Incremental Task-level Reasoning in a Competitive Factory Automation Scenario

Tim Niemueller and **Gerhard Lakemeyer**
Knowledge-based Systems Group
RWTH Aachen University, Aachen, Germany
{niemueller, gerhard}@cs.rwth-aachen.de

Alexander Ferrein
Electrical Engineering Department
FH Aachen, Aachen, Germany
ferrein@fh-aachen.de

Abstract

Facing the fourth industrial revolution, *autonomous mobile* robots are expected to play an important role in the production processes of the future. The new Logistics League Sponsored by Festo (LLSF) under the RoboCup umbrella focuses on this aspect of robotics to provide a benchmark testbed on a common robot platform. We describe certain aspects of the integrated robot system of our *Carologistics* RoboCup team, in particular our reasoning system for the supply chain problem of the LLSF. We approach the problem by deploying the CLIPS rules engine for product planning and dealing with the incomplete knowledge that exists in the domain and show that it is suitable for computationally limited platforms.

1 Introduction

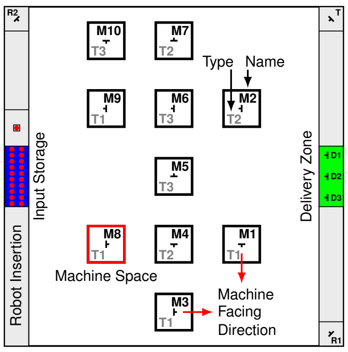
Today, we are in the middle of another industrial revolution. It is sometimes called the fourth industrial revolution; the first came with the invention of the steam engine, the second came with the invention of the assembly line and the third came with the computer and the Internet. Now we are facing the fourth industrial revolution. Production is changing right now and is going to change dramatically in the near future. Instead of mass production, customized products will be the future. This has among others the effect that the designer of the product will be in closer contact with its manufacturer and it is believed, for instance, that by 2020 10–30 % of the products that the USA are importing from China today could be produced inland (The Economist 2012). The new production will be supported by so-called cyber-physical systems (CPS). These systems combine computation with physical processes. They include embedded computers and networks which monitor and control the physical processes and have a wide range of applications in assisted living, advanced automotive systems, energy conservation, environmental control, critical infrastructure control, smart structure or manufacturing (Lee 2008). In the new scenario, *autonomous mobile robots* will play an important role in customizing the production and logistics processes, demonstrated by the recent acquisition of Kiva Systems, which produce logistics robots, by Amazon (Kucera 2012).

CPS will challenge the field of mobile robotics and AI. While robot systems today start to work reliably in the lab over extended periods of time, we are far away from operating our mobile robots as reliably as today's production robotic arms soldering car parts. Reasons lie in the much higher complexity of the environment that these robots have to operate in and which, moreover, is no longer for their exclusive use but shared with humans, as well as the more variable, deliberative tasks that these robots have to fulfill.

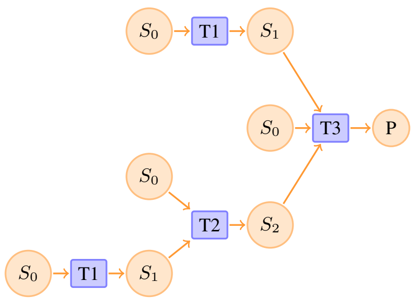
In this paper we address a new AI and robotics challenge that might become a benchmark for the next generation factory scenario. Under the umbrella of RoboCup, in 2012 the first competition in the *Logistics League sponsored by Festo* (LLSF) took place. The idea of this new RoboCup league is to simulate a production process where an assembly robot has to bring goods to production places to refine them step by step to a final product. A team of robots has to explore which machine is producing/refining what semi-finished products and produce as many products as possible.

As our main contribution to "*reintegrating AI*", we present our reasoning component implemented in the rule-based system CLIPS (Wygant 1989). Exemplarily, we show how the domain is encoded, and how rules are structured to combine sensing and reasoning to learn about the environment, and how incremental reasoning is used to handle incomplete knowledge by providing a next-best action at any point in time, that is, whenever the robot is idle. We also demonstrate the suitability of the approach for computationally limited platforms. In addition we want to discuss if the LLSF is suited as a benchmark for real-world supply chain optimization tasks in the future. Quite possibly it may make new areas like factory automation engineering accessible that are otherwise less receptive to AI methods.

