Interactive formal specification for efficient preparation of intelligent automation systems

CIRP Journal of Manufacturing Science and Technology 38 (2022) 129–138

Contents lists available at ScienceDirect

CIRP Journal of Manufacturing Science and Technology

journal homepage: www.elsevier.com/locate/cirpj

Interactive formal specification for efficient preparation of intelligent automation systems☆

Martin Dahl[a,*], Christian Larsen[b], Endre Eros[a], Kristofer Bengtsson[a], Martin Fabian[a], Petter Falkman[a]

[a] Chalmers University of Technology, Department of Electrical Engineering, Division of Systems and Control, Gothenburg, Sweden
[b] Fraunhofer-Chalmers Centre, Gothenburg, Sweden

ARTICLE INFO

Available online xxxx

Keywords:
Virtual engineering
Virtual preparation
Artificial intelligence
Control architectures and programming
Factory automation
Collaborative robotics

ABSTRACT

The automation system of the future will consist of an increasing amount of complex resources, such as collaborative robots and/or autonomously roaming robots for material handling. To control these devices in an environment shared with human operators require state of the art computer perception and motion planning algorithms to be used as part of the automation system. This new type of intelligent automation system, where intelligent machines and learning algorithms are replacing more traditional automation solutions, requires new methods and workflows to keep up with the increase in complexity. This paper presents an interactive and iterative framework for solving some of these new challenges. The framework supports model-based control system preparation performed simultaneously to preparation of 3D geometries, positioning of robots, and tool design. The workflow enables an interactive preparation process, where new resources and constraints can be added to a live (real or simulated) automation system and control system failures can be analyzed in familiar tools for virtual preparation. Additionally, the paper describes how the integrated preparation process was applied to reconfiguring an industrial use case that includes a collaborative robot working side by side with a human operator, smart tools, and a vision system for localizing both work objects and tools.

Introduction

Current trends in industrial automation for automotive final assembly aim for *re-configurable* and *self-balancing* production systems that can handle rapid changes, unpredictable demands, production disturbances, product diversity, etc.

Collaborative robots [1] are often seen as one enabler to reach this flexibility, and many research initiatives in academia and industry have tried to introduce them for automation of final assembly tasks [2–4]. Despite their relative advantages, namely that they are sometimes cheaper and easier to program and teach [4] compared to conventional industrial robots, they are mostly deployed as robots "without fences" for co-active tasks [5].

Current collaborative robot installations are in most cases not as flexible, robust or scalable as required by many tasks in manual assembly. Combined with a lack of industrial preparation processes for these types of systems, new methods and technologies must be developed to better support the imminent industrial challenges [6].

To get the most out of collaborative robots, they need to be paired with additional capabilities to perceive their environment, in order to know where products and operators are located [7,8]. The resulting automation system needs to handle very dynamic environments, which leads to increased complexity also for the control system(s).

To tackle this, many have approached automated planning as a solution [9–11]. By using automated planning, the control system can itself take decisions on when to take certain actions in order to achieve a goal (for example, producing a product). This means that traditional preparation work, such as optimizing robot programs offline, can no longer be performed with the expectation that these programs will run uninterrupted. The control system may take other decisions due to external events. Careful preparation is still required though, as moving to a completely flexible control system based on

☆ This work has been supported by UNIFICATION, Vinnova, Produktion 2030 and UNICORN, Vinnova, Effektiva och uppkopplade transportsystem.
* Corresponding author.
E-mail addresses: martin.dahl@chalmers.se (M. Dahl), christian.larsen@fcc.chalmers.se (C. Larsen), endree@chalmers.se (E. Erős), kristofer.bengtsson@chalmers.se (K. Bengtsson), fabian@chalmers.se (M. Fabian), petter.falkman@chalmers.se (P. Falkman).

https://doi.org/10.1016/j.cirpj.2022.04.013
1755-5817/

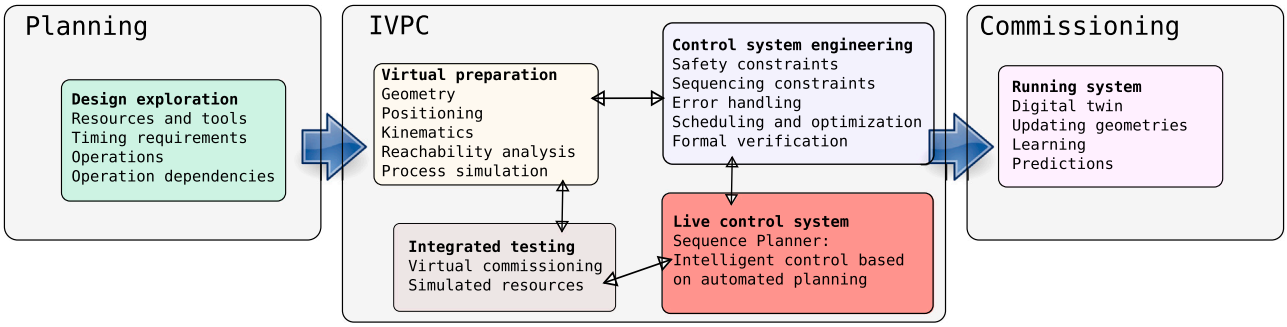

Fig. 1. Integrated virtual preparation and commissioning (IVPC).

automated planning may perform tasks in arbitrary order and in difficult to predict ways.

For preparation of traditional (i.e. fenced) industrial automation systems, virtual commissioning (VC) is commonly used as a validation tool. With virtual commissioning, an automation system's control system can be virtually wired up to a number of simulated resources, physics engines, and material simulators [12]. This allows for testing and validation to be performed in a virtual environment, which can increase the efficiency of the commissioning of a new or updated control system. These tests can also be formalized as model-based testing [13]. Traditionally, this process is done as a final validation step, but it has been argued that this process can be performed even before the control system is finished [14], especially when coupled with model-based control engineering [15]. In [15], a framework called Integrated virtual preparation and commissioning (IVPC) is introduced, where a model of the control system is used in order to generate sequences of operations that can be executed in a simulation environment that supports VC. This was extended to support a closed loop model-based control system in [16], where control was managed over an OPC UA connection. While an integrated approach for the *design* of a production station has been researched before (e.g. [17–19]), the aim of IVPC is to include also *control engineering* into an iterative approach to automation *system design*.

However, with tasks such as robot programming being replaced by online algorithms, performing VC in the traditional sense becomes also an integration challenge, due to all software involved. In [11,20], the model-based control system of [16] was extended to incorporate the Robot Operating System (ROS) as a middleware for facilitating these softwares. Coupled with a physics engine, for example Gazebo [21], ROS-based systems are also well suited for validation via simulation.

