Multi-Robot Motion and Task Planning in Automotive Production Using Controller-based Safe Reinforcement Learning

Eric Wete
Leibniz University Hannover
Hannover, Germany
eric.roslin.wete.poaka@stud.uni-hannover.de

Joel Greenyer
FHDW Hannover
Hannover, Germany
joel.greenyer@fhdw.de

Daniel Kudenko
Leibniz University Hannover
Hannover, Germany
kudenko@l3s.de

Wolfgang Nejdl
Leibniz University Hannover
Hannover, Germany
nejdl@kbs.uni-hannover.de

ABSTRACT

Using synthesis- and AI-planning-based approaches, recent works investigated methods to support engineers with the automation of design, planning, and execution of multi-robot cells. However, real-time constraints and stochastic processes were not well covered due, e.g., to the high abstraction level of the problem modeling, and these methods do not scale well. In this paper, using probabilistic model checking, we construct a controller and integrate it with reinforcement learning approaches to synthesize the most efficient and correct multi-robot task schedules. Statistical Model Checking (SMC) is applied for system requirement verification. Our method is aware of uncertainties and considers robot movement times, interruption times, and stochastic interruptions that can be learned during multi-robot cell operations. We developed a model-at-runtime that integrates the execution of the production cell and optimizes its performance using a controller-based AI system. For this purpose and to derive the best policy, we implemented and compared AI-based methods, namely, Monte Carlo Tree Search, a heuristic AI-planning technique, and Q-learning, a model-free reinforcement learning method. Our results show that our methodology can choose time-efficient task sequences that consequently improve the cycle time and efficiently adapt to stochastic events, e.g., robot interruptions. Moreover, our approach scales well compared to previous investigations using SMC, which did not reveal any violation of the requirements.

KEYWORDS

Multi-robot Motion Planning; Multi-robot Task Planning; Model Checking; Safe Reinforcement Learning; Q-Learning

ACM Reference Format:
Eric Wete, Joel Greenyer, Daniel Kudenko, and Wolfgang Nejdl. 2024. Multi-Robot Motion and Task Planning in Automotive Production Using Controller-based Safe Reinforcement Learning. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 10 pages.

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand. © 2024 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

1 INTRODUCTION

Automotive production involves the design, planning, and commissioning of multi-robot cells that are usually set up on the assembly line site [4]. The manufacturing process, e.g., spot welding [34], defines how parts must be added, or assembled, and moved from one production cell to another for further processing. In each production cell, the manufacturing process defines requirements that must be fulfilled to achieve a high-quality end product. Among others, we distinguish critical, safety, quality, and performance requirements. Due to the intrinsic complexity of requirements and manufacturing processes, e.g., spot welding, the design, and planning of the orchestration of multiple robots in a production cell is a challenging, usually manual error-prone, and time-consuming task. For example, robot engineers must ensure that no collision occurs during the manufacturing process execution. Moreover, multi-robot cell systems can be subjected to unexpected events that include, e.g., the manufacturing process interruptions, mechanical failure on a robot welding gun, and amendments to the production process. Due to the commissioning time constraints and production process requirements, this aspect of uncertainties is usually not well covered in practice. Using a dynamic task reallocation mechanism, self-adaptive systems [7, 26] can provide a solution to address uncertainties.

To address the problem of planning and task allocation, recent approaches [12, 13, 27, 28, 32, 42, 43] provided tools for the formal specification of system requirements, the build and execution of correct-by-construction system controllers. However, at the reactive controller level, these approaches do not address real-time constraints or probabilistic processes. Moreover, these methods do not scale well as the size of the system increases, e.g., the number of robots, the number of robot tasks. Reinforcement learning (RL) methods, such as Q-Learning [36], can optimize system decisions. To this end, the RL agent interacts with its environment, receives rewards according to selected actions, and adapts its behavior to optimize its accumulated reward. However, in some contexts, such as multi-robot cells, the RL agent actions must be guarded to always select requirements-compliant actions, e.g., collision-free actions. Additionally, the set of RL agent actions varies from a system state to another.

In this paper, we address the challenges raised above in the following contributions.

1. Time- and probabilistic-constrained model. To address real-time constraints and probabilistic approaches at the controller level, we specify the system requirements and environment assumptions using Prism [22], a probabilistic model checker for formal modeling, and analysis of systems with probabilistic behavior. The specification considers multi-robot cell components, such as tasks, robots, collision requirements, and task dependency constraints. The specification model also includes the movement, task and interruption times of each robot, and the interruption probabilities. In fact, the model supports the occurrence of unexpected events during production cell operation. The specification model can be produced from the robot cell description provided by robot programmers. Using model checking, the specification model is checked against requirements, such as cycle completion or that a robot must not be interrupted indefinitely.

2. Integrated model execution: Model-at-runtime. We also provide an executor engine based on the Prism simulation engine to execute the Prism model. It permits the computation of all possible states given a system action, and also all possible system actions given a state. The executor engine can be integrated with robot services, such as ABB Robot Web Services, to connect the executor to a virtual or real production cell. This paper introduces a controller-based safe reinforcement learning architecture, as illustrated in Fig. 1. Using Prism, we produce a safe controller, including a (sub optimal) policy that is optimized using RL-based methods. In fact, the RL agent learns the optimal policy from valid actions guaranteed by a controller. Therefore, the correctness of the system execution is guaranteed since the controller implements the system requirements that are formally specified using Prism in the previous stage. Moreover, using Prism reward structures and reward-based properties, given an environment observation, each available safe action can be evaluated, and the evaluations are inputted to the RL agent model. This complementary information can significantly speed up the convergence of the learning process, namely, the exploration phase. The executor engine that includes the controller, *observes*

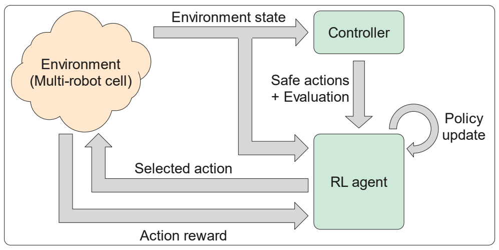

Figure 1: The controller-based safe reinforcement learning. The controller synthesizes safe actions from the environment observation including action evaluations for the RL agent.

the multi-robot cell and outputs a set of possible, safe and requirements-compliant system actions including their evaluation, e.g., cycle time, given the environment current state. The RL agent selects actions to apply on the environment. Then, the environment new state and the reward related to the applied action are computed. The new state and reward are used to update the agent policy. The goal of the RL agent is to optimize its accumulated reward during its interactions.

