Winning the RoboCup Logistics League with Fast Navigation, Precise Manipulation, and Robust Goal Reasoning

Till Hofmann[1(✉)], Nicolas Limpert[2], Victor Mataré[2], Alexander Ferrein[2], and Gerhard Lakemeyer[1]

[1] Knowledge-Based Systems Group, RWTH Aachen University, Aachen, Germany
{hofmann,gerhard}@kbsg.rwth-aachen.de
[2] MASCOR Institute, FH Aachen University of Applied Sciences, Aachen, Germany
{limpert,matare,ferrein}@fh-aachen.de

Abstract. The RoboCup Logistics League is a robotics competition in a Smart Factory scenario in which a team of robots has to assemble products for dynamically generated orders. In 2019, the Carologistics was able to win the competition with a redesigned manipulation system, improved navigation, and an incremental and distributed goal reasoning system. In this paper, we describe the major components of our approach that enabled us to win the competition, with a particular focus on this year's changes.

1 Introduction

The Carologistics RoboCup Team[1] is a cooperation of the Knowledge-Based Systems Group (RWTH Aachen University) and the MASCOR Institute (FH Aachen University of Applied Sciences), which was initiated in 2012. Doctoral, master's, and bachelor' students of both partners participate in the project and bring in their specific strengths to tackle the various aspects of the RoboCup Logistics League.

In the RoboCup Logistics League (RCLL), the goal is to maintain and optimize the material flow in a simplified Smart Factory scenario. Two competing teams of three robots each need to fulfill dynamically generated orders by assembling workpieces to products of varying complexities, ordered from `C0` to `C3`. To assemble such products, the robots operate and transport workpieces between Modular Production System stations (MPSs). Each team has an exclusive set of seven machines of five different types, where each type of machine is capable to perform a different step of the production. The major challenges of the RCLL include navigation, perception and manipulation, as well as reasoning tasks such as planning, plan execution, and execution monitoring.

[1] https://carologistics.org/.

S. Chalup et al. (Eds.): RoboCup 2019, LNAI 11531, pp. 504–516, 2019.
https://doi.org/10.1007/978-3-030-35699-6_41

In the following, we describe our approach to the RCLL with a particular focus on the components that led to the success in 2019. To foster the development of the league, we have publicly released our software stack used in 2019[2]. We begin with the software architecture and major building blocks in Sect. 2 and summarize our development workflow in Sect. 3, which provided the means to effectively coordinate a team of ten developers in a competition environment. In Sect. 4, we describe our redesigned gripper system that allows precise grasping, and we summarize a multi-stage procedure using data from a laser range finder and an RGB/D camera to quickly and precisely align to a machine. We continue with improvements to path planning in Sect. 5, which enabled our robots to move across the playing field more quickly, a crucial aspect of a competitive production. In Sect. 6, we summarize our approach to high-level decision making using a goal reasoning approach with an incremental and distributed multi-agent strategy that is capable of an efficient production flow while reacting quickly to unexpected events, before we conclude in Sect. 7.

2 Architecture and Middleware

The software system of the Carologistics robots combines two different middlewares, Fawkes [13] and ROS [20]. This allows us to use software components from both systems. The overall system, however, is integrated using Fawkes. Adapter plugins connect the systems, for example to use ROS' 3D visualization capabilities. In addition, using ROS within Fawkes allows to benefit from well-tested software solutions that solve lower level problems taking important roles in the RCLL such as navigation or several debugging functionalities including ROSBag to record the behavior and sensory of a robot and allow analyzation of the robots behavior. The overall software structure is inspired by the three-layer architecture paradigm [6]. As shown in Fig. 1, it consists of a deliberative layer for high-level reasoning, a reactive execution layer for breaking down high-level commands and monitoring their execution, and a feedback control layer for hardware access and functional components. The changes to the manipulation workflow, described in Sect. 4 give an insight on these aspects. The topmost layer is detailed in Sect. 6. The communication between single components – implemented as *plugins* – is realized by a hybrid blackboard and messaging approach [13].

