RoboCup Logistics League
Graz Robust and Intelligent Production System
GRIPS

Vanessa Egger, Leo Fürbaß, Lukas Knoflach, Stefan Krickl, Jakob Ludwiger,
Stefan Moser, Gerald Steinbauer, Thomas Ulz, Manuel Weichselbaum

April 16, 2019

Abstract

The present paper presents team GRIPS (Graz Robust and Intelligent Production System) and its approaches to the challenges in the RoboCup Logistics League 2019. Festo's Robotino, a customized additional construction and some further hardware, such as laser scanners, cameras, PC, and PLC, are used as a hardware platform. The software architecture is built in a multi-layer format. A scheduler/-planner functions as a central decision-making instance for all robots at the top player. Beneath this layer, a procedural reasoning engine (Open PRS) allows for individual decisions. At the lowest level, the robot operating system (ROS) executes the tasks assigned to the robot by the top layer. To improve the robustness of this system, observers monitor the lowest level. For this purpose, a diagnosis system is implemented which also reports abnormal behavior and likely underlying reasons to the upper layers. Malfunctioning components are thus detected and can be repaired.

1 Introduction

The main aim of the RoboCup Logistics League is to promote research in automated and flexible production and serves as a testbed for methods dealing with smart production. For this purpose, the league's competition simulates a smart factory in which different products must be manufactured in several steps at different production machines. During production, the various machines must be supplied with blanks and resources required for product manufacturing. A so-called RefBox orders products in a random order and thus simulates demands for products. All transportation tasks for products and intermediate products are performed by a fleet of autonomous robots. As this setting involves continuous changing of the product demands, the autonomous robots simulate the logistics challenges in future production systems. Therefore, the RoboCup Logistics League is an ideal testbed for innovative planning/scheduling algorithms and control methods for fleets of robots.

Table 1: Hardware components used in our approach.

Component	Quantity	Description
Robotino 3	3	Basis of all three robots.
Intel NUC	3	Additional computation device on all three robots.
Sick TIM551	3	Laserscanner for robot navigation.
Linksys Router	4	Network connections for all three robots and teamserver.
Axis Camera	3	Camera used for QR code detection.
Sick Camera	3	Camera used for conveyor alignment.

Team GRIPS was founded in 2015 as part of the practical course "Construction of Mobile Robots" held at Graz University of Technology. Students in this class have to find solutions for different challenges relevant to the Robocup Logistics League. When GRIPS was first founded, it already participated in the RoboCup World Cup which in2016 was held in Leipzig, Germany (team description paper [1]). GRIPS finished in third place in its first year of participation and was thus voted rookie of the year. At the RoboCup 2017 held in Nagoya, GRIPS achieved second place. In 2018, GRIPS has already participated at the RoboCup German Open in Magdeburg, Germany and finished in second place. After winning the thrilling finals at the RoboCup 2018 in Montreal, Canada, GRIPS earned its first world champion title.

The following section of this paper describe the hardware modifications performed on the Robotino and any additional hardware used. Section 3 explores the algorithms and software architecture implemented on the configured hardware platform. The next section briefly discusses the overall mission strategy. In Section 5 the implemented development process are presented. The next section bridges this content to current research at the institute. Finally, this paper concludes with Section 7.

2 Hardware

GRIPS uses a multi-layer system architecture (see Fig. 1). An external PC acts as a so-called teamserver which is used to coordinate the autonomous robots by assigning tasks to them. The robotic base we are using is Festo's Robotino [2] (see Fig. 2b). The robot team consists of the three robots allowed per team according to the rulebook. All three robots are identical in terms of hardware and are equipped with a customized construction to achieve the correct height for the mounted gripper in order to be able to grab the products from the conveyors and shelves. Furthermore, we equipped the Robotinos with an additional external computer to improve computational power. For navigating the robots we use a Sick TIM551 laser scanner. Furthermore, we mounted a front facing camera on all robots to detect the machines' QR-tags. To achieve the necessary precision for grabbing and delivering products at the conveyors, an additional camera is mounted at the level of the gripper. A network router is used to connect all these network devices and to provide reliable WiFi capabilities for the robots. The used equipment is listed in Table 1.