3. Results and Evaluation. We experiment and showcase the performance of our approach by implementing the RL agent using RL techniques, such as Q-learning, a model-free RL technique, and compared it with Monte Carlo Tree Search (MCTS), a heuristic sample-based planning technique. We show that the RL agent can produce, at run-time, event in robot interruption cases, time-efficient action sequences to achieve optimal cycle time. Prism can compute estimations of system properties, by sampling random system executions using Statistical Model Checking (SMC) [24]. This approach is useful in complex model checking cases or for large models and improves our approach scalability.

2 BACKGROUND

2.1 Multi-Robot Task Planning

A robot *task* is defined as an activity that a robot can perform during a certain time. We define a task as *re-allocable* when at least two robots can execute the task. The *task plan* of a robot describes which tasks must be achieved by the robot and the order in which those tasks must be executed.

The *task planning* defines the *task plan* for each robot of the production cell. We define *multi-robot task planning* as the robot task planning for a multi-robot cell.

2.2 Multi-Robot Motion Planning

The robot's *motion* is defined by the combination of a path and a trajectory. A robot *path* defines the geometric representation of a robot movement, i.e., the sequence of waypoints that the robot must pass through during its motion. A robot *trajectory* defines a time-constrained path, i.e., it defines, giving a path, the time constraints, such as velocity, accelerations, and jerks on waypoints, including the start and end path points.

The *motion planning* finds the collision-free path and trajectory of a robot to move from an initial to a final robot configuration. In the multi-robot context, we rather speak about *multi-robot motion planning*.

2.3 Discrete-Time Markov Chain (DTMC)

A *discrete-time Markov chain* (DTMC) is defined by a tuple $M = (S, s_{init}, P, L)$, where:

- $S \neq \emptyset$, is a finite set of states
- $s_{init} \in S$ is the initial state
- $P : S \times S \rightarrow [0, 1]$ represents the transition probability matrix with the constraint $\sum_{s' \in S} P(s, s') = 1, \forall s \in S$
- $L : S \rightarrow 2^{AP}$, state labeling function with atomic propositions, where AP is the set of atomic propositions.

An infinite or finite state sequence of a DTMC is called a *path*, defined by $\rho = s_0 s_1 s_2 \ldots$, where $s_i \in S$, and $\forall i : P(s_i, s_{i+1}) > 0$.

A state $q \in S$ is *reachable* from the state $p \in S$ if a path can be found from p to q, i.e., $\exists\, \rho : \rho = p \ldots q$. A state is *reachable* if it is reachable from the initial state.

The research works in [15] introduced the Probabilistic Computation Tree Logic (PCTL), an extension of the CTL [5, 6] that defines temporal logic with time constraints. In the PCTL, the temporal logic is specified as in CTL and the PCTL adds probabilities to describe DTMC properties. The PCTL abstract syntax is defined by:

$\varphi ::= \top \mid q \mid \neg\,\varphi \mid \varphi \;\wedge\; \varphi \mid P_{*p}[\;\varphi\; U\; \varphi\;]$, where q is an atomic proposition, p is a probability, $p \in [0,1]$, and $* \in \{<, >, \leq, \geq\}$. U is the temporal modal operator "until". P expresses the probability bound of the specified formula, i.e., the truthiness of the formula with the probability $*p$.

Given a DTMC model and a PCTL formula that expresses a property, the PCTL *model checking* checks if the model satisfies the formula [15]. For example, the production cell must reach the end of a cycle. Thus, the designed model must verify that the end of a cycle can be reached with respect to environment assumptions. In our problem settings, to verify that the robot cell cycle can complete, the underlying DTMC model of the robot cell must be checked against the reachability of the cycle end state. Indeed, we can use the LTL F (eventually) operator: $F\,\varphi \equiv \mathit{true}\;U\,\varphi$.

2.4 Q-learning

In RL [19], Q-learning is known as a model-free method [40], i.e., it does not require the environment (transition) model. The RL agent learns a behavior or a policy by interacting with its environment to reach its goal.

Given a state s, inferred from the agent environment observation, the agent selects an action a, among possible actions in that state, observes the environment to retrieve a new environment state, and receives a reward that is used to update its Q-table. The intuition behind this is that by making several iterations in the future and trying the possible actions, the Q-table will converge to the optimal policy.

Given S, the set of states, and A, the set of actions, the Q-values are updated using temporal difference learning [40] inspired by the following Bellman equation [2]:

$$Q(s,a) \longleftarrow Q(s,a) + \alpha \left[r + \gamma \cdot \max_{a' \in A} Q(s',a') - Q(s,a) \right] \qquad (1)$$

Where:

- $Q(s,a)$ is the Q-value of action $a \in A$ selected by the RL agent among the possible actions of the state $s \in S$
- r is the reward earned by the RL agent
- $\alpha \in [0,1]$ represents the learning rate
- γ is the discount factor
- s' represents the state obtained applying action a to the environment.

The Q-learning method updates the optimal policy using Eq. (1) while iterating over episodes.

During the system operation, the most promising action is selected using the action with the highest Q-value given the system current state.

3 RELATED WORK

Planning safe robot motion can be achieved using linear temporal logic (LTL) synthesis-based approaches, and the model can control robots during their motion [16, 20, 42, 43]. These works described how to specify, using LTL or its subset GR(1) [3, 32], the system behavior as guarantees and the environment behavior as assumptions. These approaches support the generation of robot motion and robot task orchestration. These approaches can be extended by taking into account probabilistic and real-time constraints and can be improved using learning techniques as we do in our approach. Moreover, these approaches require improvements in scalability due to the LTL state space blow up. Using SMC, our approach addresses this problem that occurs mainly during the system verification process.

Some approaches [17, 30, 31, 37] used the behavior trees (BTs) framework as a middle layer to integrate LTL synthesis with robot motion and task planning to find maximal satisfying task schedules. In fact, the BT models sets of tasks, where each that is defined is a sub-tree. The goal is to provide independent modular task formal representations. Thus, BT sub-trees can be replaced or moved according to defined requirements. In contrast to our approach that constructs a model-at-runtime, the BT-based approaches do not yet cover run time constraints due to the offline construction of the BT model.

To address the task allocation and planning under uncertainties, Markov decision process (MDP)-based models model the robot behavior and LTL for constraint specification [11, 23, 35]. These approaches design a team of MDPs that are solved for LTL constraints using Prism. However, the joint policy built using the MDPs does not consider task dependencies and robot motion time constraints to improve the cycle time. In contrast, our approach build a Prism model that specifies tasks dependency constraints (using guards) for valid action selection.

