GOLEX — Bridging the Gap between Logic (GOLOG) and a Real Robot

Dirk Hähnel[†], Wolfram Burgard[†], and Gerhard Lakemeyer[‡]

[†] Universität Bonn, Institut für Informatik III, Römerstr. 164, D-53117 Bonn
{haehnel,wolfram}@cs.uni-bonn.de
[‡] RWTH Aachen, Lehrgebiet Informatik V, Ahornstr. 55, D-52056 Aachen
gerhard@cs.rwth-aachen.de

Abstract. The control of mobile robots acting autonomously in the real world is one of the long-term goals of the field of artificial intelligence. So far the field lacks methods bridging the gap between the sophisticated symbolic techniques to represent and reason about action and more and more reliable low-level robot control and navigation systems. In this paper we present GOLEX, an execution and monitoring system for the logic-based action language GOLOG and the complex and distributed RHINO control software which operates on RWI B21 and B14 mobile robots. GOLEX provides the following features: it maps abstract primitive actions into low-level commands of the robot control system, thus allowing the user to concentrate on the application rather than the inner workings of the robot; it monitors the execution of the primitive GOLOG actions, making it possible to detect simple execution failures and timeouts; and it includes means to deal with sensing and user input and to continue the operation appropriately. We present two different real-world applications in which GOLEX successfully operated a mobile robot in dynamic and even unstructured environments. These results suggest that the time is ripe for using symbolic action languages for mobile robot applications.

1 Introduction

To successfully perform their missions mobile robots must reliably fulfill and coordinate different kinds of subtasks like navigation, manipulation, perception, and interaction. Programming such robot applications using conventional programming languages is a very delicate and often frustrating enterprise, especially for people other than the designers of the robot. Partly as an answer to this dilemma the logic-based action language GOLOG [15] has been developed. It allows a user to specify at a very abstract level how primitive actions like *pickup(x)* change the world. Moreover, such actions can be combined into high-level programs using familiar control structures such as loops or recursive procedures. GOLOG then verifies for a given description of the world whether a program is executable and, if successful, delivers a sequence of primitive actions which can then be handed to the robot for immediate execution.

Or so the theory goes. The problem is, of course, that there is still a big gap to bridge between a primitive action like *pickup(x)* and the actual (and hopefully correct) movements of the robot arm. This paper is about bridging this gap. At the moment there is no accepted theory of how to map abstract (logical) descriptions of actions into actual robot behavior, and this paper does not attempt to provide one either. In fact, we believe that experiments are needed first so that we can begin to understand what this mapping is about. Also, there is still widespread skepticism whether logic has a role to play at all in robotics, since previous attempts starting with SHAKEY [18] never left the laboratory and worked only in toy domains. Our work, perhaps for the first time, demonstrates that logic can be used successfully to control a robot in real world domains.

There are many issues that need to be addressed when implementing a high-level control language such as GOLOG. Here we focus on the following three, which cover both inherent problems when moving from the abstract to the concrete and problems due to current limitations of GOLOG.

Level of Abstraction: Primitive actions in GOLOG generally cannot be directly executed by the low-level control system.[1] For example, the action *pickup(letter)* has to be transformed in several control directives such as stop moving, initialize the gripper, move the gripper to the position of the letter, and close the gripper.

Execution monitoring: In GOLOG it is assumed that actions succeed provided their preconditions are met. This, however, is generally not the case in robotics. As an example, consider the situation in which the robot wants to go to a specific room, whose door it assumes to be open. However, it turns out that the door is in fact locked. Without monitoring the progress of the robot, the system would easily get stuck in front of the closed doorway.

Sensing and Interaction: GOLOG in its current form does not provide any means for incorporation sensing actions or interaction. Interaction with users, however, is a crucial aspect of a mobile robot operating, for example, as an interactive tour-guide in a museum. On the other hand, an office delivery robot has to detect the number of letters to be picked up in order to verify that none of them was lost during delivery.