Next we give a brief introduction to the LLSF and describe the *Carologistics* RoboCup Team's enhanced Robotino platform. In Section 3 we give some background on CLIPS, which is used for a first solution to the LLSF domain. In Section 4, we show how scheduling, planning and execution are intertwined on our system accounting for the four different aspects of the high-level system: *exploration*, *production*, *execution monitoring*, and *simulation*. Depending on the world state and phase, the CLIPS rule engine will schedule which product should be produced next and will instruct the behavior execution system. We give several examples of CLIPS world model and production process rules. We close with a discussion and an outlook to future work.



(a) LLSF Competition Area



(b) LLSF Production Tree

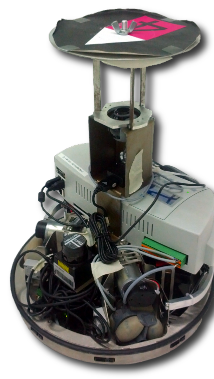

(c) The Carologistics Robotino

Figure 1: The competition area, production sequence tree, and robot used in the RoboCup Logistics League

2 Scenario

RoboCup (Kitano et al. 1997) is an international competition for academia and industry that serves as a testbed for robot applications. To address different aspects of mobile robotics and AI research, RoboCup competitions so far focused on robotic soccer with different types of robots, disaster and rescue missions, and domestic service robots. Recently, new leagues have been established for industrial and logistics applications. In the following, we describe the new Logistics League and the robot of our *Carologistics* RoboCup team.

RoboCup Logistics League

In 2012 the Logistics League Sponsored by Festo (LLSF) was officially founded. The general intention is to create a simplified and abstracted *factory automation scenario*. Teams of up to three robots operate in a fenced area of about 5.6 m $\times$ 5.6 m as shown in Figures 1(a) and 2(a). The task is to complete a production chain by carrying a (semi-finished) product (a puck in the game) along different machines (signal lights on the playing field). Points are awarded for intermediate and completed products.

The field contains an input storage containing the "raw material", and a delivery zone with gates to which final products must be delivered. Each puck has a programmable radio frequency identification (RFID) chip with which the different product states S_0, S_1, S_2, and P are distinguished. Initially, all pucks are in state S_0. In the enclosed inner field, ten signals equipped with an RFID device mounted on its front represent production machines. Each machine is assigned a random but defined type out of the types T1–T3, which is initially unknown to the robots. The type determines the input and output of a machine. Pucks transition through their states by being processed by machines. The complete production tree is shown in Figure 1(b). Circular nodes indicate a puck's state and rectangular nodes show the respective machine type. For example, the T1 machine in the upper branch takes an S_0 puck as input with an S_1 puck as output. If a machine, like T2, requires multiple inputs, these can be presented to the machine in any order. However, until the machine cycle completes, all involved pucks must remain in the machine space. The last input puck will be converted to the output puck, all others become junk and must be recycled.

The machines indicate their state after processing a puck using light signals. A green signal means that the particular machine production cycle has been completed, i.e., all required input products have been presented one after another, and now the puck has been transformed to the machines respective output, for example, after a T1 machine transformed a puck from state S_0 to S_1. An orange light indicates partial success (more pucks are required).

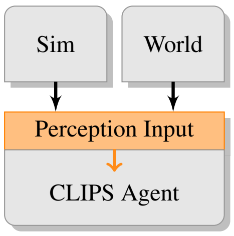
Besides typical robotics tasks such as motion planning or self-localization, the robot needs to plan an efficient sequence of actions to produce as many products as possible in a fixed amount of time. Moreover, the robot has to deal with incomplete knowledge as it is not known in advance what machine has which type. Thus, the robots need to combine *sensing and reasoning* to incrementally update their beliefs about the world. Based on the knowledge gained, it has to to find a strategy to maximize its production output, ideally minimizing costs such as travel distance.

