A Tool for the Automation of Efficient Multi-Robot Choreography Planning and Execution

Eric Wete
eric.roslin.wete.poaka@stud.uni-hannover.de
Leibniz Universität Hannover - Volkswagen AG
Hannover - Wolfsburg, Germany

Joel Greenyer
FHDW Hannover
Hannover, Germany

Daniel Kudenko
Wolfgang Nejdl
Leibniz Universität Hannover
Hannover, Germany

Oliver Flegel
Dennes Eisner
Volkswagen AG
Wolfsburg, Germany

ABSTRACT

In the automotive industry, the design, modeling, and planning of multi-robot cells are manual error-prone, and time-expensive tasks. A recent work investigated, using reactive synthesis, approaches to automate robot task planning, and execution. In this paper, we present a tool that realizes a model-at-runtime approach. The tool is integrated with a robot simulation tool, to automate efficient multi-robot choreography planning, and execution. We illustrate the tool using a multi-robot spot welding cell, inspired from an industrial case. Given a virtual model of the production cell, and user constraints definition, the tool can derive a specification for the reactive synthesis. The tool integrates the synthesized controller with the production cell execution, and in real time, optimizes the strategies by considering the uncertainties. The system can select among several correct, and safe actions, the optimal action using AI-based planning techniques, such as the Monte Carlo Tree Search (MCTS) algorithm. We showcase our tool, illustrate its implementation architecture, including how it can support robot experts for automated planning and execution of production cells.

CCS CONCEPTS

• **Software and its engineering** → **Formal language definitions**; • **Theory of computation** → **Automated reasoning**; • **Computing methodologies** → **Robotic planning**; **Planning under uncertainty**.

KEYWORDS

model-driven engineering, multi-robot motion planning, task scheduling, reactive synthesis, AI-based optimization

ACM Reference Format:
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 *ACM/IEEE 25th International Conference on Model Driven Engineering Languages and Systems (MODELS '22 Companion), October 23–28, 2022, Montreal, QC, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3550356.3559090

MODELS '22 Companion, October 23–28, 2022, Montreal, QC, Canada

ACM ISBN 978-1-4503-9467-3/22/10.
https://doi.org/10.1145/3550356.3559090

1 INTRODUCTION

Designing, and planning production cells, like multi-robot spot welding cells, is time-expensive, and error-prone, due to the high complexity of production processes, and requirements. Usually, the key performance indicator of a robot cell performance is its cycle time. Some performance criteria, such as energy consumption, robot path length, robot velocity in the workspace, define nonfunctional requirements. In most cases, robot trajectories must be optimized for minimal cycle time. Moreover, during the planning phase, the designed system usually does not consider unexpected events, such as robot damages, process interruptions, that can occur during system operation. Considering a car production line, which consists of multiple multi-robot cells, and changes that can occur during the planning phase, such as robot workspace constraints, workpiece changes, positioner, or device modifications, it is challenging to plan robot cells, and more challenging when it is done manually.

To address these challenges, researchers recently proposed an approach for multi-robot task scheduling [21] using Monte Carlo Tree Search (MCTS) [4, 9], and reactive synthesis [3, 16] algorithms. Reactive synthesis is defined as an automated process to produce a correct-by-construction implementation from the linear temporal logic (LTL)-based system specification [17]. From the production cell requirements, we showed how to design the corresponding reactive system using the LTL-based specification language SPECTRA [12]. SPECTRA is a specification language to formally describe reactive system requirements, including analysis and reactive synthesis tools such as a synthesizer that ships a correct-by-construction (executable) controller. Furthermore, we illustrated how to integrate the synthesized controller, from the just-in-time synthesis [13], with the robot cell execution. Indeed, the synthesized controller can generate correct and safe strategies. Using the heuristic AI-planning technique MCTS, we can synthesize among these strategies the efficient strategy considering the defined optimization criteria.

