A Robust and Flexible System Architecture for Facing the RoboCup Logistics League Challenge

Thomas Ulz[1], Jakob Ludwiger[2], and Gerald Steinbauer[3](✉)

[1] Institute for Technical Informatics,
Graz University of Technology, Graz, Austria
[2] Institute for Control and Automation,
Graz University of Technology, Graz, Austria
[3] Institute for Software Technology,
Graz University of Technology, Graz, Austria
steinbauer@ist.tugraz.at

Abstract. In this paper we present the software architecture of the GRIPS team for addressing the challenges of the RoboCup Logistics League. The guiding principles for the development of the architecture origin in the research focus of the involved institutes on dependable intelligent systems. The architecture enables most flexible planning of the tasks as well as a most reliable execution of the generated task list.

1 Introduction

Due to increasing demands on flexibility in terms of product configuration as well as delivery time triggered by the trend in e-commerce (e.g. on-line configurators, on-line shopping) production needs to become more flexible as well as more digitized. This trend is well known under terms like *flexible production* or *Industry 4.0*. Usually in order to facilitate reasonable prices for products as well as to guarantee sustainable product quality and fast availability of goods production is heavily automatized. Often, this automation is not very flexible, and thus, in contradiction with the demands on flexibility in configuration (in extreme cases lot size one) and availability. Fortunately, these demands on flexibility and digitization in production require new concepts and open interesting and challenging research questions ranging from Robotics over the Internet of Things (IoT) and multi-agent systems to planning and scheduling. In order to provide an interesting and appealing show case that allows research and teaching in the area of flexible production within the RoboCup initiative [16] a competition called the RoboCup Logistics League (RCLL) was founded. It resembles the setting of a flexible production plant. The RCLL competition posts a number of challenges ranging from Robotics over IoT to Artificial Intelligence and can be used to develop and evaluate new concepts in production.

In this paper we like to introduce the system architecture of the team *Graz Intelligent Robust Production System* (GRIPS) which allowed GRIPS to win

© Springer Nature Switzerland AG 2019
D. Holz et al. (Eds.): RoboCup 2018, LNAI 11374, pp. 488–499, 2019.
https://doi.org/10.1007/978-3-030-27544-0_40

the international RCLL competition in 2018. The team comprises of students and researchers of 3 different institutes of the Graz University of Technology that share a common interest in safe and dependable intelligent systems [2,6,8]. The dynamic setting of the RCLL involving numerous items such as robot and production machines interacting in a real world environment is a perfect testbed for techniques to realize robust complex systems. Thus, in this paper we focus on the aspects of the developed system architecture related to robustness, reactivity, and liveness. We will describe how these properties are achieved on the different system layers ranging from an abstract planning and scheduling module over a robust executive layer to reactive behaviors.

2 Logistics League

The RCLL [1,10] is part of the RoboCup initiative and focuses on the stimulation of the development of approaches in Robotics and Artificial Intelligence using robotics competitions. In this league the goal is that a team of autonomous robots in cooperation with a set of production machines produces individualized products on demand. Two teams share a common factory floor of the size of 14 m × 8 m. Each team comprises of up to 3 autonomous robots and owns 7 machines. Machines are represented by Modular Production Systems (MPS) provided by Festo. See Fig. 1 for an example setup.

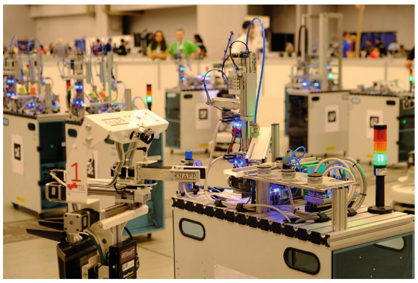
Fig. 1. Physical setup of the RoboCup Logistics League.

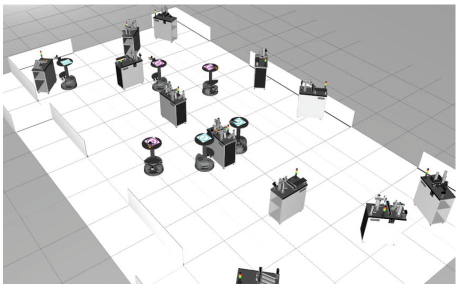
Fig. 2. Simulation setup of the RoboCup Logistics League.