To tackle these problems we developed GOLEX, a runtime and execution monitoring system for GOLOG that

- decomposes the primitive actions specified in GOLOG into a sequence of appropriate directives for the low-level robot control system, thus providing the necessary mapping from the abstract task level to the low level of the robot control software.
- permanently monitors the execution of the various and possibly parallel actions of the underlying robot control system. In case of a failure it chooses appropriate actions to recover from this situation and updates the logical representation of the state so that the system can restart from this situation.
- integrates simple forms of sensing and interaction into GOLOG. It permits the programmer to formulate actions that include interaction and sensing. To deal with situations in which the sensing or interaction has side-effects on the whole

[1] Although nothing prevents a user from specifying very low-level primitive actions in GOLOG, this would defeat the whole purpose of the language.

plan and not only on the currently executed action, it provides means for deriving a new plan by calling GOLOG using the description of the current state as the initial situation.

GOLEX has been implemented on top of the existing control software of the Bonn mobile robot RHINO [3,21], which consists of a number of asynchronous processes, loosely coupled through message passing. This software is being used by over 20 groups worldwide. GOLEX has been extensively tested in different applications and proved to be extremely robust in a real-world scenario, in which RHINO was deployed over several days as an interactive and autonomous tour-guide in the "Deutsches Museum Bonn" [4].

As for related work, issues such as execution monitoring certainly come up and need to be solved in any implementation of a robot control language. Most of these languages such as COLBERT [13] remain well below the level of abstraction of GOLOG and hence are hard to compare. Using logic to control a real robot was first attempted in the SHAKEY project [18]. It never made it beyond toy domains mainly because the robot hardware and low-level control software were neither reliable nor sophisticated enough in those days. In fact, between then and now most of the work in robotics has concentrated on fixing those shortcomings and issues of high-level control have regained interest only recently. The work closest to our own is perhaps [12]. There PRODIGY [6], a non-linear planning system, is coupled with a mobile robot. While the authors address issues similar to ours such as monitoring and failure recovery, their execution system ROGUE is tightly coupled with PRODIGY and thus cannot be readily used for GOLOG. Also, as far as we know, ROGUE has only been tested in controlled office environments.

The remainder of this paper is organized as follows. After a very brief description of GOLOG in Section 2 we present the architecture of the RHINO system in Section 3. The GOLEX system is described in Section 4. Finally, Section 5 describes two applications of GOLOG, a tour-guide robot and an office delivery task.

2 GOLOG — A Language for Specifying Actions

Since GOLEX only sees the output of a GOLOG program, which is a linear sequence of primitive actions, we will only hint at what GOLOG is about and refer the reader to [15] for details. GOLOG is based on the situation calculus [17], which itself is a dialect of the predicate calculus with three sorts: ordinary objects, actions, and situations. Situations are action histories constructed from an initial situation S_0 and a special two-place function *do* where $do(a, s)$ denotes the successor situation to s resulting from performing the action a. What is true at a situation is described in terms of *fluents*, which are predicates whose last argument is a situation.

Following [16], a situation calculus theory AX consists of a number of foundational axioms, precondition axioms for the actions, a specification of what is true after doing an action a in situation s,[2] and a description of what the world is like initially.

[2] This employs a solution to the frame problem by Reiter [19].

Given such an AX, GOLOG allows a user to write high-level programs where the primitive actions are those defined in AX and where complex actions can be formed using control structures known from conventional imperative languages. Here is an example of a simple procedure specifying coffee delivery:

proc *deliverCoffee*
  while $(\exists p)$ *wantsCoffee(p)* $\wedge$ $\neg$*hasCoffee(p)* **do**
    (πp)*.goto(coffeeMachine); pickupCoffee; goto(p); giveCoffee(p)*
  endWhile
endProc

proc *goto(loc)*
  if *robotLocation(rloc)* $\wedge$ *rloc* $\neq$ *loc* **then** *drive(rloc, loc)* **endIf**
endProc

In this example, the operator πp non-deterministically chooses an action argument p for *goto(p)* that is a solution for the existentially quantified condition. The semantics of a GOLOG program is defined completely within the situation calculus. This is done using a special three-place macro *Do* which expands into a sentence of the situation calculus and where $Do(\rho, s, s')$ should be read as "the program ρ, when executed in situation s, leads to situation s'." The job of the GOLOG interpreter is then to find a sequence s' such that

$$\mathrm{AX} \models Do(\rho, S_0, s').$$

If $\rho =$ deliverCoffee, then s' could be

do(. . . , do(giveCoffee(ray), do(drive(coffeeMachine, ray), do(pickupCoffee, s0))). . .)

provided we specified in AX that Ray wants coffee and the robot starts at the coffee machine in situation S_0. It is s' which gets handed down to GOLEX for further processing.