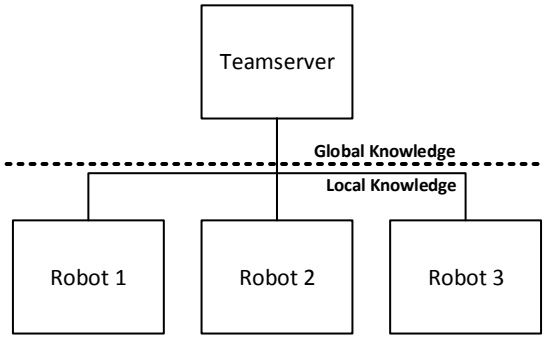


Figure 1: System architecture used in our approach. The teamserver holds global knowledge of the game state while each robot only possesses local knowledge.

3 Software

Following the idea of a three-layer architecture [3], the software is structured (as described in [4]) as follows: scheduler/planner, mid-level control and low-level. We introduced a strict communication scheme where only adjacent layers communicate with each other to achieve an increasing abstraction of the real world from layer to layer. Lower layers provide functionality to the higher layers (see Fig. 2a).

The *high-level* is responsible for coordinating the robots to achieve a common goal. It is also in charge of distributing tasks to the robots and to ensure that two robots do not use the same resource at the same time. Thus, the high-level needs to have global knowledge of the game state. As can be seen in Fig. 1, the high-level is running on a dedicated PC.

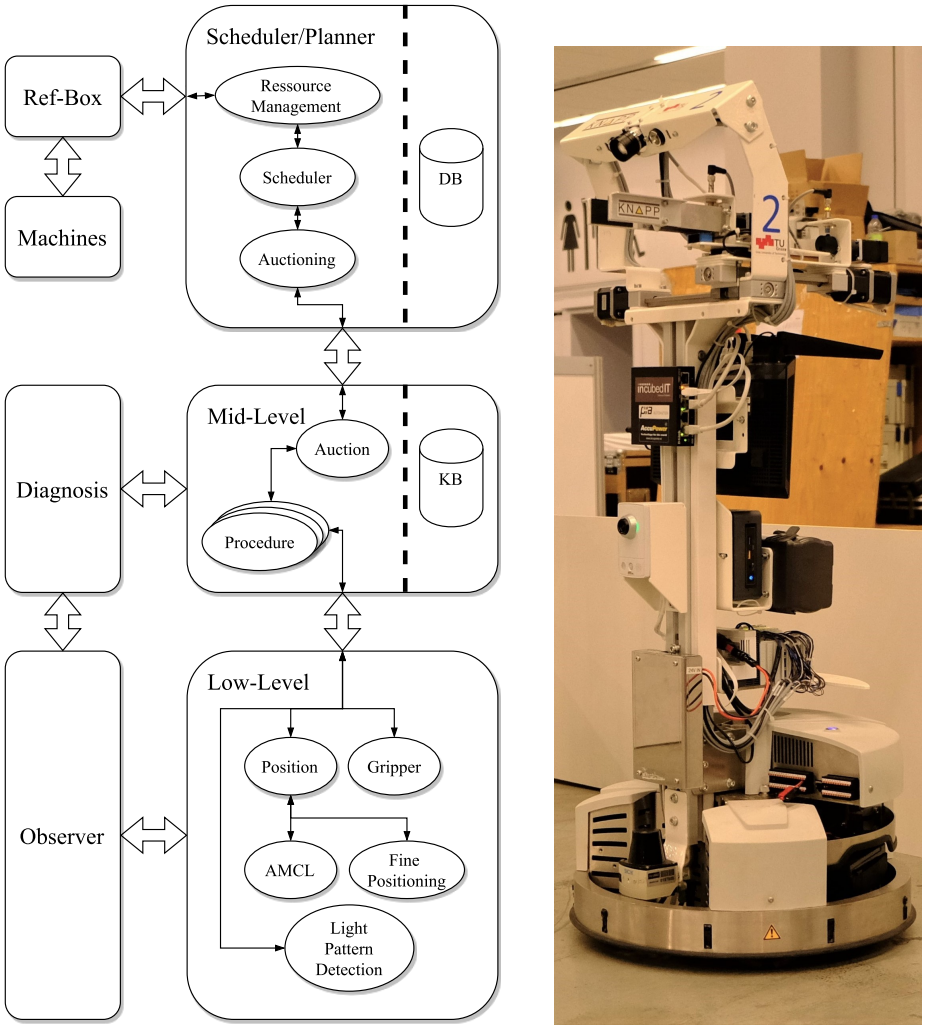
The *mid-level* controls the individual robots, i.e. the mid-level plans the actions required to finish the tasks assigned to the robot by the high-level. Additionally, the mid-level is responsible for rectifying faults in the task execution as well as possible for the robot to resolve the problem locally by the respective robot.