The Carologistics Robotino Robots

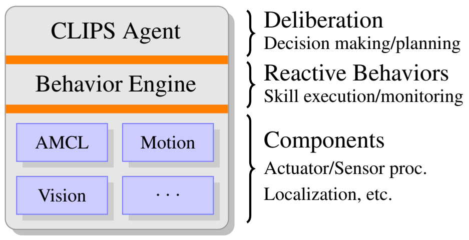
As the company Festo is sponsoring this exciting new RoboCup league, one condition is to enter the competition with Robotino robots, a small mobile robot platform designed for educational purposes by Festo Didactic.[1] It employs omni-directional locomotion, features twelve infrared distance sensors and bumpers mounted around the base, a CruizCore gyroscope, and a webcam facing forward. The Carologistics Robotinos have an additional omni-directional camera system, taken from the AllemaniACs' former middle-size league soccer robots (Beck and Niemueller 2009), which allows for a 360° view around the robot. It is used to detect pucks around the robot. The webcam is used for recognizing the signal lights of the production machines. An additional 2D Hokuyo URG laser scanner provides data for collision avoidance and self-localization. Our robot is shown in Figure 1(c). The major draw-back of the current Robotino platform is its limited computing power

[1]Information is available at `http://www.robotino.de`

(a) LLSF Scenario



(b) External Interface



(c) Behavior Layer Separation

Figure 2: LLSF scenario in-game photo, the equality of simulation and real-world input, and the behavior layer separation

with an AMD LX800 processor at 500 MHz and 256 MB of RAM. This is barely enough for typical robot software components, let alone a state-of-the-art reasoning system. This makes it particularly challenging to solve the LLSF task. In 2012 no extension in terms of computing power was allowed, a restriction no longer present in 2013.

The software system of the Carologistics robots combines two different middlewares, Fawkes (Niemueller et al. 2010) and ROS (Quigley et al. 2009). This allows us to use software components from both systems. The overall system, however, is integrated using Fawkes. Adapter plugins connect ROS and Fawkes, for example to use ROS' navigation and 3D visualization capabilities. Most of the functional components are implemented in Fawkes. For self-localization we use the Adaptive Monte Carlo Localization. Two image processing plugins were created, one for the omni-directional and one for the directed camera. The former is used to detect pucks around the robot. The directed camera is used to recognize the light signals. From ROS we use the locomotion package (move_base) which implements a dynamic window approach for local motion and collision avoidance and a Dijkstra search for a global path. The behavior components have been developed on top of Fawkes, but could easily be used in ROS. Since the computing power is rather limited on the Robotino, the behavior components need to coordinate activation and deactivation of the lower level components to solve computing resource conflicts. The behavior components are described in more detail in Section 4. Next, we briefly introduce the CLIPS rule engine.

3 CLIPS Rules Engine

CLIPS is a rule-based production system using forward chaining inference based on the Rete algorithm (Forgy 1982). The CLIPS rule engine (Wygant 1989) has been developed and used since 1985 and is thus mature and stable. It was designed to integrate well with the C programming language[2], which specifically helps to integrate with robot software like Fawkes or ROS. Its syntax is based on LISP.