3 The RHINO System Architecture

The RHINO system consists of more than 25 modules, each of which is designed to monitor or control dedicated aspects of the robot or its environment and to provide information to other modules. There is no centralized clock to synchronize the different components and the robot has to function even if major parts of its software fail or are temporarily unavailable (e.g., due to radio link failure).

The building blocks of this architecture are depicted in Figure 1. The on-board user interface consists of an on-board lap-top, a speech- and sound-board, and four colored buttons which allow users to interact with the robot. The task planner uses GOLOG to plan the activities of the robot and GOLEX to perform and monitor them. The navigation system or low-level control system of the robot consists of the collision avoidance routine [8], the map updating and path planning modules [21] and the localization system [5]. The task of the map building system is to update the information about the environment given new sensory input. The path planner computes cost-optimal paths from a starting position to a given target position given the current map of the environment. The localization module permanently estimates the position of the robot within its environment. Finally, the RHINO system contains various hardware control interfaces, for example, to a top-mounted pan-tilt-head carrying a stereo camera system or to the speech-board. All software modules communicate using TCX [7], a communication manager for point-to-point socket communication.

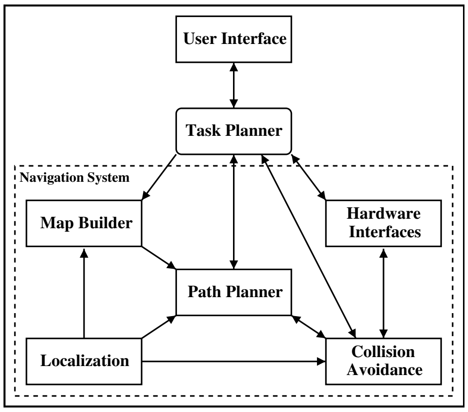


Figure 1: Major components of the RHINO control system

The RHINO navigation system has been proven robust and reliable in different applications [3,21]. The modularity provides a high flexibility with respect to new applications and supports the specialization of dedicated modules according to specific subtasks. The disadvantage of the decentralized architecture is that each module generally has to communicate with a variety of other modules. For example, the path planning system receives information from the localization system to estimate the current position of the robot, receives map updates from the map building module and transmits the next target point along the planned path to the collision avoidance system. Due to the asynchronous nature of the concurrently and independently running modules, the complexity of coordinating the various modules increases with the number of active modules. Moreover modules generally do not broadcast the successful completion of their tasks. For example, the collision avoidance simply stops the robot after reaching the target point obtained from the path planning module. Thus, the path planner has to monitor the progress of the robot and to transmit a new (intermediate) target points if the final destination has not yet been reached. Finally, modules may fail or abort and have to be re-initialized after restart. Thus, a high-level control system has to communicate with a series of different low-level control modules and has to permanently monitor the progress of the system to detect possible failures or the completion of the task.

4 GOLEX — The GOLOG Execution System

The GOLOG execution system GOLEX, which has been implemented in Prolog (together with a C-interface), is designed to support the control of a mobile robot equipped with the RHINO navigation system and to provide the programmer with the necessary level of abstraction on the Prolog level. During execution it handles user inputs and deals with possible timeouts of actions of the user or the robot. To summarize, the task of the GOLEX system is to

- refine primitive GOLOG actions into actions that can be carried out by the low-level controls system.
- permanently monitor the actions of the robot to verify that the primitive GOLOG actions have been successfully carried out, this way ensuring that the relevant state of the world is consistent with GOLOG's model.
- integrate simple forms of sensing as well as interaction with users.

How these different tasks are accomplished is described in more detail in the remainder of this section.