There are different types of machines that resemble different production steps like fetching raw material, assembling parts, or delivering final products. The task of the teams is to develop methods that coordinate the robots (which are mobile) and machines (which are static) that allow producing and delivering requested goods in time. All involved entities are allowed to communicate via WiFi. Robots are cooperative in the sense that they need to interact physically with the machines, e.g. fetching raw material from a dispenser machine or provide an intermediate product to a machine that refines it. Usually teams use some coordination server that collects information from robots, machines, and a central production management system that coordinates the necessary tasks. The products are mimicked by stacks of bases, rings, and caps of different colors. The configuration of the components is flexible and determines the complexity of a product. In general, several refining steps of intermediate products by different machines are required to produce a final product.

This setup for products was selected to have a physical interaction among the robots, the machines, and the products. A central agent named referee box randomly generates product orders with varying configurations and delivery time windows. These orders are communicated to the teams that need to derive a production schedule and distribute the tasks among the robots and machines. Based on the complexity of the product and if the delivery windows was met points are awarded to the teams. For the most complex products usually up to 10 different steps like fetching and delivering material to machines are needed. The actual number depends on the planning representation. Some of them might be parallelized or rescheduled in order to optimize the awarded points. In order to simulate a real world production environment, machines go out of service on a random basis which asks for flexibility in the production planning. The team that collects the most points during 17 min of production time wins the game.

The referee box is able to run games and scoring automatically. Together with a full-fledged simulation [19] (see Fig. 2 for an example setup), it forms an advanced benchmarking system for flexible production approaches [10,11].

The interesting aspect of the RCLL setting is its resemblance to flexible on-demand production sites while abstracting it to not involve any physically changes of the product. Given that, this the RCLL posts challenges in the full range from Robotics (e.g. navigation, precise manipulation) over communication and multi-agent systems (e.g. reliable communication, reliable task execution) to planning and scheduling (e.g. generate production plans, execution monitoring and re-planning).

3 Software Architecture

The main aspects for realizing a multi-robot system as required in the RCLL are (1) planning and scheduling, (2) plan refinement and execution, (3) behavior and control, and (4) low-level functionality. In the following sections we are going to present selected topics for all aspects, except low-level functionality such as navigation. A general overview of the software architecture we apply in our system can be seen in Fig. 3. As depicted there, the software architecture spans over multiple physical systems. The planning and scheduling instance is deployed on a so-called *teamserver* which has *global* knowledge of the current game comprising information from the RCLL referee box such as requested orders and from all active robots such as the status of task execution. The teamserver controls all robots and interacts as a gateway between them and the referee box. The modules running on the robots comprises an executive, a behavior and control module, and low-level functionality.

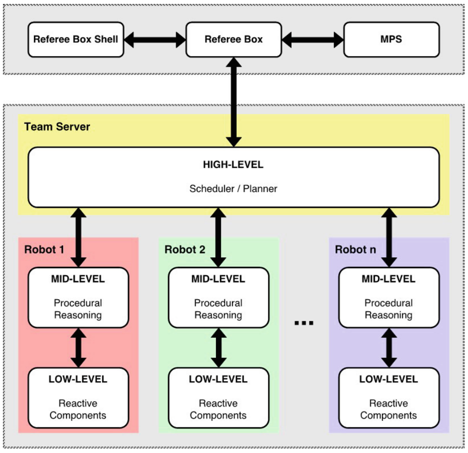

Fig. 3. Overview of distributed software architecture in our approach.

3.1 Planning and Scheduling

Planning and scheduling in our approach is based on splitting any order that is received from the RCLL referee box into subtasks that cannot be split further. The representation of the task on this level is rather abstract because we like to limit the complexity in planning and there exists a task refinement in the executive layer. This idea is inspired by the concept of hierarchical task network planning [5]. In our system, we distinguish between two subtask categories where we assume that one robot is only able to carry a single item:

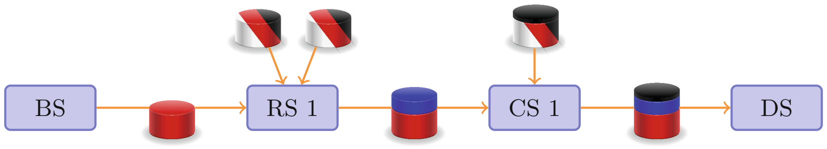

Fig. 4. Example production chain for an order of complexity *C1*, adapted from RCLL rulebook [3]. This order requires 2 additional workpieces at the ringstation *RS1* and a cap loaded at capstation *CS1*.