This paper presents a tool that implements a model-at-runtime technique, supports robot engineers during the planning of production cells, and obtains a correct-by-construction reactive system. The tool is integrated into a software for 3D modeling, programming and simulation of production cells. Given a 3D model of a production cell, and the requirements, the tool can generate collision-free, and requirements-compliant trajectories of all robots using planning algorithms. Moreover, it can find, through simulations of all possible trajectories combinations, potential collisions that may occur: these collision risks define additional constraints. The tool simulates all trajectories, and stores their time for optimization. In fact, the integrated execution of the production cell with our tool uses these trajectories times for efficient planning. Using a domain-specific language (DSL), the tool formalizes the robot cell definition, the requirements, and constraints including the collision risks. We illustrate our tool using an industrial case of a spot welding multi-robot cell. In a nutshell, we describe, using the tool, the multi-robot cell planning process, and show how to integrate the production cell execution with the reactive system. By continuous monitoring of the production cell, the tool can safe control the robots' choreography under the supervision of the synthesized controller.

This paper is structured as follows. Section 2 describes the development process for the planning and execution of a multi-robot cell. Section 3 explains the tool architecture, and Sect. 4 introduces a use case by showing some highlights of the tool usage. Sect. 5 contrasts our technique, and Sect. 6 concludes this paper.

2 DEVELOPMENT PROCESS OVERVIEW

Figure 1 illustrates the planning and execution process of a multi-robot cell that consists of the following steps.

Step 1: 3D Modeling & System requirements: This step consists in modeling the production cell using a 3D modeling software for robotic systems, and CAD-data of cell components, such as robots, tools, workpiece, and positioner. We used the modeling software RobotStudio[1], and integrated our tool with the software. Using the tool, the robot expert can describe the tasks to be performed including the working points, as well as production cell requirements, and constraints, such as task dependencies. We formalize these requirements, and constraints inside a specification. This is the only manual task activity of our process.

Step 2: Trajectories generation & Collision analysis: Based on the robot cell 3D model, and the requirements, the tool uses an iterative A* algorithm with decreasing step size to generate, for all robots, collisions-free trajectories. The A* algorithm, proposed in [7], addresses the challenge of finding a path with minimal cost using a heuristic cost function based on domain-/problem-specific information. The step size permits to build a grid-layout based environment from the Euclidean space. The tool checks the reachability of each robot, finds the working points each robot can reach, and thus the tasks each robot can perform. Then, the tool simulates all pairs of trajectories to find trajectory combinations having a risk of collision, and store robot movement times for later use. Using the additional derived requirements, such as robot constraints due to robot working range, and collision constraints, the tool updates the formal robot cell specification.

Step 3: Reactive specification & Synthesis: In this step, we generate the reactive specification, namely, the SPECTRA specification, from the formal robot cell specification that was generated in the previous step. SPECTRA includes tools to analyse specifications. It can check if the SPECTRA specification is realizable, and generate, in that case, a controller strategy, otherwise, it can compute a counter-strategy that shows how at least one guarantee of the SPECTRA specification can be violated. Moreover, SPECTRA can check if the environment is well-separated [8, 11], that means, that the system cannot force the environment to violate its assumptions. If an implementation of the SPECTRA specification, also known as strategy, cannot be found, the robot expert must update the requirements, or 3D robot cell model, by restarting from the first step. If a strategy is found, it can be used for the production cell operation, as described in the next step.

Step 4: Optimized & integrated strategy execution: The controller strategy can be executed using a standalone Java application. We extended the SPECTRA controller executor to optimize the system output. In fact, given an environment state, the extended controller executor must select the optimal action among multiple system outputs. The SPECTRA selection strategy does neither consider the robot movement times recorded in the first step, nor the robot interruption model if it exists. Using a Java application, we integrated the controller executor with the heuristic search technique Monte Carlo Tree Search (MCTS), and with the production cell execution. MCTS showed good performance in AI and game tree domains [5, 6, 14, 19], and it can find optimal strategies through look-ahead simulations. Figure 2 illustrates how the SPECTRA controller is integrated with the production cell execution, and how the controller execution is tuned in real time with an AI-based optimizer. The controller execution proactively reacts to unexpected environment events, and the controller can always (re)schedule, within the allocated time, optimal task sequences. Indeed, once the production cell state, i.e., robots' and tasks' status, is captured, and passed to the controller, the controller executor outputs a set of possible actions, guaranteed by the reactive specification. Instead of the SPECTRA selection strategy of the controller output, we use the MCTS-based selection strategy that includes domain-knowledge to find the best action. The chosen output updates the internal controller status, and is transformed to robot specific function calls that execute the corresponding trajectories generated in Step 2.