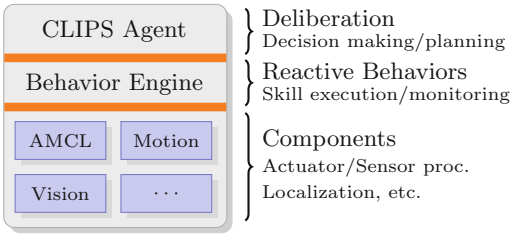


Fig. 1. Behavior layer separation [17]

[2] https://fawkesrobotics.org/p/rcll2019-release/.

2.1 Lua-Based Behavior Engine

In previous work we have developed the Lua-Based Behavior Engine (BE) [14]. It serves as the reactive layer to interface between the low- and high-level systems. The BE is based on hybrid state machines (HSM). They can be depicted as a directed graph with nodes representing states for action execution, and/or monitoring of actuation, perception, and internal state. Edges denote jump conditions implemented as Boolean functions. If a condition fires, the active state is changed to the target node of the edge. A table of variables holds information like the world model, for example storing numeric values for object positions. It remedies typical problems of state machines like fast growing number of states or variable data passing from one state to another. Behaviors, so-called *skills*, are implemented using the light-weight, extensible scripting language Lua.

3 Development Workflow

Over the years, it has become increasingly apparent that a healthy development workflow is an often underrated factor that can make or break a fast-paced robotics competition such as the RCLL. During an RCLL competition, matches are often played in quick succession, sometimes with only 1-h breaks before a team must be back on the playing field. Due to its relative expensiveness and complexity, the playing field is often only partially available for testing, so testing time is precious and must be well-coordinated. The short development windows create a tendency towards "quick-and-dirty" fixes, and the scarce testing often leads to uncertainty whether a feature can be considered stable or not.

In order to not end up with unusable code after a competition, these issues must be actively managed. During a typical, busy RCLL competition, there can easily be 50 or more feature branches in concurrent development. Here, it is most important to strictly separate branches that are supposed to become a stable feature (i.e. be merged back into the `master` branch) from location-specific tuning and dirty hacks. Although deployment to the robots is (aside from the merge conflicts) a straightforward task of simply merging all desired (stable or experimental) feature branches into a temporary (so-called `current`) branch, the job of feeding things back into the `master` remains important for a different reason: Since the `master` branch is the preferred starting point for all feature branches, it cannot fall too far back behind the on-going development. If some feature `A` is not in the `master`, any branch `B` that patches it has to be started from branch `A`, which should be avoided since it introduces an additional dependency. Even worse, if some feature `C` depends on two independent features `A` and `B`, at least one of `A` and `B` *must* be merged back into the master. Most importantly, branching any features off of the `current` branch is strictly forbidden, as `current`, and therefore anything branching from it, may contain things that turn out to be a bad idea.

To improve our assessment of code quality when merging back into the `master`, in 2019 we introduced continuous integration builds, a review process,

as well as syntactic and semantic linter checks. To further improve reproducibility and reliability of our setup, we use Ansible [7] for configuration management. This allows us to quickly set up a robot from scratch and guarantees that all robots are set up with the same configuration. Additionally, we use Ansible to deploy the most recent changes to the robots by updating all repositories on the robot with a single Ansible playbook.

4 Improvements to Manipulation

In an ongoing effort to optimize picking and placing actions in terms of time and reliability, we modified both the gripper's hardware design and the design of the software components. Combining mechanical gripper adaptions for more robustness and reliability with a fast model-based perception to precisely estimate poses of either conveyor belts or the MPS's slides has had a major influence on the success of our system this year. Most of our hardware modifications have been described in our previous team description papers [8–10,19], so here we will focus on new developments only.

4.1 Gripper System

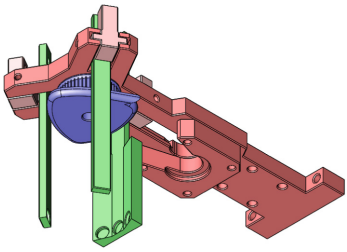


Fig. 2. New gripper with three self-centering fingers.