CLIPS has three building blocks (Giarratano 2007): a fact base or working memory, the knowledge base, and an inference engine. *Facts* are basic forms representing pieces of information which have been placed in the fact base. They are the fundamental unit of data used by rules. Facts can adhere to a specified template. It is established with a certain set of slots, properties with a name which take on values of various types. The *knowledge base* comprises heuristic knowledge in the form of rules, and procedural knowledge in the form of functions. *Rules* are a core part of the production system. They are composed of an antecedent and consequent. The antecedent is a set of conditions, typically patterns which are a set of restrictions that determine which facts satisfy the condition. If all conditions are satisfied based on the existence, non-existence, or content of facts in the fact base the rule is activated and added to the agenda. The consequent is a series of actions which are executed for the currently selected rule on the agenda, for example to modify the fact base. *Functions* carry procedural knowledge and may have side effects. They can also be implemented in C++. In our framework, we use them to utilize the underlying robot software, for instance to communicate with the reactive behavior layer described below. CLIPS' *inference engine* combines working memory and knowledge base. Fact updates, rule activation, and agenda execution are performed until stability is reached and no more rules are activated. Modifications of the fact base are evaluated if they activate (or deactivate) rules from the knowledge base. Activated rules are put onto the agenda. As there might be multiple active rules at a time, a conflict resolution strategy is required to decide which rule's actions to execute first. In our case, we order rules by their salience, a numeric value where higher value means higher priority. If rules with the same salience are active at a time, they are executed in the order of their activation, and thus in the order of their specification. The execution of the selected rule might itself trigger changes to the working memory, causing a repetition of the cycle.

4 Behavior Components for the LLSF

In the described scenario, tasks that the high-level reasoning component of the robot should fulfill 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 behavior engine.

Simulation: Simulate the perception inputs of the high-level system's decisions for an arbitrary world situation to perform offline spot tests of the agent program.

[2]And C++ using clipsmm, see `http://clipsmm.sf.net`

Naturally, these steps are intertwined. While the robot explores the machine types, it already takes steps in the production chain, and needs to execute and monitor behaviors. Especially this entanglement of tasks calls for an incremental reasoning approach. As facts become known, the robot needs to adjust its plan. The simulation allows to perform offline tests evaluating the agent with a particular machine type assignment replacing real world input with simulated data. An additional requirement, due to the constrained platform, is the need for a computationally frugal approach.

Behavior Components

In previous work we have developed the Lua-based Behavior Engine (BE) (Niemueller, Ferrein, and Lakemeyer 2009). It mandates a separation of the behavior in three layers, as depicted in Figure 2(c), 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 interfacing layer.

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 jump conditions are evaluated, typically at about 15 Hz. If a jump condition fires, the active state is changed to the target node of the edge. A table of variables intrinsic to an HSM holds information like the world model, for example storing numeric values for object positions or strings describing its properties. It remedies typical problems of state machines like fast growing number of states or variable data passing from one state to another.

In systems like HERB (Srinivasa et al. 2012) we have previously used an HSM to serve as overall task state machine on top of the reactive middle layer. This works nicely if only execution parameters depend on the world situation, but the action sequence is (mostly) static and not to be re-planned depending on the perceived environment. However, in the LLSF domain the action sequence is variable and depends on employing sensing and reasoning to learn about the world. If the HSM mechanism is used in this situation it degrades. For example, a world model is typically built in the variable table, and rather than states representing the action flow, a large number of transitions among the states capture the incompleteness and make the HSM hard to follow.

Therefore, coming back to the layer separation, we decided to have an incremental reasoning agent program on the top-most layer. It relies on basic actions provided by the BE. To simplify the rules for exploration and production, most of the error handling capabilities are provided by the BE, and only critical failures are propagated.

Incremental Reasoning Agent

The problem at hand with its intertwined exploration, world model updating and execution and production phases naturally lends itself to a representation as a fact base with update rules for the exploration phase, and triggering behavior for certain beliefs. We have chosen the CLIPS rules engine,

```
(defrule s0-t23-s1
  (state IDLE) (holding S0)
  (machine (mtype ?mt&T2_3) (name ?n)
           (loaded-with $?l&:(contains$ S1 ?l)) )
  ?g <- (goto (machines $?ms&~:(contains$ ?n ?ms))
              (min-prio ?mp&:(<= ?mp (m-prio ?mt))))
  =>
  (modify ?g (machines (merge ?mp (m-prio ?mt) ?ms ?n))
             (min-prio (m-prio ?mt)))
)
```

Figure 3: CLIPS Production Process Rule

because using incremental reasoning the robot can take the next best action at any point in time, that is whenever the robot is idle, without costly re-planning (as with approaches using classical planners) and it allows us to cope with incomplete knowledge about the world, required in the LLSF scenario. Additionally, it is computationally inexpensive.