3 TOOL OVERVIEW

Figure 3 depicts the overall architecture of our tool. It shows the components of the tool including the produced artifacts. We extended the 3D modeling and simulation software RobotStudio to support the robot experts during the production cell design. The extension collects relevant information on the robot cell, like robots, workpiece, positioners, robot devices, as well as requirements that the user defines, such as robot constraints, task dependencies. The tool automatic formalizes the requirements inside a DSL-based requirement specification. Based on the robot cell CAD-data, the tool computes cost-efficient collision-free trajectories of each robot with respect to the requirements, and generates robot programs, and configurations accordingly. The simulation of trajectories computes the movement time model,

[1] https://new.abb.com/products/robotics/robotstudio

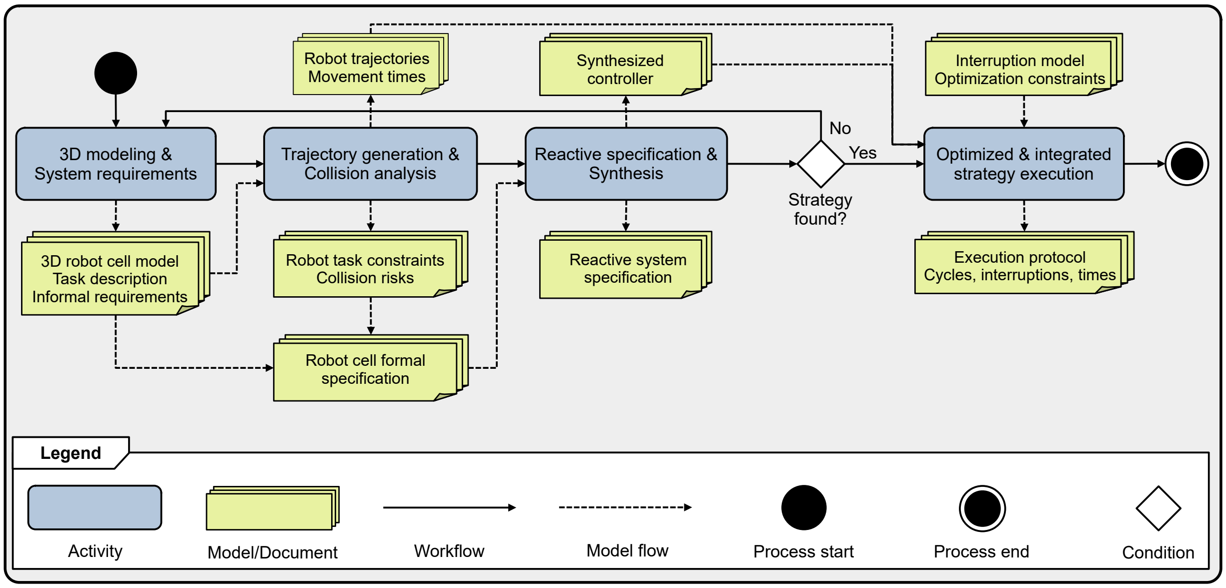

Figure 1: The multi-robot cell planning and execution process

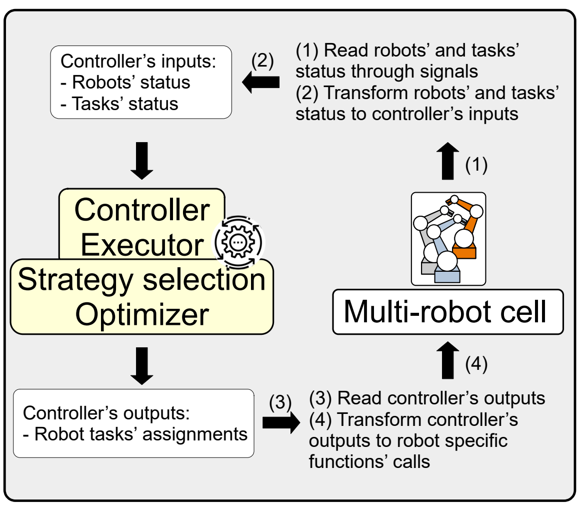

Figure 2: The integrated strategy execution architecture

tasks each robot can perform, and identifies collision-prone trajectories' pairs, and the requirement specification model is updated accordingly. We implemented an eclipse plugin to automatic transform this DSL-based specification to a SPECTRA specification, to obtain a controller through the reactive synthesis [3, 16]. The robot cell execution is integrated with the controller strategy using a Java application based on Spring Boot[2]. Spring Boot is a java-based framework to create standalone based applications, and micro services. The Java application uses the movement times, and the environment model, i.e., robot interruptions model, to enhance the controller execution. The application is hosted on a PC on the same local area network (LAN) where the robot cell is deployed. It interacts with the robot cell through HTTP using REST APIs[3][4] to get robot cell state, and trigger the execution of trajectories and programs.