1. **GET:** A GET task implies that the robot needs to navigate to a given MPS, where a workpiece is fetched by the robot, usually after sending some instruction to the MPS to initiate for instance a material dispense.
2. **DELIVER:** A DELIVER task involves the robot navigating to a given MPS, where the carried workpiece is then deposed. This is usually followed by the robot sending some instruction to the MPS like mounting a ring.

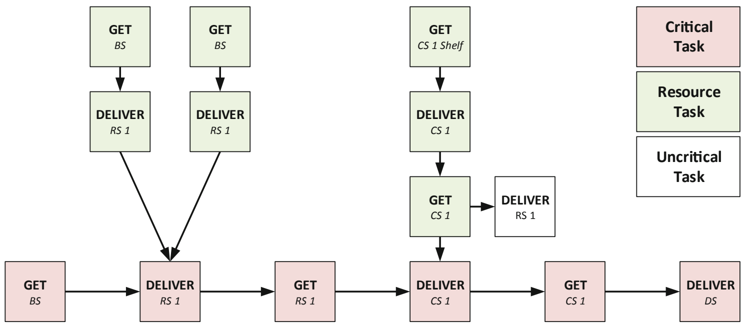

Fig. 5. Dependency graph for tasks required to build and deliver a product for the *C1* order that is shown in Fig. 4. Failure of critical tasks lead to complete cancellation of the respective product, while resource tasks can be reassigned. Failure of uncritical tasks have no influence on the overall goal of delivering a complete C1.

Since getting and delivering a workpiece reasonably needs to be done by the same robot, this specific choice of subtask types might seem counterintuitive. However, by separating the pickup and deliver process, MPSs can be freed from a workpiece that would otherwise block the MPS for other robots. The successful execution of these subtasks is then arranged and monitored by the system that will be discussed in Sect. 3.2. For simplicity, we assume here, that this system is capable of providing information of successful or unsuccessful task execution.

Task Generation. Any order that is received from the RCLL referee box specifies the color required for each workpiece that is used in the production process. Therefore, any order implicitly defines which MPSs need to be used during the production process. An example production chain for an order of easy-to-medium complexity *C1* (meaning that one ring needs to be mounted before the cap) is shown in Fig. 4. Based on this production chain, our scheduler creates the required subtasks and the corresponding dependency graph for this tasks using the ideas of HTN refinement. The resulting dependency graph for the production chain shown in Fig. 4 is then depicted in Fig. 5. As can be seen there, subtasks belong to one of the following three categories:

1. **Critical Tasks** represent the actual production flow where the requested product is assembled by the MPSs using the workpieces already loaded into the respective MPSs. If such a critical task fails, the product that currently is assembled cannot be reasonably recovered, and thus, production of this product is canceled. Depending on the current game's context, assembly of the same product might be started again.
2. **Resource Tasks** load the MPSs with workpieces that are required for the assembly of products. If resource tasks fail, the actual assembly of the product is not harmed and thus, these tasks can be reassigned until successfully completed. However, assembly of a product might be severely delayed due to resource tasks failing.

3. **Uncritical Tasks** neither influence the successful completion of the currently assembled product, nor do they (directly) influence assembly time of that product. However, if successfully completed, these tasks might have a positive effect by speeding up future assembly processes.

Task Scheduling. The assignment of tasks to respective robots in our system is done based on a *request-response* approach. This means, robots that currently do not own tasks request new tasks from the central planning and scheduling instance. For task scheduling, three scenarios might occur that we are going to discuss in the following paragraphs. To do so, we define the following symbols:

- τ: a given task.
- $pred(\tau)$: set of all predecessor tasks of τ based on the task dependency graph.
- $\tau.type$: the task's type which is one of $\{GET, DELIVER\}$.
- $\tau.state$: the task's state which is one of $\{SUCCESS, FAIL, UNASSIG-NED\}$.
- $\tau.robot$: the robot to which the task τ was assigned.
- $\tau.machine$: the machine which which the robot interacts in this task. The machine is one of $\{BS, CS1, CS2, RS1, RS2, SS, DS\}$.
- ξ: a given product.
- $\xi.\tau$: all tasks that are required to assembly product ξ.
- $\xi.machines$: all machines the robots need to interact with during assembly of this product.
- $\mathcal{T}$: the set containing all currently active tasks.
- ρ: the current robot that is requesting a task.