The integration of model checking with motion planning shows that a solution for the task specification is proposed to address the challenge of motion and action planning given LTL constraints [14]. In fact, the approach leverages the domain-specific knowledge and the domain-independent knowledge. The former defines the robot motion along with the workspace model, and the latter describes the system actions, representing the robot capabilities. This increases the loose coupling, scalability, and flexibility of the framework. We take advantage of this architecture model to improve the scalability of our approach. Instead of synthesizing the controller, which raises scalability issues for large models, as in our use case, we verify the system via SMC. During the system operation, we execute the defined model that follows the model strategy. Moreover, we improve the synthesized strategy using RL-based approaches.

RL is often used to solve decision-making problems, e.g., to avoid the system making unrecoverable errors, but even with low rates, such unrecoverable errors can appear because the agent can select unsafe actions [29]. To address this problem, some approaches proposed shield synthesis for safe RL [18, 21] to restrict agent action space while optimizing agent performance. The shield can correct the agent-selected action if it is unsafe [1, 9] or it first filters out the unsafe actions, and the agent selects among the filtered actions. The latter is known as the preemptive shield. The shield is built using reactive synthesis from an LTL-based specification and an abstraction of the environment model specified using MDP. Our methodology uses a preemptive strategy. Our Prism model is specified to avoid such unrecoverable actions to be selected during the system execution or agent learning. The unrecoverable actions are identified during the design phase and checked against the Prism model using the Prism property specification.

4 PROBLEM DESCRIPTION

We consider a multi-robot cell that consists of robots, robot tools, peripheral devices, workpieces, and objects such as part positioners. We assume that the work cell is heterogeneous, i.e., the robot capabilities can differ from one robot to another, e.g., set of spot welding, with pick-and-place robots. Each robot can move in the work cell and achieve a set of tasks. Each robot has a motion range and can share a workspace with other robots. Thus, there are some task sets that a robot set can execute.

4.1 System Design

The system design requires the identification of relevant work cell components and component properties. We identify the system- and environment-related transitions. In fact, the robot current location, the robot status (e.g., interrupted or not), the status of tasks (completed or not), and the robot remaining time estimation (when the robot will be available for a new task assignment) define the *environment variables*. The robot task assignments define the *system variables*, i.e., a system variable defines for a robot the task, which is assigned to the robot. We define a *multi-robot state* as a full assignment of environment and system variables. We define a *system action* as a joint assignment of task to robots, i.e., an action describes a full assignment of the robots with tasks. The assignment of a task to a robot will trigger the corresponding robot movement and task execution. For example, a welding task assignment will make the robot to move to the welding point and perform the welding process. This task assignment description is similarly defined for glue, paint, pick and place tasks.

To understand these definitions, let us consider a multi-robot cell with two robots r_1 and r_2 with their the home locations h_1 and h_2, respectively. The tasks that robot r_1 (resp. r_2) can execute are t_1, t_2 and t_3 (resp. t_2, t_3 and t_4). We consider the movement of a robot to its home location as a task that we define as t_{h1} (resp. t_{h2}) for robot r_1 (resp. r_2). Tab. 1 defines the variables of this multi-robot cell example. Each defined variable has a type (environment or system), a description, and a domain. A variable domain defines the values that this variable can take. According to this example, a system or an RL agent action is defined as the tuple (t_i, t_j) that assigns the task $t_i \in \{t_{h1}, t_1, t_2, t_3\}$ to robot r_1 and the task $t_j \in \{t_{h2}, t_2, t_3, t_4\}$ to robot r_2.

4.2 System Rules

The production cell system to be built must manage the orchestration of multiple robots, i.e., parallel movements, under the consideration of the work cell requirements described in the following rules.

- Each robot has a restricted workspace, thus having a set of tasks that it can perform.
- A task assignment can be reallocated if the robot is interrupted.
- A task must not be assigned to many robots at the same.
- An interrupted robot must go to its home location (e.g., for repair).
- A robot returns to its home location when its tasks are completed.
- A task must not be assigned to a robot if all its task dependencies are not cleared.
- Any task assignment representing a potential collision must be avoided.
- Incomplete tasks must be assigned so that completed tasks are not assigned infinitely to the robots.

Table 1: Variable definition of a multi robot cell example

Variable type	Description	Domain
Environment	Robot r_1 current location	t_{h1}, t_1, t_2, t_3
Environment	Robot r_2 current location	t_{h2}, t_2, t_3, t_4
Environment	Robot r_1 remaining time	Time in ms
Environment	Robot r_2 remaining time	Time in ms
Environment	Robot r_1 status	Boolean
Environment	Robot r_2 status	Boolean
Environment	Task t_{h1} status	Boolean
Environment	Task t_{h2} status	Boolean
Environment	Task t_1 status	Boolean
Environment	Task t_2 status	Boolean
Environment	Task t_3 status	Boolean
Environment	Task t_4 status	Boolean
System	Robot r_1 task assignment	t_{h1}, t_1, t_2, t_3
System	Robot r_2 task assignment	t_{h2}, t_2, t_3, t_4

4.3 Environment Assumptions

The system engineer must define how the environment behaves, more precisely, how the environment variables change during production cell operation. We define the environment behavior in assumptions as follows:

- When a cycle starts, all robots are ready to achieve tasks, are not interrupted, are at their home position, and no tasks are completed yet.
- At the end of a cycle, a new cycle starts.
- A robot first moves to the location where the task must be completed and then executes it.
- A robot task cannot be unassigned if it is not completed except if the robot is interrupted, i.e., if a robot is at a task location or gets a new task and the task is not yet completed it performs the task.
- A robot eventually completes its assigned task unless interrupted.
- A robot can be interrupted at any time (robot failure and repair).
- If a robot is interrupted, it must not complete a task.
- A robot is repaired only at its home position, i.e., if a robot is interrupted, its status can only change at the home position..

5 METHODOLOGY

Our approach combines formal system specification and RL techniques. The former permits the formal modeling, analysis and verification of the system requirements, along with the computation of a controller based on the Prism [22] simulation engine. The later allows us to improve the system actions while interacting with its environment according to optimization criteria.

As shown in Fig. 2, we describe in three levels our approach architecture. The first level, i.e., the informal level, consists of the environment assumptions, e.g., a robot can be interrupted with some probability, and system requirements, e.g., a robot can only execute a set of task due to its working range. The second level, i.e., formal level (Prism), consists of a Prism model that formally specifies the environment assumptions, the system requirements, and the reward structures, which are required for optimizations at run time. The third level or run time level (Java) is made up of the Prism simulator engine, a controller that uses Prism APIs and the simulator engine to provide the next possible and requirement-compliant system actions, and a controller executor. The controller executor implements the strategy or agent policy to optimize the making-decision system, i.e., finding the most efficient task schedules giving the unexpected environment behavior. In the following sub-sections we take a closer look at the three levels.