In Fig. 1, the IVPC activities are highlighted in the middle box. In this paper we look specifically at the combination of preparation work for robot motions with a control system based on automated planning. We apply a *model-based* control design combined with state of the art virtual preparation of robot motions, and virtually validate the result using simulation. The model-based control system design is supported by the open source software *Sequence Planner* [11] (SP). SP is a model-based framework for control of ROS-based (see Section 3.1) automation systems, which tries to handle complexity by formal reasoning. It uses online planning to automatically try to reach the currently active goals, which enable flexible error recovery [22]. Additionally, it relies on formal synthesis for safely composing reusable components [20]. This allows for constraint-based modeling approach, where an initially free system is constrained by formal specifications to only perform safe actions. These constraints can be changed *online*, which allows for flexibility as well as quick iterations during preparation. Combined with VC, the model-based control framework can be used to solve some of the engineering challenges in incorporating human-robot collaboration in final assembly. The intelligent control system based on SP becomes an enabler for IVPC in that the system can be created in an iterative fashion. Simulation and testing is performed continuously using simulated resources which are ROS components.

Contribution

When preparing the control logic for automation systems like the one in the use-case, where the resources, including the control system, are more or less autonomous, it is difficult to anticipate all different behaviors that can arise when the system tries to fulfill its current goals. The main contribution and focus of this paper is to demonstrate how to interactively, and iteratively, solve some of the engineering challenges in a virtual environment.

Applying the IVPC framework, we show how the combination of a state of the art software for virtual preparation of robot motion and product geometries with the SP control framework enables a truly interactive workflow for automation system preparation. In this framework, the virtual environment and the logic of the control system can be continuously updated and validated.

We describe an application where a collaborative robot, with the help of an operator, should perform various assembly tasks on a diesel engine. Neither the position of the robot, nor the autonomous guided vehicle carrying the engine (see Fig. 2), is static, which makes it difficult to apply traditional off-line programming of the robots. Additionally, the control system must be able to react properly to the actions performed by the operator, who is allowed to intervene freely.

To support multiple types of robots with dynamic work object positions, the Robot Optimization Module of the IPS [23] is used. The Robot Optimization Module of IPS contains algorithms for motion planning and sequence optimization, enabling fast commissioning of industrial robots. We have developed a simple integration of IPS into ROS, which makes it possible to compute high quality robot paths *online*. The online motion planner can be constrained during

Fig. 2. Experimental setup of collaborative robot assembly station.

preparation to disallow paths which are deemed not suitable (e.g. unnecessarily complicated paths).

Outline

Section 3, briefly introduces the notation and the workings of SP and IPS. Section 4.2, describes the proposed workflow. Then a case study that highlights the interactive nature of the work is presented in Section 4, with examples of details added to make the automation system work given in Section 5. The paper ends with some concluding remarks in Section 6.

Related work

Since the type of automation system presented in this paper is a concept of a *future* automation, the standard methodologies for preparation do not apply. Especially when introducing intelligent control that supports decision taking in an online fashion. In the robotics community, several approaches that could support a similar interactive preparation work flow exist. For example, the framework ROSPlan [24] that uses PDDL-based models for automated task planning and dispatching, SkiROS [25] that simplifies the planning with the use of a skill-based ontology, MaestROB [26] that adds natural language processing and machine learning to teach robots new skills that are executed using ontology based planning, eTaSL/eTC [27] that defines a constraint-based task specification language for both discrete and continuous control tasks, and CoSTAR [28] that uses Behavior Trees to define complex tasks, combined with a novel way of defining computer perception pipelines. Applications that use planning of robot skills have seen successful experimentation in industrial settings [9,10].

However, these systems have been mainly robot-oriented (in contrast to automation-oriented) and often focus on a single robot. On the production side, research on how to design collaborative stations have also been researched before (e.g. [17–19]), but this body of research does not generally take flexible planning from the control system into the perspective.

The IVPC framework built around SP and IPS provides a combination of both high-level robotic tasks with more “traditional automation” tasks – both low-level execution and state management of a variety of different devices and offline preparation of geometry and motion planning.

Preliminaries

This section briefly introduces the different tools and frameworks used throughout the work. Specifically it covers: using ROS as an integration layer, preparation and planning of robot motions in IPS, and finally the model based control framework SP.

ROS

In order to ease integration and development of different types of online algorithms for sensing, planning, and control of hardware, various platforms have emerged as middleware solutions. One that stands out is the Robot Operating System (ROS) [29]. In the current version of ROS (ROS2 [30]), the communication architecture is based on the Data Distribution Service (DDS) [31] to enable large scale distributed systems to be built on top of it. ROS2 systems are composed of a set of nodes communicating by sending typed messages over named topics using a publish/subscribe mechanism. The use of standardized message types enables a quick and semi-standardized way to introduce new drivers and algorithms to a system. For example, any robot that publishes its joint values in a certain way can use *MoveIt!* [32], which builds on the Open Motion Planning Library [33] that provides a multitude of state of the art motion planning algorithms. ROS also gives easy access to a large amount of *driver* software, which makes it easy to move from simulation to reality. Since all communication is based on publish/subscribe, it is possible to run systems where some nodes are simulations and some control real hardware.

IPS

IPS is a software tool that implements algorithms for virtual product realization [23]. It includes software modules for different applications ranging from cable simulation to robot optimization, etc. The IPS Robot Optimization Module, used in this paper, contains a toolbox of algorithms for automatic off-line programming of industrial robots. It allows virtual preparation by supporting kinematic modeling, virtual geometry preparation for large meshes and point clouds as well as algorithms for motion planning and robot motion sequence optimization.

IPS contains a deterministic motion planner for industrial robots that is inspired by probabilistic methods for motion planning such as Rapidly-Exploring Random Trees (RRT) [34] and Probabilistic Roadmap Methods (PRM) [35]. The robot motion sequence optimization determines the order in which a set of planning targets should be executed by the robot in a collision-free and cycle-time minimal manner. This is done by formulating the problem as a Generalized Traveling Salesman problem that is iteratively solved to minimize the cycle-time, similar to the method proposed in [36].

Sequence Planner

SP is a model-based framework for control of intelligent automation systems. To ease modeling and control, SP uses formal models together with online planning to reach the current *goal* of the automation system. It is an open source software and available on github [37].

SP is based on models of resources in terms of their state, with non-deterministic guarded actions (transitions) that can transition the system between these states. Transitions exist in three kinds: *controllable*, *automatic*, and *effect*. Controllable and automatic transitions are taken by SP, while effect transitions are used to model the environment. The resource models are composed and constrained by formal specifications which automatically remove undesired states. See Fig. 3 for an overview of how an automation system in SP is structured. SP is based on planning in two levels: first with the *operation planner* to determine which operations should be run, then with the *transition planner* to determine exactly how resources should perform the operations.

SP uses a state based control design, where resources continuously receive *goal states* from SP, and continuously update *measured states*, which are inputs to SP. For example, consider a simple indicator light, it may have a goal state $\in \{off, on\}$ and a measured state with the same domain. Contrary, a robot may have a complex goal state that includes a frame to reach in space, perhaps with additional active constraints such as speed or joint limits. SP models the resources using a formalism with finite domain variables, states that are unique valuations of the variables, and non-deterministic transitions between states. Some definitions to clarify follows.