The *low-level* is responsible for executing primitive actions. These actions range from detecting the orientation of the machine to moving between way-points or opening the gripper. This level is the most reactive one and is the only level which performs near real-time activities.

Each level will be described in more detail in the remaining subsections of this section.

3.1 High-Level (Scheduler/Planner)

The scheduler/planner, being in the top layer of the structure, has the most abstract view. It is the only instance communicating with the referee box and is therefore responsible for keeping track of the orders and of forwarding the robot status arriving from the lower layers. As there is only one scheduler/planner for all robots, the resource management is also done on this level. Thus, the scheduler/planner is the only instance in our architecture that is in possession of a global knowledge base containing



(a) Overview of the software architecture.

(b) Adapted Robotino 2.

Figure 2: Used software architecture and hardware setup.

the complete game state. A resource manager included in the scheduler/planner keeps track of the current state of the production machines and their usage. That is, only one robot is granted permission to use a production machine at a time. This resource managements helps to avoid inconsistent production machine configurations as well as possible conflicts between robots.

In addition to the resource management, the scheduler/planner also maintains a database containing a pool of active tasks, the game state, and the state of the robots and machines. This is used to distribute the tasks to the robots (i.e. atomic production steps to be made to fabricate a product) in order to maximize the points awarded. In our approach, each production order is split into a set of three atomic tasks: *GetProduct*, *DeliverProduct* and *PrepareCap*.

GetProduct Task: This task comprises the following two steps for a robot:

- The robot moves to the specified machine and side.
- The robot then grabs the product with its gripper.

DeliverProduct Task: This task comprises the following two steps for a robot:

- The robot moves to the specified machine and machine side.
- The product that is currently in the robot's gripper is fed into the machine.

PrepareCap Task: This task comprises the following steps for a robot:

- The robot moves to the specified cap stations input side.
- The robot detects the products with mounted cap on the shelf
- The robot picks one of those products and feeds it in the input of the cap station

3.1.1 Teamserver Visualization

Since every team uses three robots and the seven machines provided in the production hall during the RCLL competitions, it is hard to observe the tasks of robots and the current states of machines simultaneously. However, knowing what a robot is currently doing or planning to do is essential in many situations to foresee errors or possible problems. Issues, such as bad localization or wrong task execution, could disrupt the manufacturing process. If we are able to foresee such issues, maintenance can be executed in an earlier stage and thus avoid the occurence of errors beforehand. As a solution, we developed a simple website to visualize information on the robots and machines. The information about each robot contains its state, current task, next task, current product and many more. The machine's information contains the current state, whether it is currently being used by a robot, etc. As our team server uses the Spring framework, which is based on a model-view-control architecture and already supports

web developement, the creation of a website skeleton linked to our team server was not a difficult task. We use web sockets to provide the data from our team server to the website. To transfer the information to the website, a class needs to be defined holding the information to be displayed. This class is referred to as the model. In addition, a second class, the controller, needs to be generated. The controller class holds the methods executed when get, post or similar types are called from the webpage. The controller provides requested data, and also writes received data to some data structure. Finally, the webpage displays the requested data in a certain format.

