Integrated, Plan-Based Control of Autonomous Robots in Human Environments

Michael Beetz, *Munich University of Technology*
Tom Arbuckle, Thorsten Belker, Armin B. Cremers, Dirk Schulz, *University of Bonn*
Maren Bennewitz, Wolfram Burgard, Dirk Hähnel, *University of Freiburg*
Dieter Fox, *University of Washington*
Henrik Grosskreutz, *Aachen University of Technology*

The authors extend the Rhino robot by adding the means for plan-based high-level control and plan transformation, further enhancing its probabilistic reasoning capabilities. The result: an autonomous robot capable of accomplishing prolonged, complex, and dynamically changing tasks in the real world.

Autonomous robots have operated reliably and safely in unstructured environments: they have been museum tour guides,[1] performed simple delivery tasks,[2] and explored dangerous environments.[3] These tasks were carefully chosen: they made great demands on the robots´ navigation skills and on safe and reliable operation in unstructured, dynamic environments but their structure was kept deliberately simple. Creating an autonomous robot capable of accomplishing prolonged, complex, and dynamically changing tasks in the real world is a key challenge for the next generation of autonomous robots.

To perform complex activities effectively, robots must possess rich perceptual capabilities to recognize objects and places and interactively communicate with people. Robots must also reason about and manage concurrent tasks and subtasks while executing them. Moreover, for prolonged success, robots must improve their control routines based on experiences to adapt to environment and task changes.

Our research investigates the computational principles enabling autonomous-robot control systems to accomplish complex and diverse tasks in the real world. Our work focuses on developing advanced perceptual, learning, and adaptation capabilities as well as planning mechanisms for robotic agents. This article describes how we added these capabilities to Rhino, an RWI B21 mobile robot (see Figure 1).

Design principles

We developed a unique control system for Rhino. The sidebar "Rhino at Work as an Office Courier" illustrates how we implemented the following key design principles in a specific application.

Dynamic-system perspective

Robots must be flexible and responsive to changing situations. Therefore, we used dynamic systems as the primary abstract model for programming the integrated, plan-based controller (see Figure 2). In this model, the state of the world evolves through the interaction of two processes: the controlling process (the robot's control system) and the controlled process (events in the environment, the robot's physical movements, and sensing operations). For complex dynamic systems, we further decompose the controlled process into an environment process, which changes the world state, and a sensing process, which maps the world state into the sensor data the robot receives. We similarly decompose the controlling process into state estimation and action generation processes. The state estimation process computes the robot's beliefs about the controlled system's state. An action generation process specifies the control signals supplied to the controlled process as a response to the estimated system state. Auxiliary monitoring processes signal system states that the controlling process is waiting for.

The main consequence of this model is that robots

1094-7167/01/$10.00 © 2001 IEEE

Figure 1. Rhino, an RWI B21 mobile robot, has a unique control system enabling successful operation in increasingly complex environments.

must control concurrent and continuous processes both flexibly and reliably. Rhino does this through control routines that specify and synchronize concurrent percept-driven behavior. We implemented the routines using the *reactive plan language (RPL),*[4] a control language providing traditional control abstractions and high-level constructs (interrupts, monitors) that synchronize actions. RPL also allows specification of interactive behavior, concurrent control processes, failure recovery methods, and temporal coordination of processes. We use these features to make plans reactive and robust by incorporating sensing and monitoring actions as well as reactions triggered by observed events.

Plan-based high-level control

To be reliable and efficient, our robot courier must flexibly interleave tasks, exploit opportunities, quickly plan a course of action, and, if necessary, revise its intended activities. Recomputing the best possible course of action whenever a situational detail changes is typically not feasible, but can often be made feasible by explicitly managing the robot's respective beliefs, goals, and intended plans.

In Rhino, plans are symbolic representations of future activity and have two roles: *executable prescriptions* the robot interprets to accomplish its jobs, and *syntactic objects* the robot synthesizes and revises to meet its specific success criteria. RPL has tools that support plan formation and adaptation. This enables planning processes to

- project what might happen when a robot controller executes a plan,
- return the result as an execution scenario,[5]
- infer what might be wrong with a robot controller given an execution scenario,[6] and
- perform complex revisions on robot controllers.6

Probabilistic state estimation

The plan-based, high-level control of the robot operates from abstract perceptions of the current state of objects, the robot, and the environment. To derive robust abstract perceptions from its local and inaccurate sensors, the Rhino system uses probabilistic state estimation techniques.[7] The state estimators maintain the probability densities for the states of objects over time. Whenever the planning component requests state information, the state estimators provide the object's most likely state. An object's probability density at that moment contains all the information available. Based on this density, we can determine the object's state and derive even more meaningful statistics (for example, variance and entropy of the current estimate). In this way, the high-level system can reason about an estimate's reliability.

