The Carologistics RoboCup Logistics Team 2013

Tim Niemueller[1], Daniel Ewert[2], Sebastian Reuter[2],
Alexander Ferrein[3], Sabina Jeschke[2], and Gerhard Lakemeyer[1]

[1] Knowledge-based Systems Group, RWTH Aachen University, Germany
[2] Institute Cluster IMA/ZLW & IfU, RWTH Aachen University, Germany
[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. We outline the hardware modifications and describe our software systems and describe our efforts towards a fully referee box.
The team members of the 2013 team are Andre Burghof, Daniel Ewert, Alexander Ferrein, Bahram Maleki-Fard, Victor Mataré, Tobias Neumann, Tim Niemueller, Florian Nolden, Sebastian Reuter, Johannes Rothe, Alexander von Wirth, Frederik Zwilling.

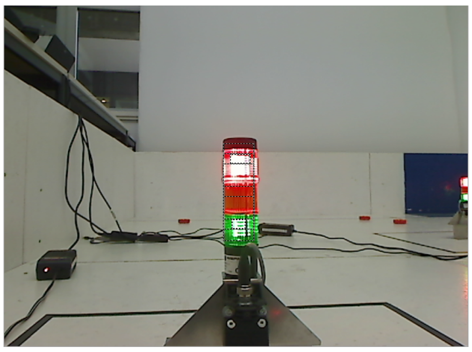
1 The Carologistics Robotino Robots

The basic platform 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 as shown in Figure 1, taken from the AllemaniACs' former middle-size league soccer robots [1], 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 Hokuyo URG laser scanner is used for collision avoidance and self-localization. In 2013, the robots will be outfitted with an additional laptop on the robot.

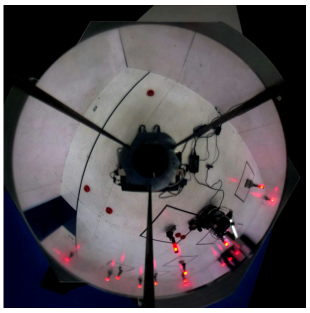
2 Middleware Concepts: Deploying Fawkes and ROS

The software system of the Carologistics robots combines two different middlewares, Fawkes [2] and ROS [3]. 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' navigation and 3D visualization capabilities. Most of the functional components are implemented in Fawkes. For example self-localization is done using Adaptive Monte Carlo Localization. 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. For computational and energy efficiency, the behavior components need to coordinate activation and

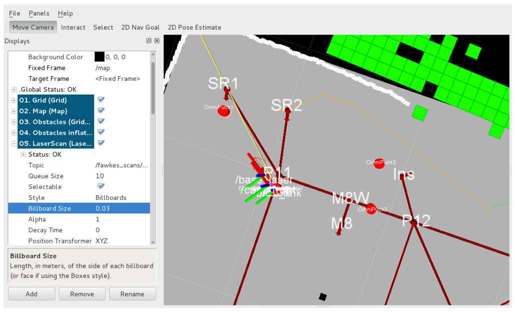
(a) Modified Robotino of the Carologistics RoboCup team.



(b) Image from the directed camera detecting the light signal of a machine.



(c) Image from the omni-directional camera.



Background Color	0, 0, 0
Fixed Frame	/map
Target Frame	<Fixed Frame>
.Global Status: OK	
01. Grid (Grid)	✓
02. Map (Map)	✓
03. Obstacles (Grid...	✓
04. Obstacles inflat...	✓
05. LaserScan (Lase...	✓
Status: OK	
Topic	/fawkes_scans/...
Queue Size	10
Selectable	✓
Style	Billboards
Billboard Size	0.03
Alpha	1
Decay Time	0
Position Transformer	XYZ

Billboard Size
Length, in meters, of the side of each billboard (or face if using the Boxes style).

(d) Visualization of the scene in Rviz

Fig. 1. Carologistics Robotino, sensor processing, and visualization

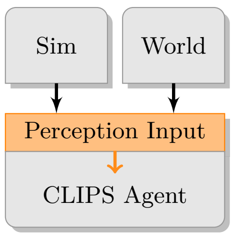
deactivation of the lower level components to solve computing resource conflicts. The behavior components are described in more detail in Section 3.2. Next, we briefly describe the task coordination components.