3.2 Mid-Level Control

The mid-level control is based on the Open Procedural Reasoning System (Open PRS) [5]. Open PRS implements the belief-desire-intention (BDI, [6]) system and allows operational plans (OPs) to be designed using a graphical user interface. Within a BDI system the agent keeps track of the world's current state and the goals that should be achieved. To achieve a goal, the agent selects one suitable OP and executes it. Through this paradigm the robot can achieve a task depending on the environment and the robot's status. The integrated reasoning engine also allows these OPs to be executed in parallel. We utilize this feature, for instance, in the exploration phase to simultaneously explore the world while also looking for unseen QR-tags.

An example OP that is used to close the gripper can be seen in Figure 3. This example should illustrate some functionalities and the syntax. This OP is initiated by posting the invocation part `(achieve (gripobject))` using the message passing (MP) interface of OPRS. The plan is only applicable if the gripper is open at the moment, i.e. `((test (gripper open))`. After successful execution of this plan, `(gripper-holds-object)` is written in the database. On the edges of the graph, subgoals can be posted and (evaluable) predicates can be tested. E.g. the posting of the goal `(gripper closed)` is directly forwarded to the lower level as it is an atomic part of the execution chain.

Additionally, error detection and error handling of low-level functions is done in the mid-level. The system tries to determine what went wrong and what is the most effective way to recover from the given error. In order to do so the mid-level comprises a knowledge base containing the state of the robot and its environment to find an efficient recovery if there exists one. It is important to note that any failure detected by the diagnosis system (see Section 3.4 for details) is also reported to the mid-level and is stored in the knowledge base. Thus, the mid-level also tries to recover from faults detected through the diagnosis system.

3.3 Low-Level

The lowest layer is mainly based on the open robot operating system (ROS [7]). Here, basic atomic actions are advertised to the layer above as a ROS action server. Different services are advertised here, e.g. opening/closing the gripper, locally navigating to a machine, aligning in front of a machine and so on. Performing these actions, all the error detection and a possible error recovery is made by the ROS service. In the

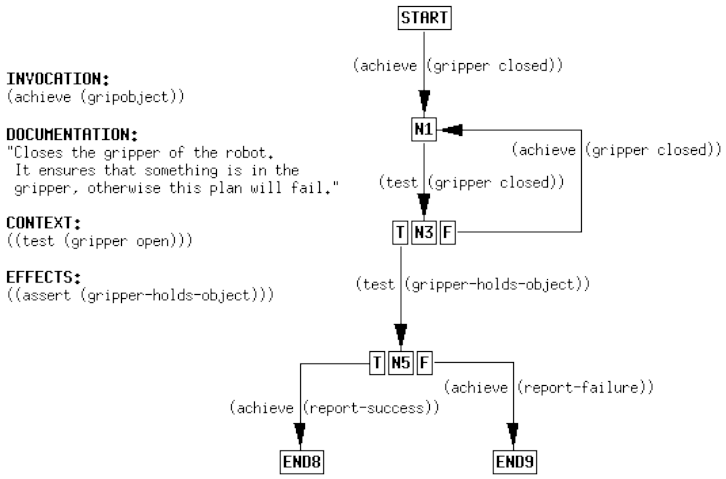


Figure 3: Sample OPRS plan to grab some object with the gripper.

following part of this subsection we will briefly discuss the most important parts of the low-level functions.

3.3.1 Way-Point Navigation

In order to retrieve or deliver objects, the robot moves between defined way-points. These way-points are stored in the mid-level as abstract identifiers e.g. the output of the base-station would be stored as `BS-O`. In order to move to a given way-point, the low-level uses a table to look up the real world coordinates for the abstract identifier and afterwards plan to move to this way-point. To move to the given coordinates, we use the move_base package of ROS [8]. This package comprises a global planner that uses the map to find a path between the current robot position and the desired final position. In order to avoid the different production machines, these machines are added to the navigation map. Furthermore, after finding a global plan a local planner is used to move along the path by considering and avoiding detected objects on the planned path.

