Proceedings of the OAGM&ARW Joint Workshop 2017 DOI: 10.3217/978-3-85125-524-9-12

A Robust and Flexible Software Architecture for Autonomous Robots in the Context of Industrie 4.0

Marco Wallner[1], Clemens Mühlbacher[1], Gerald Steinbauer[1], Sarah Haas[2], Thomas Ulz[2] and Jakob Chrysant Ludwiger[3]

Abstract— **The next industrial revolution should allow the production of individually configured items at the cost of a currently mass-produced commodity. To make this possible, autonomous robots play an essential role. These robots use their knowledge about the world and the task as well as sensor information to derive the next action to achieve a common goal. To give research in this area the possibility to test novel methods and algorithms for the next industrial revolution, the RoboCup Logistic League was established. The league uses a small shop floor environment wherein a group of robots has to produce customized goods within a given time-frame.**

In this paper, we present a software framework for a group of autonomous robots which deal with the problems of the RoboCup Logistic League. The software is separated into three distinct layers allowing modularity as well as maintainability. The framework provides all the needed functionality starting from creating plans for a fleet of robots, to the hardware skills to detect machines, to move between locations and interact with the physical world. To show the use of the software framework a use-case is presented. This use-case is the exploration of an unknown factory hall with a fleet of autonomous robots. All the presented solutions in this paper were tested in the RoboCup world championship 2016 in Leipzig. There the system showed its robustness and its capability to solve issues arising with the next industrial revolution.

I. Introduction

To increase customer satisfaction and sales, the industry tries to fulfill the desire of the customer for an individual product to the price of a product created by mass-production. As current methods in industrial production cannot provide these low costs for highly customized products, new approaches need to be developed. This is done through an ongoing automation in the industry which leads to the so-called Industrie 4.0 [1]. This term, originated by the German government, describes the abstract shape of the next generation in industry. One of the manifestations of this vision is the idea of a fully autonomous fabrication with smart machines.

To create such a smart factory two parts are essential. On the one hand, there are configurable smart machines necessary which are capable of performing the manufacturing steps. On the other hand, an intelligent delivery system between these devices is needed to allow the transportation of intermediate products to produce compound goods. As the usage of the machines and the scheduling can no longer be statically determined for a production line but need to adapt to the current requests, new algorithms need to be developed. To test these algorithms, the RoboCup Logistic League [2] was initiated as part of the annual RoboCup world championship [3]. The idea is to create a simplified version of a smart factory which serves as a testbed and a standardized benchmark for novel algorithms and approaches. With this, different aspects can be tested and evaluated regarding distinct components as well as the complete framework.

In this paper, we present a software framework to solve the challenges of the RoboCup Logistic League. The framework allows to schedule the entire fleet of robots and to react to changes in the environment in a reactive manner. Furthermore, the system is capable of reacting to faults during the execution of a task assigned to a robot or even a full drop-out of a robot in the fleet. To allow an efficient, modular and maintainable implementation, the framework uses a layered architecture. This approach allows to schedule the fleet on the highest level while considering the reactive interaction between the robot and its environment on the lowest level. To show how this software framework is used we present in this paper how the robotic fleet can explore an unknown shop floor. With the help of this example, we will show how the fleet is scheduled to cover the shop floor efficiently. Additionally, we will demonstrate how the machines are detected and identified.

The remainder of the paper is organized as follows. In the next section, we will discuss the RoboCup Logistic League in more detail. The proceeding section discusses the layered architecture. Afterward, we will discuss the software components which are used for the exploration of the shop floor environment. Before we conclude the paper we will discuss some related research. Finally, we conclude the paper and point out some future work.

II. RoboCup Logistic League

RoboCup, proposed and founded in 1997, is an annual international robotics competition. There, teams from all over the world compete in different disciplines, such as humanoid robots, soccer robots, rescue robots and the mentioned logistics competition.