4.1 Refinement of Primitive Actions

Primitive actions in GOLOG have to be translated into a set of corresponding directives to the low-level control system accomplishing the desired task. For this purpose GOLEX includes a set of generic commands for possible actions carried out by the robot. These actions include primitives such as *robot_drive_path(L)* which successively transmits the list L of target points to the path planner so that the robot passes this sequence of points. Furthermore, it includes actions for turning the robot, for example, the predicate *robot_turn_to_point(P)* forces the robot to rotate so that it is heading towards P. It also contains commands for displaying images or texts on the on-board screen, to speak texts to play sounds, or to request user input. Finally, GOLEX provides primitives for receiving raw as well as abstract and preprocessed sensory input. For example, the primitive *free_space* is true, if there is no obstacle in front of the robot. This way, GOLEX allows to check, whether the path of the robot is blocked by a closed door.

Based on these primitives, the user can easily specify, how the primitive actions in GOLOG have to be translated into an appropriate sequence of navigation or display directives or user input requests. For example, in the tour-guide application the primitive *do(go(L), S)* is realized within GOLEX by the following Prolog clause:

```
exec(go(L)) :-
   position(L, (X, Y)),
   pan_tilt_set_track_point((X, Y)),
   target_message(L, M),
   speech_talk_text(["Please follow me to", L]),
   sound_play(horn),
   robot_drive_path([(X, Y)]),
   robot_turn_to_point((X, Y)).
```

4.2 Execution Monitoring

Because the other software components of the RHINO system generally do not report a failure, the GOLEX system has to constantly monitor the progress of the robot and to detect whether the robot has accomplished the particular action. For example, the primitive *robot_drive_path(L)* monitors, whether the robot has reached the target point. The path planning module generally regards a target position as reached if the robot is about 50cm away from it. To provide a higher accuracy at the final destination, the *robot_drive_path(L)* primitive directly communicates with the collision avoidance module to move the robot closer to the target and to achieve the desired heading. Furthermore, it re-transmits the necessary information to each connected module in case of transmission failures, resource conflicts between different modules or even in situations in which a module fails or is restarted again. For example, the path planner stops the robot whenever it arrives at the target position. If *robot_turn_to_point((X, Y))* starts to rotate the robot before this stop command is received by the collision avoidance after the rotate command, the robot also stops the final rotation and is facing the wrong direction. Thus, GOLEX monitors whether this situation occurs and re-transmits the rotation command to the collision avoidance.

Several commands have additional arguments for timeouts. For example, *button_get_input(B, T)* waits the time T until a button is pressed. If no button was pressed, it succeeds with B=0. Otherwise, B is bound to the number of the button that was pressed. GOLEX also supports timeouts during the execution of actions. For example the primitive *robot_go_path([P], T)* succeeds, if the robot reaches the given P before the end of the time interval T and fails otherwise. This way the system is able to consider time constraints while acting as a coffee delivery robot [20]:

```
exec_plan([startGo( Person, Place )| Other_actions], Done) :-
   wants_coffee(Person, Time_interval),
   ( robot_go_path([Place], Time_interval) ->
       execPlan(Other_actions, do(startGo(Person, Place), Done))
   ;
       write("Skipping "), write(Person), nl,
       retract(wants_coffee(Person, Time_interval)),
       ( do( deliver_coffee(Now), Done, New_actions) ->
            split(Done, New_actions, Remaining_actions),
            execPlan(Remaining_actions, Done)
       ;
            write("Aborting execution"),nl
       )
   ).
```

If GOLEX detects that the robot can no longer reach the destination before the end of T it drops the user requests and re-plans using the current state as initial state.

4.3 Interaction

By providing primitives for user input, GOLEX adds the ability to interact with the user. For example, the coffee delivery robot needs a confirmation that the user has already taken his coffee. If the user does not respond after a certain amount of time, the robot has to proceed with serving other people:

```
exec(serve_coffee_to(Person, Location)) :-
   robot_drive_path([Location]),
   robot_turn_to_point(Location),
   speech_talk_text(["Hi", Person,
            ". Please take your coffee and push a button"]),
   button_get_input(B, 10),
   ( B = 0 ->
       speech_talk_text(["Did you forget to push a button?"]),
       button_get_input(B1, 10),
       ( B1 = 0 ->
           speech_talk_text(["You seem to be very busy."])
         ;
           true
       )
     ;
       true
   ),
   speech_talk_text(["Thanks and good bye."]).
```

In both applications, GOLEX requires further information about the environment and the task. For example, in the tour-guide application it requires a map specifying the locations of the exhibits in the museum.