3 High-level Decision Making and Task Coordination

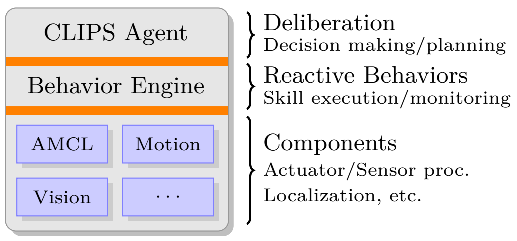
Task coordination is performed using an incremental reasoning approach [4]. In the following we introduce the rule-based production system CLIPS, describe the behavior components, and briefly describe the reasoning process in two particular situations from the rules in 2012.

3.1 CLIPS Rules Engine

CLIPS is a rule-based production system using forward chaining inference based on the Rete algorithm [5]. The CLIPS rule engine [6] has been developed and used since 1985 and is thus mature and stable. It was designed to integrate well



(a) External Interface



(b) Behavior Layer Separation

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

with the C programming language[4], which specifically helps to integrate with robot software like Fawkes or ROS. Its syntax is based on LISP.

CLIPS has three building blocks [7]: 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 specified name which can hold information. 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] And C++ using clipsmm, see `http://clipsmm.sf.net`

3.2 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 Lua-based 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.

In 2012, these steps were 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.

3.3 Behavior Components

In previous work we have developed the Lua-based Behavior Engine (BE) [8]. It mandates a separation of the behavior in three layers, as depicted in Figure 2(b), 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 the 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.

3.4 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, because using incremental reasoning the robot can take the next best action at any point in time

```
(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)))
)
```

Fig. 3. CLIPS Production Process Rule

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.

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 or changing an action usually incurs much higher costs.

The rules explained in the following demonstrate what we mean by *incremental reasoning*. In 2012, with an emphasis on incomplete knowledge, 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 for the *production process*. The robot is currently idle and got a 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 yellow 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 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. In the first rule, the conditions `(state GOTO-FINAL)(goto-target ?name)` denote that the robot finished locomotion and 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 and it can update its belief. The following action resets the loaded pucks, increases the junk count by the number of pucks in the machine area, sets the type to T2, and increments the production count.

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

The world model can also change due to information received from other robots, in particular regarding which pucks a machine is currently loaded with. Likewise, the own belief is published to other robots.

```
(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)))
)
```

Fig. 4. CLIPS World Model Update Rules

3.5 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.

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(a) 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.

4 LLSF Referee Box

The Carologistics team has developed an autonomous referee box (refbox) for the LLSF which will be deployed in 2013 [9]. It strives for full autonomy on the controlling side of the game, 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. This is in particular necessary because at the moment the refbox cannot detect if a puck was moved out of the machine area when a production was currently work in progress. In the future, we plan to achieve full autonomy of the refbox by integrating and adapting the overhead camera vision system of the Small Size League.

5 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. A great effort was made to re-use the agent principles to develop an autonomous referee box for the league.

References

1. Beck, D., Niemueller, T.: AllemaniACs 2009 Team Description. Technical report, Knowledge-based Systems Group, RWTH Aachen University (2009)
2. Niemueller, T., Ferrein, A., Beck, D., Lakemeyer, G.: Design Principles of the Component-Based Robot Software Framework Fawkes. In: Int. Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR). (2010)
3. Quigley, M., Conley, K., Gerkey, B.P., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: ROS: an open-source Robot Operating System. In: ICRA Workshop on Open Source Software. (2009)
4. Niemueller, T., Lakemeyer, G., Ferrein, A.: Incremental Task-level Reasoning in a Competitive Factory Automation Scenario. In: Proc. of AAAI Spring Symposium 2013 - Designing Intelligent Robots: Reintegrating AI. (2013)
5. Forgy, C.L.: Rete: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence **19**(1) (September 1982)
6. Wygant, R.M.: CLIPS: A powerful development and delivery expert system tool. Computers & Industrial Engineering **17**(1–4) (1989)
7. Giarratano, J.C.: CLIPS Reference Manuals. (2007) `http://clipsrules.sf.net/OnlineDocs.html`.
8. Niemueller, T., Ferrein, A., Lakemeyer, G.: A Lua-based Behavior Engine for Controlling the Humanoid Robot Nao. In: RoboCup Symposium 2009. (2009)
9. Niemueller, T., Ewert, D., Reuter, S., Ferrein, A., Jeschke, S., Lakemeyer, G.: RoboCup Logistics League Sponsored by Festo: A Competitive Factory Automation Benchmark. In: RoboCup Symposium 2013. (2013)