–54).

Fitzgerald, T., Goel, A. K., & Thomaz, A. L. (2018). Human-Guided object mapping for task transfer. *ACM Transactions on Human-Robot Interaction (THRI), 7*, 1–24.

Fitzgerald, T., Short, E., Goel, A. K., & Thomaz, A. L. (2019). Human-guided trajectory adaptation for tool transfer. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems* (pp. 1350–1358).

Goel, A. K., Fitzgerald, T., & Parashar, P. (2020). Analogy and metareasoning: Cognitive strategies for robot learning. In *Human-Machine shared contexts* (pp. 23–44). Elsevier.

Goel, A. K., & Jones, J. K. (2011). Meta-Reasoning for self-adaptation in intelligent agents. In: *Metareasoning: Thinking about thinking* (p. 151). Cambridge, MA: MIT Press

Goel, A. K., & Rugaber, S. (2017). GAIA: A CAD-like environment for designing game-playing agents. *IEEE Intelligent Systems, 32*, 60–67.

Huckaby, J. O. (2014). *Knowledge transfer in robot manipulation tasks*. Georgia Institute of Technology.

IFR (International Federation of Robotics). (2020). *World robotics report*. Frankfurt.

Jia, Z., Bhatia, A., Aronson, R. M., Bourne, D., & Mason, M. T. (2018). A survey of automated threaded fastening. *IEEE Transactions on Automation Science and Engineering, 16*, 298–310.

Jones, J. K., & Goel, A. K. (2012). Perceptually grounded self-diagnosis and self-repair of domain knowledge. *Knowledge-Based Systems, 27*, 281–301. https://doi.org/10.1016/j.knosys.2011.09.012

Kaelbling, L. P., & Lozano-Pérez, T. (2011). Hierarchical task and motion planning in the now. In *2011 IEEE International Conference on Robotics and Automation* (pp. 1470–1477).

Kortenkamp, D., Simmons, R., & Brugali, D. (2016). Robotic systems architectures and programming. In *Springer handbook of robotics* (pp. 283–306). Springer.

Laursen, J. S., Schultz, U. P., & Ellekilde, L.-P. (2015). Automatic error recovery in robot assembly operations using reverse execution. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 1785–1792).

Lawless, W., Mittu, R., & Sofge, D. (2020). *Human-machine shared contexts*. Academic Press.

Liu, H.-C., Liu, L., & Liu, N. (2013). Risk evaluation approaches in failure mode and effects analysis: A literature review. *Expert Systems with Applications, 40*, 828–838.

Müller, A., Kirsch, A., & Beetz, M. (2007). Transformational planning for everyday activity. In *Proceedings of the Seventeenth International Conference on International Conference on Automated Planning and Scheduling, ICAPS'07* (pp. 248–255). Providence, Rhode Island, USA: AAAI Press.

Muñoz-Avila, H., Jaidee, U., Aha, D. W., & Carter, E. (2010). Goal-driven autonomy with case-based reasoning. In *International Conference on Case-Based Reasoning* (pp. 228–241).

Murdock, J. W., & Goel, A. K. (2008). Meta-case-based reasoning: Self-improvement through self-understanding. *Journal of Experimental & Theoretical Artificial Intelligence, 20*, 1–36.

Nau, D. S., Au, T.-C., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., & Yaman, F. (2003). SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research, 20*, 379–404.

Nau, D. S., Cao, Y., Lotem, A., & Muñoz-Avila, H. (1999). SHOP: Simple hierarchical ordered planner. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence* (Vol. 2, pp. 968–973).

Parashar, P., Goel, A. K., Sheneman, B., & Christensen, H. I. (2018). Towards life-long adaptive agents: using metareasoning for combining knowledge-based planning with situated learning. *The Knowledge Engineering Review, 33*, e24. https://doi.org/10.1017/S0269888918000279.

Parashar, P., Naik, A., Hu, J., & Christensen, H. I. (2021). Meta-Modeling of assembly contingencies and planning for repair. arXiv:2103.07544 [cs].

Parashar, P., Sanneman, L. M., Shah, J. A., & Christensen, H. I. (2019). A taxonomy for characterizing modes of interactions in goal-driven, human-robot teams. In IROS (pp. 2213–2220).

Pourazin, S., & Barforoush, A. A. (2006). Concurrent metareasoning. *The Journal of Supercomputing, 35*, 51–64.

Russell, S., & Wefald, E. (1991). *Principles of Metareasoning. Artificial Intelligence, 49*, 361–395.

Shivashankar, V., Kaipa, K. N., Nau, D. S., & Gupta, S. K. (2014). Towards integrating hierarchical goal networks and motion planners to support planning for human-robot teams. In *International Conference on Intelligent Robots and Systems* (pp. 1–3).

Stroulia, E., & Goel, A. K. (1999). Evaluating PSMs in evolutionary design: The A UTOGNOSTIC experiments. *International Journal of Human-Computer Studies, 51*, 825–847.

Stroulia, E., & Goel, A. K. (1995). Functional representation and reasoning for reflective systems. *Applied Artificial Intelligence, 9*, 101–124. https://doi.org/10.1080/08839519508945470

Tenorth, M., & Beetz, M. (2013). KnowRob: A knowledge processing infrastructure for cognition-enabled robots. *The International Journal of Robotics Research, 32*, 566–590. https://doi.org/10.1177/0278364913481635

Wolfe, J., Marthi, B., & Russell, S. (2010). Combined task and motion planning for mobile manipulation. In *Proceedings of the Twentieth International Conference on International Conference on Automated Planning and Scheduling, ICAPS'10*. (pp. 254–257). Toronto, ON, Canada: AAAI Press.

WRC (World Robotic Challenge). (2020). *Industrial robotics category assembly challenge—rules and regulations 2020*, Tokyo, Japan. Retrieved January 16, 2020.