The logistics league simulates the problems arising in a smart factory. Primarily it provides a standardized testbed for test new algorithms and approaches for smart factories.

[1]Marco Wallner, Clemens Mühlbacher and Gerald Steinbauer are with the Institute for Software Technology, Graz University of Technology, Graz, Austria. {mwallner,cmuehlba,steinbauer}@ist.tugraz.at.

[2]Sarah Haas and Thomas Ulz are with the Institute of Technical Informatics, Graz University of Technology, Graz, Austria. {thomas.ulz,sarah.haas}@tugraz.at.

[3]Jakob Chrysant Ludwiger are with the Institute of Automation and Control, Graz University of Technology, Graz, Austria. jakob.ludwiger@tugraz.at.

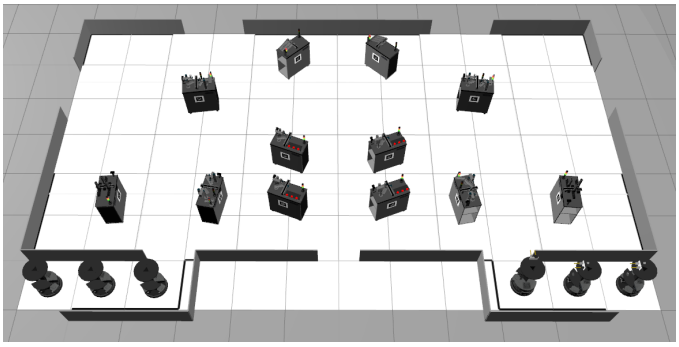

Fig. 1. RoboCup Logistics game-field in the simulator. Two attending teams with 3 robots and 6 machines each.

This is achieved by a controlled environment which contains a modular production system and a fleet of robots which need to be controlled. To allow fair conditions, a standardized robot platform (Robotino by Festo [4], three per team) is used for the mobile robots as well as standardized modular production systems (MPS by Festo, six per side) for the fabrication steps.

To emphasize the idea of a smart factory, the rules require that the fleet performs its task completely autonomously. Thus no intervention from humans is allowed. The idea is to put a robot in a workshop and let it explore the environment on its own, find machines to work with and produce products according to arriving orders. For this, the whole scenario is split into two phases, the exploration, and the production phase.

A. Exploration Phase

In this first phase, the robots have no knowledge about their environment. They have to explore the game-field (see Figure 1, a screenshot of the RoboCup Logistic League simulation [5]) and find the machines located there. To award points for the detection of such a machine, the robots have to report their observations to a central referee box. Each report contains the type of the machine, the shown status light as well as it is position in the field. If all the machines have been found, or after some deadline has passed, the next phase is invoked.

B. Production Phase

In this phase the actual production takes place. Random orders are placed by the central referee box, and both teams try to produce these as fast as possible.

1) Products: The products are mocked up as cups (base) with a defined number of rings pressed on it and a cap. The color of each part of the product is defined in the order.

2) Order: An order consists of the demanded product (e.g. a red base cup with two rings, the first ring blue, the second one yellow and a black cap) and its earliest delivery time as well as the deadline for the delivery of this product.

3) Modular Production System: To produce the ordered product, the mobile robots can use the six production systems of their team. There are four types of these workstations:

- *1x Base Station:* Providing bases in the demanded color.
- *2x Ring Station:* Mounting a ring in requested color on the provided base.
- *2x Cap Station:* Mounting a cap in required color on the provided base.
- *1x Delivery Station:* Point to deliver a product in the given time window.

As the mounting of a ring represents the addition of some feature to a product, some ring colors require additional bases as ”raw“ material. Thus also the need of deliveries for supply material is modeled in this scenario.

III. Software Architecture

To solve the tasks of the Logistics League, we propose the following software architecture. The software is split into three distinct layers, namely high-level, mid-level and low-level. Each layer is independent of the other layers within this concept. The lower layers provide functionality to the upper one [6]. Furthermore, higher layers command the actions of the lower layers.