The Robotino base faces challenges on precise positioning in front of machines. The overall positioning tolerance of roughly ±2 mm on handling products at the MPS' conveyor belts requires precise motion. In 2018 we decided to extend the motion capabilities of the gripper [9] to overcome this issue. We added two axes for precise frontwards and sidewards gripper movement to get a higher placement tolerance of the Robotino base during product placing or picking actions. Previously, we could not align the Robotino with a sufficient precision, as it only allows motions with a certain speed threshold preventing position correction in the sub-centimeter range. The motor controllers have a limited minimal amount of rotations per minute which result in minimum translational velocities of 0.006 $\frac{\text{m}}{\text{s}}$. In practice, the motion reported by the wheels lags behind the frequency of our software framework. In turn, the robot oscillates when the positioning tolerance is set too low. Our current setup sets a minimum translational tolerance of 0.02 m. Additionally, the position estimation reported by the global localization is inaccurate in front of a machine, due to the limited number of laser beams matching global map components and imprecise absolute machine positions. As a result we have to rely on the sensory gathering precise machine and conveyor belt positions (as denoted in the following section).

To overcome issues resulting from mechanical requirements of the gripper itself it has been redesigned in 2019 to grip the workpiece from above with three

instead of two fingers, as shown in Fig. 2. The advantage is increased robustness and precision because the workpiece always centers between the three spring-loaded fingers, independently of any positioning error. Another advantage of the new design is that it retracts fully behind the robot's circular base shape, which significantly reduces the risk of damage and simplifies path planning.

4.2 Conveyor Belt Detection

Workpiece manipulation is one of the central challenges of the RCLL. In fact, production and delivery of a mere `C0` already involves six `pick` or `place` operations. For a `C2` with expensive rings, up to 18 manipulation operations are required. A manipulation failure is likely to result in a total loss of the handled product, so a single failure in handling an almost finished `C2` will easily cause the loss of an entire game.

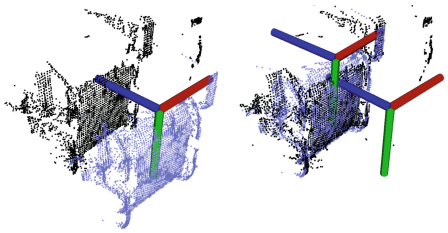


Fig. 3. Left: Model pointcloud (blue) roughly aligned to scene (black) based on initial guess. Right: After running ICP, the model is aligned to the scene precisely [8]. (Color figure online)

Consequentially, reliability is at the top of the priority list for manipulation in the RCLL. Since the conveyor belts are only a few millimeters wider than the workpieces, much of the reliability comes down to the precision of both the manipulator mechanics and the conveyor detection. We chose not to attempt detection of the workpieces themselves, but to concentrate on detecting the conveyor belts as reliably as possible.

The first step in that direction was to replace the notoriously unreliable RealSense F200 3D cameras with the newer SR300 model, which is equally suited for near-field operation with high resolution. It features a much more reliable USB 3.0 implementation and is supported by the redesigned `librealsense2`[3], which improves both OS support and stability. It is also better at dealing with the dark and reflective surfaces of most of the conveyor assembly, yielding less spotty pointclouds.

For maximum manipulation reliability, we need to determine the 6D pose (translation & rotation) of the conveyor belt. Our detection algorithm works purely on the 3D pointcloud data from the RealSense camera, disregarding any color information. For the RoboCup 2019, we built a multi-modal 3-stage pipeline that incorporates the two previous solutions as the first two stages, which each stage supplying an initial configuration for the next.

The first stage uses the LiDAR sensor mounted in the bottom base of the robot to roughly determine the pose of the MPS box in front of the robot. It uses a RANSAC approach [5] to fit a line parametrization that matches the width of

[3] https://github.com/IntelRealSense/librealsense.

the MPS table's flat side panel (error ≃20 mm). The second stage uses the rough MPS pose to run a RANSAC-based plane fitting, this time where the conveyor belt can be expected in the point cloud data from the RealSense 3D camera. This stage reduces the translational error to ∼5 mm, supplying a suitable initial estimate for the next stage. The third stage is a highly precise ICP-based model fitting approach [4], as shown in Fig. 3. Its iterative nature makes it much more computationally expensive and sensitive to bad initial estimates, but it turns out to be the most reliable way to further reduce the pose jitter to something below 2 mm.