1. Task in active assembly. In the simpler of the two cases, a task in an already active production process for a given product can be found for the robot requesting a new task. That is, the set of tasks Φ that could be assigned to the robot according to (1) is not empty. In our system, this is the preferred case, and thus, scheduling of tasks is always greedy in a sense that the scheduler aims at finishing products as quickly as possible. Any robot requesting a task is assigned a randomly selected one $\tau \subseteq \Phi$.

$$\Phi = \begin{cases} \Psi, & \text{if } \Psi \neq \emptyset \\ \Theta, & \text{if } \Psi = \emptyset \end{cases} \tag{1}$$

Where Ψ and Θ are defined as follows.

$$\begin{aligned} \Psi = \{\tau : & \tau.type = DELIVER \\ & \wedge \exists\ pred(\tau).robot = \rho \wedge \forall\ pred(\tau).state = SUCCESS\} \end{aligned} \tag{2}$$

That is, the set Ψ contains all tasks of type DELIVER for which a successfully finished predecessor task was already assigned to the same robot. In general, the set will only contain one task.

$$\Theta = \{\tau : \forall\ pred(\tau).state = SUCCESS\} \cup \{\tau : pred(\tau) = \emptyset\} \tag{3}$$

That is, the set Θ contains all tasks for which all predecessor tasks have been finished successfully. Of course, $\Psi \subseteq \Theta$ holds.

2. Start new assembly. If no task in the current assembly process needs to be done, the planning and scheduling instance determines whether the assembly of an additional product can be started. To do so, it is determined if a parallel production chain can be found where no machine (besides BS and DS) overlap, such that no deadlock can occur. This mechanism is formalized in (4).

$$\Omega = \{\xi : \xi.machines \ \cap \mathcal{T}.machines = \emptyset\} \tag{4}$$

If the set Ω contains an additional product for which assembly can be started. Tasks are then selected according to the previous section for the newly to be assembled product. However, considering that any production chain includes mounting a cap to finish the currently assembled product, in our current architecture a maximum of two parallel production chains can be processed. Note that each team has 1 base station, 2 ring stations, 2 cap stations, and 1 delivery station in their MPS set. We did not use the 7^{th} machine - the storage station - in this implementation.

3. "Dummy" task. If no production relevant task can be found for a robot requesting a new task, that is, if $\Phi = \emptyset \wedge \Omega = \emptyset$, the robot is assigned so-called dummy tasks such that it is not blocking any relevant MPS while having no task. In our system, a dummy task consists of sending the robot to a random zone, such that it is constantly moving while having no production relevant task.

3.2 Executive

The bridging between the abstract planning and scheduling and the practical behavior layer is established by an executive layer that runs separately on each robot. The two main functions of the executive are the refinement of the abstract tasks to executable behaviors and the supervision of the entire task execution. The separation of the two functions contributes to the robustness of the overall architecture as the former allows the system to use a flexible abstracted planning approach while the latter allows to reactive to uncertainties and unexpected situations in the interaction between the physical robot and its environment.

We realized the executive layer following the well-known concept of belief-desire-intention (BDI) [4] using the open-source implementation OpenPRS [7]. In order to allow robust and reactive control of robots the approach follows the idea of practical reasoning where the tasks to be fulfilled are represented by goals and goals are pursuit using scripted recipes called procedure. Procedures are represented as directed graph with further sub-goals on the edges. Possible sub-goals are non-primitive goals (further goals), queries (simple queries to a knowledge base), information updates (asserting and retracting facts to the knowledge base), and primitive actions (representing executable behaviors). Goals may also be combined using special modifiers such as maintain where one goal is permanently active until another goal is achieved. The robustness and reactivity of a BDI system results from the execution semantics where the interpreter tries all applicable procedures and valid execution traces within recipes to achieve a given goal and the fact that instead of expensive reasoning (e.g. resolution) a simpler matching process between goals and procedures is used. The response to a posted goal or sub-goal is either success (all sub-goals were achieved) or fail (the interpreter were not able to achieve all sub-goals).