The highest level of our software architecture is responsible for the connection of the different parts. It connects to the central referee box as well as an arbitrary number of connected robots as it can be seen in Figure 2.

To allow independent development and testing of each layer defined interfaces are necessary. Additionally, to feature different programming languages for each layer, Google's protocol buffers are used for these interfaces. This independence is used as the high-level is written in Java, the mid-level using a belief-desire-intention [7] engine (openPRS [8], C) and the low-level is written in C++ using the ROS (Robot Operating System [9]) framework. The communication scheme for one robot can be seen in Figure 3.

For each interface dedicated protocol buffer (protobuf) messages are defined. With this structure, an increasing abstraction of the physical world can be achieved from the bottom up to the top. The message used between the high-level to the midlevel can be seen as an example in Listing 1.

Listing 1. Protobuf message to communicate between the layers.

```
message PrsTask {
  required Team teamColor = 1;
  required uint32 taskId = 2;
  required uint32 robotId = 3;

  optional ExecutionResult result = 4;

  optional ReportMachinesTask reportTask = 5;
  optional ExploreMachineTask explTask =  6;
  optional GetWorkPieceTask getWPTask = 7;
  optional PrepareCapTask prepCapTask = 8;
  optional DisposeProdTask dispProdTask = 9;
  optional DeliverProdTask deliProdTask = 10;
}
```

The lowest layer is responsible for small tasks close to the hardware, e.g. to move to a waypoint, grab an object, detect an AR-tag or analyze the status light of a machine (see Section IV-A.2 and Section IV-A.1 for further details). We call the execution of these small tasks skills in the remainder

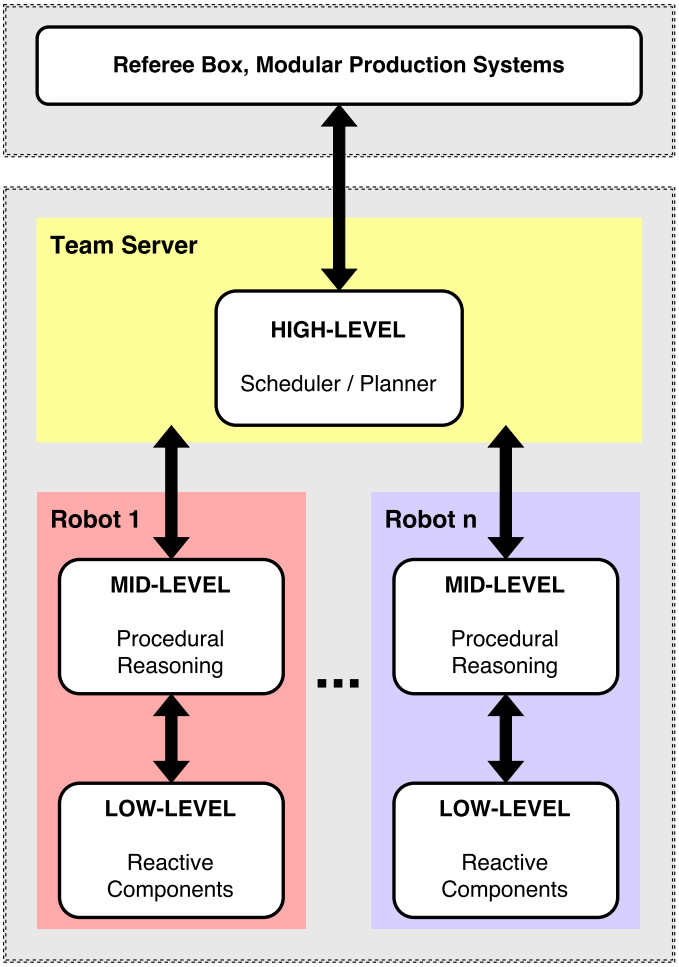

Fig. 2. Overall software architecture with links to the central refbox, the team server and the connected n robots.