Definition 1. A transition system (TS) is a tuple $\langle S, \rightarrow, I \rangle$, where S is a set of states, $\rightarrow \subseteq S \times S$ is the transition relation and $I \subseteq S$ is the nonempty set of initial states.

Definition 2. A state $s \in S$ is a unique valuation of each *variable* in the system. E.g. $s = \langle v_1, v_2, \ldots, v_n \rangle$.

Variables have finite discrete domains, i.e. Boolean or enumeration types.

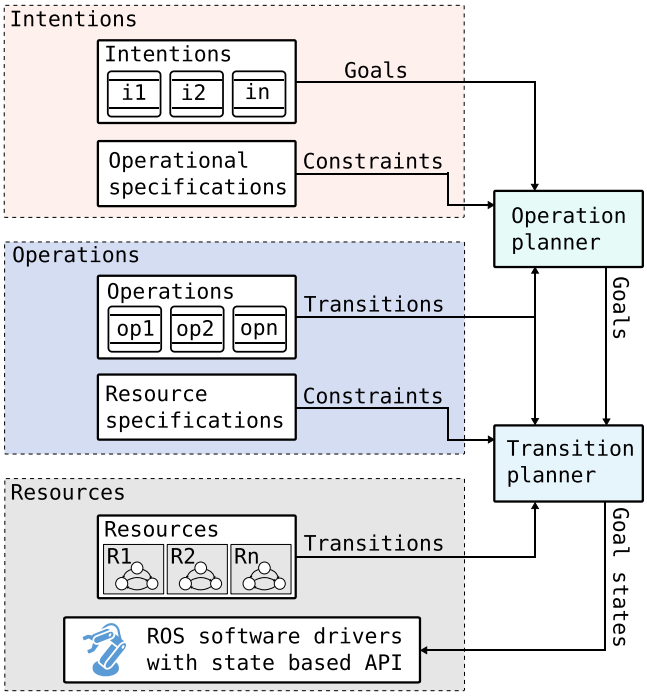

Fig. 3. The structure of a control system model in SP.

The transition relation → define the transitions that modify the system state:

Definition 3. A transition t has a guard g, which is a boolean function over a state, g: $S \rightarrow \{false, true\}$, and a set of action functions A where a: $S \rightarrow S$ which updates the valuations of the state variables in a state. We often write a transition as g/A to save space.

In SP, resources include models of the behavior of the resources bundled with ROS nodes as reusable components [20]. The behavior models are non-deterministic transition systems that the *operations* navigate using planning to reach their goals.

Definition 4. A resource i is defined as $r_i = \langle V_i^M, V_i^G, V_i^E, T_i^c, T_i^a, T_i^e \rangle$ where V_i^M is a set of *measured* state variables, V_i^G is a set of *goal* state variables, V_i^E is a set of *estimated* state variables. Variables are of finite domain. The set $V_i = V_i^M \cup V_i^G \cup V_i^E$ defines all state variables of a resource. The sets T_i^c and T_i^a define *controlled* and *automatic* transitions respectively. T_i^e is a set of *effect* transitions describing the possible consequences to V_i^M of being in certain states.

T_i^c, T_i^a, and T_i^e have the same formal semantics, but are separated due to their different uses: *Controlled transitions* T_i^c are taken when their guard condition evaluates to true, only if they are also activated by the planning system. *Automatic transitions* T_i^a are always taken when their guard condition evaluates to true, regardless of if there are any plans active or not. All automatic transitions are taken before any controlled transitions can be taken. This ensures that automatic transitions can never be delayed by the planner. *Effect transitions* T_i^e define how the *measured state* is updated, and as such they are not used during control like the control transitions T_i^c and T_i^a. They are important to keep track of however, as they are needed for online planning and formal verification algorithms. They are also used to know if the plan is correctly followed – if expected effects do not occur it can be due to an error.

As a clarifying example, consider a model of a *door* resource that has a sensor for measuring whether it is opened or closed ($c_?$ ∈ {*false*, *true*}, where $c_?$ being true means the door is closed), and an actuator for opening and closing the door ($c_!$ ∈ {*false*, *true*}, where $c_!$ being true activates the actuator that opens the door).

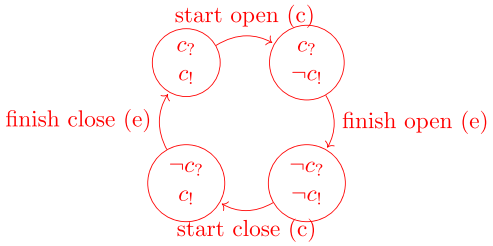

Fig. 4. Transition system modeling the example door resource.

The nodes in Fig. 4 represent the state combinations of the two variables $c_?$ and $c_!$ and the edges represent transitions that change the state. For example "start open (c)" is a controlled transition. Regardless of whether its guard is true ($c_? \wedge c_!$), it needs to be activated by the planning system in order to update the state. If the transition was automatic, SP would always (immediately) take the transition in this state. The two "finish" transitions are effect transitions. As such, they are only part of the formal model and not included when the system is being executed.

Given a set of *resources*, the system is initially allowed to take any actions. *Resource specifications*, are defined as invariants over the system state, and constrain the system to avoid unsafe regions during execution. The negation of invariant formulas (e.g. the forbidden states) are extended into larger sets of states using symbolic backwards reachability analysis to properly deal with the uncontrollability of automatic and effect transitions. The method used is described in [38].

In addition to the variables defined by the resources, another set of variables exist, *decision variables*. These define the state of the system in abstract terms to plan which operations to execute. For example, a decision variable could be the abstract state of a particular resource, or the state of a product in the system. Sometimes decision variables can be directly measured by resources, in which case these measurements are *copied* into the decision variables, usually after undergoing some form of transformation (for example discretization).

Definition 5. An operation j is defined as $o_j = \langle p_j, e_j, g_j, a_j, s_j \rangle$, where p_j is a precondition over the decision variables defining when the operation can start, a set of effect actions e_j of completing the operation, which are actions defined on the decision variables, as well as a goal predicate g_j defined over the resource variables. a_j is a set of actions for synchronizing the operation with the resource state. Finally, the operation has an associated state variable s_j ∈ {*i*, *e*, *error*}. Throughout the paper, operations are graphically depicted as in Fig. 5.

When the precondition of an operation is satisfied, the operation can start. The effect actions are then evaluated against the current state, and the difference between the current state and the next state is converted into a predicate. This predicate becomes the post-condition of the operation. E.g. if the effect of the operation is $x := y$, and $y = 5$ when the operation starts, then the post-condition (and thus its planning goal) becomes $x = 5$. If the operation needs to update the resource state, a_j can additionally include actions from a transition

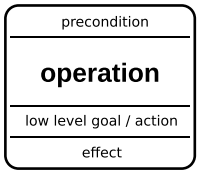

Fig. 5. We use this graphical notation to visualize operations. For the operation j, the precondition is p_j, the effect is e_j, the goal is g_j and the set of actions is a_j.

among the resources, in which case g_j is also conjuncted with the guard of this transition.