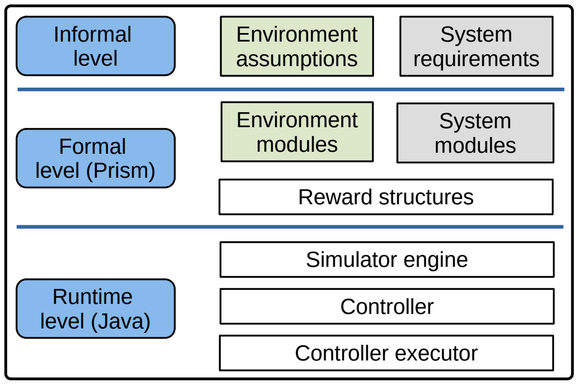

Figure 2: The methodology architecture in three levels

5.1 System Specification

The environment as well as the system can be modeled as a DTMC, where environment and system transitions are performed one after the other. Environment transitions update only the environment variables, whereas system transitions update only the system variables. Our approach requires the system description and its requirements. As illustrated in Fig. 3, our process is threefold.

First, the prerequisites (i.e., the informal level in Fig. 2) are made up of system requirements, robot cell constraints, and collision constraints. A robot cell constraint can define for example dependencies between robot tasks, i.e., task prerequisites, or tasks that can only be performed when some tasks are already completed. Due to working range constraints and requirements, a robot can only execute a set of tasks. We leverage analysis techniques for robot reachability and capacity to identify tasks that each robot can execute [33, 38]. Collision constraints define robot movements that must not occur during the robot cell operation. Robot programmers can provide these prerequisites in a robot development environment tool or a formal robot cell specification [42]. Given the set of tasks a robot can perform, we leverage trajectory planning techniques such as A-Star [8, 10, 25] to compute all trajectories for each task. Synchronously, we run all trajectory pairs to find all potential collisions. We mention that the trajectories in the pair are from two different robots. Moreover,

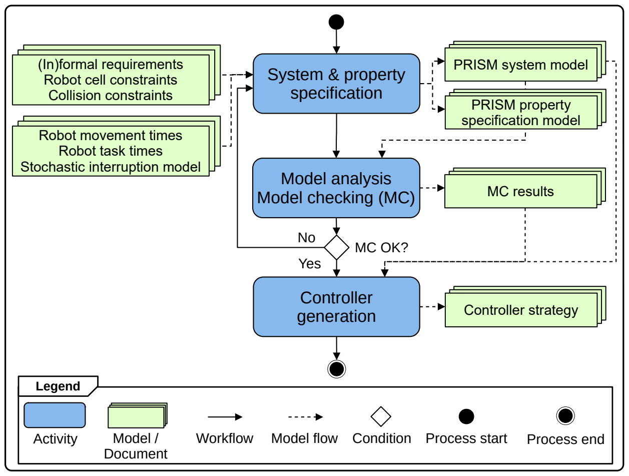

Figure 3: The system specification process. The figure shows how we build the controller based on Prism models, MC of LTL properties to synthesize safe actions for RL agents.

this step requires time constraints on movements, interruptions, and interruption probabilities. Based on the inputs above, we write the Prism system model (see the formal level in Fig. 2) that defines the system and the environment behavior, including the stochastic environment events. The environment behavior also known as environment assumptions (resp. system behavior or system requirements) are specified using Prism modules that only control environment (resp. system) variables and are called environment (resp. system) modules. We also defined the reward structures for system action evaluation at run time. This step also outputs the Prism property specification model that defines, using probabilistic temporal logic, the properties that the system must verify.

Second, we perform model analysis and model checking using the Prism tool. In fact, the Prism system model is checked w.r.t. the system requirements encoded in the Prism property specification model. If model checking fails, the Prism system model is refined accordingly to fulfill the system requirements. To this end, Prism can generate, in some cases, a counterexample that shows a sequence of states leading to a property violation. This verification process permits at run time to avoid unsafe action selections, e.g., an action that assigns two tasks to two robots, which cause a collision between the two robots.

Finally, we integrate the Prism system model with a controller that can be executed while the system is running (see the run time level in Fig. 2). The controller executor runs the Prism system and computes safe and requirements-compliant system actions given the state of the environment. To this end, we use the Prism simulator engine that uses Java-based APIs to run the Prism model. Moreover, using the reward structures, the Prism simulator permits computing, given a state, quantitative property estimations such as cycle time for each possible system action. The controller strategy refers to the controller implementation of the underlying Prism model.

5.2 Prism Model Specification

The Prism model consists of constant definition, environment and system modules. The constants define relevant multi-robot cell configurations: the number of robots and tasks, robot bases, interruption probability constraints, and time constraints for robot movements, tasks and interruptions. The environment modules specify environment transitions by defining rules to update environment variables. In contrast, the system modules define system transitions that update system variables. For each variable, we define a Prism module and declare a the variable in the module. Using the Prism synchronization feature, the environment and system modules are activated in a turn-based fashion during the Prism model execution.

Each environment (resp. system) module defines a local variable that represents an environment (resp. a system) variable. At a given time, the variable values of all modules represent a state of the multi-robot cell. Each module behavior is specified by commands that define how the variables change. A Prism command is defined in Eq. (2) as follows.

$$[\mathrm{L}]\ \mathrm{C} \rightarrow \mathrm{prob}_1 : \mathrm{up}_1 + \cdots + \mathrm{prob}_n : \mathrm{up}_n; \tag{2}$$

where L is the command annotation label that permits the synchronization of commands having the same label; C is the condition over the model variables. It activates the command transitions that follow the right part of the right arrow (→). Transitions are separated by the plus (+) operator. For $i \in \{1, \dots, n\}$, prob_i defines the transition probability and up_n the update expression that specifies new variable values of the module.

Lst. 1 shows some module command examples. Each command has a label, a guard condition for the command transitions. The first line updates the current location of robot R00 (`cl00`) when its movement to its target (`tl00`) is completed. At line 2, the second command updates the interruption status (`ii00`) using the interruption probability constant (`INT_PROB`). The next line specifies the task completion of the assigned task 0 (`iv000`). The last command assigns the task 0 to the robot R00 (`tl00`).

The Prism model is verified using model checking with Prism property specification. For example, to check that the production cell completes its cycle infinitely often, we specified the following property and checked against the Prism model: $\mathbf{P}_{\geq 1}$[**G F cycleCompleted**]. Moreover, we verified that once a cycle is completed a new cycle start using the property $\mathbf{P}_{\geq 1}$[**G F (noTaskCompleted U cycleCompleted)**]. The following non-exhaustive properties contribute to avoid collisions: $\mathbf{P}_{\geq 1}$[**!(F `tl00` = `tl01`)**] prevents a task to be assigned to two robots, and $\mathbf{P}_{\geq 1}$[**!(F `cl00` = `cl01`)**] prevents two robots to be on a same position. An important property must also check that no robot interrupts indefinitely $\mathbf{P}_{\geq 1}$[**GF !`ii00`**]. This property must be verified for all robots.