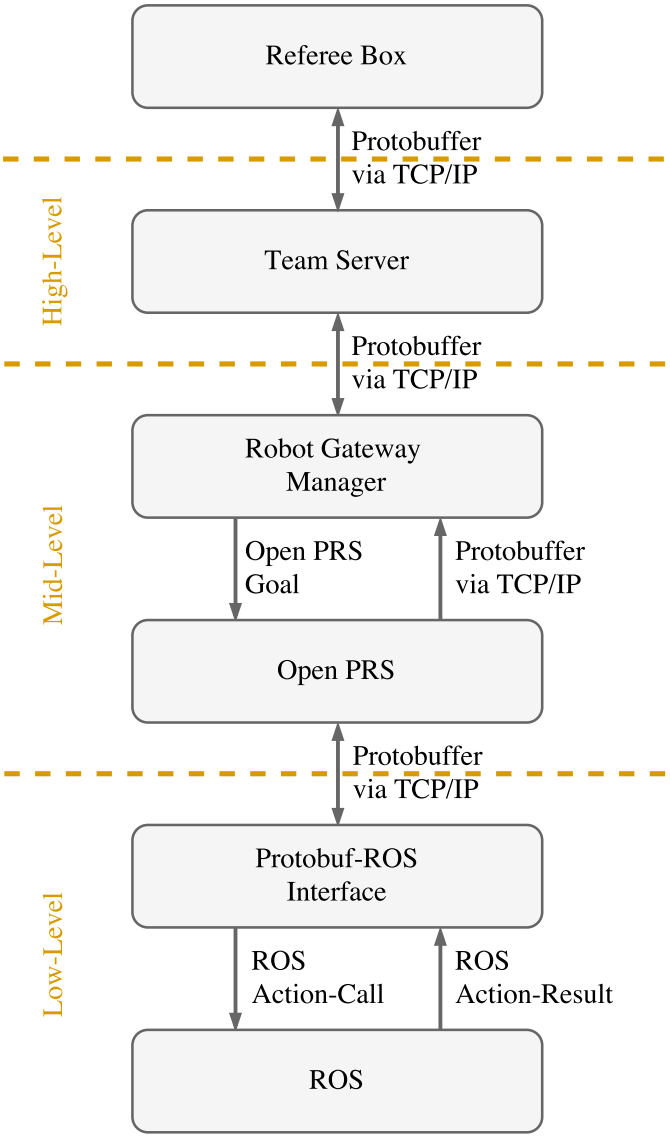

Fig. 3. Communication between the different layers for one robot using Google's protocol buffers.

of the paper. These skills are provided to the next higher layer, the mid-layer, via defined messages.

The mid-layer, therefore, can use these skills to perform more complex tasks such as exploring a zone of the game-field, get a base from the base station or deliver the product holding in its gripper. Additionally, a first error detection and recovery behavior are implemented here, e.g. the system checks if there is a product in the gripper after the low-level has successfully grabbed something. These complex tasks are again provided via defined protobuf messages to our highest layer, the team server.

Here a central knowledge-base is held and a game strategy is derived (see Section IV-B for further details). This central point enables the system to conclude a global optimal game strategy for the complete robotic fleet. The global strategy is derived using a simple planning system which uses a hierarchical task network [10] to properly create the products. Due to the centralized knowledge base one does not need to deal with synchronization of knowledge bases of the robot or distributed planning. Instead a "simpler" approach for planning can be applied.

IV. Selected Software Components

To get an idea of the functional interaction of our robot system some selected components are presented. First of all, two low-level modules necessary to detect and identify a machine are presented in SectionIV-A. Additionally, the scheduling algorithm (located at the high-level component of our system) which manages the discovery of the unknown game-field is presented in Section IV-B.

A. Machine Detection and Identification

To be able to gather information about the unknown environment it is necessary for the robot to recognize elements surrounding it. One important type of these elements is a modular production system, i.e. the machines capable of producing the ordered products. To identify the machines AR-tags are used which are placed at two sides of these machines.