The interaction with the other parts of the architecture works as follows. Any time the robot becomes idle it requests a new task from the planning and scheduling component. The tasks assigned to a robot by this component are mapped to configurable goals. Currently we have corresponding goals for the get, delivery, and dummy task with corresponding hand-crafted procedures. The executable basic behaviors like navigating to a given position, alignment at a machine, or grasping an item are represented by primitive goals that lead directly to a behavior execution. The physical execution is realized using the action-server concept of the Robot Operating System (ROS) [14]. But each primitive goal is wrapped by a safe version of the original goal to achieve dependable execution. These goals comprise additional hand-crafted monitoring and fault-recovery recipes. These safe goals are reused when structuring the recipes for the top-level goals.

The communication between the planning and execution layer is based on abstract positions like `C-BS-Input` representing the position for the conveyor input of the cyan base station. In order to ground such positions or make conclusions such as that robot is close we use the transformation framework of ROS (there is a proper transformation for each abstract position maintained by the behavior layer) and the concept of evaluable predicates and functions provided by OpenPRS (oracles for the evaluation of predicates and functions are implemented in the behavior layer).

We like to point out the difference in planning and execution to previous attempts reported in [12]. In contrast we use OpenPRS only as an executive to execute tasks while task scheduling is done in the team server. Moreover, in contrast to the Clips-based approach we follow a clear separation of the abstract task scheduling and the task execution rather than performing the overall reasoning in an reactive manner using a rule-based system.

3.3 Behavior and Control

Several software components are implemented in the behavior and control layer of the software architecture. These components comprise navigation, alignment to machines, identification and localization of machines, and identifying and manipulation of products. In this paper the control strategy which enables the precise alignment of the robot in front of the machine during production will be explained in detail. This behavior is the base for reliable manipulation of products. In order to grasp or place products during production, the robot needs the ability to align itself at very short distances and with very high precision in front of machines. Achieving these criteria with the usual navigation approaches [9] already implemented in ROS is not possible. However, this is a typical task for classical feedback control. The two parts necessary for feedback control are the error computation and the controller design. These two parts will be described in detail in the following two subsections.

Error Computation. To perform closed loop control, the current positioning error has to be computed. The robots are equipped with a laser scanner at the front, which can be used to compute the position of the machine relative to the robot. Given the fact that all machines in the logistics league have the same rectangular base shape and assuming that the robot is roughly facing the machine (this is achieved using the navigation methods mentioned above), the relative position of the machine can be estimated using a very basic clustering algorithm. As the robot faces the machine, the central laser scan measurement is the root of the cluster. Starting from this root, every measurement value with Euclidean distance to the cluster smaller than a predefined threshold is added to the cluster. Applying classical least squares line fitting (see [13]) gives the angular error and using the edge points of the cluster results in the positioning error. Figure 6 shows a typical laser scan reading in gray rays with red tips where the robot faces a machine. The clustered data is depicted in blue and the estimated position of the machine as a green square. Based on the estimate of the machine pose, the three errors for position e_x and e_y as well as the angular error e_φ are computed and fed into the controller.

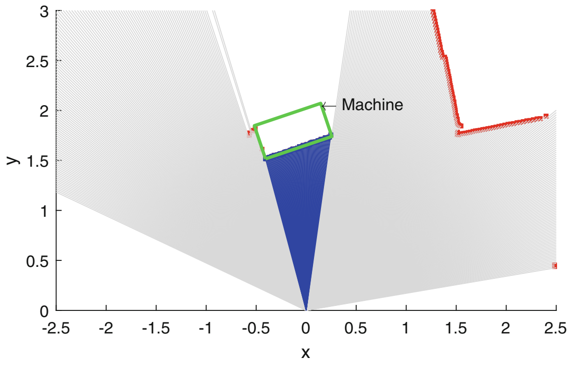

Fig. 6. Machine position estimation (Color figure online)