For the last stage, the quality of the model (i.e. the reference point cloud) that is fitted into the live data is of great importance. It should be sufficiently complex, i.e. it should contain points that describe multiple, non-parallel surfaces so as to be sufficiently defined in the translational component. Symmetries should also be minimal, and it should be sufficiently large in order to achieve high rotational precision.

With these conditions satisfied, and some tuning to compensate mechanical assembly tolerances, we were able to play entire games with no manipulation failures towards the end of the competition.

5 Path Planning

Throughout an entire game, navigation is one of the tasks that takes up most of the time. Changing to ROS Navigation [12] in 2017 has proven to be a flexible and reliable approach well-suited for the difficult navigation tasks in the RCLL. Path planning in such an environment requires a robust but fast navigation solution suitable to provide collision-free navigation with dynamic obstacles. From an agent's perspective it is desired to decrease the average time to reach navigational goals, which mainly depends on the average velocity, but also on path efficiency.

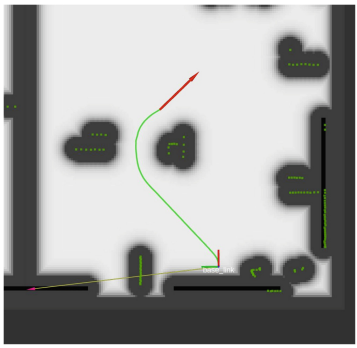
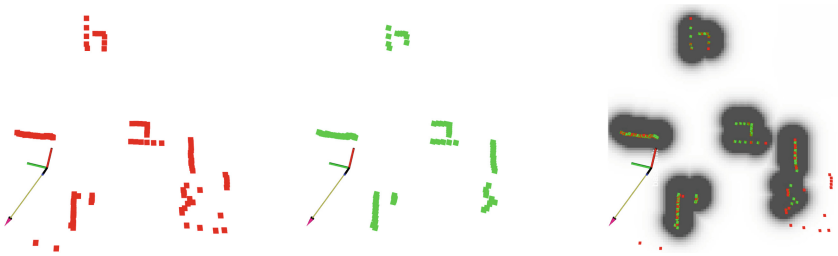


Fig. 4. Path planning in the RCLL. Obstacles are represented in occupancy grids which in turn are used for the path planning environment.

The classic separation between global and local path planning is of major use to quickly react to obstacles crossing the way in a small local frame. Figure 4 depicts an example of a robot navigating in the RCLL. The global path planner does not have to take kinematic constraints into account, as the Robotino base is omni-directional.

Since 2017, we have made use of a local planner based on a timed elastic band approach [22] as a controller running within our setup of ROS Navigation responsible for executing the planned path and optimizing a trajectory based on several properties of the robot base such as acceleration and velocity limits. The major advantage of this planner is its strafing capability. However, due to the

kinematic freedom of the platform, we found that the complex workflow of the timed elastic band approach is not needed for us. As such, we implemented our own local planner based on the Vector Field Histogram approach [3], giving us considerably high loop rates of roughly 90 Hz (compared to an average maximum of 10 Hz of the previously used planner) resulting in much lower reaction times on dynamic obstacles, which allows to move closer to obstacles and use narrow passages that otherwise might lead to collisions as the robots do not react in appropriate time. However, we eventually realized that this approach requires a more elaborate solution to be a drop-in replacement for our local planner. The timed-elastic-band–based local planner has built-in features to smoothen translational and angular accelerations given the current obstacles around the robot, the current robot's velocities, the calculated trajectory and the difference between the robot and the absolute goal, which are lacking from the Vector Field Histogram planner. Thus, we decided to keep the previously used setup using the well-tested local planner and instead consider other aspects of navigation that will increase efficiency.