The CLIPS rules are roughly structured using a fact named *state* whose value 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 an action already started usually incurs much higher costs.

The rules explained in the following demonstrate what we mean by *incremental reasoning*. The fact base is updated as the robot gains more knowledge or commits to certain actions. This can also be triggered by information about the world published by other robots. The robot does not create a full-fledged 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 four areas: *world modeling*, *production process execution*, *simulation*, and *utilities*.

In Figure 3 we show a rule handling the *production process*. The robot is currently idle and just got raw material from the input storage: `(state IDLE)(holding S0)`. For this example, we assume to only know a T1 machine and another one that could be either of type T2 or T3, which is denoted by `(mtype ?mt&T2_3)`. This knowledge was acquired earlier bringing an S_1 puck to the machine, after which it signaled with an orange light that production is still in progress. In this situation, as the rule suggests, it is best to take the S_0 puck to this machine. Afterwards, the type of the machine will have been determined. The rule matches a `goto` fact which holds a list of potential targets to move to in the `machines` slot. The additional condition in this rule, `(min-prio ?mp&:(<= ?mp (m-prio ?mt)))`, makes sure that only a higher or same priority target compared to the current best target is considered. With the following action the rule updates the potential targets and updates the new minimum priority:

```
(modify ?g (machines (merge ?mp (m-prio ?mt) ?ms ?n))
           (min-prio (m-prio ?mt)))
```

Machine priorities are ordered by the type of the machine, e.g., a T2 machine has a higher value than a T1 machine. This is to prefer the completion of higher valued sub-goals. For example, if the robot was holding an S_0 puck, and it knew a T1 machine, and a T2 which was already loaded with an S_1 puck, it makes sense to prefer the T2 machine, because it can complete a production step to produce an S_2 puck, which scores more points in the competition than another S_1. The priority is also required to avoid getting stuck in local minima, e.g., producing lots of S_1 pucks but not completing the higher value goals.

The production process rules guide the robot to commit to the highest value action that can be taken, similar to a reward function. In our environment this has two particular benefits. First, aborting an action is expensive on the existing robot. The computational bounds and low-frequent control loops prohibit high motion speeds. Second the low memory and computational requirements make it suitable for the limited platform. The incurred overhead is virtually negligible.

The *world model* holds facts about the (partly) known or unknown machine types, what kind of puck the robot is currently holding (if any) and the motion state of the robot. Two examples for world model updates are shown in Figure 4. The rules are invoked after the action from the production rule presented above was successfully completed, i.e., an S_0 puck was taken to a machine of a yet undetermined type T2 or T3. The first rule shows the inference of the output puck type given a machine's reaction, the second handles a world model update based on this new.

In the first rule, the conditions

```
(state GOTO-FINAL)  (goto-target ?name)
```

denote that the robot finished going to a machine. Its name is bound to the variable `?name`.

The type of this machine, as known from the world model by matching a machine fact with the same name, was not yet determined as explained above. There was a single puck in this machine's area, matched by the following pattern. First, the list of loaded pucks is assigned to the list `$?w`, then it is constrained to have a length of one by the condition

```
(loaded-with $?w&:(= (length$ ?w) 1))
```

Further, the light turned green. This means that the production cycle has been completed and the robot now *knows* to be holding an S_2 puck. We retract the `light` fact and update the `holding` fact (by retracting and asserting it).