Definition 6. The *intention* k is defined as $i_k = \langle p_k, g_k, \phi_k, a_k, s_k \rangle$, where p_k is a predicate over (all) the variables in the system, defining when the intention starts (automatically), g_k is a goal predicate defined over the decision variables, ϕ_k is an optional LTL formula over the decision variables, a_k are a set of actions that can update (all) variables in the system, that are applied when the intention finishes, and $s_k \in \{i, e, f\}$ is the state of the intention.

The goals defined by the intentions over the decision variables is the way the system is driven forward. The decision variables are meant to be allowed to be changed *at any time* from the outside. It can be that they can be changed to a high level state from which the goal cannot be reached, in which case *replanning* occurs automatically. Not only does the planner allow SP to be agnostic about the current resource state, it also allows for interrupting or canceling currently running operations in a safe way – by simply changing the goal state, the planning system will find the correct way to instead reach the new goal.

In model checking [39], temporal properties are verified by means of state space-exploration based on a set of initial states and a set of transitions. The temporal properties are specified in extensions to propositional logic such as Computation Tree Logic (CTL) or Linear Temporal Logic (LTL) [40]. For example, LTL has temporal operators for expressing properties on the next state ($\bigcirc$), that some formula should *always* ($\Box$) hold, that it should *eventually* hold ($\Diamond$), and that one formula should hold until another one does (U). For example, the formula $\Box(x \rightarrow \bigcirc y)$ expresses that it is always the case that x implies y in the next state. Then an LTL model checking problem is to prove that given a set of valid initial states, this formula always holds, or if this cannot be proven produce a counterexample.

By turning the problem around, and having the model checking proving that a desired future state is *not* reachable, one can use the counterexample as a *plan*, which, if followed, will reach that particular state. In contrast to a more simplistic forward search, using model checking allows SP to restrict the plans by providing additional temporal specifications that need to hold.

SP continuously tries to find the shortest path that reaches the goal of all currently executing operations. For this nuXmv [41], a well known off the shelf model checker, is used. nuXmv supports *bounded* model checking (BMC) [42]. In BMC, the model checking problem is reduced to a boolean satisfiability problem with a bounded length in the number of discrete "timesteps" from the initial states. One advantage of BMC in this setting is that it produces counterexamples of minimal length [42], i.e. the plans will never be longer than necessary.

Case study

The application used as a case study in this paper is the result of transforming an existing manual assembly station from a truck engine final assembly line, shown in Fig. 6, into an intelligent and collaborative robot assembly station, shown in Fig. 2.

Description of assembly station

Diesel engines are transported to the assembly station on an Automated Guided Vehicle (AGV). Material to be mounted on a specific engine is loaded by an operator from kitting facades located adjacent to the production line. An autonomous mobile platform (MiR100) carries the kitted material to be mounted on the engine to the collaborative robot assembly station.

In the station, a robot and an operator work together to mount parts on the engine by using different tools suspended from the ceiling. A dedicated camera system keeps track of operators, ensuring safe coexistence with machines. The camera system is also used for gesture recognition.

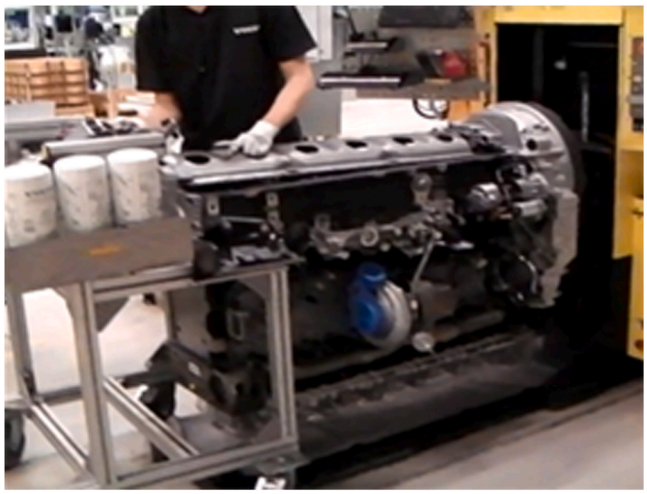

Fig. 6. The original manual assembly station.

After the MiR100 has arrived with the kitting material, a Universal Robots (UR10) robot and the operator collaborate to lift a heavy ladder frame on to the engine. After placing the ladder frame on the engine, the operator informs the control system with a button press on a smartwatch or with a gesture, after which the UR10 switches tools; the lifting end-effector is replaced with a nutrunner for tightening bolts. During this tool change, the operator starts to insert 24 bolts that the UR10 will tighten.

During the tightening of the bolts, the operator can mount three oil filters. If the robot finishes the tightening operation first, it leaves the nutrunner in a floating position above the engine and waits for the operator. When the operator is finished mounting the oil filters, the robot attaches a new end-effector for oil filter tightening. During that time, the operator attaches two oil transport pipes on the engine, and uses the same nutrunner previously used by the robot to tighten plates that hold the pipes to the engine. After executing these operations, the AGV with the assembled engine, and the empty MiR100 both leave the collaborative robot assembly station.

Application of IVPC to the case study

Fig. 7 highlights how the IVPC framework was used to interactively work on various aspects of the automation system for the application described in the previous section.

The top left box corresponds to IPS in its normal off-line commissioning usage. This includes positioning geometries, targets and tool center points as well as iterating the available off-line robot optimization algorithms. The SP model box corresponds to writing constraints for the model based control system, for example using input from the off-line preparation done in IPS. The SP model can be subjected to formal verification via model checking, as shown in the bottom box, "formal analysis". Formal verification is used to prove that the system operates according to specifications. But the SP model describe only the discrete behavior of the system, and this does not capture nuances w.r.t. dynamics and robot motions. Additionally, because it is not always trivial to anticipate how certain specifications affect the system, there is also a need to *validate* that the specifications are correct [15]. This validation can be performed by means of simulation, using SP in its execution mode (i.e. when driving the control system). SP controls a number of simulated ROS resources for validation via simulation. The top right box represents the IPS online motion planner, which is used by the ROS resource for

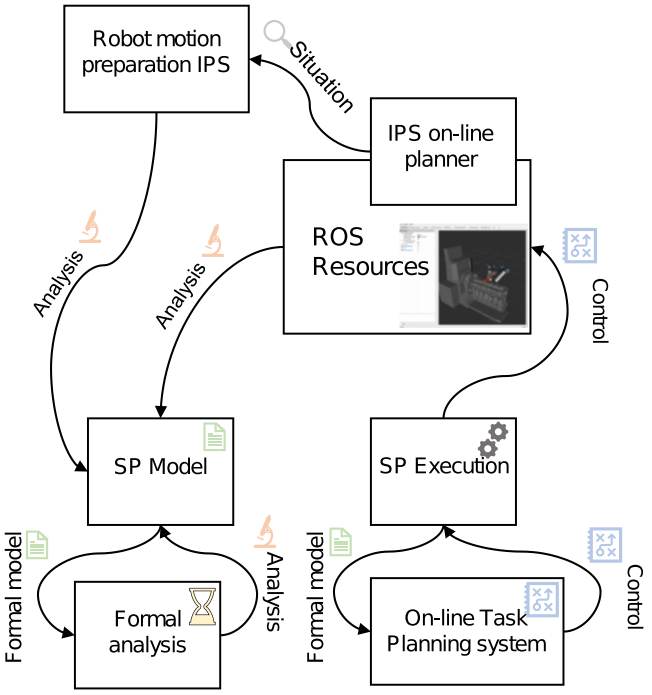

Fig. 7. Schematic of the workflow in the proposed architecture.

the robot. In the case of a robot motion planning failure, the current situation can be inspected with IPS used in its normal off-line mode. Section 5 will show a concrete examples of the arrows marked with "analysis" as a case study, by highlighting specific parts of the implemented automation system. The point of the case study is to show how to interactively incorporate new information and iterating the design based on what is learned from simulation.