(a) Raw laser data (b) Filtered laser data (c) Resulting costmap

Fig. 5. Laser filtering process. Figure 5(a) shows the unfiltered laser readings and the position of the robot in the bottom left. Figure 5(b) depicts the filtered laser. Figure 5(c) is the resulting costmap representing filtered obstacles in an overlay to the filtered and unfiltered laser beams. Note the single laser beams in the bottom left and right corners that are not taken into account.

The planning environment in ROS Navigation is represented with occupancy grids [11]. The local planner's computation time is largely determined by the size of the occupancy grid. Therefore, shrinking its size reduces the planning time significantly. However, this approach impairs the local planner's foresight, which we remedy by implementing an additional controller for the local planner's maximum velocity based on plain laser data. To make the approach fast and reliable, we limit the maximum velocity when any laser beam reports obstacles within certain thresholds. We use three thresholds, which were empirically determined: A maximum velocity of 0.8 $\frac{\text{m}}{\text{s}}$ if there are no obstacles, 0.6 $\frac{\text{m}}{\text{s}}$ if there are obstacles within 0.6 m, and 0.3 $\frac{\text{m}}{\text{s}}$ if there are obstacles within 0.3 m.

While this allows a maximum speed of 0.8 $\frac{\text{m}}{\text{s}}$ in free environments, the system suffers from sensor noise gathered by raw laser data. In many cases, the laser

reports false positives, showing obstacles at positions that are actually free, which results in occupied cells of the occupancy grid which are traversable. As a result, the path planner tries to avoid phantom obstacles. To overcome this issue, we filter the raw laser data by using an implementation of the fixed-radius near neighbour problem [2] in the PointCloud Library. Figure 5 shows an example scenario in which 247 beams are filtered down to 142 beams, while Fig. 5(c) shows the resulting costmap.

6 Goal Reasoning with the CLIPS Executive

We implemented an agent based on the the CLIPS Executive (CX) [16], which uses a goal reasoning model [1] similar to ActorSim [21]. We refer to [16] for an in-depth discussion of related work.

A goal describes objectives that the agent should pursue and can either *achieve* or *maintain* a condition or state. The program flow is determined by the *goal mode*, which describes the current progress of the goal. The mode transitions are determined by the goal lifecycle, as shown in Fig. 6. When a goal is created, it is first *formulated*, merely meaning that it may be relevant to consider. The goal reasoner may decide to *select* a goal, which is then *expanded* into one or multiple plans, either by using manually specified plans or automatic planners such as PDDL planners [15]. The reasoner then *commits* to one of those plans, which is *dispatched* after all required resources have been acquired, typically by executing a skill of the behavior engine. Eventually, the goal is *finished* and the outcome is *evaluated* to determine the goal's success.

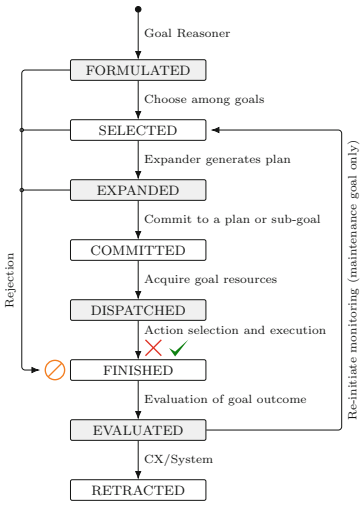


Fig. 6. The goal lifecycle with all possible goal modes [16]

The CX provides an explicit representation of the agent's world model, and its goals, plans, and actions. It separates the *domain model* with the available operators, predicates, and known facts from the *execution model*, which enhances the domain model by features that are only relevant for the execution of the plan, e.g., *exogenous actions* and *sensed predicates*.

Multi-robot Coordination. The CX also provides means for multi-robot coordination, in particular *world model synchronization*, *mutual exclusion*, and *resource allocation* [16]. To cooperate effectively, each agent must share (parts of) its world model with the other agents. The CX implements world model synchronization using a shared database [18,23]. Each robot uses a database instance for local (agent-specific) and global (shared) world model facts. The

global world model database is synchronized as part of a replica set with the global instances of the other robots.