[2]https://spring.io/projects/spring-boot

4 CASE STUDY

Let us consider the spot welding multi-robot cell illustrated in Fig. 4. The robot cell consists of 2 robots *R0* and *R1*, located at their base location *0* and *5*, respectively. On a positioner, a car body must be weld on 8 locations: *1* to *4*, and *6* to *9*. The welding process requires that the robots weld the corners first, namely, *2*, and *7*.

Once the robot expert defines the production cell, and inputs the tasks, including the dependencies, the extension produces robot trajectories, and necessary configurations of robots *R0* and *R1*. The extension identifies the tasks each robot can perform according to its reach specification, and via simulations of trajectory pairs, detects possible collisions. For example, *R0* can only perform the tasks at welding points *1*, *2*, *3*, *4*, and *6*. A collision is detected when *R0* moves to location *6*, while *R1* moves to location *1*. These constraints, including the requirements, are formally defined in the specification as illustrated in Lst. 1.

The SPECTRA specification encodes the system rules as guarantees, and describes the environment behavior as assumptions. We highlight some aspects of the reactive specification. Listing 2 shows an excerpt of the SPECTRA specification. It states that each robot eventually completes its assigned task except if the robot reports failure or is interrupted. The guarantee defined in Lst. 3 specifies the task constraints of robot *R0*, derived from our tool. The corner constraint related to the corner task *2* is specified in Lst. 4.

[3]https://developercenter.robotstudio.com/api/rwsApi/
[4]https://developercenter.robotstudio.com/api/RWS