5.3 System Performance Optimization with RL

We construct a learner that takes as input: the multi-robot cell state and the available safe actions according to the environment observation, including the action evaluations obtained by the controller (see Fig. 1). The multi-robot cell state consists of the value of the environment and system variables. The controller defines the set of available safe actions given the multi-robot cell current state. The computation of actions including their evaluation, e.g., estimated cycle time, must be performed since the action set can differ from an environment observation to another. The goal of the RL agent is to learn the optimal policy by iterating over episodes and updating its policy (Q-table for classic Q-learning method using the temporal difference learning, as defined in Eq. (1)). The RL agent selects the most promising action with respect to its policy given the current environment state.

In tabular Q-learning, a multi-robot cell state (i.e., robot tasks, robot status, and task status) is encoded in a Q-table, along with the safe system actions provided by the controller given the environment state. To illustrate the Q-table encoding, let us consider a multi-robot cell of N robots with T tasks. We define the set of robots $\mathcal{R} = \{1, \dots, N\}$, and the set of tasks $\mathcal{T} = \{1, ..., T, T+1, ..., T+N\}$, where $T+i$ is the task for the home position of the robot $i \in \mathcal{R}$, which defines the movement of robot i to its home position. For each robot $r \in \mathcal{R}$, we define: CL_r as the current location, RT_r as the remaining time, RS_r as the robot status, TT_r as the assigned task of the robot r. We apply unity-based normalization to normalize time values into the range $[0, 5]$, as shown in Eq. (3).

$$X = \left\lfloor \frac{5 \cdot (x - x_{min})}{(x_{max} - x_{min})} \right\rfloor \tag{3}$$

For each task $t \in \mathcal{T}$, we define TS_t as the task status of the task t. A state is defined by $(CL_r, RT_r, RS_r, TT_r, TS_t)\ \forall r \in \mathcal{R}, t \in \mathcal{T}$, where TT_r defines the current task assignment. An action is defined by $(TT_r)\ \forall r \in \mathcal{R}$, which is the next task assignment. Let $\mathcal{S}$ be the set of states and $\mathcal{A}$ be the set of actions. Then, the Q-table builds a matrix and defines the function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$. The Q-values are updated using the Bellman equation Eq. (1).

Thanks to the controller, learner exploration can be improved using action estimations. Therefore, the available actions can be sorted along with action estimations. The initialization of the RL agent is performed using the action evaluation provided by the controller, e.g., Q-values initialization: action estimation value for all nonterminal states or 0 for the terminal state. The terminal state corresponds to the end of cycle. Each training episode runs until the cycle ends. We implemented the learner exploration applying the ϵ-greedy strategy for action selection at each training step. Once the selected action is applied to the environment, the RL agent receives an immediate reward defined as $r(s_i, t_i)$ that refers to the immediate reward after reaching state s_i and time t_i. As illustrated in Eq. (4), it sets the negative time elapsed as the reward on non-terminal state and 0 otherwise.

$$r(s_i, t_i) = \begin{cases} -(t_i - t_{i-1}), & \text{if } s_i \text{ is not terminal} \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

To find the best learner hyper-parameter settings, we ran experiments with different hyper-parameter values, and compared

Listing 1: Sample of Prism model module commands

```
[envRTPos] isEnvRTPosUp & !nextRMTNotDone00 & !atTarget00 -> (cl00'=tl00);
[envInt] isEnvIntUp & notInterrupted00 & isNewAssign00 -> INT_PROB : (ii00'=true) + (1-INT_PROB) : (ii00'=false);
[envTask] isEnvTaskUp & notVisitedAndAssignedToNotInterrupted000 -> (iv000'=true);
[sys00] !isEnv & !sysPlayed00 & tl00!=0 & atTarget00 & !ii00 & !allCanDoTasksCompleted00 & isHomeVisited00 &
     isCurLocVisited00 & mustBeCompleted000 & !iv000 & potentialCollisionR00L000 -> (tl00'=0) & (isNewAssign00'=(tl00
     !=0));
```

the learner performance. The RL agent learns an optimal action value in a given state or an action value function that models the optimal expected long-term value or reward for an action in a given state. The optimal action selection policy is based on the Q-function and selects the action having the highest Q-value.

We integrated the Prism model execution with an RL agent to optimize the model execution policy and compared the cycle times obtained with state-of-the-art methods, namely, random and MCTS. The former used random system action selection, while the latter applied MCTS simulations for the system action selection. We used the random policy execution as our baseline. The baseline results are used for the model initialization of the RL agent, which permits the speed-up of the agent policy convergence. Moreover, the model is enriched with the cycle time estimations of possible actions provided by the controller at run time. We use the following settings for the RL agent: $\epsilon = \max(0.02,\ e^{-0.01\ t})$ for the ϵ-greedy strategy, learning rate $\alpha = 0.1$, and the discount factor $\gamma = 1$. This value of the discount faction significantly considers RL agent future rewards. The action reward corresponds to the negative value of the elapsed time when applying the agent-chosen action.

We leverage Prism to produce a controller that runs using an underlying Prism model to produce safe actions for the RL agent. Q-learning is used to further optimize this controller. More interestingly, in Q-learning, the controller restricts or defines the action space of the RL agent keeping the optimized controller safe (see Fig. 1).

6 USE CASE

The motivation for our approach is inspired by an industrial case study of a multi-robot spot welding production cell. Let us consider the multi-robot cell of a car underbody, as illustrated in Fig. 4. The production cell consists of four spot welding robots and a workpiece (car underbody) placed on a part positioner. The robot cell consists of four robots: *R00*, *R01*, *R02*, *R03*, and twelve spot welding tasks on a car underbody: *0*, *1*, …, *11*. A robot has a base (home) position: the locations *12*, *13*, *14*, *15* are bases for the robots *R00*, *R01*, *R02*, *R03* respectively, and the task feasibility of robot are depicted in Tab. 2. In Fig. 4, we also highlighted the tasks that each robot can perform due to the robot reachability range and capability [33, 38]. Environment variables such as robot location, robot status, and task status are updated via robot signals and spot welding case sensors whose data is collected via specific function calls. Simultaneous simulations of all trajectory pairs can identify all potential collisions, e.g., if robot *R00* must perform task *11*, while robot *R02* must perform task *0*. The process quality requirements also define task dependencies, e.g., task *1* depends on tasks *2*, and *7*. These requirements and constraints are specified in the Prism model.