Plan transformation

Because the robot is uncertain about the state of its dynamic environment, it must also quickly form and adapt its plans. Plan transformation enables Rhino to adapt during execution and to learn canned plans based on experience.[8]

A *plan library* provides a collection of canned plans for achieving standard tasks in standard situations. However, canned plans do not always execute optimally. If an unexpected opportunity presents itself while the robot is executing its tasks, for example, a canned plan will have trouble testing for subtle consequences. The decision criteria to

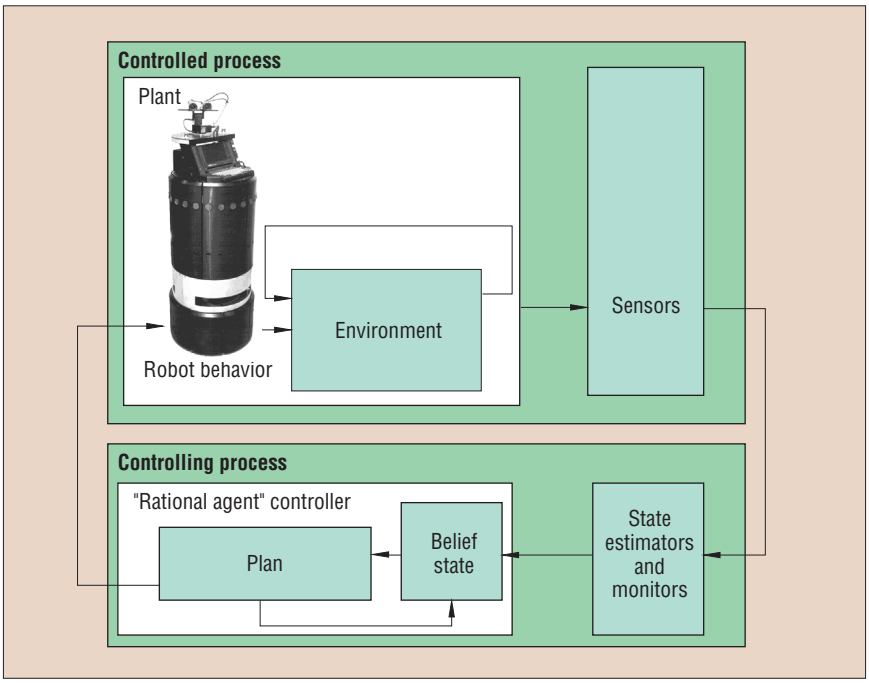

Figure 2. Block diagram of our dynamic system model for autonomous robot control. Boxes depict processes; arrows depict interactions.

Rhino at Work as an Office Courier

The key to Rhino's self-adapting, plan-based control is its *structured reactive controller.*[3] The SRC manages activity and enables the robot to reliably and efficiently carry out schedules of multiple jobs. It consists of a collection of concurrent control routines that specify routine activities and adapt to nonstandard situations. The SRC employs three kinds of control processes:

- routine processes handle standard jobs in standard situations,
- monitoring processes detect nonstandard situations, and
- planning processes adapt standard routines to nonstandard situations.

Consider a problem-solving episode: Rhino, working as an autonomous office courier, receives two commands (see Figure A):

1. "Put the red letter on the meeting table in room A-111 on Michael's desk in A-120."
2. "Deliver the green book from the librarian's desk in room A-110 to the desk in A-113."

To accomplish its commands, Rhino uses a library of routine plans. The routine delivery plan specifies that Rhino is to navigate to the pickup point, ask somebody to load the letter, wait until it is loaded, navigate to the destination, ask somebody to unload the letter, and wait for it to be unloaded.

Whenever Rhino is to carry out a set of delivery jobs, it quickly computes an appropriate schedule. Rhino's initial schedule is to deliver the red letter first and the green book afterwards. Proposing a schedule implies making assumptions about whether doors are open and closed. In our experiment, Rhino assumes that the rooms A-110, A-111, A-113, and A-120 are open. To adapt its schedule flexibly, Rhino monitors the scheduling assumptions while performing its deliveries: whenever it passes a door, it estimates the opening angle of that door and revises the schedule if necessary.

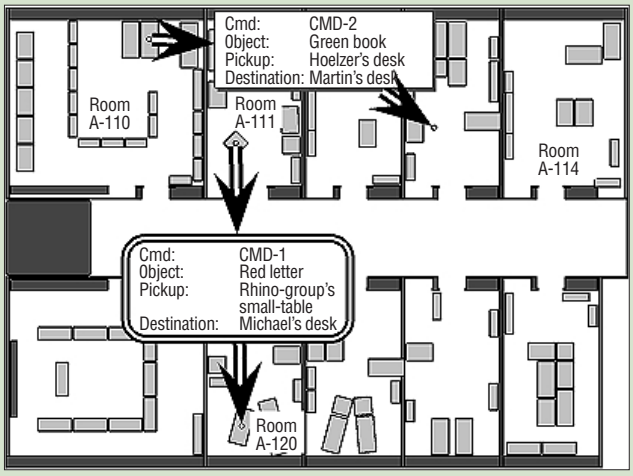

Figure A. Rhino's operating environment-a small hallway with adjacent offices, a library, and a classroom-and two delivery requests.