Based on the replicated database, the CX also implements a locking mechanism. To lock a mutex, an agent must request a *majority acknowledgment*, thereby avoiding two agents to hold the same mutex. To allow *mutual exclusion*, the CX specifies two actions *lock* and *unlock*, which may be used by the agent just as any other action. Additionally, each goal may be associated with one or multiple resources that are required in order to dispatch the goal. If one resource is currently unavailable, the goal is *rejected*. The agent holds the resource for the whole lifetime of the goal, once a goal is *retracted*, its acquired resources are released automatically. In contrast to that, other mutexes locked by *lock* and *unlock* actions are acquired and released explicitly by plan actions during the execution of the goal's plan. Resource locks are typically used if the resource is consumed or changed by the goal's plan and any other goal related to the resource may become invalid after the goal has finished, while lock actions are typically used to guarantee short-term mutual exclusion, e.g., to avoid two robots moving to the same location.

6.1 Goal Reasoning in the RCLL

Using goal reasoning in the CX, we implemented an incremental and distributed strategy for the RCLL. We split the production of an order into multiple goals, such that each step of the production of one order is a separate goal and each goal starts and ends at an MPS without a workpiece in the robot's gripper. With this approach, a robot can easily switch between tasks and orders, e.g., first mount a ring for a `C2` and then switch to mounting a cap for a `C0`. All production goals are structured in a goal tree. The root of the tree is a `PRODUCTION` maintenance goal, with separate maintenance sub-trees for `URGENT`, `FULFILL-ORDERS`, `PREPARE-RESOURCES`, and `NO-PROGRESS` goals, with decreasing priorities of the order of the sub-trees. This way, the agent always selects an urgent goal if there is any, and otherwise tries to fulfill orders, e.g, by delivering a product. If this is not possible, it prepares resources not tied to a specific order, e.g., feeding raw material into the ring station. If there is no goal in any of the sub-trees, it selects a goal that does not progress the game, e.g., going to a waiting position.

Goal selection is solely based on the priority of the goal, i.e., we always select the formulated goal with the highest priority, which is currently defined manually for each goal class. Goals that continue the production of an already started product have higher priority than goals that start a new product, products of higher complexity have higher priority than products of lower complexity. Our goal selection implements an incremental strategy as our goal reasoner only decides which goal to pursue next rather than scheduling a set of goals ahead of time. We obtain a distributed multi-agent strategy that effectively fulfills order without an explicit decision which orders to pursue, thereby removing the need for a central agent.

Although the CX is capable of using a PDDL planner [15], we use a database of hand-crafted plans instead. This allows tighter control of the resulting plans

and better execution monitoring and also avoids planning overhead during execution, but reduces the flexibility of the goal reasoner as we cannot easily dispatch the same goal in slightly different situations, e.g., start dispatch a goal with the robot holding a workpiece. Currently, PDDL is only used for the execution model to check whether all preconditions of an action are satisfied before executing an action, and also to specify the actions' effects.

To coordinate the three robots, we use a distributed approach using the locking mechanisms of the CX. More specifically, a production goal requires a workpiece and an MPS as resource if they are altered by the goal's plan. Additionally, all plans contain lock actions for locations and machines. The resource locks guarantee that we only dispatch one goal that depends on the current state of the workpiece (e.g., the number of rings mounted on the base) or the machine (e.g., whether a cap station has a buffered cap). The additional lock actions guarantee that no two robots try to move to the same location or operate the same MPS station at the same time, even if the station is not a required resource. This allows more efficient plan dispatching, as one robot may already start the execution of a plan, even if the plan contains an action at a station that is currently occupied.