The second rule shows the inference of a machine type in that situation. A world model evaluation is triggered after a transportation step has been completed. Like before, the robot was at a machine of type T2 or T3. It held an S_0 or S_1 puck when it got there, and afterwards an S_2 puck. The robot can now be certain that the machine is of type T2 (in accordance with the known production tree as shown in Figure 1(b)) and it can update its belief. The following action resets the loaded pucks and increases the junk count by the number of puck in the machine area, fixes the type to T2, and increases the production count.

```
(modify ?m (mtype T2) (loaded-with)
  (junk (+ ?junk (length$ ?lw))) (productions (+ ?p 1)))
```

```
(defrule wm-holding-t23-one-green-s2
  (declare (salience ?*PRIORITY_WM*))
  (state GOTO-FINAL)  (goto-target ?name)
  ?h <- (holding ?any)
  (machine (name ?name) (mtype T2_3)
           (loaded-with $?lw&:(= (length$ ?lw) 1)))
  ?l <- (light green)
  =>
  (retract ?l ?h)
  (assert (holding S2))
)

(defrule wm-determine-t23-s0-or-s1-now-s2
  (declare (salience ?*PRIORITY_WM*))
  ?w <- (wm-eval (machine ?name) (junk ?junk)
                 (was-holding S0|S1) (now-holding S2))
  ?m <- (machine (name ?name) (mtype T2_3)
                 (loaded-with $?lw) (productions ?p))
  =>
  (retract ?w)
  (modify ?m (mtype T2) (loaded-with)
             (junk (+ ?junk (length$ ?lw)))
             (productions (+ ?p 1)))
)
```

Figure 4: CLIPS World Model Update Rules

The world model and the robot's beliefs can also be communicated from and to other robots, in particular regarding which pucks a machine is currently loaded with.

System Integration

The *overall system* consists of an initial fact base containing about two dozen facts, for example holding information about the machines (we know that there will be 10 machines on the playing field, but we do not know their type assignments). The rule base comprises a total of 74 rules, 38 are used for processing and publishing world model updates, 24 are concerned with the production process, 7 serve for the simulation and 5 for house keeping. So we see that the system requires only a small number of rules to maintain a world model and exhibit the behavior for the logistics scenario in 2012. It already handles communication and will extend to a more dynamic production scenario.

The *simulation* is used to perform spot tests for the agent program. When setting up a robot system for a new task, many software components on all levels need to be developed and integrated at the same time. The more a component can be tested independently of the others, the easier its integration into the full system typically gets. The agent simulation operates by disconnecting the agent from the actual robot system. It creates ground-truth data — either manually defined or randomized — for a particular scenario, i.e., a machine type assignment. Then, all actions that the robot executes, like fetching a puck or moving with it to a machine, are assumed to be successful. Then, perception input like a signal light response is generated based on the input and ground-truth information. Hence, the game can be played rapidly and often. The excellent tracing features of rule activation allow to verify the sequence of actions, detect planning dead ends, and optimize the sequence of actions. This makes simulation a valuable tool for debugging. Also, if an error is encountered, particular scenarios can be replayed. Figure 2(b) shows how the perception input can be provided either by the simulation, or by real world perception. The interface of the agent towards the input remains the same for either simulation or perception.

Actions do fail sooner or later when operating on the real robot, of course. For example, the puck might be lost when taking it to another machine. The *error handling* is done on the reactive layer as much as possible, e.g., the puck is constantly observed during transport and if lost a behavior is triggered to recover it. If at some point the reactive layer must conclude that it cannot recover the agent is notified. For example, recovering a puck could endanger another puck in a machine area. The CLIPS agent in that case aborts the action, updates its belief about the world, e.g., that it is no longer holding a puck, and goes to the idle state from where it can commit again to the next best action.

5 Discussion

In this paper we presented our CLIPS-based approach to a supply chain optimization (SCO) problem in the LLSF. Rules encoding certain situations guide the robot for the next best action to take. This can be exploration by taking pucks to yet unknown machines, sensing its reactions and reasoning about the actual type of the machine, or to continue in the production process. This incremental planning is required to cope with the incomplete knowledge in the domain, the result of an initially unknown machine type assignment. Priorities prefer the completion of higher valued goals — in the sense of winning points in the game. The system requires only a small number of rules to complete the task and has been implemented on a computationally limited platform, showing the feasibility and efficiency of the approach. We have experienced the LLSF as an interesting testbed and robotic competition for benchmarking our high-level system for the robotic SCO problem.