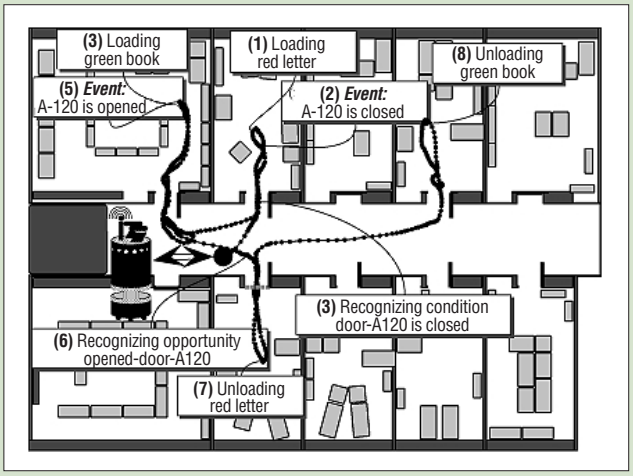

Figure B. Complete trajectory and event trace for the two deliveries if A-120 is closed while the robot is in A-111 and opened again while it is in A-110.

Here's how the SRC produces the event trace depicted in Figure B. Initially, all doors in the environment are open. Rhino starts with the delivery of the red letter and heads to the meeting table in A-111, where someone loads the letter (step 1). Then someone closes A-120's door (step 2). Thus, when Rhino enters the hallway to deliver the red letter to Michael's desk, it estimates the opening angle of A-120's door and detects that the door has been closed (step 3). The delivery plan signals a failure that informs the SRC that the delivery cannot be completed at this time.

Because Rhino does not know when room A-120 will be open again, it revises the schedule so that it now delivers the green book first and suspends the delivery that could not be completed until the office is open again. Thus, Rhino navigates to the librarian's desk in A-110 to pick up the green book and deliver it to room A-113 (step 4). At this moment, room A-120 is opened again (step 5). As Rhino heads towards A-113 to deliver the green book, it passes room A-120 (step 6). At this point, the door estimation process signals an opportunity: A-120 is open. Therefore, Rhino interrupts its current delivery to complete the delivery of the red letter. After delivering the red letter (step 7), Rhino delivers the green book (step 8). Figure C shows the behavior generated by the same SRC (C1) if all doors stay open and (C2) if A-120 is closed but not opened again.

The self-adapting plan consists of the primary activities that specify the course of action for satisfying the user requests-the delivery tour. The plan also contains two plan adaptors. The first one makes sure that the intended course of action contains delivery tasks for every actual user request and no unnecessary delivery tasks. Thus, whenever the systems adds,

take or ignore such opportunities must typically be hardwired into the canned plans when the plan library is built.

An alternative is to equip a robot with self-adapting plans. Whenever a specific belief of the robot changes, self-adapting plans trigger a runtime plan adaptation process. When triggered, the adaptors decide, possibly based on predictions, whether plan revisions are necessary and, if so, perform them. Plan adaptation processes are specified explicitly, modularly, and transparently and are implemented using declarative plan transformation rules.

Context- and resource-adaptive operation

To make its control decisions in a timely manner, the plan-based controller applies various resource-adaptive inference methods. Sampling-based inference methods perform probabilistic state estimation, for example.[7] These enable the controller to trade off accuracy and the risk of making wrong decisions against the computational resources consumed to make those decisions. Moreover, the results of resource-adaptive reasoning enable execution modes to change

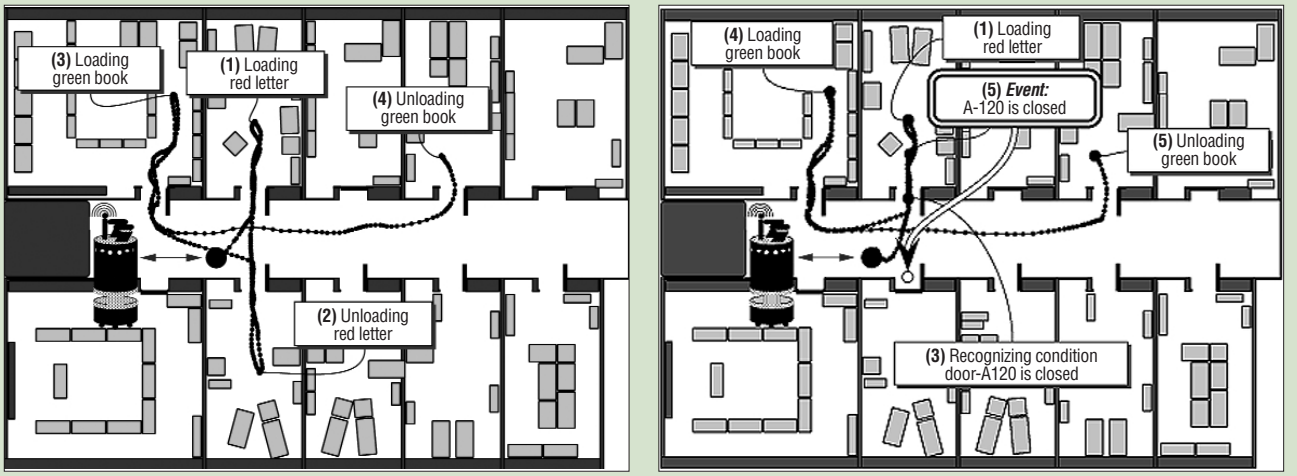

Figure C. Complete trajectory and event trace for the two deliveries if (a) all doors are open, and (b) if A-120 is closed and not opened again.

deletes, or revises a user request, the plan adaptor performs the necessary changes on the intended primary activities. The second plan adaptor orders the collection and delivery steps of the tour. This eliminates any unwanted interactions between steps and ensures quick completion of the tour.