Control Algorithm. For each of the three component of the error (position e_x, e_y and angular e_φ) a sliding mode controller is designed (see [18]). Sliding mode control was chosen, because it is a very simple to implement, easy to tune but also represents a robust control strategy. The basic concept of first order sliding mode control will be explained by means of an example. Consider a continuous integrator

$$\frac{dx}{dt} = u + f \tag{5}$$

with state $x \in \mathbb{R}$, input $u \in \mathbb{R}$ and the bounded perturbation $\sup|f| = \bar{f}$. Applying a first order sliding mode control law

$$u = -\rho \operatorname{sign}(x) \tag{6}$$

with parameter $\rho > \bar{f}$ yields the closed loop system

$$\frac{dx}{dt} = -\rho \operatorname{sign}(x) + f. \tag{7}$$

As the parameter $\rho > \bar{f}$, the controller always dominates the perturbation f. The reader interested in the theoretical property of sliding mode control exactly compensating perturbations is referred to [15,17,18] and getting familiar with differential inclusions as well as with Filipov's theory. The part $-\rho \operatorname{sign}(x)$ also dominates the perturbation f which results in a movement towards the origin from any initial condition. Typical trajectories for the unperturbed case ($f = 0$) using first order sliding mode control is shown in Fig. 7 for the two initial conditions $x_0^{(1)} = 1.125$ and $x_0^{(2)} = -2.125$. One can see the typical finite time convergence which is also a very good property of sliding mode control. However, the main drawback of this control strategy is also visible in this figure because the state converges to a vicinity around zero and performs a zig-zag motion called chattering. This chattering appears in real world applications due to finite switching frequencies.

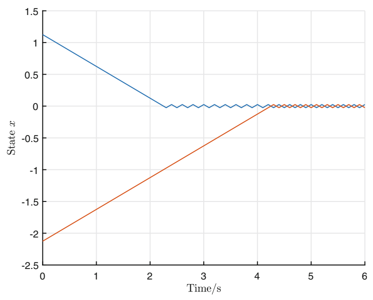

Fig. 7. Typical trajectories resulting with first order sliding mode control for two initial conditions and perturbation.

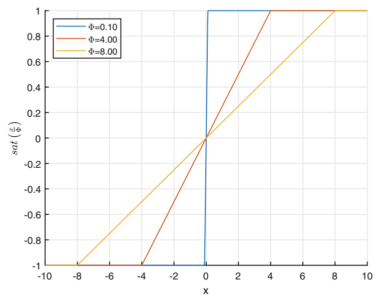

Fig. 8. Influence of parameter Φ on sign function approximation.

In order to reduce this chattering phenomenon, several approximations of the sign function are proposed (see [15]). In the remainder of this paper the $\operatorname{sign}(\cdot)$ function is approximated by the saturation function

$$\operatorname{sat}(\gamma) = \begin{cases} \gamma & -1 \leq \gamma \leq 1 \\ \operatorname{sign}(\gamma) & \text{else} \end{cases} \tag{8}$$

which results in the control law

$$u = -\rho \operatorname{sat}\left(\frac{x}{\Phi}\right) \tag{9}$$

with parameter $\Phi \in \mathbb{R}^+$ specifying the slope of the approximation as depicted in Fig. 8.

Remark 1. The parameter Φ offers a possibility to find a tradeoff between accuracy and chattering alleviation. Please note that $\operatorname{sat}\left(\frac{x}{\Phi}\right) = \operatorname{sign}(x)$ for $\Phi \rightarrow 0$.

As the used holonomic robot takes velocity commands, the dynamics of the three errors e_x, e_y and e_φ can be formulated as integrators (5). In the application the control law (9) is then independently applied to these three systems.

4 Conclusions and Future Work

Following the common interest of the institutes involved in the GRIPS RoboCup Logistics League team in a holistic approach to develop methods for dependable intelligent systems we developed a software architecture that allows flexible and robust execution of the demanded production tasks. The basic idea is to separate different concerns such as abstract planning and scheduling, refinement of task execution, and behavioral control and equip each layer with proper motioning and fault-recovery capabilities. The flexible planning and task assignment paired with robust task execution allowed us to realize more complex products reliably and constantly than in the past competitions.

In future work we will aim for an optimization based selection of suitable products to be build (using more context information such as travel time) as well as a better parallelization (using an improved resource management). Moreover, we like to better team up monitoring and recovery between different layers. Often one layer lacks of sufficient knowledge about the actual situation to make a consistent final conclusion about errors and recovery. Sharing and combining information of different layers may help to address this issue.

Acknowledgement. The team members in 2018 are Sarah Haas, Vanessa Egger, Stefan Krickl, Leo Fürbaß, Ivan Martin, Thomas Ulz, Jakob Ludwiger, and Gerald Steinbauer.