Furthermore, GOLEX supports two different types of user input that influences the currently executed plan of the robot. For example, in the museum tour-guide application visitors had the opportunity to request specific information about the currently visited exhibit, or even abort or change the current tour. The interaction with the users that want specific information about the current exhibit cannot be planned a priori in GOLOG, since it is unclear at the planning phase, which button will be pressed. The normal situation, however, is that the robot explains the exhibit to the visitors, reacts on the user input and after that continues with the currently planned sequence of exhibits. Thus, by providing a primitive action *explain(E)* to GOLOG, this type of interaction with the users can be achieved within GOLEX.

However, if the user requests to abort the tour or to be guided to another exhibit, GOLEX uses GOLOG to compute a new plan given the current state of the world as starting state. In such a situation it proceeds in a similar way as when a timeout occurs during a delivery job (see above).

5 Experimental Results

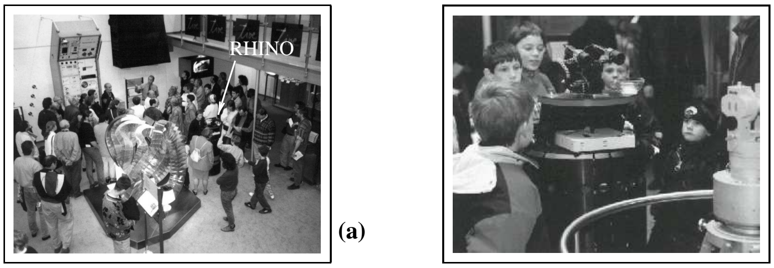
To evaluate the GOLEX system we performed two different experiments in different partially unstructured environments with the mobile robots RHINO (Figure 2a) from the University of Bonn and GOLEM from the University of Toronto (Figure 2b).

5.1 The Museum Tour-Guide Application

The first experiment has been carried out during the six day deployment of the mobile robot RHINO as an interactive personal tour-guide in the "Deutsches Museum Bonn", Germany [4]. The task of the robot was to guide visitors through the exhibition and to provide explanations about the exhibits (see Figure 3). During the 47 hours of operation, the actions of the robot were planned using GOLOG and the generated plans were executed by GOLEX.


(a) (b)

Figure 2: The mobile robots RHINO (a) and GOLEM (b) used for the experiments



(a) (b)

Figure 3: RHINO giving a tour to visitors (a) and explaining an exhibit (b).

Over this period of time, the robot travelled approximately 18.6km. More than 2,000 real visitors and over 600 "virtual" Web-based visitors were guided by RHINO (see Figure 3). The robot fulfilled 2,400 tour requests by real and virtual visitors of the museum. Only six requests were not fulfilled which corresponds to an overall success-rate of 99.75%. None of these failures were caused by the high-level planning system or by GOLEX. The key results are also summarized in Table 1. A typical path of the robot as well as the user interface are depicted in Figures 4 and 5.

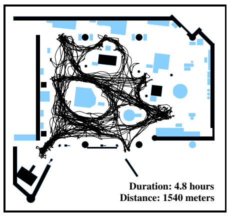


Figure 4: Typical path of the robot.

Figure 5: RHINO's user interface.

Hours of operation	47
Number of visitors	> 2000
Kilometers traveled	18.6
Maximal speed of travel	> 80
Average speed during motion	36.6
Number of requests	2400
Number of failures	6
Success rate	99.75%

Table 1: Some key results.