During the course of the episode, the SRC modifies itself. Rhino initially performs no primary activities. Its plan adaptors ensure that the robot receives and processes new commands. When it receives the two jobs (event "start" in Figure B), the plan adaptor P-1 puts plans for accomplishing them into the primary activities. Inserting the commands triggers plan adaptor P-2's scheduler. The plan adaptor for rescheduling also adds an additional plan adaptor P-3 that monitors the assumptions underlying the schedule-that is, that the rooms A-110, A-111, A-113, and A-120 are open. Figure D shows the SRC after this revision.

After Rhino picks up the red letter and leaves room A-111, it notices that room A-120 has closed in the meantime (event 3 in Figure B). Because Rhino cannot finish delivering the red letter, the corresponding command fails, signaling an incompleteable delivery. This failure triggers the replanning plan adaptor P-2, which transforms the completion of the delivery into a subplan that is suspended until the office is open again (see Figure E). Thus, as soon as Rhino notices that room A-120 is open (event 6), it interrupts its current mission, finishes delivering the red letter (event 7), and continues with the remaining jobs.

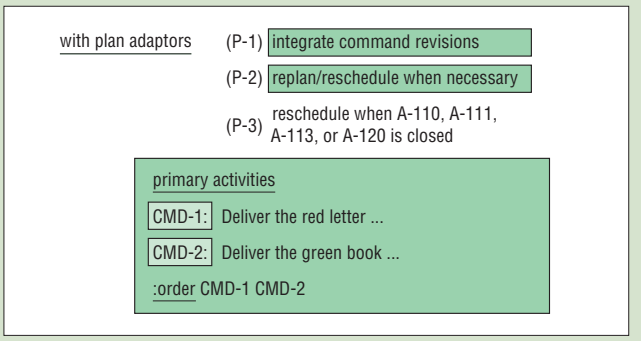

Figure D. The top-level structure of the SRC after inserting the plans for the two delivery requests. The SRC also contains a plan adaptor that triggers a rescheduling process when a door closes.

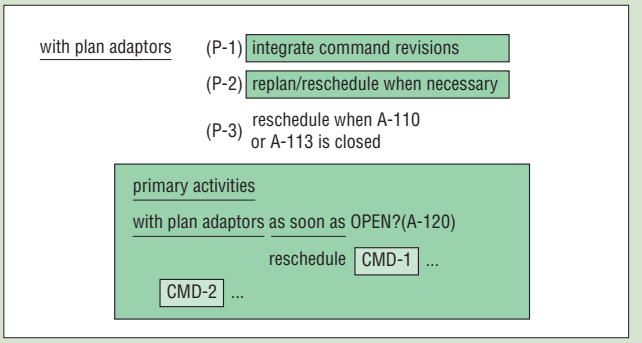

Figure E. Structure of the SRC after it detects A-120 is closed. Rhino transforms completion of the delivery for room A-120 into an opportunity. As soon as Rhino learns that A-120 is open, it inserts the delivery request in the actual schedule.

Reference

1. M. Beetz, "Structured Reactive Controllers-a Computational Model of Everyday Activity," *Proc.3rd Int'l Conf. Autonomous Agents*, ACM Press, New York, 1999, pp. 228–235.

in response to the robot's context.[9]

As an example, consider the case described in the "Rhino at Work" sidebar, where the robot is required to find a letter on a desk for future delivery. The robot must visually sense the letter but the postprocessing of this sensing will depend heavily on the current context. The controlling process is aware of internal and external factors such as lighting, distance to object, object color, visual clutter, and current processing load and employs them to alter the way it detects the envelopes.

Control system architecture

Figure 3 depicts Rhino's current control system software architecture. The perceptual subsystem is shown on the right and the action subsystem on the left; both are part of the low-level control system (LLCS). The central box is the high-level controller, which we call a structured reactive controller (SRC). The interface between the low-level and high-level control systems is the high-level interface (HLI).

The low-level control system

The LLCS consists of approximately 20 distributed modules, each designed to moni-

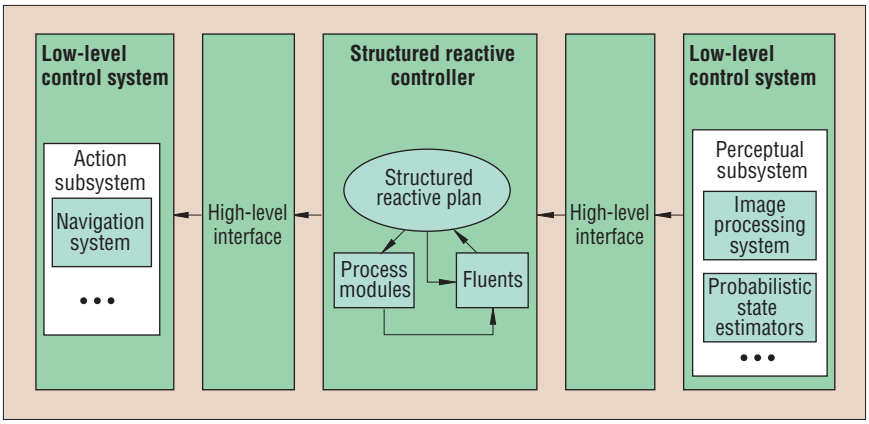

Figure 3. A block diagram of the Rhino control system showing the control and communication links for plan-based control. The control and communication links in the low-level control system have been omitted.