Technical setup

For the implementation described in the remainder of the paper, three standard consumer PC:s connected over wired Ethernet was used. IPS and SP were each run on a separate computer, as was the simulated ROS nodes and software for visualizing the state of the simulation. Communication between the systems is handled entirely over ROS. Real-time aspects are not taken into consideration.

Preparation of the control system

To give an understanding of how the model-based control system works, this section starts by defining the resources and states used for control of a subset of the assembly procedures, specifically placing and tightening the bolts, which can be done both by the operator and the robot, locating the position of the engine and the tool, as well as performing tool change.

The resources are all controlled by individual ROS nodes, which continuously receive *goal states* from SP. Resources include the robots, the IPS motion planner, the smart tool, and the camera used for localization. In this paper we do not describe the autonomous kitting robot, the safety of the system, nor the mounting of the pipes and oil-filters.

The building blocks of the model-based control system are intentions and operations. At a high level, we want to express that eventually, all the bolts should be tightened. This is the intention *tighten all bolts*. To achieve the goal that all bolts should be tightened, operations that control the resources are needed. The intention and the operations used are depicted in Fig. 8.

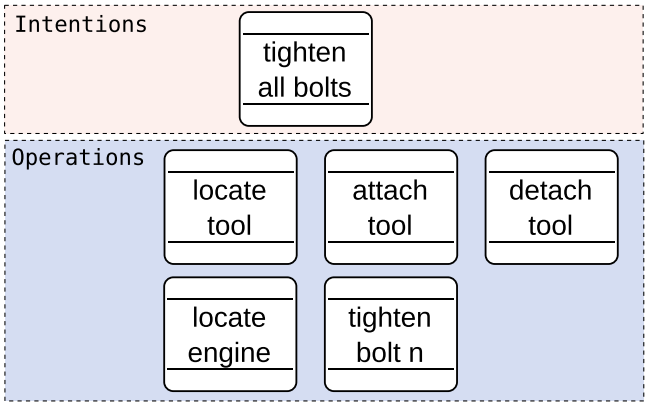

Fig. 8. The intention tighten all bolts together with the operations available to the system.

The next sections (Sections 5.1–5.5) relate to the initial development of the SP model in Fig. 7. In these steps, the basic logic behavior of how the resources interact with each other and the products are defined. In Sections 5.6.1–5.8, the upper parts of Fig. 7 are exemplified. These examples highlight how the interactive nature with IPS in the loop is beneficial when developing the formal model.

Resources

The operations define the low-level actions that need to be taken by the resources in the system in terms of goal states. Often operations use more than one resource. Consider for example the "locate" operations, which use the robot resource and the 3d camera resource, or the tighten bolt operations, which use both the robot resource and the smart tool resource. Fig. 9 depicts how these operations interact with the different resources. The green color highlights what is currently executing in each layer.

Decision variables

Because control in the framework is based on moving from the current state to a particular goal state, some key state variables need to be defined. The engine, the ladder frame, the bolts, and the pipes, are considered the *products* of the system. Other important states are whether the positions of the engine and the tools are known or not. This information is modeled as *decision variables*. See Table 1 for their definitions.

The decision variables represent the high-level state of the system, which is available to the intentions and operations.

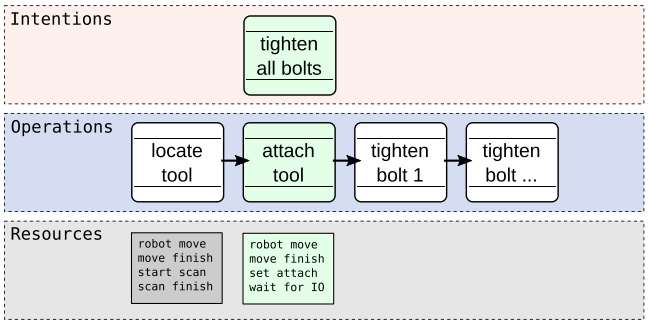

Fig. 9. The intention tighten all bolts is executing, which results in a sequence of operations that each trigger a number of resource transitions in order to reach the goal. The green color highlights what is currently executing in each layer.

Table 1
Decision variables in the example. $n = 1, \ldots, 24$.

Variable	Domain
`b_n`	{*not_placed*, *placed*, *untightened*, *tightened*}
`engine_present`	{*false*, *true*}
`engine_scanned`	{*false*, *true*}
`tool_scanned`	{*false*, *true*}
`tool_attached`	{*false*, *true*}

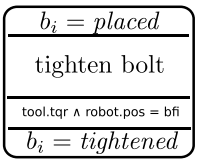

Fig. 10. Operation defining the tightening of bolt *i*. The low-level goal is that the tool should have registered torque reached (`tool.tqr`) with the robot at bolt position *i* (bf_i).

Operations

The operations are the glue between the resources in the system and the product state. Consider for example a bolting operation as shown in Fig 10. This operation has as its goal that the tool should have registered that the correct torque has been reached (`tool.tqr`) with the robot at bolt position *i* (bf_i): *tool.tqr* ∧ *robot.pos* = bf_i. The result of reaching this state is b_i := *tightened*.

The other operations in Fig. 8 are defined in a similar way. For example, for the 3d camera, the goal state is simply *camera.scan* = `done`. Without additional constraint, this allows the robot be in any location when performing the 3d scanning, which could result in performing the scanning in the wrong locations. Specifications that prevent this are added in Section 5.4.

Constraining the resources

To ensure that the planner produces the correct outcomes, the resources in the automation system can be constrained either by providing invariant propositions, or conjuncting the transitions resources state machines with additional guard expressions. For example, the invariant

$$\bigwedge_{1 \le i \le 24} ap = bf_i \;\Rightarrow\; sa$$

where *ap* is the position last visited by the robot, bf_i, is the target frame of the robot above bolt *i*, and *sa* is a boolean variable indicating that the spinner tool is attached to the robot. This effectively forbids the robot from moving to above the bolts without holding the spinner tool.

For the 3d camera, a similar constraint could be written:

$$(camera.\,scan \wedge (camera.\,object = \text{tool})) \Rightarrow (ap = \text{tool}_{\text{scanposition}} \wedge rs)$$

where *ap* is the position last visited by the robot, *rs* is a predicate for "robot still", "camera.scan" is an I/O that starts the scanning procedure and "camera.object" defines which object to perform matching against. The implication ensures that this state cannot be reached unless the robot is in the correct location and is not moving.