5.2 The Coffee Delivery Application

The second experiment is designed to demonstrate the applicability of GOLEX in a typical office delivery scenario. In this application the task of the robot GOLEM was to serve people with coffee [20]. Thereby it had to deal with time constraints given

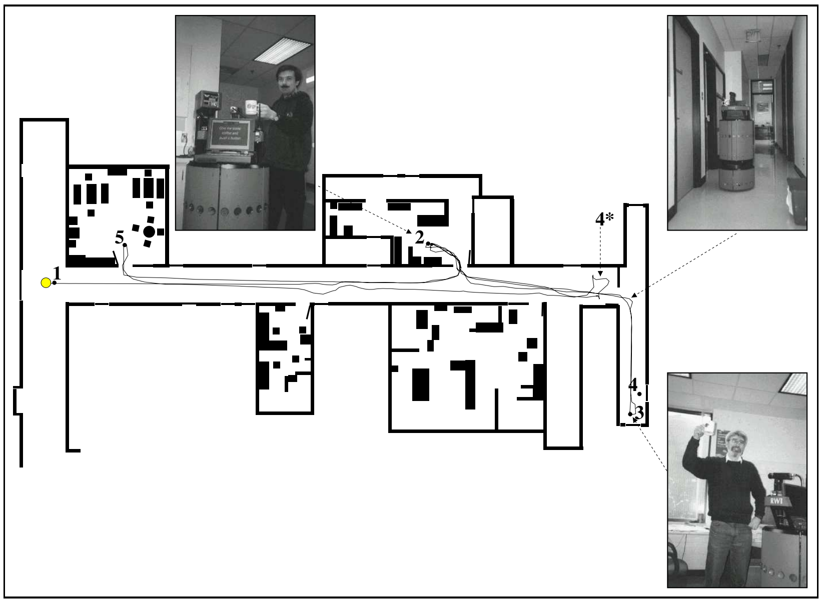


Figure 6: Path of the robot during the coffee delivery

by the time interval when the people had to be served. Whenever GOLEX detects, that the next customer cannot be served within the given amount of time, it skips this sub-plan from the current schedule and re-plans after removing this person from the list of persons to be served.

Figure 6 shows a typical run of GOLEM while serving three different people at the University of Toronto. It starts at position 1 and goes to the coffee machine at position 2. There GOLEM request for coffee which is uploaded by Mikhail and afterwards proceeds to Ray's office at position 3. After Ray took the coffee from the robot, GOLEM returns to the coffee machine to upload a further cup of coffee which has to be brought to Hector whose office is at position 4. On the way to Hector, the path of the robot is blocked by a crowd of people (position 4*). At the moment when the due time for serving Hector is expired, GOLEX retracts Hector's request and then computes a new plan using GOLOG. According to this plan, the visitor at position 5 is served next. After completing this task, the robot returns to position 2 and stops its delivery job.

6 Conclusions

In this paper we presented GOLEX, a runtime and execution monitoring system for GOLOG, that demonstrates, how mobile robot applications specified in a logical action language can be realized on top of existing low-level robot control software like that of the RWI B14 and B21 mobile robots [21]. GOLEX maps abstract primitive actions into low-level commands of the robot control system, thus allowing the user to concentrate on the application rather than the inner workings of the robot. Furthermore, it monitors the execution of the primitive GOLOG actions thus allowing to detect possible execution failures or timeouts. Finally, it incorporates user input and sensing and is able to continue the operation appropriately in such situations.

GOLEX has been implemented and tested extensively in different applications. In a long-term experiment GOLEX served as the execution monitoring system during a 47 hours lasting deployment of the mobile robot RHINO as an interactive tour-guide in the "Deutsches Museum Bonn". The robot fulfilled 99.75% of the user requests and none of the failures were caused by the high-level control system. It furthermore was applied in a typical office-delivery application where it controlled the mobile robot GOLEM to deliver coffee.

Although GOLEX has been proven to allow a robust execution of plans in these real world applications, there are many opportunities for future research:

- While there is a parallel version of GOLOG [9], GOLEX currently does not support parallel actions. For example, GOLEX is not able to react on user input while the robot travels to its destination. Supporting parallelism in general would require a resource management system since certain resources such as motors cannot be shared simultaneously.
- Specifying how to react to the outcome of a sensing action really should be handled at the abstract level of GOLOG. In fact, very recently there have been proposals to extend GOLOG with sensing actions [10,14]. Work is under way to adapt GOLEX to use these more expressive action languages.
- Similarly, it seems desirable to leave at least some part of execution monitoring under high-level user control. First results on how to specify such monitors in GOLOG have recently appeared [11], but they are still at a preliminary stage.
- Finally, when moving to more complex applications than the ones we reported on, the current techniques in GOLEX to deal with execution failures will probably not suffice. For example, rather than simply generating a new plan from scratch after a failure, plan transformation techniques as in [1] may be preferable.