tor or control a dedicated aspect of the robot and provide information to other modules. The modules communicate using an asynchronous message-passing library. The LLCS contains various server modules that interface directly with the robot's sensors and effectors (lasers, cameras, motors, sonar, a pan and tilt unit, a speech unit, buttons, or touch-sensitive displays, for example). It also contains various navigation modules for mapping, path planning, localization, collision avoidance, and other functions. The LLCS modules share important design characteristics. Many of them solve computational problems that can be compactly described in mathematical terms. The localization module, for example, computes the probability distribution over the robot's position, and the path planner computes the shortest paths from all locations to a given destination. The LLCS modules use iterative algorithms with simple update rules and can therefore run many iterations per second. Such fast iterative algorithms are particularly well suited to the LLCS. They can react almost immediately and promptly accommodate asynchronously arriving information. Some of these iterative algorithms are anytime algorithms: they produce initial results quickly and can improve these results given more computation time.

Perceptual subsystem. We enhanced Rhino's first generation components.[10] First, we added two additional probabilistic state estimators beside the localization and mapping modules.[11] The first one handles objects such as doors, chairs, and wastebaskets-permanent parts of the dynamic world model subject to infrequent state changes.[7] A template-matching approach estimates the objects' state density, comparing the real sensor measurements against ideal measurements obtained by simulating each sensor in a 3D world model. The second estimates the trajectory of other robots or of people walking by. In contrast to the static objects handled by the first estimator, these objects are not permanent parts of the environment; they can enter and leave the scene frequently. Therefore, the state estimation method for these objects is based on detecting and tracking dedicated features indicating their presence. For instance, the robot must often track several people simultaneously. If their paths cross, the robot must still keep the people separate to function properly. The estimator relies on a probabilistic data association algorithm for this purpose.

A second component is an image-processing subsystem. Successfully integrating image processing in the Rhino system requires making it dynamically configurable and runtime controlled. Programmers developed Rhino's visual and image-processing components using a programming framework called Recipe (*r*econfigurable, *e*xtensible, *c*apture and *i*mage *p*rocessing *e*nvironment).[12] Recipe provides a standardized, multithreaded environment that loads modules-for image processing or other tasks-at runtime to take advantage of the robot's context. The framework provides standardized facilities for managing resources and capturing images while improving system robustness and promoting code reuse. Recipe modules, as well as being runtime loadable, are scriptable, active entities well able to mirror the SRC's requirements. The SRC can reason about its image-processing plans in the same way as any other plans. The SRC calculates contextual modifications and delegates operations to the Recipe image-processing server. Creating Recipe modules is simple, and the their content is not restricted in any way. So far, we've created Recipe modules for object recognition, gesture recognition, image segmentation, motion-based object recognition, and encapsulation of standard image-processing operations.

Another component deals with communication. Natural language conversation significantly enhances robot capabilities. It augments their ability to both perceive and change their environment. For instance, a robot can ask a person to open a door it cannot open by itself. Natural language communication also enables robots to receive a wider range of job specifications and to acquire information they cannot perceive using sensors. We extended the Rhino control system by adding a module to send and receive electronic mail written in a constrained subset of English. A simple definite-clause grammar parses incoming electronic mail, transforming it into an internal speech act representation (see Figure 4). Object descriptions, such as "the right desk in room A-120," are then interpreted using Rhino's symbolic world model, which makes task specifications such as "pick up the red letter from the right desk in room A-120" effective. The computational processes for communication, including parsing, interpretation, and sentence construction, are then integrated into the RPL plan language. This lets the robot control conversational actions. Using these plan language extensions, we were able to concisely specify a wide spectrum of communication behaviors.

Navigation system. Here, we briefly review the operation of Rhino's navigation system, a component of the action subsystem, to provide the necessary background for later explaining how Rhino learns symbolic navigation plans (see Burgard[10] for a detailed description). To solve a navigation problem, Rhino first transforms a pair of locations, s and d (where s is the start location and d the destination), into a Markov decision problem (MDP). Then, a path planner solves the MDP using a value iteration algorithm. The solution is a mapping from every possible location to the optimal heading to reach the target. The reactive collision avoidance module gets this mapping and, taking the robot's actual sensor readings and the dynamics into account, uses it to reach the target.