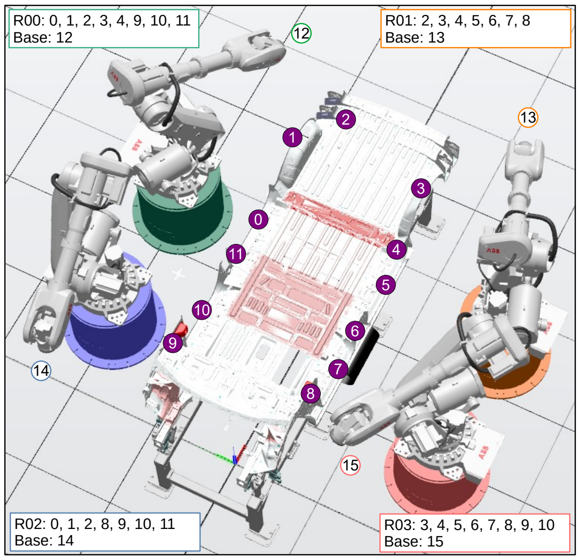
R00: 0, 1, 2, 3, 4, 9, 10, 11
Base: 12
R01: 2, 3, 4, 5, 6, 7, 8
Base: 13
R02: 0, 1, 2, 8, 9, 10, 11
Base: 14
R03: 3, 4, 5, 6, 7, 8, 9, 10
Base: 15
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Figure 4: A multi-robot spot welding cell of a car underbody. The figure shows four robots at their base location and the spot welding points that each robot can process.

Table 2: The task feasibility of each robot. This figure shows the spot welding task distribution by robot.

Robot	Tasks
R00	0, 1, 2, 3, 4, 9, 10, 11
R01	2, 3, 4, 5, 6, 7, 8
R02	0, 1, 2, 8, 9, 10, 11
R03	3, 4, 5, 6, 7, 8, 9, 10

7 RESULTS AND DISCUSSION

In our experiments, we used a PC, with an Intel (R) Core (TM) i5-8250U CPU, RAM 16.0 GB, on a x64-based processor, running Windows 11.

To verify that the system reaches the end of the cycle, and apply SMC, we specify the following property: $\mathbf{P}_{\geq 0.98}$[**F cycleCompleted**].

The Prism **P** operator is `true` in a state s where the specified bound probability (here ≥ 0.98) is met for paths starting from the state s, which satisfy the specified property (here **F cycleCompleted**). This property specifies the reachability of the cycle end using the LTL operator F (eventually). The Boolean expression **cycleCompleted** specifies that the cycle of the robot

Table 3: SMC results using SPRT

Robots	Tasks	Avg. iteration time (s)	Path length Avg.	Min.	Max.
2	8	0.009	6.6×10^2	74	2554
2	12	0.016	8.4×10^2	99	4778
2	20	0.031	1.1×10^3	144	3254
2	30	0.100	1.1×10^3	229	3398
2	40	0.270	1.2×10^3	339	5544
3	12	0.019	8.3×10^2	149	2068
3	30	0.130	1.0×10^3	245	2255
3	40	0.340	1.0×10^3	371	3233
4	12	0.037	1.1×10^3	202	3323
4	30	0.190	1.2×10^3	335	2630
4	40	0.340	1.3×10^3	461	4030
5	30	0.220	1.4×10^3	343	2342
5	50	0.740	1.6×10^3	767	2790
6	30	0.240	1.6×10^3	701	2932
6	40	0.490	1.7×10^3	872	3301
7	50	1.000	2.0×10^3	928	3238
7	60	1.800	2.2×10^3	989	3168
8	40	0.690	2.1×10^3	692	3452
9	50	1.400	2.5×10^3	1306	3898
10	40	0.890	2.6×10^3	844	4080

cell is complete, i.e., all tasks are completed, all robots are at their base and no robot is interrupted.

Using the Sequential Probability Ratio Test (SPRT) method [39], we verified the Prism model of the use case (see [41]) and obtained a positive result with the probability of 0.99, with 113 iterations performed in 4.165 s. The average, minimum, and maximum path lengths computed are 1.1×10^3, 202, and 3323, respectively, with the maximal path length set to 10 000. For scalability testing purposes, we varied the robot cell settings. The results of our experiments are shown in Tab. 3. For all the experiments, the SPRT method computed 113 iterations and obtained a positive result with the probability of 0.99. Tab. 3 shows that the time required for the SMC and path length increase with the number of robots and tasks. This can be explained by the increase in the state space size. For example, the system has at most $r \times t$ possible actions, where r is the number of robots and t the number of tasks. Our approach can at least support up to 10 robots and 40 tasks. Considering the scalability, these results are satisfying for the industry current needs and our use case.

In our experiments, we performed 2000 cycles, corresponding to the number of proceeded car underbodies. We performed experiments with and without probabilistic interruptions using Prism guards. We should mention that all the compared methods rely on the Prism model. Thus, the policy strategies are safe with respect to safety requirements such as collision freedom and task dependency constraints, but they differ in the action selection strategy used. We did not find any violation of the safety requirements. Illustrated in Tab. 4, the results of our experiment

Table 4: Cycle times achieved with 2000 cycles

Method	Int.	Cycle time (s) Overall ($\times 10^3$)	Avg.	Imp. (%)	Min.	Max.
Random		38.09	19.04		11.63	29.94
MCTS	0	37.75	18.87	0.9	12.49	26.70
QL		37.60	18.80	1.3	11.59	25.89
Random		58.59	29.29		11.34	88.49
MCTS	0.0125	57.17	28.58	2.4	11.14	90.87
QL		56.52	28.26	3.5	11.91	92.49

show that the RL agent provided the best performance compared to random and MCTS-based policies. The RL agent can therefore optimally schedule task sequences under the consideration of stochastic interruptions. The RL agent can find the most-efficient task sequences, even in interruption cases and outperform random and MCTS policies.

8 CONCLUSION

In this paper, we addressed the problem of efficient multi-robot motion and task planning under the consideration of uncertainties, tasks and real-time motion constraints using safe RL where the controller guarantees the safety of the agent action selection. We described the controller-constrained RL architecture, which extends the traditional RL approach with a controller that guarantees the correctness or validity of the agent actions. We also described how we address the problem using the three level architecture from the informal level that describes the environment assumptions, along with the system requirements and constraints, to the runtime level that specifies our model-at-runtime based on the second level that formalizes and design the environment and system behaviors.

