The Carologistics RoboCup Logistics Team 2014

Tim Niemueller\textsuperscript{1}, Sebastian Reuter\textsuperscript{2}, Daniel Ewert\textsuperscript{2}, Alexander Ferrein\textsuperscript{3}, Sabina Jeschke\textsuperscript{2}, and Gerhard Lakemeyer\textsuperscript{1}

\textsuperscript{1} Knowledge-based Systems Group, RWTH Aachen University, Germany  
\textsuperscript{2} Institute Cluster IMA/ZLW & IfU, RWTH Aachen University, Germany  
\textsuperscript{3} Electrical Engineering Department, FH Aachen, Germany

Abstract. In this team description paper, we outline the approach of the Carologistics team with an emphasis on the high-level reasoning system and our simulation to develop it. We outline the hardware modifications and describe our software systems and describe our efforts towards a fully autonomous referee box. The team members in 2014 are Daniel Ewert, Alexander Ferrein, Nicolas Limpert, Matthias Löbach, Randolph Maassen, Victor Mataré, Tobias Neumann, Tim Niemueller, Florian Nolden, Sebastian Reuter, Johannes Rothe, and Frederik Zwilling.

1 Introduction

The Carologistics RoboCup Team is a cooperation of the Knowledge-based Systems Group, the IMA/ZLW & IfU Institute Cluster (both RWTH Aachen University), and the Electrical Engineering Department (Aachen University of Applied Sciences) initiated in 2012. Doctoral, master, and bachelor students of all three partners participate in the project and bring in their specific strengths tackling the various aspects of the RoboCup Logistics League sponsored by Festo (LLSF): designing hardware modifications, developing functional software components, system integration, and high-level control of a group of mobile robots.

Fig. 1. Carologistics (three robots with omni-vision tower) and TUMBendingUnits (robots on the left and right) during the LLSF finale at the German Open 2014
Our team has participated in RoboCups 2012 and 2013, and the RoboCup German Open (GO) 2013 and 2014. We were able to win the GO 2014 (cf. Figure 1) in particular demonstrating robust collision avoidance and self-localization, and with a flexible high-level control system. We have been active members of the Technical and Organizational Committees and proposed various ground-breaking changes for the league like merging the two playing fields into one or using physical processing machines in 2015 [1,2].

In the following we describe our robots and software components (Section 2), aspects of our task coordination (Section 3), our new simulation with real-world agency and multi-level abstraction (Section 4), and our involvement in the development of the LLSF Referee Box (Section 5). We conclude in Section 6.

2 The Carologistics Robotino Robots

The modified Robotino depicted in Figure 2(a) used by the Carologistics RoboCup team features two additional Logitech webcams and a Sick laser range finder. One of the webcams is used for recognizing the signal lights of the production machines, the other to detect pucks in front of the robot. The former omnidirectional camera is no longer used as it was prone to distortion and its time-intensive calibration. The webcams are mounted with serrated locking plates for a firm adjustment to defined angles. The Sick TiM551 laser scanner is used for collision avoidance and self-localization. In comparison to the Hokuyo laser scanner with a scanning range of 4 meters last year, the Sick TiM551 has a maximal scanning of 10 meters. An additional laptop on the robot increases the computation power and allows for more elaborate methods for self-localization, computer vision, or navigation. A custom-made passive guidance device is mounted to the front of the robots to allow for proper control of the pucks. Optical sensors mounted to the guidance device are used to measure the longitudinal distance for approaching the signal lights.

(a) Carologistics Robotino 2014

(b) Visualization of a scene in rviz

Fig. 2. Carologistics Robotino, sensor processing, and visualization
2.1 Middleware and Functional Software Components

The software system of the Carologistics robots combines two different middlewares, Fawkes [3] and ROS [4]. 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 (cf. Figure 2(b)). All functional components, like self-localization based on Adaptive Monte Carlo Localization, are implemented in Fawkes. For locomotion, we integrated the collision avoidance module [5] which is also used by the AllemaniACs® RoboCup@Home robot. A new component currently in development is a vision-based machine detection module. It will allow to detect and approach the machines more precisely as it yields a 3D pose. Figure 3 shows the visualization of the extracted features.

3 High-level Decision Making and Task Coordination

Task coordination is performed using an incremental reasoning approach [6]. In the following we describe the behavior components, and the reasoning process in two particular situations from the rules in 2014. For computational and energy efficiency, the behavior components need also to coordinate activation and deactivation of the lower level components to solve computing resource conflicts.

3.1 Behavior Components for the LLSF

asks that the high-level reasoning component of the robot must fulfill in the LLSF are:

**Exploration:** Gather information about the machine types by sensing and reasoning to gain more knowledge, e.g., the signal lights’ response to certain types of pucks.