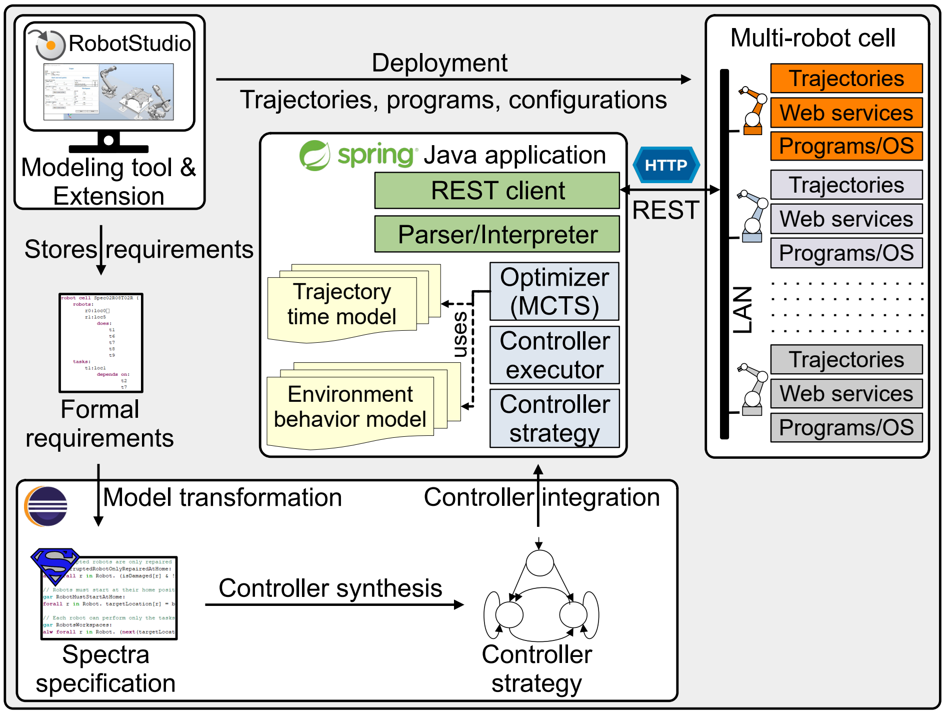

Figure 3: The overall architecture of our tool

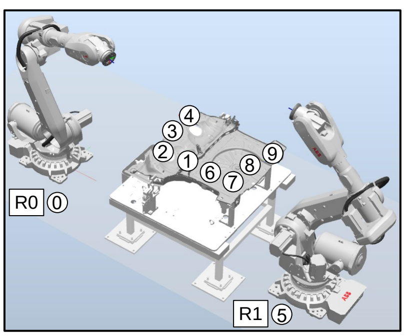

Figure 4: Production cell case study

```
robot cell Models22UC02R08T02R {
    robots:
        r0:p0
            does:
                t1 t2 t3 t4 t6
        r1:p5
            does:
                t1 t6 t7 t8 t9
    tasks:
        t1:p1
            depends on:
                t2 t7
        t2:p2
        t3:p3
            depends on:
                t2
        t4:p4
            depends on:
                t2
        t6:p6
            depends on:
                t2 t7
        t7:p7
        t8:p8
            depends on:
                t7
        t9:p9
            depends on:
                t7
    locations:
        p0 p1 p2 p3 p4 p5 p6 p7 p8 p9
    collisions:
        r0:p6, r1:p1
}
```

Listing 1: The multi-robot cell specification of the use case.

5 RELATED WORK

Sampling-based approaches, like the PRM or RTT methods proposed, respectively, in [1, 15], and [2, 10], address the motion planning problem by building a representation of the environment to support the planning. There also exist graph-based approaches, such as A* to obtain optimal paths [18, 20]. However, we propose an approach that does not only address the motion planning problem, but also can schedule and execute safe multi-robot tasks and movements to complete cycles, considering unexpected environment events. To this end, LTL-based specifications can be used, and combined with planning based methods. This paper's aim is to contribute in this direction by showing the feasibility of this technique (cf. [21]), and its integration with an industrial case including the tools used in the industry by the robot engineers.

6 CONCLUSION

In this paper, we presented a tool for the planning and execution of multi-robot cells. The tool supports robot engineers to

```
asm RobotsWillEventuallyCompleteATask:
  alwEv forall r in Robot.  !isDamaged[r] -> (forall t in
        Location. (targetLocation[r] = t) -> ((isCompleted[t])
        ));
```

Listing 2: An assumption from SPECTRA

```
gar RobotsWorkspace_R0:
  alw targetLocation[0] in {0, 1, 2, 3, 4, 6};
```

Listing 3: A workspace guarantee from SPECTRA

```
gar Dependency_for_2{Robot r}:
  alw (!isCompleted[2]) -> (targetLocation[r] not in {1, 3, 4,
         6});
```

Listing 4: A task dependency guarantee from SPECTRA

automatic produce cost-efficient robot trajectories, and computes, in real time, safe, and optimal task sequences guaranteed by a reactive specification. We described the tool-supported development process, and the tool architecture that includes the components required to construct the multi-robot system. We combined our tool with the integrated development environment of robot engineers to collect requirements, constraints, and produce robot programs including trajectories, as well as configurations for each robot. We illustrated how to obtain a correct-by-construction reactive system from the formal description of the requirements. Moreover, we highlighted how to integrate, with a Java application, the synthesized controller, and the production cell execution. More interesting, our tool can be used easily used by robot experts, since it does not require prior knowledge on writing reactive specification. The provided DSL can easily be used at early stages to check some properties, such as system realizability, before starting with the 3D modeling. For example, given the type of robot, or the robot-mounted position in the production cell, a robot workspace can be approximated. Then, using the DSL, the robot expert knows beforehand if all tasks can be done using the chosen robot or cell configuration.

For future development, it would be interesting to suggest to the robot engineers, in case of unrealizability, hints to produce a realizable system, for example, by proposing new locations where robots can be mounted. An investigation could also be to propose, the optimal robot-mounted positions, as well as robot specifications, given the description of tasks, and the available robots.

ACKNOWLEDGMENTS

Thanks to Shahar Maoz, and Ilia Shevrin for their support for the integration, and usage of Spectra.