Using Prism, we specified the environment and the system behaviors, including stochastic and real-time constraints. The designed Prism model defines also reward structures to evaluate, at run time, agent actions, and therefore considers quality requirements for the online optimization of the system agent performance. Using LTL formulae, we specified a property specification model to verify the Prism model against the system requirements. In fact, to validate the correctness of the system specification, the system verification is performed using LTL-based model checking approaches, such as SMC, which is appropriate for large models or when traditional model-checking methods do not scale well. The system execution is monitored, controlled using a model-at-runtime, and optimized at run time through RL learning. We applied our approach and experimented with an industrial use case, and showed that the RL agent performs with the best results over state-of-the-art methods.

Further research could involve experimentation on real multi-robot cells, and on-the-fly model constraint changes, since the model can become obsolete due to online requirement and constraint adjustments. In the future, it could be interesting to investigate how to improve the scalability of model-checking by using for example learning approaches.

REFERENCES

[1] Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. 2018. Safe Reinforcement Learning via Shielding. *Proceedings of the AAAI Conference on Artificial Intelligence* 32, 1 (Apr. 2018). https://ojs.aaai.org/index.php/AAAI/article/view/11797

[2] EN Barron and H Ishii. 1989. The Bellman equation for minimizing the maximum cost. *Nonlinear Analysis: Theory, Methods & Applications* 13, 9 (1989), 1067–1090.

[3] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. 2012. Synthesis of Reactive(1) designs. *J. Comput. System Sci.* 78, 3 (2012), 911 – 938. https://doi.org/10.1016/j.jcss.2011.08.007 In Commemoration of Amir Pnueli.

[4] Nils Boysen, Malte Fliedner, and Armin Scholl. 2008. Assembly line balancing: Which model to use when? *International Journal of Production Economics* 111, 2 (2008), 509–528. https://doi.org/10.1016/j.ijpe.2007.02.026 Special Section on Sustainable Supply Chain.

[5] Edmund M. Clarke and E. Allen Emerson. 1982. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logics of Programs*, Dexter Kozen (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 52–71.

[6] E. M. Clarke, E. A. Emerson, and A. P. Sistla. 1986. Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications. *ACM Trans. Program. Lang. Syst.* 8, 2 (apr 1986), 244–263. https://doi.org/10.1145/5397.5399

[7] R. de Lemos and P. Potena. 2017. Chapter 14 - Identifying and Handling Uncertainties in the Feedback Control Loop. In *Managing Trade-Offs in Adaptable Software Architectures*, Ivan Mistrik, Nour Ali, Rick Kazman, John Grundy, and Bradley Schmerl (Eds.). Morgan Kaufmann, Boston, 353–367. https://doi.org/10.1016/B978-0-12-802855-1.00014-9

[8] František Duchoň, Andrej Babinec, Martin Kajan, Peter Beňo, Martin Florek, Tomáš Fico, and Ladislav Jurišica. 2014. Path Planning with Modified a Star Algorithm for a Mobile Robot. *Procedia Engineering* 96 (2014), 59–69. https://doi.org/10.1016/j.proeng.2014.12.098 Modelling of Mechanical and Mechatronic Systems.

[9] Ingy ElSayed-Aly, Suda Bharadwaj, Christopher Amato, Rüdiger Ehlers, Ufuk Topcu, and Lu Feng. 2021. *Safe Multi-Agent Reinforcement Learning via Shielding.* International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 483–491.

[10] Shang Erke, Dai Bin, Nie Yiming, Zhu Qi, Xiao Liang, and Zhao Dawei. 2020. An improved A-Star based path planning algorithm for autonomous land vehicles. *International Journal of Advanced Robotic Systems* 17, 5 (2020), 1729881420962263. https://doi.org/10.1177/1729881420962263 arXiv:https://doi.org/10.1177/1729881420962263

[11] Fatma Faruq, David Parker, Bruno Laccrda, and Nick Hawes. 2018. Simultaneous Task Allocation and Planning Under Uncertainty. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 3559–3564. https://doi.org/10.1109/IROS.2018.8594404

[12] Joel Greenyer, Larissa Chazette, Daniel Gritzner, and Eric Wete. 2018. A Scenario-Based MDE Process for Dynamic Topology Collaborative Reactive Systems - Early Virtual Prototyping of Car-to-X System Specifications. In *Joint Proceedings of the Workshops at Modellierung 2018 co-located with Modellierung 2018, Braunschweig, Germany, February 21, 2018.* 111–120. http://ceur-ws.org/Vol-2060/mekes8.pdf

[13] Joel Greenyer, Daniel Gritzner, Timo Gutjahr, Florian König, Nils Glade, Assaf Marron, and Guy Katz. 2017. ScenarioTools – A tool suite for the scenario-based modeling and analysis of reactive systems. *Science of Computer Programming* 149 (2017), 15 – 27. https://doi.org/10.1016/j.scico.2017.07.004 Special Issue on MODELS'16.

[14] Meng Guo, Karl H. Johansson, and Dimos V. Dimarogonas. 2013. Motion and action planning under LTL specifications using navigation functions and action description language. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 240–245. https://doi.org/10.1109/IROS.2013.6696359

[15] Hans Hansson and Bengt Jonsson. 1994. A logic for reasoning about time and reliability. *Formal Aspects of Computing* 6, 5 (01 Sep 1994), 512–535. https://doi.org/10.1007/BF01211866

[16] Keliang He, Morteza Lahijanian, Lydia E. Kavraki, and Moshe Y. Vardi. 2017. Reactive synthesis for finite tasks under resource constraints. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 5326–5332. https://doi.org/10.1109/IROS.2017.8206426

[17] Georg Heppnerl, Nils Berg, David Oberacker, Niklas Spielbauer, Arne Roennau, and Rüdiger Dillmann. 2023. Distributed Behavior Trees for Heterogeneous Robot Teams. In *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*. 1–8. https://doi.org/10.1109/CASE56687.2023.10260300

[18] Nils Jansen, Bettina Könighofer, Sebastian Junges, Alex Serban, and Roderick Bloem. 2020. Safe Reinforcement Learning Using Probabilistic Shields (Invited Paper). In *31st International Conference on Concurrency Theory (CONCUR 2020) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 171)*, Igor Konnov and Laura Kovács (Eds.). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 3:1–3:16. https://doi.org/10.4230/LIPIcs.CONCUR.2020.3

[19] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. 1996. Reinforcement learning: A survey. *Journal of artificial intelligence research* 4 (1996), 237–285.