We gratefully acknowledge the financial support of Graz University of Technology, Knapp AG, IncubedIT GmbH, AccuPower GmbH, and pia automation. In particular, the team is grateful to Knapp AG for the mechanical and electrical design and integration of the GRIPS robot platforms.

References

1. RoboCup Logistics League. http://www.robocup-logistics.org/
2. Boano, C., Römer, K., Bloem, R., Witrisal, K., Baunach, M., Horn, M.: Dependability for the internet of things - from dependable networking in harsh environments to a holistic view on dependability. e & i Elektrotechnik & Informationstechnik **133**, 304–309 (2016)
3. Coelen, V., Deppe, C., Hoffmann, T., Karras, U., Niemueller, T., Rohr, A.: Rules and Regulations - RoboCup Logistics League. http://www.robocup-logistics.org/rules. Accessed 19 Dec 2018

4. Georgeff, M., Pell, B., Pollack, M., Tambe, M., Wooldridge, M.: The belief-desire-intention model of agency. In: Müller, J.P., Rao, A.S., Singh, M.P. (eds.) ATAL 1998. LNCS, vol. 1555, pp. 1–10. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-49057-4_1
5. Ghallab, M., Lau, D., Traverso, P.: Automated Planning - Theory and Practice. Morgan Kaufmann (2004)
6. Gspandl, S., Pill, I., Reip, M., Steinbauer, G., Ferrein, A.: Belief management for high-level robot programs. In: Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2011, Barcelona, Spain, pp. 900–905 (2011)
7. Ingrand, F.F., Chatila, R., Alami, R., Robert, F.: PRS: a high level supervision and control language for autonomous mobile robots. In: Proceedings of IEEE International Conference on Robotics and Automation, vol. 1, pp. 43–49 (1996)
8. Ludwiger, J., Steinberger, M., Horn, M., Kubin, G., Ferrara, A.: Discrete time sliding mode control strategies for buffered networked systems. In: 2018 IEEE 57th Annual Conference on Decision and Control (CDC). IEEE (2018)
9. Marder-Eppstein, E., Berger, E., Foote, T., Gerkey, B., Konolige, K.: The office marathon: robust navigation in an indoor office environment. In: International Conference on Robotics and Automation (2010)
10. Niemueller, T., Karpas, E., Vaquero, T., Timmons, E.: Planning competition for logistics robots in simulation. In: WS on Planning and Robotics (PlanRob) at International Conference on Automated Planning and Scheduling (ICAPS), London, UK (2016)
11. Niemueller, T., Zug, S., Schneider, S., Karras, U.: Knowledge-based instrumentation and control for competitive industry-inspired robotic domains. KI - Künstliche Intelligenz **30**(3–4), 289–299 (2016)
12. Niemueller, T., et al.: Cyber-physical system intelligence – knowledge-based mobile robot autonomy in an industrial scenario. In: Jeschke, S., Brecher, C., Song, H., Rawat, D.B. (eds.) Industrial Internet of Things: Cybermanufacturing Systems. SSWT, pp. 447–472. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-42559-7_17
13. Penrose, R.: On best approximate solutions of linear matrix equations. Math. Proc. Camb. Philos. Soc. **52**(1), 17–19 (1956)
14. Quigley, M., et al.: ROS: an open-source robot operating system. In: ICRA Workshop on Open Source Software (2009)
15. Shtessel, Y., Edwards, C., Fridman, L., Levant, A.: Sliding Mode Control and Observation. Springer, New York (2013). https://doi.org/10.1007/978-0-8176-4893-0
16. Steinbauer, G., Ferrein, A.: 20 years of RoboCup. Künstliche Intelligenz **30**(3–4), 221–224 (2016)
17. Utkin, V., Guldner, J., Shi, J.: Sliding Mode Control in Electro-Mechanical Systems. CRC Press, Boca Raton (2009)
18. Utkin, V.I.: Sliding Modes in Control and Optimization. Springer, Heidelberg (1992). https://doi.org/10.1007/978-3-642-84379-2
19. Zwilling, F., Niemueller, T., Lakemeyer, G.: Simulation for the RoboCup logistics league with real-world environment agency and multi-level abstraction. In: Bianchi, R.A.C., Akin, H.L., Ramamoorthy, S., Sugiura, K. (eds.) RoboCup 2014. LNCS (LNAI), vol. 8992, pp. 220–232. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18615-3_18