3.3.2 Conveyor Alignment

During the production phase the robot needs to deliver and retrieve products from the conveyor multiple times. This is done through a way-point that is in close proximity to the desired conveyor. Thus, a robot is able to move near the intended machine. After that, the QR-Code and the laser scanner is used to approach the machine and to align the robot close to the conveyor belt. Since combining the information provided by the QR code detecting camera and the laser scanner might be inaccurate, we use an additional elevated camera on our robot for fine aligning the robots. For this, the conveyor (also products, slides, and shelves) are detected through an additional camera

and a HOG detector.

HOG Detection: The histogram of oriented gradients (HOG) is used to detect the positions of products, conveyor bands, slides and shelfs on different machines. The HOG detector identifies regions of interest (ROI) in images based on previously learned features. To identify the features, images for training and testing need to be taken. In all images the ROI needs to be marked (see Figure 4) for an example). Then the training and testing images are forwareded to the HOG detector. The HOG then tries to identify the features of the given ROIs and generates a feature set. The training images are used to find the features, and the testing images are used to test if the features identified from the training images match the ROIs defined in the testing images. In the logistics league, the robot approaches a machine and stops at a certain position close enough to the desired objective. When in position, the camera takes four images with different light conditions provided by the 2 headlights. The head lights are used to reduce the effects of changing environmental light such as overhead lights or daylight on the taken images. The images are analyzed by the HOG detector and the resulting ROIs are averaged to find the correspoing feature (see Figure **??** for an example). Four images are taken to increase the possibility of finding a correct ROI in one of the images even when the lighting conditions differ from the conditions the HOG detector was trained on. In our case, we use several different HOG detectors to detect different objects such as products or the conveyor band. Each of these detectors was trained with different images.

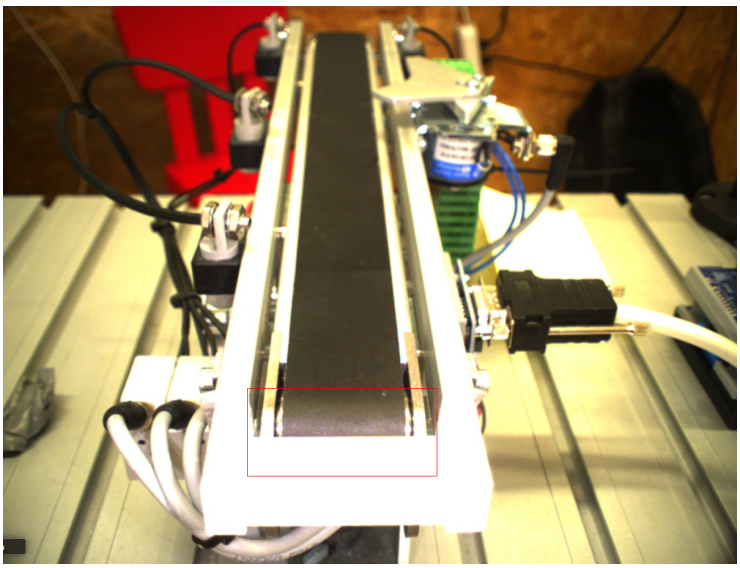


Figure 4: Example of a marked ROI for HOG training.

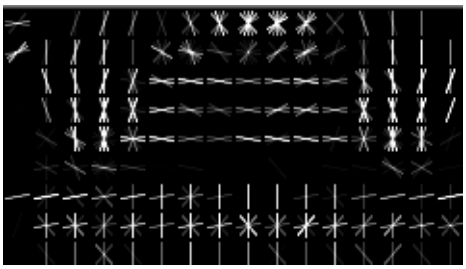


Figure 5: Example learned features of our HOG detector.

3.4 Diagnosis System

In order to guarantee a certain degree of dependability, an online diagnosis system is used in our approach [9]. The diagnosis system uses a set of observers to monitor different properties of the system e.g. the frequency of the communication between two different low-level modules. These observations are then compared to defined normal system behavior. If the observation yields a discrepancy, a diagnosis on how to fix the problem is calculated. This is done using a diagnosis engine based on PyMDB [10] which calculates a consistency based diagnosis [11]. Thus, the result of the diagnosis is a set of low-level modules that, if assigned to be faulty, would explain the undesired observations. The information of the possible faulty components is reported to the mid-layer in order to allow a repair mechanism to be triggered. Moreover, a simple rule engine is used to directly trigger simple actions (e.g. write a log file) once a faulty component has been identified.