Execution Monitoring. In most robotics scenarios and in the RCLL in particular, plan execution may fail for a number of reasons. For one, an action may simply fail due to imprecise sensors or actuators, e.g., picking a workpiece from an MPS may fail because the gripper is not properly aligned to the MPS. Also, another robot, either from the same or the opposite team, may interfere, e.g., by blocking a location or resetting an MPS. Additionally, a plan may fail due to the inherent uncertainties in the RCLL, e.g., an MPS being `DOWN`. Finally, we also need to deal with exogenous events and their effects on the agent's world model. We can distinguish three kinds of events: (1) an action may fail to execute, (2) a plan no longer has the intended effect, and (3) a goal is no longer useful. The CX already supports generic monitoring rules, e.g., retrying a failed action a number of times. In addition to that, we also implemented domain-specific monitoring rules, which deal with the specific aspects of the RCLL. Most of these domain-specific rules were created by observing undesired behaviors in test environments. Typical examples include resetting an MPS if it is in an unexpected state and removing unknown workpieces from the input or output of a machine. To implement and test our execution monitoring, we adapted the Gazebo-based simulation of the RCLL [24] to incorporate random failures. With this strategy, we were able to develop extended monitoring rules even for scenarios that we could not observe frequently in the real world.

7 Conclusion

In 2019, we continued the development of the CLIPS Executive (CX), a goal reasoning system which we use to pursue an incremental and distributed multi-agent strategy for the RCLL in a principled way. We redesigned our gripper

system and replaced it with a three-finger manipulator that grasps workpieces from the top. We took a new approach to MPS alignment with a multi-stage strategy using RANSAC with a LiDAR sensor and ICP on an RGB/D image. We further improved our navigation, which allowed our robots to move more quickly across the playing field while avoiding collisions with static objects and robots of both our and the opposite team. All those changes provided the means to a more efficient production, which resulted in the team's success in the RoboCup Logistics League.

Acknowledgements. The team members in 2019 were David Bosen, Mario Claer, Sebastian Eltester, Christoph Gollok, Daniel Habering, Till Hofmann, Nicolas Limpert, Victor Mataré, Morian Sonnet and Tarik Viehmann.

Our special thanks go to T. Niemueller for his continued support and contributions both to the league and to our team.

We gratefully acknowledge the financial support of RWTH Aachen University and FH Aachen University of Applied Sciences.

T. Hofmann and V. Mataré were supported by the DFG grants *GL-747/23-1* and *FE-1077/4-1* (respectively) on Constraint-based Transformations of Abstract Task Plans into Executable Actions for Autonomous Robots (http://gepris.dfg.de/gepris/projekt/288705857).

N. Limpert was partly supported by the H2020 ROSIN project under grant agreement No 732287 (https://cordis.europa.eu/project/rcn/206395/factsheet/en) on ROS-Industrial quality assured software components.

We appreciate the financial and organizational support by the Cybernetics Lab IMA & IfU, RWTH Aachen University.

We are especially thankful to *Hans-Hermann-Voss-Stiftung* (https://www.hans-hermann-voss-stiftung.de/) for their financial support.

We thank our sponsors *ELTROPULS GmbH* (https://www.eltropuls.de/) and *Magazino GmbH* (https://www.magazino.eu/) for travel funding, as well as *AGVR GmbH* (http://www.agvr.eu/), *igus GmbH* (https://www.igus.de/) and *SICK AG* (https://www.sick.com/) for providing hardware and development support.

References

1. Aha, D.W.: Goal reasoning: foundations, emerging applications, and prospects. AI Mag. **39**(2), 3–24 (2018)
2. Bentley, J.L., Stanat, D.F., Williams Jr., E.H.: The complexity of finding fixed-radius near neighbors. Inf. Process. Lett. **6**(6), 209–212 (1977)
3. Borenstein, J., Koren, Y.: The vector field histogram-fast obstacle avoidance for mobile robots. IEEE Trans. Robot. Autom. **7**(3), 278–288 (1991)
4. Chen, Y., Medioni, G.: Object modelling by registration of multiple range images. Image Vis. Comput. **10**(3), 145–155 (1992)
5. Fischler, M.A., Bolles, R.C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM **24**(6), 381–395 (1981)
6. Gat, E.: Three-layer architectures. In: Kortenkamp, D., Bonasso, R.P., Murphy, R. (eds.) Artificial Intelligence and Mobile Robots, pp. 195–210. MIT Press, Cambridge (1998)