While GOLEX was designed for the purpose of linking GOLOG to a real robot, GOLEX really only gets to see the output of GOLOG, that is, a sequence of primitive actions. Hence we expect it to be relatively easy to use GOLEX in conjunction with other action or planning languages such as TL [2] and RPL [1].

Finally, we strongly believe that both robotics and symbolic AI can benefit from each other. GOLEX is but one step in bringing the two fields closer together.

References

1. M. Beetz and D. McDermott. Improving robot plans during their execution. In Kris Hammond, editor, *Second International Conference on AI Planning Systems*, pages 3–12, Morgan Kaufmann, 1994.
2. W. Bibel. Let's plan it deductively. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, volume 2, pages 1549–1562, August 1997.

3. J. Buhmann, W. Burgard, A.B. Cremers, D. Fox, T. Hofmann, F. Schneider, J. Strikos, and S. Thrun. The mobile robot RHINO. *AI Magazine*, 16(2):31–38, Summer 1995.
4. W. Burgard, Armin B. Cremers, D. Fox, D. Hähnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. The interactive museum tour-guide robot. In *Proc. of the Sixteenth National Conference on Artificial Intelligence (AAAI-98)*, 1998. To appear.
5. W. Burgard, D. Fox, D. Hennig, and T. Schmidt. Estimating the absolute position of a mobile robot using position probability grids. In *Proc. of the Fourteenth National Conference on Artificial Intelligence*, pages 896–901, 1996.
6. J. G. Carbonell, C. A. Knoblock, and S. Minton. Prodigy: An integrated architecture for planning and learning. In K. Van Lehn, editor, *Architectures for Intelligence*. Erlbaum, 1990.
7. C. Fedor. *TCX. An interprocess communication system for building robotic architectures. Programmer's guide to version 10.xx.* Carnegie Mellon University, Pittsburgh, PA 15213, 12 1993.
8. D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, 1997.
9. G. de Giacamo, Y. Lespérance, and H.J. Levesque. Reasoning about concurrent execution, prioritized interrupts, and exogenous actions in the situation calculus. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, volume 2, pages 1221–1226, 1997.
10. G. de Giacomo and H. J. Levesque. An incremental interpreter for high-level programs with sensing. Technical report, University of Toronto, 1998.
11. G. de Giacomo, R. Reiter, and M. M. Soutchanski. Execution monitoring of high-level robot programs. In *Proceedings of the Sixth International Conference on Knowledge Representation (KR 98)*, 1998. To appear.
12. K. Z. Haigh and M. M. Veloso. High-level planning and low-level execution: towards a complete robotic agent. In *Proceedings of the First International Conference on Autonomous Agents*, Marina del Rey, CA, February 1997.
13. K. Konolige. Colbert: A language for reactive control in saphira. In *KI-97: Advances in Artificial Intelligence*, pages 31–52. LNAI Series, Springer Verlag, 1997.
14. G. Lakemeyer. On sensing and off-line interpreting in golog. Technical report, RWTH Aachen, 1998.
15. H.J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl. GOLOG: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31:59–84, 1997.
16. F. Lin and R. Reiter. State constraints revisited. *Journal of Logic and Computation, special issue on actions and processes*, 4:665–678, 1994.
17. J. McCarthy. Situations, actions and causal laws. In *Semantic Information Processing*, pages 410–417. MIT Press, 1968.
18. Nils J. Nilsson. SHAKEY the robot. Technical report, SRI International, 1984.
19. R. Reiter. The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression. In V. Lifshitz, editor, *Artificial Intelligence and Mathematical Theory of Computation*, pages 359–380. Academic Press, 1991.
20. R. Reiter. Sequential, temporal golog. In *Proceedings of the Sixth International Conference on Knowledge Representation (KR 98)*, 1998. To appear.
21. S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schimdt. Map learning and high-speed navigation in RHINO. In D. Kortenkamp, R.P. Bonasso, and R. Murphy, editors, *AI-based Mobile Robots: Case studies of successful robot systems*. MIT Press, Cambridge, MA, 1998.