3.5 Communication

The communication between the scheduler/planner and the mid-level control is done in a similar way as the communication with the referee box where serialized messages generated with the open protocol buffer data format (protobuf) developed by Google are used. Also, the internal communication in the mid-layer is implemented using such serialized streams since the standard communication interface of Open PRS has some limitations that are mitigated by our implementation. The mid-level control communicates with the low-level using so-called ROS action servers, i.e. the low-level offers actions that can be triggered by the mid-layer. Implementing ROS action servers allows fetching the current status of the action during execution as well as a return status after the execution is finished or failed.

4 Mission Strategy

The game can be split up into two main phases, exploration and production that will be discussed separately in this section.

4.1 Exploration

In the exploration phase each robot will be in charge of exploring one column of our half of the field. To do so, the robot moves on the field and searches for unseen QR-tags. Whenever one is found, the robot moves to the corresponding machine and measures the machine's orientation using the laser scanner. The collected information is then sent to the high-level Control. The scheduler/planner gathers all reports from the robots with additional information about the certainty of these observations. Having multiple observations with given probabilities, a safe report can be sent to the referee box to avoid negative points resulting from wrong reports. Exploring only one half of the field is sufficient due to the symmetry of the game field. Currently we develop a constraint based consistency check for the reported machines and zones.

4.2 Production

The benefits offered by the three-layer architecture (see Section 3) can be utilized best in this phase of the game. The scheduler/planer splits up the orders into atomic tasks to build a desired product and stores them in a task pool. The tasks are auctioned to the robots ordered by their priority to achieve a maximum number of points. The scheduler/planner maintains a global view on the current game to achieve a global maximum of points. Atomic tasks are, for instance, providing a cap, mounting a cap, discarding the base and so on. All the robots bid for the tasks (if they are currently not executing a task) with their current local cost, i.e. the robot doing the job most efficiently wins the bid. As a robot needs a modular production system (MPS), it tries to get a mutual lock for the machine via the scheduler/planner. There, a global locking table is kept to avoid congestion and multiple configurations of machines.

5 Development

In order to build up a robust system during our development, we use a continuous integration [12] server that executes builds as well as unit and integration tests. These integration tests are executed with the help of the Gazebo simulation that is provided by BBUnits and the Carologistics team [13, 14]. Through this continuous testing software faults and integration problems can be found more easily. To encourage other teams to follow the idea of continuous integration, we plan to release the software to perform these tests under an open source license after it has reached a certain level of maturity.

6 Influence Through Current Research

Within the focus of the current research at the Institute for Software Technology (IST) hosting the team is the usage of a model-driven approach to develop dependable autonomous robots. The idea is to use models as a central part of the developing process to automatically test during software development and to diagnose the system at runtime. These processes should be automated as much as possible transforming information such as requirements to models automatically or by reusing existing models

[15, 16]. This approach has already been shown to be applicable for an industrial use case. Thus, we hope that this approach also results in a robust system for the logistic league.

Furthermore, current research is done in order to design an agent architecture which allows a model driven approach to be easily integrated into a robotic system. This architecture should allow a robotic system to be more dependable and thus run for long periods of time. We expect that the result of this research is of special interest for the logistics league as this properties are of high interest for smart production use cases.

Finally, we have investigated different high-level approaches to control the robot fleet together with the Carologistics team [17]. The high-level approach evaluated was based on YAGI (Yet Another Golog Interpreter), which is an implementation based on the ideas of the logic-based agent control language GOLOG. YAGI allows imperative as well as declarative parts to be used; thus, programming and planning can be balanced between ease of use and performance.