Fig. 4. View of the robot in front of machine through the light detection camera at the RoboCup 2016 in Leipzig, Germany.

1) AR Detection: To localize objects in a defined frame of reference, it is first necessary to localize the robot itself. For this, a laser scanner and the knowledge of the fixed outer boundaries of the factory are used to infer the position of the robot using an adaptive Monte Carlo localization approach [11]. Using the particle filter also the confidence of the current location can be inferred.

With the known location of the robot and the known position of a camera mounted on the robot, it is possible to infer the position and orientation of seen augmented reality (AR) tags. For this, the open source AR-Tag tracking library Alvar is used. These tags are of defined size (allows derivation of distance to the tag) and are mounted at the input and the output of each machine. Each machine has a defined tag id for the input as well as the output. Using this knowledge, the position of the machine can be calculated having at least one of the tags seen. The accuracy and reliability of this measurements are further improved using a moving average filter. The filter is used to correct the estimate of the machine position with the help of several measurements. This raw data of the location of the machines is used by the higher layers as described in Section IV-B to determine which zone the machine is in. The information about the occupied zone is then reported to the referee box to earn points during the exploration.

2) Light Detection: To fully identify a machine, additionally to the AR-tag, the position and orientation of it, as well as the shown light pattern, needs to be reported. The light pattern is used to uniquely identify the machine. For this, the robot moves to a point in front of the machine. Afterwards, the robot captures an image of the machine. The captured image can be seen in Figure 4. These views have random backgrounds with arbitrary components, colors, and structures in it. Therefore, a detection of the light with the help of a blob detection is difficult to configure and is unreliable. Instead, one can exploit the fixed structure of the traffic lights. All of them have the same geometry regardless

Fig. 5. Cropped traffic light by the histogram of oriented gradients detector.

of the shown light pattern. They have a defined ratio between length and height, are sectored in three parts and are always upright.

This knowledge could be exploited by applying different manually generated and adjusted rules to determine the position of the traffic light in the image. Instead of these manually created rules, our approach uses a machine learning approach allowing the method to be more reliable, easy to configure and adapt to new environments with no effort. We use the static feature of the structure to train a histogram of oriented gradients (HOG) detector as described by [12]. This detector exploits that the mentioned static features manifest in a static gradient pattern.

Using the results of the HOG detector, a region of interest (ROI) can be extracted. The result of this cropping can be seen in Figure 5. Here the cropped traffic light is shown for all possible light combinations. The HOG detector has the advantage of almost no false-positive detections, i.e. if a ROI is found, there is a traffic light in it with a high probability.

To report the type of shown light pattern, a mapping from the traffic light image (which light is on and which is off) to a representing number is needed. For this, the lighting condition is encoded in a binary fashion, i.e. the representing state is calculated as:

$$\text{state} = s(\text{green})^0 + s(\text{yellow})^1 + s(\text{red})^2 \tag{1}$$

with

$$s(x) = \begin{cases} 2, & \text{if } x \text{ is on} \\ 0, & \text{else.} \end{cases} \tag{2}$$

With this mapping, a feed forward artificial neural network can be trained. We used the scaled conjugate gradient descent algorithm described in [13] to train the network. With this trained network, it is possible to map a newly seen image to a vector of probabilities describing the likelihood of each class as described in [14].

This gathered information can then be used by the higher layers to build up a knowledge base about the environment as described in Section IV-B.

The chain of a HOG-detector and a neural network was chosen as none of these approaches need a lot of computing power during the execution (only once at training time) to avoid the tuning of several parameters. The used neural network further increases the reliability as it can be trained to be resilient to different lighting situations.

B. Scheduling Algorithm

The robots have no information about their environment at the start of the game. Therefore, they have to use sensors

to observe the environment and to gather information. This is achieved by using a laser scanner for localization and cameras for machine detection.