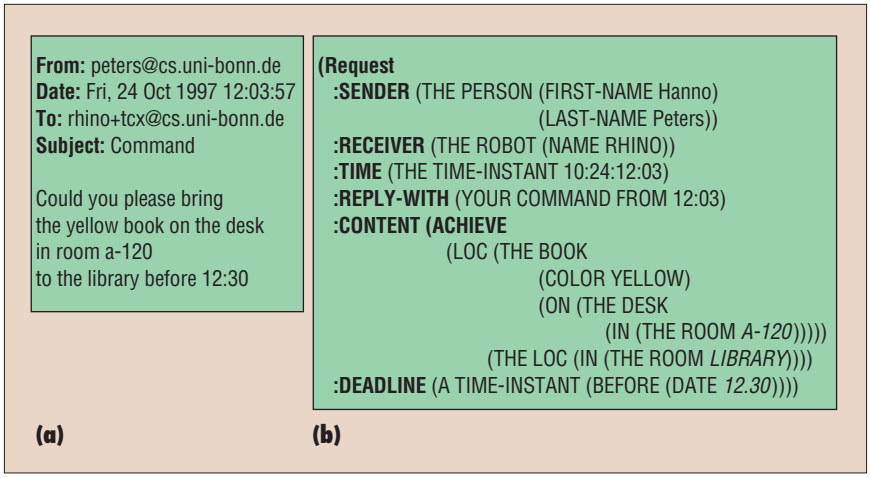

Figure 4. (a) Electronic mail sent to Rhino's computer account, and (b) its translation into the internal speech act representation.

High-level interface

Although the LLCS provides the robot's basic functionality, activating and deactivating continuous control processes, it does not effectively combine the processes into coherent task-directed behavior. Without additional functionality, specifying robust and efficient control programs would be tedious and error-prone. Thus, the Rhino software architecture includes a high-level interface to the LLCS.[13] From the LLCS point of view, the HLI module supports the reliable, effective, and efficient execution of nontrivial tasks. It does so by synchronizing the operations at the LLCS level in task-specific ways and by recovering from local execution failures.

From the SRC point of view, HLI provides tasks as a convenient programming abstraction for the low-level control processes and as a reliable execution component. Using HLI, the SRC can start and terminate tasks and wait for their completion. In addition, HLI maintains the robot's current operational status as well as relevant feedback from the control processes-such as task completion signals and so on-for the SRC's immediate access.

Structured reactive controllers

Given a set of jobs, an SRC concurrently executes the default routines for each individual job. These general, flexible routines work well in standard situations. They cope well with partly unknown and changing environments, run concurrently, handle interrupts, and control robots without assistance over extended periods. While it executes routine activities, the SRC also tries to determine whether its routines might interfere with each other and monitors robot operation for nonstandard situations. If it finds one, the robot will try to anticipate behavior flaws by predicting how its routine activities might work in this nonstandard situation. If necessary, it revises its routines to make them more robust. Finally, it integrates the proposed revisions smoothly into the stream of its actions.

Prediction in structured reactive controllers. Temporal projection, the process of predicting what will happen when a plan is executed, is essential for autonomous robots to successfully plan their missions. To project their plans, robots must have causal models that represent the effects of their actions. Rhino uses causal models for predicting the behavior generated by modern autonomous-robot controllers accurately enough to foresee a wide range of realistic execution problems. This is possible because Rhino's action models reflect that

- physical robot actions cause continuous change;
- controllers are reactive systems;
- the robot is executing multiple physical and sensing actions; and
- the robot is uncertain about the effects of its actions and the state of the environment.

The problem of using such realistic action models is obvious. Nontrivial concurrent plans for controlling robots reliably are complex. Several control processes are usually active, and many more are dormant, waiting for conditions that trigger their execution. The branching factors for possible future states-not to mention the distribution of execution scenarios that they might generate-are immense. The accurate computation of this probability distribution is prohibitively expensive in terms of computational resources.

Therefore, Rhino employs probabilistic sampling-based temporal projection methods that can infer information from such complex distributions quickly and with bounded risk.[11] Rhino's temporal-projection methods solve the following prediction problem: given a set of possible plan failure modes and a risk that the robot is willing to take that its prediction is wrong, infer whether the plan is likely to produce a failure mode with a probability greater than a given threshold.

Transformational planning of concurrent reactive plans. Rhino's plan adaptors perform execution-time transformational planning with concurrent reactive plans.[6,14] A library of modular, efficient default plans enables Rhino to cope with most eventualities. Using this library, plan adaptors can compute a default plan for a set of tasks by retrieving and instantiating plans for individual tasks and pasting them together to form the overall plan. Although pasting default plans together is fast, it is prone to producing plans that might fail under contingencies. So, whenever the robot detects a contingency, a plan adaptor will project the this contingency's effects on its current plan and revise the plan to make it more robust. Whenever a plan adaptor thinks its plan is better than the executed one, it will replace the current plan with the better one and restart the new plan.

Planning is implemented as a search in plan space. A node in the space is a proposed plan; the initial node is the default plan created using the plan library. A step in the space requires three phases. First, a plan adaptor *projects* a plan to generate sample execution scenarios for it. Then, in the *criticism* phase, a plan adaptor examines these execution scenarios to estimate how good the plan is and to predict possible plan failures. It diagnoses the projected plan failures by classifying them in a taxonomy of failure models. The failure models serve as indices into a set of transformation rules that are applied in the third phase, *revision*, to produce new versions of the plan that are, one hopes, improvements.