7 Conclusion

In order to get a global view of the current game state we use a central planning and scheduling instance that distributes the robot's tasks through an auctioning system. The robots use a BDI system in order to execute the tasks in a reliable and reactive manner. Additionally, the robots will be observed during run-time in order to detect faults and allow a fast recovery from failures. The robotic base we use for our robots is Festo's Robotino 3 which was modified such that the robot is capable of fulfilling all necessary tasks. Based on the research performed at the Institute for Software Technology at Graz University of Technology, we expect to provide the league with new ideas, to design more dependable systems.

Conformity with the Rules *We hereby declare that our presented hardware and software architecture satisfies all the requirements stated in the RoboCup Logistics Rulebook 2018 (published January 25th, 2018).*

References

[1] Haas, S., Keskic, D., Mühlbacher, C., Steinbauer, G., Ulz, T., Wallner, M.: Robocup logistics league tdp graz robust and intelligent production system grips. (2016)

[2] Karras, U., Pensky, D., Rojas, O.: Mobile robotics in education and research of logistics. In: IROS 2011–Workshop on Metrics and Methodologies for Autonomous Robot Teams in Logistics. (2011)

[3] Gat, E., et al.: On three-layer architectures. Artificial intelligence and mobile robots **195** (1998) 210

[4] Wallner, M., Muehlbacher, C., Steinbauer, G., Haas, S., Ulz, T., Ludwiger, J.: A robust and flexible software architecture for autonomous robots in the context of industrie 4.0. In: Austrian Robotics Workshop. (2017) 67–73

[5] Ingrand, F.F., Chatila, R., Alami, R., Robert, F.: Prs: A high level supervision and control language for autonomous mobile robots. In: Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on. Volume 1., IEEE (1996) 43–49

[6] Georgeff, M., Pell, B., Pollack, M., Tambe, M., Wooldridge, M.: The belief-desire-intention model of agency. In: Intelligent Agents V: Agents Theories, Architectures, and Languages. Springer (1998) 1–10

[7] Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system. In: ICRA workshop on open source software. Volume 3. (2009) 5

[8] 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)

[9] Loigge, S., Mühlbacher, C., Steinbauer, G., Gspandl, S., Reip, M.: A Model-Based Fault Detection - Diagnosis and Repair for Autonomous Robotics systems. In: Austrian Robotics Workshop. (2017)

[10] Quaritsch, T., Pill, I.: Pymbd: A library of mbd algorithms and a light-weight evaluation platform. Proceedings of International Workshop on Priciples of Diagnosis (DX) (2014)

[11] Reiter, R.: A theory of diagnosis from first principles. Artificial intelligence **32**(1) (1987) 57–95

[12] Duvall, P.M., Matyas, S., Glover, A.: Continuous integration: improving software quality and reducing risk. Pearson Education (2007)

[13] Zwilling, F., Niemueller, T., Lakemeyer, G.: Simulation for the robocup logistics league with real-world environment agency and multi-level abstraction. In: RoboCup 2014: Robot World Cup XVIII. Springer (2014) 220–232

[14] Niemueller, T., Reuter, S., Ewert, D., Ferrein, A., Jeschke, S., Lakemeyer, G.: Decisive factors for the success of the carologistics robocup team in the robocup logistics league 2014. In: RoboCup 2014: Robot World Cup XVIII. Springer (2014) 155–167

[15] Simón, J.S., Mühlbacher, C., Steinbauer, G.: Automatic model generation to diagnose autonomous systems. In: Proceedings of the 26th International Workshop on Principles of Diagnosis (DX-2015) co-located with 9th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes (Safeprocess 2015), Paris, France, August 31 - September 3, 2015. (2015) 153–158

[16] Mühlbacher, C., Gspandl, S., Reip, M., Steinbauer, G.: Improving dependability of industrial transport robots using model-based techniques. In: Robotics and Automation (ICRA), 2016 IEEE International Conference on, IEEE (2016) 3133–3140

[17] Ferrein, A., Maier, C., Mühlbacher, C., Niemueller, T., Steinbauer, G., Vassos, S.: Controlling logistics robots with the action-based language yagi. In: IROS Workshop on Taks Planning for Intelligent Robots in Service and Manufacturing. (2015)