7. Hochstein, L., Moser, R.: Ansible: Up and Running. O'Reilly, Sebastopol (2014)
8. Hofmann, T., Limpert, N., Mataré, V., Ferrein, A., Lakemeyer, G.: The Carologistics RoboCup Logistics Team 2019. Technical report, RWTH Aachen University and FH Aachen University of Applied Sciences (2019). https://kbsg.rwth-aachen.de/~hofmann/papers/carologistics-2019-tdp.pdf
9. Hofmann, T., et al.: The Carologistics RoboCup Logistics Team 2018. Technical report, RWTH Aachen University and FH Aachen University of Applied Sciences (2018). https://kbsg.rwth-aachen.de/~hofmann/papers/carologistics-2018-tdp.pdf
10. Hofmann, T., et al.: Enhancing software and hardware reliability for a successful participation in the RoboCup Logistics League 2017. In: Akiyama, H., Obst, O., Sammut, C., Tonidandel, F. (eds.) RoboCup 2017. LNCS (LNAI), vol. 11175, pp. 486–497. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00308-1_40
11. Lu, D.V., Hershberger, D., Smart, W.D.: Layered costmaps for context-sensitive navigation. In: 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014), pp. 709–715. IEEE (2014)
12. 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)
13. Niemueller, T., Ferrein, A., Beck, D., Lakemeyer, G.: Design principles of the component-based robot software framework fawkes. In: International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR) (2010)
14. Niemüller, T., Ferrein, A., Lakemeyer, G.: A Lua-based behavior engine for controlling the humanoid robot Nao. In: Baltes, J., Lagoudakis, M.G., Naruse, T., Ghidary, S.S. (eds.) RoboCup 2009. LNCS (LNAI), vol. 5949, pp. 240–251. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-11876-0_21
15. Niemueller, T., Hofmann, T., Lakemeyer, G.: CLIPS-based execution for PDDL planners. In: ICAPS Workshop on Integrated Planning, Acting and Execution (IntEx) (2018)
16. Niemueller, T., Hofmann, T., Lakemeyer, G.: Goal reasoning in the CLIPS Executive for integrated planning and execution. In: Proceedings of the 29th International Conference on Planning and Scheduling (ICAPS) (2019)
17. Niemueller, T., Lakemeyer, G., Ferrein, A.: Incremental task-level reasoning in a competitive factory automation scenario. In: Proceedings of AAAI Spring Symposium 2013 - Designing Intelligent Robots: Reintegrating AI (2013)
18. Niemueller, T., Lakemeyer, G., Srinivasa, S.: A generic robot database and its application in fault analysis and performance evaluation. In: IEEE International Conference on Intelligent Robots and Systems (IROS) (2012)
19. Niemueller, T., et al.: Improvements for a robust production in the RoboCup Logistics League 2016. In: Behnke, S., Sheh, R., Sarıel, S., Lee, D.D. (eds.) RoboCup 2016. LNCS (LNAI), vol. 9776, pp. 589–600. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68792-6_49
20. Quigley, M., et al.: ROS: an open-source Robot Operating System. In: ICRA Workshop on Open Source Software (2009)
21. Roberts, M., Alford, R., Shivashankar, V., Leece, M., Gupta, S., Aha, D.W.: ActorSim: a toolkit for studying goal reasoning, planning, and acting. In: WS on Planning and Robotics (PlanRob) at International Conference on Automated Planning and Scheduling (ICAPS), London, UK (2016)

22. Rosmann, C., Feiten, W., Wosch, T., Hoffmann, F., Bertram, T.: Efficient trajectory optimization using a sparse model. In: European Conference on Mobile Robots (ECMR), pp. 138–143. IEEE (2013)
23. Zwilling, F.: A Document-oriented robot memory for knowledge sharing and hybrid reasoning on mobile robots. Master's thesis, RWTH Aachen University (2017)
24. Zwilling, F., Niemueller, T., Lakemeyer, G.: Simulation for the RoboCup logistics league with real-world environment agency and multi-level abstraction. In: RoboCup Symposium (2014)