**Production:** Complete the production chains as often as possible dealing with incomplete knowledge.

**Execution Monitoring:** Instruct and monitor the reactive mid-level Lua-based behavior engine.

A group of three robots perform these steps cooperatively, that is, they communicate information about their current intentions, acquire exclusive control over resources like machines, and share their beliefs about the current state of the environment. This continuous updating of information suggests an incremental reasoning approach. As facts become known, the robot needs to adjust its plan.

---

*See the AllemaniACs website at [http://robocup.rwth-aachen.de](http://robocup.rwth-aachen.de)*
3.2 Lua-based Behavior Engine

In previous work we have developed the Lua-based Behavior Engine (BE) [7]. It mandates a separation of the behavior in three layers, as depicted in Figure 4: the low-level processing for perception and actuation, a mid-level reactive layer, and a high-level reasoning layer. The layers are combined following an adapted hybrid deliberative-reactive coordination paradigm with the BE serving 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. For the active state of a state machine, all outgoing conditions are evaluated, typically at about 15 Hz. 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. Skills are implemented using the light-weight, extensible scripting language Lua.

3.3 Incremental Reasoning Agent

The problem at hand with its intertwined world model updating and execution naturally lends itself to a representation as a fact base with update rules for triggering behavior for certain beliefs. We have chosen the CLIPS rules engine [8], because using incremental reasoning the robot can take the next best action at any point in time whenever the robot is idle. This avoids costly re-planning (as with approaches using classical planners) and it allows us to cope with incomplete knowledge about the world. Additionally, it is computationally inexpensive.

```
(defrule load-T5-with-S0
  (declare (salience ?*PRIORITY-LOAD-T5-WITH-S0*)))
  (phase PRODUCTION)
  (?s <- (state IDLE))
  (holding NONE |SO)
  (team-color ?team-color)
  (machine (mtype T5) (loaded-with $?l &~:(member$ S0 ?l))
    (incoming $?i &~:(member$ BRING_S0 ?i)) (name ?name)
    (produced-puck NONE) (team ?team-color))
  =>
  (printout t "PROD: Loading T5 " ?name " with S0" crlf)
  (assert (proposed-task (name load-with-S0) (args (create$ ?name))))
  (retract ?s)
  (assert (state TASK-PROPOSED)))
```

Fig. 5. CLIPS Production Process Rule
The CLIPS rules are roughly structured using a fact to denote the current overall state that determines which subset of the rules is applicable at any given time. For example, the robot can be idle and ready to start a new sub-task, or it may be busy moving to another location. Rules involved with physical interaction typically depend on this state, while world model belief updates often do not. The state is also required to commit to a certain action and avoid switching to another one if new information, e.g., contributed by other robots on the field, becomes available. While it may be better in the current situation to pursue another goal, aborting or changing an action usually incurs much higher costs.

The rules explained in the following demonstrate what we mean by incremental reasoning. The robot does not create a full-edged plan at a certain point in time and then executes it until this fails. Rather, when idle it commits to the ‘then-best’ action. As soon as the action is completed and based on its knowledge, the next best action is chosen. The rule base is structured in six areas: exploration, production step decision, coordination with other robots, process execution, world modeling, and utilities.

In Figure 5 we show a simplified rule for the production process. The game is in the Production phase, the robot is currently idle and holds a raw material puck $S_0$ or no puck: (phase PRODUCTION) (state IDLE) (holding NONE|S0). Furthermore there is a T5-machine, whose team-color matches the team-color of the robot, which has no produced puck, is not already loaded with an $S_0$, and no other robot is currently bringing an $S_0$. If these conditions are satisfied and *PRIORITY-LOAD-T5-WITH-S0* is the highest priority of currently active rules, the rule fires proposing to load the machine with the name ?name with an puck in state $S_0$. It also switches the state.

There is a set of such production rules with their conditions and priorities determining what the robot does in a certain situation, or – in other terms – based on a certain belief about the world in the fact base. This simplifies adding new decision rules. The decisions can be made more granular by adding rules with more restrictive conditions and a higher priority.

```clips
(defrule goto-proc-complete
  (declare (salience ?*PRIORITY-WM*)))
  (state GOTO-FINAL)
  (goto-target ?name)
  ?h <- (holding ?)
  (lights GREEN-ON YELLOW-OFF RED-OFF)
  (machine (name ?name) (output ?output) (loaded-with ?$lw) (junk ?$jn))
=>
  (printout t "Production completed at " ?name "|" ?mtype crlf)
  (retract ?h)
  (assert (holding ?output))
  (foreach ?puck ?$lw
    (assert (worldmodel-change (machine ?name)
        (change REMOVE_LOADED_WITH) (value ?puck)))
  )
  (assert (worldmodel-change (machine ?name) (change SET_NUM_CO)
        (amount (+ ?$jn (length$ ?$lw))))))
```
After a proposed task was chosen, the coordination rules of the agent cause communication with the other robots to announce the intention and ensure that there are no conflicts. If the coordination rules accept the proposed task, process execution rules perform the steps of the task (e.g., getting an $S_0$ from the input storage and bringing it to the machine). Here, the agent calls the Behavior Engine to execute the actual skills like driving to the input storage and loading a puck.