To explore the game-field in an efficient manner with multiple robots, a scheduling algorithm has to be implemented. For this, our software architecture described in SectionIII comes into play. With the centralized team server, it is possible to generate a global exploration strategy and to combine the information delivered by all the robots into one reliable and consistent database.

During the exploration phase, all robots have the non-blocking task to report all seen machines, i.e. the zone, orientation (in discrete steps) and light pattern as well as the corresponding confidence. These updates are sent to the team server if parts of the information changes (e.g. orientation is corrected), new information is added (e.g. a light pattern is detected) or the confidence of a property rises. This information is then collected at the team server as an `observations` database.

To start the exploration with no observed date (i.e. at the beginning of the exploration phase) the default task for the first robot is to explore the top most left zone of the game field if the team starts at the right start box or the top right zone of the game field if the team starts at the left start box (see Figure 1). Using this simple strategy, the probability is very high that on the way to the destination zone the robot observed other machines and reported them to the team server. As soon as another robot is ready for a task or the first robot has finished its navigation, the robot gets the task assigned to visited a zone. During the visiting of a zone, the robot detects if a machine is within the zone. If a machine is present, the robot performs a light detection of the machine. If no machine position is reported so far, the robot gets a backup task to visit a randomly chosen zone which was not visited before. Otherwise, the robot is sent to a zone with a high probability that a machine is in this zone (one robot has reported that there should be a machine) but was not visited before. If all zones are visited, the zone with the lowest confidence is chosen as the next task. This allows maximizing the confidence of the machine information. The simplified algorithm can be seen in Algorithm 1.

With the start position in the team boxes (as it can be seen in Figure 1) it is very likely that at least one machine is seen already in the start position. Thus the usual procedure is that the first robot directly reports at least one machine at start-up. The team server creates a task for this robot and sends it to discover the light state and the correct orientation. On its way, the robot reports other machines, and so the other robots can be sent to zones with machines too. Thus the backup solution to drive to some randomly chosen zone is rarely used.

This dynamic scheduling allows a very efficient and fast exploration of the whole game field. This is necessary as the game field is rather large ($12m \times 6m$) for the low speed these robots are able to move.

Another advantage of the global view of the team server can be used here too. The machines are distributed at the game-field in a symmetric fashion to allow fair conditions for both teams. This constraint can be used for a sanity check of the reports, i.e. before the final result is sent to the referee box, it is checked if it makes sense and the most probable consistent set of observations is reported.

Algorithm 1: Exploration Algorithm

Input: observations, notVisitedZones, #MPS, thresh
Output: task

```
1:  if observations = ∅ then
2:    if oppositeZone ∈ notVisitedZones then
3:      return exploreZone(oppositeZone)
4:    else
5:      zone = chooseRandom(notVisitedZones)
6:      return exploreZone(zone)
7:    end if
8:  else
9:    if numZonesNotVisited(observations) > 0 then
10:     zones = zonesNotVisited(observations)
11:     zone = getZoneWithLowestConfidence(zones)
12:     return exploreZone(zone)
13:   end if
14:   if mFound(observations, thresh) < #MPS then
15:     zone = chooseRandom(notVisitedZones)
16:     return exploreZone(zone)
17:   else
18:     zones = zonesNotVisited(observations)
19:     zone = getZoneWithLowestConfidence(zones)
20:     return exploreZone(zone)
21:   end if
22: end if
```

After the exploration phase, the set of reliable machine positions and orientations is then broadcasted to the connected robots to allow them to work during the production phase with the gathered information. Also if one robot has to be restarted during the production phase, the information about the position of the machines is provided as a new (or in this case restarted) robot connects to the team server.

V. Related Research

In the previous section we have discussed our software architecture how to solve the challenges in the RoboCup logistic league. Within this section we will discuss another approach to solve the problems in RoboCup Logistic League. We will compare our approach to the Carologistics Team which won the world championships several times. As the Carologistics Team describes in its team description paper [15], they also use a three-layer architecture.