Intentions

The intentions define goals over the decision variables. In this example, the goal is simple: all bolts should be in the tightened state.

Table 2 defines the intentions in the system. These define the different modes of operation, for example tightening the bolts. Section 5.6 will make clear why bolting is separated into two different intentions, and what ϕ_c and ϕ_r are.

Constraining operations

The operations chosen for execution depend on the currently active intentions, combined with the currently active specifications. Some specifications are known before the preparation work has started. Usually these specifications relate to the production process. We call these specifications *product specifications*. For example, the ladder frame has a requirement on the order in which the bolts are tightened. Other constraints on the operations relate to more practical aspects. For example, in order to even start tightening the bolts, the system must be confident in its location measurement for the engine. I.e. `engine_scanned` must be true. This can be expressed as a simple precondition on the bolting operations. Having the planner in the loop means that a bolting operation could potentially be aborted if `engine_scanned` suddenly becomes false.

We will exemplify implementing the product specifications using the bolts and the ladder frame.

Sequence specification

The product specification is that the bolts at the corners of the ladder frame should be tightened before the other ones. To simplify the formulas below, the corner bolts have been given indices 1–4, see Fig. 11. This is a hard requirement that should lead to an error if the operator does it in any other way. As such, it is naturally given as an invariant proposition over the variables representing the state of the bolts.

$$\bigwedge_{2 \le i \le 4} \left[b_i = tightened \;\Rightarrow\; \bigwedge_{1 \le j \le i-1} b_j = tightened \right] \tag{1}$$

For the intention "tighten corner bolts", ϕ_c in Table 2 is the constraint in (1).

If there is no reason for the other bolts to be tightened in a specific order, there should not be a product specification that restricts the system unnecessarily. Such design choices can then be done at the later stages of preparation, where more details are known about the specific resources in the system – perhaps there are *physical* constraints that in practice produce a certain ordering anyway.

Now, a suitable order for visiting the remaining bolts by the robot could be prepared offline using IPS. This ordering can take into account optimization aspects such as execution time, but it could also take into account, for example, where the operator is expected to be working during assembly, to reduce interference. This is represented by the top left "analysis" arrow in Fig. 7. Since the control system is

Table 2
Intentions describe the main production activities.

Intention	Precondition	Postcondition	Spec
Tighten corner bolts	$\bigwedge_{i=1}^{4} bolt_i \neq tigthened$	$\bigwedge_{i=1}^{4} bolt_i = tightened$	ϕ_c
Tighten remaining bolts	$\bigwedge_{i=1}^{4} bolt_i = tigthened \wedge \bigwedge_{i=5}^{24} bolt_i \neq tigthened$	$\bigwedge_{i=5}^{24} bolt_i = tightened$	ϕ_r

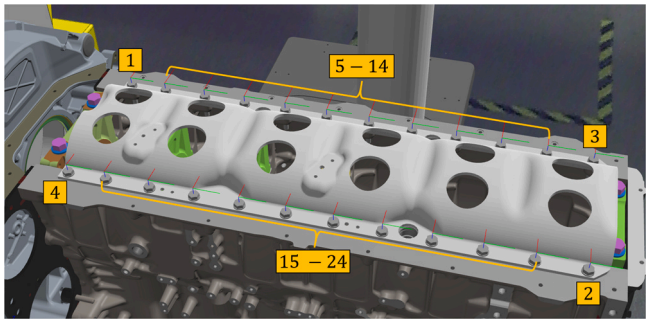

Fig. 11. Indices and locations of the bolts.

always available for simulation (bottom right part of Fig. 7), the ordering suggested by IPS can be applied *online* to try different scenarios in a simulated setting, highlighted by the middle "analysis" arrow in Fig. 7.

Suppose for simplicity that the order in which to tighten the remaining bolts should simply be to run them in the order of their indices (i.e. 5–24). This is a soft requirement; the bolts should be tightened in order when possible, but we allow bolts to be tightened out of order. This means that it should not be an error for the operator to go ahead and tighten a few bolts given the opportunity. Likewise, it should not be an error to *restart* production with a few bolts arbitrarily tightened. So this sequence constraint cannot be expressed as in (1).

Priority sequence specification

For this we introduce a slightly different type of sequence specification, the priority sequence. Since the actions of the operations are considered atomic during planning, we can instead ensure that the transition between untightened and tightened can only happen when the previous bolt in the priority sequence has been tightened. This allows freedom for the $bolt_i$ (for $i > 4$) state variables to be in the tightened state regardless of the states of the bolts with lower indices, but tightening bolts with an index higher than the lowest bolt that remains untightened is not allowed.

As the planning system supports arbitrary LTL expressions, we can express this transitioning constraint using the next operator as in (2)

$$\bigwedge_{6\le i\le 24}\left[b_i = untightened \wedge \bigcirc b_i = tightened \Rightarrow \bigwedge_{5\le j\le i-1} b_j = tightened\right] \tag{2}$$

For the intention "tighten remaining bolts", ϕ_r in Table 2 is set to (2). These two intentions can then be used to handle all bolting as done by the robot, while still allowing freedom for the operator to tighten bolts.

Interactively updating specifications

Consider the case of the bolting again. With the help of virtual validation it is possible to investigate how the specification works in different circumstances. This highlights the top left "analysis" arrow in Fig. 7, where a specific situation is analyzed in IPS and the SP model is updated accordingly.

In Fig. 12, a (virtual) pipe has been placed in such a way that bolting is not possible for the robot. When the intention in Fig. 12a is active, a planning request is made for the next bolt in the given bolting sequence. Since this bolt is not reachable (Fig. 12b), the *ur10* resource goes into an error state (*ur10.state = planningError*). When this happens, suppose the robot should just skip the unreachable bolts. The tighten bolt operation is extended with an additional goal state: `ur10.state = planningError/ bolt := skipped` and the domain of the decision variable is extended to include `skipped`. Similarly the effect of the intention can be changed to b_i *=tigthened* $\vee$ b_i *=skipped* to instead indicate that the bolts should all have been processed (but some may have failed).

The effects of changing the operation and the domain of the bolt variables to also include "skipped" results in a change to the generated planning problem. This means a user can run into this problem, change the desired operation and then continue executing with the new underlying model.

With the updated operations, it is possible to add another intention that can be run later in the process, which can tighten any remaining bolts. This sequence could be designed to be performed either by the operator or the robot.

Handling a late change request

Assume that late in the process it is decided that a different type of spinner tool also needs to be supported. Because of the reliance on robot motion planning, it is very easy to replace the tool as no additional robot programming needs to be performed.

The new tool is depicted in Fig. 13b. With it, the system can successfully tighten the bolts. However, when executing the locate engine operation we get the *planning error* from the robot resource again. Upon inspection, see Fig. 13b, it turns out that the tool would collide with the engine upon moving to the "scan engine" location.