[20] Mizuho Katayama, Shumpei Tokuda, Masaki Yamakita, and Hiroyuki Oyama. 2020. Fast LTL-Based Flexible Planning for Dual-Arm Manipulation. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 6605–6612. https://doi.org/10.1109/IROS45743.2020.9341352

[21] Bettina Könighofer, Mohammed Alshiekh, Roderick Bloem, Laura Humphrey, Robert Könighofer, Ufuk Topcu, and Chao Wang. 2017. Shield synthesis. *Formal Methods in System Design* 51, 2 (01 Nov 2017), 332–361. https://doi.org/10.1007/s10703-017-0276-9

[22] M. Kwiatkowska, G. Norman, and D. Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-time Systems. In *Proc. 23rd International Conference on Computer Aided Verification (CAV'11) (LNCS, Vol. 6806)*, G. Gopalakrishnan and S. Qadeer (Eds.). Springer, 585–591.

[23] Bruno Lacerda, David Parker, and Nick Hawes. 2014. Optimal and dynamic planning for Markov decision processes with co-safe LTL specifications. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 1511–1516. https://doi.org/10.1109/IROS.2014.6942756

[24] Axel Legay, Anna Lukina, Louis Marie Traonouez, Junxing Yang, Scott A. Smolka, and Radu Grosu. 2019. *Statistical Model Checking*. Springer International Publishing, Cham, 478–504. https://doi.org/10.1007/978-3-319-91908-9_23

[25] Yibo Li, Zixin Wang, and Senyue Zhang. 2022. Path Planning of Robots Based on an Improved A-star Algorithm. In *2022 IEEE 5th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, Vol. 5. 826–831. https://doi.org/10.1109/IMCEC55388.2022.10019799

[26] S. Mahdavi-Hezavehi, P. Avgeriou, and D. Weyns. 2017. Chapter 3 - A Classification Framework of Uncertainty in Architecture-Based Self-Adaptive Systems With Multiple Quality Requirements. In *Managing Trade-Offs in Adaptable Software Architectures*, Ivan Mistrik, Nour Ali, Rick Kazman, John Grundy, and Bradley Schmerl (Eds.). Morgan Kaufmann, Boston, 45–77. https://doi.org/10.1016/B978-0-12-802855-1.00003-4

[27] Shahar Maoz and Jan Oliver Ringert. 2018. On the software engineering challenges of applying reactive synthesis to robotics. In *Proceedings of the 1st International Workshop on Robotics Software Engineering, RoSE@ICSE 2018, Gothenburg, Sweden, May 28, 2018*, Federico Ciccozzi, Davide Di Ruscio, Ivano Malavolta, Patrizio Pelliccione, and Andreas Wortmann (Eds.). ACM, 17–22. https://doi.org/10.1145/3196558.3196561

[28] Shahar Maoz and Jan Oliver Ringert. 2021. Spectra: a specification language for reactive systems. *Software and Systems Modeling* (14 Apr 2021). https://doi.org/10.1007/s10270-021-00868-z

[29] David Martínez, Guillem Alenyà, and Carme Torras. 2015. Safe robot execution in model-based reinforcement learning. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 6422–6427. https://doi.org/10.1109/IROS.2015.7354295

[30] Alejandro Marzinotto, Michele Colledanchise, Christian Smith, and Petter Ögren. 2014. Towards a unified behavior trees framework for robot control. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 5420–5427. https://doi.org/10.1109/ICRA.2014.6907656

[31] Petter Ogren. 2012. *Increasing Modularity of UAV Control Systems using Computer Game Behavior Trees.* American Institute of Aeronautics and Astronautics. https://doi.org/10.2514/6.2012-4458 arXiv:https://arc.aiaa.org/doi/pdf/10.2514/6.2012-4458

[32] Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. 2006. Synthesis of Reactive(1) Designs. In *Verification, Model Checking, and Abstract Interpretation*, E. Allen Emerson and Kedar S. Namjoshi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 364–380.

[33] Oliver Porges, Theodoros Stouraitis, Christoph Borst, and Maximo A. Roa. 2014. Reachability and Capability Analysis for Manipulation Tasks. In *ROBOT2013: First Iberian Robotics Conference*, Manuel A. Armada, Alberto Sanfeliu, and Manuel Ferre (Eds.). Springer International Publishing, Cham, 703–718.

[34] M. Pouranvari and S. P. H. Marashi. 2013. Critical review of automotive steels spot welding: process, structure and properties. *Science and Technology of Welding and Joining* 18, 5 (01 Jul 2013), 361–403. https://doi.org/10.1179/1362171813Y.0000000120

[35] Philipp Schillinger, Mathias Bürger, and Dimos V. Dimarogonas. 2018. Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems. *The International Journal of Robotics Research* 37, 7 (2018), 818–838. https://doi.org/10.1177/0278364918774135 arXiv:https://doi.org/10.1177/0278364918774135

[36] R.S. Sutton and A.G. Barto. 1998. Reinforcement Learning: An Introduction. *IEEE Transactions on Neural Networks* 9, 5 (1998), 1054–1054. https://doi.org/10.1109/TNN.1998.712192

[37] Jana Tumova, Alejandro Marzinotto, Dimos V. Dimarogonas, and Danica Kragic. 2014. Maximally satisfying LTL action planning. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 1503–1510. https://doi.org/10.1109/IROS.2014.6942755

[38] Maximilian Wagner, Peter Heß, Sebastian Reitelshöfer, and Jörg Franke. 2017. Reachability analysis for cooperative processing with industrial robots. In *2017 22nd IEEE International Conference on Emerging Technologies and Factory*

Automation (ETFA). 1–6. https://doi.org/10.1109/ETFA.2017.8247646

[39] A. Wald. 1945. Sequential Tests of Statistical Hypotheses. *The Annals of Mathematical Statistics* 16, 2 (1945), 117 – 186. https://doi.org/10.1214/aoms/1177731118

[40] Christopher J. C. H. Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning* 8, 3 (01 May 1992), 279–292. https://doi.org/10.1007/BF00992698

[41] Eric Wete, Joel Greenyer, Daniel Kudenko, and Wolfgang Nejdl. 2024. Multi-Robot Motion and Task Planning in Automotive Production Using Controller-based Safe Reinforcement Learning. https://doi.org/10.5281/zenodo.10585700

[42] Eric Wete, Joel Greenyer, Daniel Kudenko, Wolfgang Nejdl, Oliver Flegel, and Dennes Eisner. 2022. A Tool for the Automation of Efficient Multi-Robot Choreography Planning and Execution. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings* (Montreal, Quebec, Canada) *(MODELS '22)*. Association for Computing Machinery, New York, NY, USA, 37–41. https://doi.org/10.1145/3550356.3559090

[43] Eric Wete, Joel Greenyer, Andreas Wortmann, Oliver Flegel, and Martin Klein. 2021. Monte Carlo Tree Search and GR(1) Synthesis for Robot Tasks Planning in Automotive Production Lines. In *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*. 320–330. https://doi.org/10.1109/MODELS50736.2021.00039