A. Carologistics

The main difference is that no central coordinator is used. Instead a distributed, local-scope and incremental reasoning approach [16] is chosen. This has the advantage of no single point of failure but also the disadvantage that no optimal global strategy can be derived. To keep a consistent view of

the physical world, a permanent synchronization of the robots is needed. For this purpose, one of the agents is chosen to act as a leader responsible for collecting and distributing a view of the world and manages reservation of resources.

1) Software Architecture: The function of each robot is separated into the three distinct layers responsible for deliberation (high level), i.e. decision making and planning, a reactive skill engine (mid-level) and low-level components for e.g. motion and vision.

The reasoning and planning component is implemented using a CLIPS rule engine [17]. This allows an incremental reasoning to derive at any time-point for each of robot a local optimal decision. The mid-level is designed as a Lua-based behavior engine [18]. With this, simple and complex skills can be modeled as a hybrid state machine. This modularity allows tuning and optimization of skills for specific tasks. The underlying robot framework used is Fawkes [19]. This framework is an alternative to ROS and provides several low-level functionalities as e.g. AMCL, hardware interfaces to the Robotino base and navigation plugins.

2) Light Detection: The light pattern detection (described in Section IV-A.2) is solved by the Carologistics Team using a more complex and more configuration-intense way. The region of interest (ROI) is cropped using the fusion of the camera and the laser scanner. The robot is aligned with the use of the mounted AR-tag. As this tag can be mounted arbitrarily on the machine, this only allows a course alignment. With the use of the laser scanner and the knowledge of the type of machine (via the AR-tag), the relative position of the mounted traffic light can be calculated. For this, the exact location of the light for each side of the machine is necessary with respect to the machine base. After this, the region of interest can be restricted a first time. With the knowledge of the position of the laser scanner as well as the camera, it is possible to calculate the position of this ROI in the image frame. Here several heuristics are used to find the shown traffic light, e.g. the fixed width to height ratio, that there have to be three distinct lights stacked in a vertical manner and much more. Having this, the state of the traffic light is determined using the color of the ROIs for the red, orange and green image section. This is done using a defined space for off and on in the YUV color space.

Our approach avoids the need for all the configuration by using the HOG-detector and the neural network. The detector eliminates the need for geometric heuristics and knowledge about the machine, and the neural network generalizes enough (trained with several lighting conditions) to detect the state of the traffic light without the need of a tuned color model. This allows more robustness as e.g. different lighting, or a displacement of the mounted camera or the laser scanner would lead to wrong classifications with the solution presented by the Carologistics team.

VI. Conclusion and future work

With the help of the next industrial revolution, it should be possible to produce individual configured products to the price of current mass-production. This ambitious scheme requires smart factories with modular machinery and an intelligent and flexible transportation system. Such transport can be provided by a fleet of autonomous robots. To offer a standardized testbed for different aspects of such smart factories the RoboCup Logistic League was established.

In this paper, we presented a software architecture which can be used to solve various problems appearing in the context of the RoboCup Logistic League. The software architecture consists of three layers which interact with clearly defined interfaces. The top layer manages the entire robotic fleet, generates an optimal global schedule, and is responsible for error detection and correction. For this, it uses the mid-layer which provides complex tasks (e.g. explore a zone, deliver a product). Here these skills are decomposed, and the mid-layer commands simple skills (move to a waypoint, open the gripper) to the lowest layer.

The software was successfully tested at the RoboCup world championship 2016 and allowed us to rank among the top three teams [20].

Besides the general software architecture, we described in this paper several components in more detail. These components allow the robot to explore an unknown factory. The presented components range from a scheduling mechanism to distribute the work onto the entire fleet down to mechanisms to detect the type of the machine defined by a signal light pattern.