This problem can be fixed, without stopping the simulation, by simply updating the scan engine operation to include an additional precondition on the attachment state of the tool. When this is done and the error state cleared from the intention, the system will automatically leave the tool, scan the engine and pick the tool up again, before resuming its current task. This corresponds to the middle "analysis" arrow in Fig. 7. Alternatively, an updated target frame for the scanning position could be prepared in IPS. It is a routine task to check that the position is reachable with both types of tools.

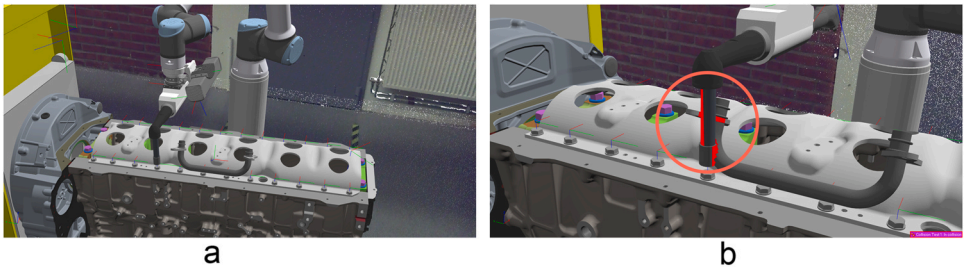

Fig. 12. Testing robot motion planning with an obstacle.

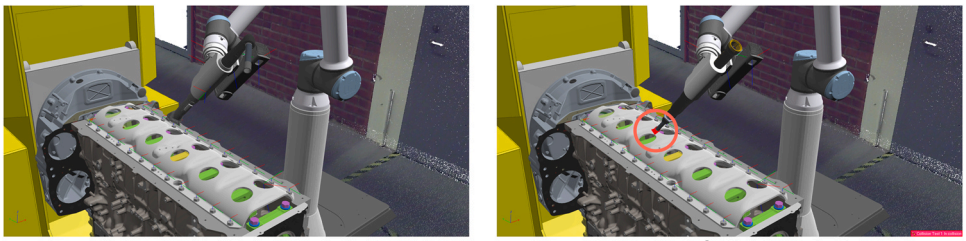

Fig. 13. Virtual validation after changing tool type.

Conclusion

This paper has shown how an automation system can be efficiently prepared using an "online" approach to formal modeling, allowing for declarative high level constraints to be written and tested on a live simulation. The prerequisites include access to simulations of the resources in the system (provided by ROS), access to an online motion planner to eliminate robot programming (provided by IPS), and a control system based on constraints and planning (provided by SP). Perhaps the main bottleneck is that formal models of the resources behaviors is required in order to use SP for safe control – something which is commonly not readily available today. However, the formal models describing the resources can be reused. This is especially true given that the composition of resources is done using specification rather than specific control policies implemented on a per-resource basis. This shifts the role of the automation engineer from implementing control policies in computer code to writing and validating specifications. While writing correct specifications is not easy, their correctness can be formally proven, as well as interactively validated by running a simulation of the system. This allows for an efficient preparation process involving iterative verification and validation procedure. Additionally, by employing automated planning as the driver of the system, flexibility that would be very difficult to implement using traditional programming can be achieved. It should be stressed that no programming is needed in the approach. This makes it is difficult to compare to traditional methods, as a fair comparison would have to involve formal verification of the developed automation software. The reliance on formal methods does pose an upper limit on the number of resources that can be included, but in practice we have seen that the framework can be applied to most "single station" sized systems before needing to break it up into subsystems. The workflow presented illustrates the latest iteration of the framework conceptualized in [15]. In addition to the case described in this paper, it has been applied to a bin-picking system at a AB Volvo facility in Sweden.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Bauer, A., Wollherr, D., Buss, M., 2008, Human-robot Collaboration: A Survey. International Journal of Humanoid Robotics, 05/01: 47–66. https://doi.org/10.1142/S0219843608001303.
[2] Tsarouchi, P., Matthaiakis, A.-S., Makris, S., Chryssolouris, G., 2017, On a Human-robot Collaboration in an Assembly Cell. International Journal of Computer Integrated Manufacturing, 30/6: 580–589.
[3] Fast-Berglund, Å., Palmkvist, F., Nyqvist, P., Ekered, S., Åkerman, M., 2016, Evaluating Cobots for Final Assembly. Procedia CIRP, 44:175–180.
[4] Villani, V., Pini, F., Leali, F., Secchi, C., 2018, Survey on Human-robot Collaborationin Industrial Settings: Safety, Intuitive Interfaces and Applications. Mechatronics, 55:248–266.
[5] He, W., Li, Z., Chen, C.L.P., 2017, A Survey of Human-centered Intelligent Robots: Issues and Challenges. IEEE/CAA Journal of Automatica Sinica, 4/4: 602–609.
[6] Hanna, A., Bengtsson, K., Dahl, M., Erős, E., Götvall, P., Ekström, M., 2019, Industrial Challenges When Planning and Preparing Collaborative and Intelligent Automation Systems for Final Assembly Stations. 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 400–406. https://doi.org/10.1109/ETFA.2019.8869014.
[7] Alterovitz, R., Koenig, S., Likhachev, M., 2016, Robot Planning in the Real World: Research Challenges and Opportunities. AI Magazine, 37/2: 76–84.
[8] Perez, L., Rodriguez, E., Rodriguez, N., Usamentiaga, R., Garcia, D.F., 2016, Robot Guidance Using Machinevision Techniques in Industrial Environments: A Comparative Review. Sensors, 16/3. https://doi.org/10.3390/s16030335⟨http://www.mdpi.com/1424-8220/16/3/335⟩.
[9] Schou, C., Andersen, R.S., Chrysostomou, D., Bøgh, S., Madsen, O., 2018, Skill-based Instruction of Collaborative Robots in Industrial Settings. Robotics and Computer-Integrated Manufacturing, 53:72–80.
[10] Krueger, V., Rovida, F., Grossmann, B., Petrick, R., Crosby, M., Charzoule, A., Garcia, G.M., Behnke, S., Toscano, C., Veiga, G., 2019, Testing the Vertical and Cyber-physical Integration of Cognitive Robots in Manufacturing. Robotics and Computer-Integrated Manufacturing, 57:213–229.
[11] Erös, E., Dahl, M., Hanna, A., Götvall, P.-L., Falkman, P., Bengtsson, K., 2020, Development of an Industry 4.0 Demonstrator Using Sequence Planner and ros2. in: Robot Operating System (ROS), Springer, pp. 3–29.
[12] Lee, C.G., Park, S.C., 2014, Survey on the Virtual Commissioning of Manufacturing Systems. Journal of Computational Design and Engineering, 1/3: 213–222.
[13] Khan, A., Falkman, P., Fabian, M., 2019, Testing and Validation of Safety Logic in the Virtual Environment. CIRP Journal of Manufacturing Science and Technology, 26:1–9. https://doi.org/10.1016/j.cirpj.2019.07.002⟨http://www.sciencedirect.com/science/article/pii/S1755581719300318⟩.
[14] Oppelt, M., Urbas, L., 2014, Integrated Virtual Commissioning an Essential Activity in the Automation Engineering Process from Virtual Commissioning to Simulation Supported Engineering. IECON 2014 - 40th Annual Conference of the IEEE Industrial Electronics Society. IEEE: 2564–2570.
[15] Dahl, M., Bengtsson, K., BergagÅrd, P., Fabian, M., Falkman, P., 2016, Integrated Virtual Preparation and Commissioning: Supporting Formal Methods During Automation Systems Development. IFAC-PapersOnLine, 49/12: 1939–1944.
[16] Dahl, M., Bengtsson, K., Fabian, M., Falkman, P., 2017, Automatic Modeling and Simulation of Robot Program Behavior in Integrated Virtual Preparation and Commissioning. Procedia Manufacturing, 11:284–291.
[17] Michalos, G., Spiliotopoulos, J., Makris, S., Chryssolouris, G., 2018, A Method for Planning Human Robot Shared Tasks. CIRP Journal of Manufacturing Science and Technology, 22:76–90. https://doi.org/10.1016/j.cirpj.2018.05.003⟨http://www.sciencedirect.com/science/article/pii/S1755581718300300⟩.
[18] Hagemann, S., Stark, R., 2020, An Optimal Algorithm for the Robotic Assembly System Design Problem: An Industrial Case Study. CIRP Journal of Manufacturing Science and Technology, 31:500–513. https://doi.org/10.1016/j.cirpj.2020.08.002⟨http://www.sciencedirect.com/science/article/pii/S1755581720300894⟩.
[19] Papakostas, N., Alexopoulos, K., Kopanakis, A., 2011, Integrating Digital Manufacturing and Simulation Tools in the Assembly Design Process: A Cooperating Robots Cell Case. CIRP Journal of Manufacturing Science and Technology, 4/1: 96–100. https://doi.org/10.1016/j.cirpj.2011.06.016. (special Section on Innovative and Cognitive Manufacturing Engineering). ⟨http://www.sciencedirect.com/science/article/pii/S1755581711000678⟩.
[20] Dahl, M., Erős, E., Hanna, A., Bengtsson, K., Fabian, M., Falkman, P., 2019, Control Components for Collaborative and Intelligent Automation Systems. 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 378–384. https://doi.org/10.1109/ETFA.2019.8869112.
[21] Koenig, N., Howard, A., 2004, Design and Use Paradigms for Gazebo, An Open-source Multi-robot Simulator. IEEE/RSJ International Conference on Intelligent Robots and Systems, Sendai, Japan, 2149–2154.
[22] Dahl, M., Bengtsson, K., Falkman, P., 2021, Application of the Sequence Planner Control Framework to an Intelligent Automation System with a Focus on Error Handling. Machines, 9/3: 59.
[23] Industrial Path Solutions. ⟨https://industrialpathsolutions.com/⟩. [Accessed 10 November 2020] (2020).
[24] Cashmore, M., Fox, M., Long, D., Magazzeni, D., Ridder, B., Carreraa, A., Palomeras, N., Hurtós, N., Carrerasa, M., 2015, Rosplan: Planning in the Robot Operating System. in: Proceedings of the Twenty-Fifth International Conference on International Conference on Automated Planning and Scheduling, ICAPS'15. AAAI Press: 333–341.