REFERENCES

[1] O. Arslan, K. Berntorp, and P. Tsiotras. 2017. Sampling-based algorithms for optimal motion planning using closed-loop prediction. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 4991–4996. https://doi.org/10.1109/ICRA.2017.7989581

[2] Oktay Arslan and Panagiotis Tsiotras. 2015. Dynamic programming guided exploration for sampling-based motion planning algorithms. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 4819–4826. https://doi.org/10.1109/ICRA.2015.7139869

[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] Rémi Coulom. 2007. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In *Computers and Games*, H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. (Jeroen) Donkers (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 72–83.

[5] Sylvain Gelly, Levente Kocsis, Marc Schoenauer, Michèle Sebag, David Silver, Csaba Szepesvári, and Olivier Teytaud. 2012. The Grand Challenge of Computer Go: Monte Carlo Tree Search and Extensions. *Commun. ACM* 55, 3 (mar 2012), 106–113. https://doi.org/10.1145/2093548.2093574

[6] Sylvain Gelly and David Silver. 2011. Monte-Carlo Tree Search and Rapid Action Value Estimation in Computer Go. *Artif. Intell.* 175, 11 (jul 2011), 1856–1875. https://doi.org/10.1016/j.artint.2011.03.007

[7] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics* 4, 2 (July 1968), 100–107. https://doi.org/10.1109/TSSC.1968.300136

[8] Uri Klein and Amir Pnueli. 2011. Revisiting Synthesis of GR(1) Specifications. In *Hardware and Software: Verification and Testing*, Sharon Barner, Ian Harris, Daniel Kroening, and Orna Raz (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 161–181.

[9] Levente Kocsis and Csaba Szepesvári. 2006. Bandit Based Monte-Carlo Planning. In *Machine Learning: ECML 2006*, Johannes Fürnkranz, Tobias Scheffer, and Myra Spiliopoulou (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 282–293.

[10] Steven M. LaValle. 1998. Rapidly-exploring random trees : a new tool for path planning. *The annual research report* (1998).

[11] Shahar Maoz and Jan Oliver Ringert. 2016. On Well-Separation of GR(1) Specifications. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Seattle, WA, USA) *(FSE 2016)*. Association for Computing Machinery, New York, NY, USA, 362–372. https://doi.org/10.1145/2950290.2950300

[12] 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

[13] Shahar Maoz and Ilia Shevrin. 2020. Just-in-Time Reactive Synthesis. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering* (Virtual Event, Australia) *(ASE '20)*. Association for Computing Machinery, New York, NY, USA, 635–646. https://doi.org/10.1145/3324884.3416557

[14] M. Müller, M. Enzenberger, B. Arneson, and R. Segal. 2010. Fuego – an open-source framework for board games and Go engine based on Monte-Carlo tree search. *IEEE Trans. Comput. Intell. AI in Games* 2 (2010). https://doi.org/10.1109/TCIAIG.2010.2083662

[15] Sean Murray, Will Floyd-Jones, Ying Qi, Daniel Sorin, and George Konidaris. 2016. Robot Motion Planning on a Chip. In *Robotics: Science and Systems*. https://doi.org/10.15607/RSS.2016.XII.004

[16] 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.

[17] A. Pnueli and R. Rosner. 1989. On the Synthesis of a Reactive Module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (Austin, Texas, USA) *(POPL '89)*. Association for Computing Machinery, New York, NY, USA, 179–190. https://doi.org/10.1145/75277.75293

[18] Erke Shang, Bin Dai, Yiming Nie, Qi Zhu, Liang Xiao, and Dawei Zhao. 2020. A Guide-line and Key-point based A-star Path Planning Algorithm For Autonomous Land Vehicles. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. 1–7. https://doi.org/10.1109/ITSC45102.2020.9294336

[19] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. 2016. Mastering the Game of Go with Deep Neural Networks and Tree Search. *Nature* 529, 7587 (Jan. 2016), 484–489. https://doi.org/10.1038/nature16961

[20] Yu Song and Pengcheng Ma. 2021. Research on Mobile Robot Path Planning Based on Improved A-star Algorithm. In *2021 International Conference on Electronic Information Engineering and Computer Science (EIECS)*. 683–687. https://doi.org/10.1109/EIECS53707.2021.9588002

[21] 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