For our future work, we plan to describe the LLSF domain in the Planning Domain Definition Language (PDDL) to gain more flexibility. The current rule-based implementation will serve as a baseline system for comparison. An example for using state-of-the-art PDDL planner for SCO problems is (Radzi, Fox, and Long 2007). There, the authors compare SGPlan, LPG-d and CRICKEY on a number of SCO problems with the result that none of the algorithms could handle all features of the domain. While the presented optimization problem is more complex than what we have to deal with in our domain, their results show that solving complex SCO problems is challenging and not easily done off-the-shelf. Using a PDDL description would also allow for trying out planners that combine task level planning with geometric planning such as (Cambon, Gravot, and Alami 2004; Kaelbling and Lozano-Perez 2011). Another option is to employ continual planning (Keller, Eyerich, and Nebel 2010), which allows for incremental reasoning in a more sophisticated way. Other areas to work on are multi-robot coordination, task optimization strategies, and handling of uncertainty. Additionally, the increasing complexity of the league in the future poses further challenges to be addressed. Our software has been released as Open Source software.[3]

Acknowledgments

We thank the Carologistics RoboCup Team for their tremendous effort to develop a system on which the presented work could base. This work was supported by the German National Science Foundation (DFG) with grants GL747/9-5 and LA747/18-1. We thank the anonymous reviewers.

References

Beck, D., and Niemueller, T. 2009. AllemaniACs 2009 Team Description. Technical report, KBSG, RWTH Aachen University.

Cambon, S.; Gravot, F.; and Alami, R. 2004. A robot task planer that merges symbolic and geometric reasoning. In *Proc. of the 16th Eu. Conf. on Artificial Intelligence (ECAI-04)*, 895–899.

Forgy, C. L. 1982. Rete: A fast algorithm for the many pattern/-many object pattern match problem. *Artificial Intelligence* 19(1).

Giarratano, J. C. 2007. *CLIPS Reference Manuals*. http://clipsrules.sf.net/OnlineDocs.html.

Kaelbling, L. P., and Lozano-Perez, T. 2011. Hierarchical planning in the now. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*.

Keller, T.; Eyerich, P.; and Nebel, B. 2010. Task Planning for an Autonomous Service Robot. In *33rd German Conference on Artificial Intelligence (KI 2010)*.

Kitano, H.; Asada, M.; Kuniyoshi, Y.; Noda, I.; and Osawa, E. 1997. RoboCup: The Robot World Cup Initiative. In *Proceedings of the 1st Int. Conf. on Autonomous Agents*.

Kucera, D. 2012. Amazon Acquires Kiva Systems in Second-Biggest Takeover. Available at http://bloom.bg/Gzo6GU.

Lee, E. 2008. Cyber Physical Systems: Design Challenges. In *11th IEEE Int. Symp. on Object Oriented Real-Time Distributed Computing (ISORC)*, 363–369.

Niemueller, T.; Ferrein, A.; Beck, D.; and Lakemeyer, G. 2010. Design Principles of the Component-Based Robot Software Framework Fawkes. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*.

Niemueller, T.; Ferrein, A.; and Lakemeyer, G. 2009. A Lua-based Behavior Engine for Controlling the Humanoid Robot Nao. In *RoboCup Symposium 2009*.

Quigley, M.; Conley, K.; Gerkey, B. P.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; and Ng, A. Y. 2009. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*.

Radzi, N. H. M.; Fox, M.; and Long, D. 2007. Planning in supply chain optimization problem. In *Proc. of the 26th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG-07)*.

Srinivasa, S. S.; Berenson, D.; Cakmak, M.; Collet, A.; Dogar, M. R.; Dragan, A. D.; Knepper, R. A.; Niemueller, T.; Strabala, K.; Vande Weghe, M.; and Ziegler, J. 2012. HERB 2.0: Lessons Learned From Developing a Mobile Manipulator for the Home. *Proceedings of the IEEE* 100(8).

The Economist. 2012. The third industrial revolution. Vol. 12(16). http://www.economist.com/node/21553017.

Wygant, R. M. 1989. CLIPS: A powerful development and delivery expert system tool. *Computers & Industrial Engineering* 17(1–4).

[3]Find code, documentation, and videos of the robot in action at http://www.fawkesrobotics.org/p/clips-agent