Learning symbolic robot plans. We have already stressed the importance of explicitly representing the plans that the robot has committed to execute so that the robot can use its limited computational resources for flexible task execution and effective action planning. However, this raises the question of how such plans can be obtained. Rhino's navigation system,[10] for example, considers navigation a Markov decision problem. It models naviga-

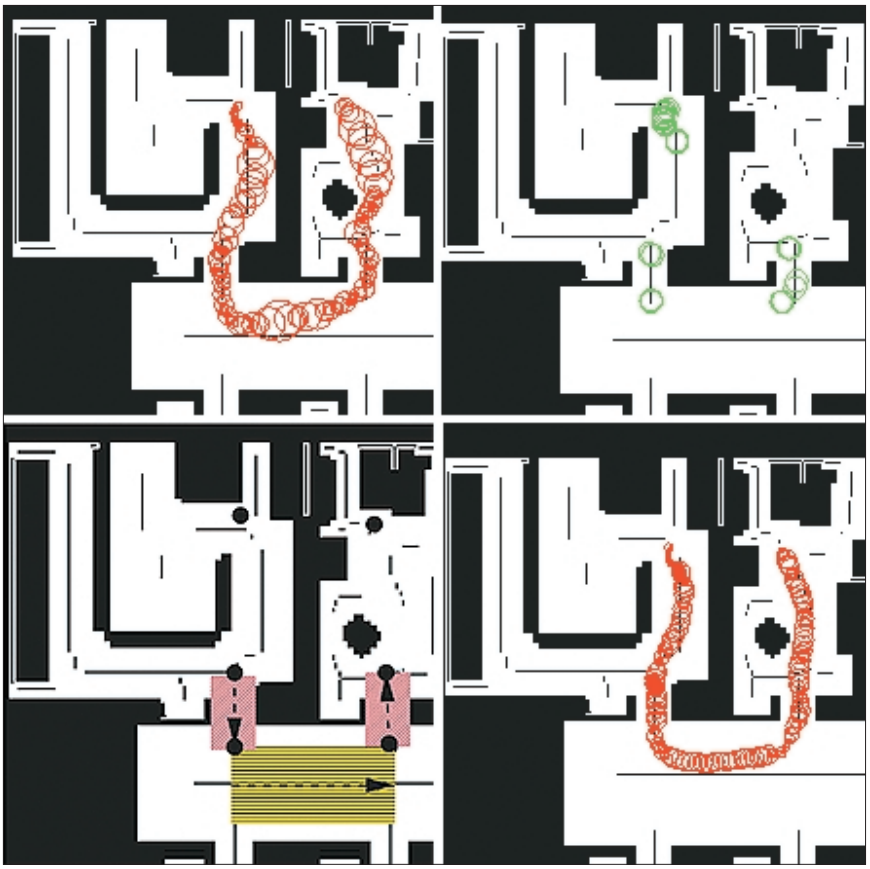

Figure 5. A sample learning session: (a) a behavior trace of the default plan, (b) behavior stretches where the robot moves conspicuously slowly, (c) the added subplans in the learned navigation plan, and (d) a behavior trace of the learned plan.

tion behavior as a finite-state automaton in which navigation actions cause stochastic state transitions. The robot is rewarded for reaching its destination quickly and reliably. he reward is mentally within the controller. A subsystem assigns rewards to execution scenarios. For example, the mental reward for a navigation task is higher if the task executes faster. This enables the control system to learn faster plans and forestall unnecessary delays that might be caused by closed doors. A solution for such problems is a mapping from states to actions that maximizes the accumulated reward. Such state-action mappings, necessary for adapting quickly to changing circumstances and quickly responding to exceptional situations, are inappropriate for high-level plan management.

We developed XFRMLEARN,[8] a learning component that builds up explicit symbolic navigation plans automatically. Given a navigation task, XFRMLEARN learns to structure continuous navigation behavior and represents the learned structure as compact and transparent plans. It obtains the structured plans by starting with monolithic default plans that are optimized for average performance and adding subplans to improve the given task's navigation performance.

XFRMLEARN's learning algorithm starts with a default plan that transforms a navigation problem into an MDP and then passes the MDP to Rhino's navigation system. After Rhino's path planner has determined the navigation policy, the navigation system activates the collision avoidance module. XFRMLEARN records the resulting navigation behavior and looks for stretches of behavior to improve. XFRMLEARN then tries to explain the improvable behavior stretches using causal knowledge and its knowledge about the environment. These explanations are used to index promising plan revision methods that introduce and modify subplans. The revisions are subsequently tested in a series of experiments to decide whether they are likely to improve navigation. The symbolic plan incorporates successful subplans. Figure 5 illustrates a sample learning session.

Using this algorithm, Rhino can autonomously learn compact and well-structured symbolic navigation plans by using MDP navigation policies as default plans and repeatedly inserting subplans that significantly improve navigation performance. The plans learned by XFRMLEARN support action planning and opportunistic task execution. These plans provide plan-based controllers with subplans such as "traverse a particular narrow passage or an open area." More specifically, navigation plans can generate qualitative events from continuous behavior (such as entering a narrow passage), support online adaptation of navigation behavior (drive more carefully while traversing a particular narrow passage),[3] and allow compact and realistic symbolic predictions of continuous, sensor-driven behavior.[5]

Long-term demonstrations

The flexibility and reliability of runtime plan management and plan transformation have been extensively tested in the Minerva

Figure 6. Minerva: (a) a close-up view and (b) Minerva leading a tour group at the Smithsonian.