[25] Rovida, F., Crosby, M., Holz, D., Polydoros, A.S., Großmann, B., Petrick, R.P.A., Krüger, V., 2017, SkiROS–A Skill-Based Robot Control Platform on Top of ROS. Springer International Publishing, Cham: 121–160. https://doi.org/10.1007/978-3-319-54927-9_4.
[26] Munawar, A., De Magistris, G., Pham, T., Kimura, D., Tatsubori, M., Moriyama, T., Tachibana, R., Booch, G., 2018, Maestrob: A Robotics Framework for Integrated Orchestration of Low-level Control and High-level Reasoning. 2018 IEEE International Conference on Robotics and Automation (ICRA), 527–534. https://doi.org/10.1109/ICRA.2018.8462870.
[27] Aertbeliën, E., De Schutter, J., 2014, etasl/etc: A Constraint-based Task Specification Language and Robot Controller Using Expression Graphs. 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, 1540–1546. https://doi.org/10.1109/IROS.2014.6942760.
[28] Paxton, C., Hundt, A., Jonathan, F., Guerin, K., Hager, G.D., 2017, Costar: Instructing Collaborative Robots with Behavior Trees and Vision. 2017 IEEE International Conference on Robotics and Automation (ICRA), 564–571. https://doi.org/10.1109/ICRA.2017.7989070.
[29] Quigley, M., Faust, J., Foote, T., Leibs, J., 2009, Ros: An Open-source Robot Operating System. ICRA Workshop on Open Source Software, 3/2.
[30] ROS 2. ⟨https://index.ros.org/doc/ros2/⟩. [Accessed 20 November 2020] (2020).
[31] Pardo-Castellote, G., 2003, Omg Data-distribution Service: Architectural Overview. 23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings, 200–206. https://doi.org/10.1109/ICDCSW.2003.1203555.
[32] Şucan, I.A., Chitta, S., 2018, MoveIt!. ⟨http://moveit.ros.org⟩. [Accessed 26 February 2019].
[33] Şucan, I.A., Moll, M., Kavraki, L.E., 2012, The Open Motion Planning Library. IEEE Robotics & Automation Magazine, 19/4: 72–82. https://doi.org/10.1109/MRA.2012.2205651⟨http://ompl.kavrakilab.org⟩.
[34] LaValle, S.M., Kuffner Jr, J.J., 2001, Randomized Kinodynamic Planning. The International Journal of Robotics Research, 20/5: 378–400.
[35] Bohlin, R., Kavraki, L.E., 2000, Path Planning Using Lazy prm. in: Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065), 1, IEEE, pp. 521–528.
[36] Spensieri, D., Carlson, J.S., Ekstedt, F., Bohlin, R., 2015, An Iterative Approach for Collision Free Routing and Scheduling in Multirobot Stations. IEEE Transactions on Automation Science and Engineering, 13/2: 950–962.
[37] Sequence Planner. ⟨https://github.com/sequenceplanner/sp-rust⟩. [Accessed 8 January 2021] (2021).
[38] Dahl, M., Bengtsson, K., Fabian, M., Falkman, P., 2020, Guard Extraction for Modeling and Control of a Collaborative Assembly Station. IFAC Workshop on Discrete Event Systems, WODES, Nov. 2020.
[39] Grumberg, O., Clarke, E., Peled, D., 1999, Model Checking.
[40] Pnueli, A., 1977, The Temporal Logic of Programs. 18th Annual Symposium on Foundations of Computer Science (sfcs 1977). IEEE: 46–57.
[41] Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S., 2014, The nuXmv Symbolic Model Checker. CAV, 334–342.
[42] Biere, A., Cimatti, A., Clarke, E., Zhu, Y., 1999, Symbolic Model Checking Without BDDs. International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer: 193–207.