The system is designed in such a way that faults are detected. Thus the system can react to faults properly. This allows the system to reliably to execute its task. To improve the reliability of our system even further the next step is to implement an online diagnosis system as described in [21]. This system can use different measures (e.g. publishing frequency of particular topics, time to respond to actions) to detect abnormal system behavior and furthermore calculate a diagnosis.

References

[1] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, "Industry 4.0," *Business & Information Systems Engineering*, vol. 6, no. 4, p. 239, 2014.

[2] T. Niemueller, D. Ewert, S. Reuter, A. Ferrein, S. Jeschke, and G. Lakemeyer, "Robocup logistics league sponsored by festo: A competitive factory automation testbed," in *Automation, Communication and Cybernetics in Science and Engineering 2015/2016*. Springer, 2016, pp. 605–618.

[3] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa, "Robocup: The robot world cup initiative," in *Proceedings of the first international conference on Autonomous agents*. ACM, 1997, pp. 340–347.

[4] U. Karras, D. Pensky, and O. Rojas, "Mobile robotics in education and research of logistics," in *IROS 2011–Workshop on Metrics and Methodologies for Autonomous Robot Teams in Logistics*, vol. 72, 2011.

[5] F. Zwilling, T. Niemueller, and G. Lakemeyer, "Simulation for the robocup logistics league with real-world environment agency and multi-level abstraction," in *Robot Soccer World Cup*. Springer, 2014, pp. 220–232.

[6] E. Gat *et al.*, "On three-layer architectures," *Artificial Intelligence and Mobile Robots*, vol. 195, p. 210, 1998.

[7] M. Georgeff, B. Pell, M. Pollack, M. Tambe, and M. Wooldridge, "The belief-desire-intention model of agency," in *International Workshop on Agent Theories, Architectures, and Languages*. Springer, 1998, pp. 1–10.

[8] F. F. Ingrand, R. Chatila, R. Alami, and F. Robert, "Prs: A high level supervision and control language for autonomous mobile robots," in *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, vol. 1. IEEE, 1996, pp. 43–49.

[9] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: An open-source robot operating system," in *ICRA Workshop on Open Source Software*, vol. 3, no. 3.2. Kobe, 2009, p. 5.

[10] K. Erol, J. A. Hendler, and D. S. Nau, "Umcp: A sound and complete procedure for hierarchical task-network planning." in *AIPS*, vol. 94, 1994, pp. 249–254.

[11] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte carlo localization: Efficient position estimation for mobile robots," *AAAI/IAAI*, vol. 1999, no. 343-349, pp. 2–2, 1999.

[12] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1. IEEE, 2005, pp. 886–893.

[13] M. F. Møller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural networks*, vol. 6, no. 4, pp. 525–533, 1993.

[14] J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neurocomputing*. Springer, 1990, pp. 227–236.

[15] T. Niemueller, T. Neumann, C. Henke, S. Schönitz, S. Reuter, A. Ferrein, S. Jeschke, and G. Lakemeyer, "Improvements for a robust production in the robocup logistics league 2016."

[16] T. Niemueller, G. Lakemeyer, and A. Ferrein, "The robocup logistics league as a benchmark for planning in robotics," in *WS on planning and robotics (PlanRob) at Int. Conf. on Aut. planning and scheduling (ICAPS)*, 2015.

[17] R. M. Wygant, "Clipsa powerful development and delivery expert system tool," *Computers & Industrial Engineering*, vol. 17, no. 1-4, pp. 546–549, 1989.

[18] T. Niemüller, A. Ferrein, and G. Lakemeyer, "A lua-based behavior engine for controlling the humanoid robot nao," in *Robot Soccer World Cup*. Springer, 2009, pp. 240–251.

[19] T. Niemueller, A. Ferrein, D. Beck, and G. Lakemeyer, "Design principles of the component-based robot software framework fawkes," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2010, pp. 300–311.

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

[21] S. Zaman, G. Steinbauer, J. Maurer, P. Lepej, and S. Uran, "An integrated model-based diagnosis and repair architecture for ros-based robot systems," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 482–489.