The world model holds facts about the machines and their state, what kind of puck the robot is currently holding (if any) and the state of the robot. A simplified example for a world model update is shown in Figure 6. The world model update rule is invoked after a task or sub-task from the production rule presented above was successfully completed, i.e., an $S_0$ puck was taken to a machine of the type $T_5$. The rule shows the inference of the output puck type given a machine’s reaction. The conditions (state GOTO-FINAL)(goto-target ?name) denote that the robot finished locomotion and production at the target machine ?name. Furthermore, the robot sees only a green light at the machine, which indicates that the machine successfully finished the production. If all these conditions hold, the rule updates the world model about what kind of puck the robot is holding. Additionally, it assumes all pucks removed that were loaded in the machine and increases the amount of consumed pucks. The world model is synchronized with other robots with another set of rules.

In comparison to 2013, the agent evolved to enable a tighter cooperation of the three agents. This required smaller atomic tasks, which are performed by the agents, a coordination mechanism to ensure the robots perform no redundant actions, more fine-grained production rules, and synchronization of the world model. The latter allows for dynamically adding or removing robots without interference to the overall production process. Furthermore, the agent became more robust against failure of behavior execution and wrong perception by adding a set of more distinctive world model update rules.

4 Multi-robot Simulation in Gazebo

The character of the LLSF game emphasizes research and application of methods for efficient planning, scheduling, and reasoning on the optimal work order of production processes handled by a group of robots. An aspect that distinctly separates this league from others is that the environment itself acts as an agent by posting orders and controlling the machines’ reactions. This is what we call environment agency. Naturally, dynamic scenarios for autonomous mobile robots are complex challenges in general, and in particular if multiple competing agents are involved. In the LLSF, the large playing field and material costs are prohibitive for teams to set up a complete scenario for testing, let alone to have two teams of robots. Additionally, members of related communities like planning and reasoning might not want to deal with the full software and system complexity. Still, they often welcome relevant scenarios to test and present their research. Therefore, we have created an open simulation environment [9] to support research and development. There are three core aspects in this context:
Fig. 7. The simulation of the LLSF in Gazebo. The circles above the robots indicate their localization and robot number.

1. The simulation should be a turn-key solution with simple interfaces,
2. the world must react as close to the real world as possible, including in particular the machine responses and signals, and
3. various levels of abstraction are desirable depending on the focus of the user, e.g. whether to simulate laser data to run a self-localization component or to simply provide the position (possibly with some noise).

In recent work [9], we provide such an environment. It is based on the well-known Gazebo simulator addressing these issues: (1.) its wide-spread use and open interfaces already adapted to several software frameworks in combination with our models and adapters provides an easy to use solution; (2.) we have connected the simulation directly to the referee box, the semi-autonomous game controller of the LLSF, so that it provides precisely the reactions and environment agency of a real-world game; (3.) we have implemented multi-level abstraction that allows to run full-system tests including self-localization and perception or to focus on high-level control reducing uncertainties by replacing some lower-level components using simulator ground truth data. This allows to develop an idealized strategy first, and only then increase uncertainty and enforce robustness by failure detection and recovery. More information, media, and the software itself are available at http://www.fawkesrobotics.org/projects/llsf-sim/.

In the LLSF, the large playing field and material costs are prohibitive for teams to set up a complete scenario, let alone to have two teams of robots. Additionally, members of related communities like planning and reasoning might not want to deal with the full software and system complexity. Still they often welcome relevant scenarios to test and present their research. Therefore, we propose a new simulation sub-league for the LLSF based on our simulation [9].

5 LLSF Referee Box

The Carologistics team has developed the autonomous referee box (refbox) for the LLSF which was deployed in 2013 [1]. It strives for full autonomy on the game controller, i.e. it tracks and monitors all puck and machine states, creates (randomized) game scenarios, handles communication with the robots, and
interacts with a human referee. In 2014 the refbox has been adapted to the merged fields and two opposing teams on the field at the same time. We have also implemented a basic encryption scheme for secured communications.

6 Conclusion

The Carologistics RoboCup team has developed extensions for the Robotino hardware platform and an open software system based on the Fawkes and ROS frameworks. An incremental task-level reasoning approach is employed to deal with incomplete knowledge, computational constraints, and formal encoding of the behavior. With our Gazebo-based physical 3D simulation environment we are able to simulate complete games in a realistic fashion.

The website of the Carologistics RoboCup Team with further information and media can be found at http://www.carologistics.org.

Acknowledgments. F. Zwilling and T. Niemueller were supported by the German National Science Foundation (DFG) research unit FOR 1513 on Hybrid Reasoning for Intelligent Systems (http://www.hybrid-reasoning.org).

References