Related Work

In recent years, a number of impressive long-term real-world demonstrations have shown the potential impact of this technology:

- In NASA's Deep Space program, a plan-based robot controller called the Remote Agent[1] has autonomously controlled scientific experiments in space.
- In the Martha project,[2] fleets of robots have been effectively controlled and coordinated.
- Xavier,[3] an autonomous mobile robot with a plan-based controller, has navigated through an office environment for more than a year, allowing people to issue navigation commands and monitor their execution over the Internet.
- Chip[4] is an autonomous mobile robot serving as a general-purpose robotic assistant, providing a variety of tasks including cleanup.

All these approaches to plan-based control differ regarding the aspects of the control problem they focus on and the assumptions they make. For example, Xavier employs Rogue, a system for interleaved task planning, execution, and situation-dependent learning.[5] In Xavier, action plans are sequences of actions and reactive behavior is generated by Xavier's plan execution system Rogue. In Rhino, on the other hand, reactive behavior is specified explicitly and transparently by the plans. As another example, a main research goal of the Chip control system is the development of a general library of low-level and situation-driven high-level plan sketches. In contrast to Chip's plans, SRCs' plans are also designed to be reasoned about and revised. Finally, our approach to plan-based control of robotic agents differs from the Flakey approach[6] in several ways. Flakey dispenses with execution time deliberation[7] and therefore cannot predict or forestall execution failures in the way SRCs do. The emphasis in developing the remote agent is the design of a plan-based control system that controls a complex physical system with extreme reliability.[1]

References

1. N. Muscettola et al., "Remote Agent: To Go Boldly Where No AI System Has Gone Before," *Artificial Intelligence*, vol. 103, no. 1–2, Aug. 1998, pp. 5–47.
2. R. Alami et al., "Multi Robot Cooperation in the Martha Project," *IEEE Robotics and Automation Magazine*, vol. 5, no. 1, Mar. 1998, pp. 36–47.
3. R. Simmons et al., "A Modular Architecture for Office Delivery Robots," *Proc. 1st Int'l Conf. Autonomous Agents*, ACM Press, New York, 1997, pp. 245–252.
4. J. Firby et al., "Programming CHIP for the IJCAI-95 Robot Competition," *AI Magazine*, vol. 17, no. 1, Spring 1996, pp. 71–81.
5. K. Haigh and M. Veloso, "High-level Planning and Low-level Execution: Towards a Complete Robotic Agent," *Proc. 1st Int'l Conf. Autonomous Agents*, ACM Press, New York, 1997, pp. 363–370.
6. K. Konolige et al., "The Saphira Architecture: A Design for Autonomy," *J. Experimental and Theoretical Artificial Intelligence*, vol. 9, no. 2, Apr. 1997, pp. 215–235.
7. K. Myers, "A Procedural Knowledge Approach to Task-Level Control," *Proc. 3rd Int'l Conf. AI Planning Systems*, AAAI Press, Menlo Park, Calif., 1996, pp. 158–165.

robot (see Figure 6). During the 13 days that Minerva operated as a robotic tour guide in the Smithsonian Museum,[1] it was in service for more than 94 hours, completed 620 tours, showed 2,668 exhibits, and traveled over more than 44 km. As its high-level controller, Minerva used an SRC, which directed the robot's course of action in a feedback loop executing more than three times a second. Minerva used plan adaptors to install new commands, delete completed plans, and schedule tours. Minerva made about 3,200 execution-time plan transformations while performing its tour guide job. Minerva's plan-based controller differs from Rhino's only with respect to its top-level plans in its plan library and some of the plan adaptors used.

In two other experiments, we evaluated specific capabilities of the plan-based controller. In one experiment, we showed how predictive plan transformation improves performance. We did this by outperforming controllers without predictive transformations in situations that require foresight. At the same time, Rhino remained a strong performer in situations that require no foresight. In a second experiment, we tested Rhino's ability to learn symbolic navigation plans. Over more than 100 hours of autonomous experimentation, Rhino's learning component significantly and substantially improved its navigation system's performance.[8]

See the "Related Work" sidebar for more discussion of projects similar to Rhino.

Our results provide promising design principles for autonomous robot control systems. Future work will focus on improving the robustness of the Rhino control system in the face of unknown situations and hardware or software failures. We will also further extend and exploit the expressive power of the system's probabilistic models and context-sensitive calculation abilities.

Another important research thread is the development of a general computational model for plan-based control of robotic agents. We research into this direction by realizing autonomous controllers based on SRCs for diverse applications including autonomous robot soccer, autonomous control of a toy factory, autonomous robot rescue missions, and distributed supply chain management. We believe that a thorough analysis of the individual plan based controllers for the different applications will give us a much deeper understanding of which concepts such a computational model should provide and support. ■

Acknowledgments

We acknowledge Sebastian Thrun's continuing contributions to the Rhino system. We also thank Denise Hurst and the anonymous reviewers for their valuable suggestions. The research described in this article has been partially funded by Deutsche Forschungsgemeinschaft (199483) and by EC contract Nr ERBFMRX-CT96-0049.

filler

References

1. S. Thrun et al., "Minerva: A Second Generation Mobile Tour-Guide Robot," *Proc. IEEE Int'l Conf. Robotics and Automation* (ICRA '99), IEEE Press, Piscataway, N.J., 1991.
2. R. Simmons et al., "Xavier: Experience with a Layered Robot Architecture," *ACM Intelligence*, ACM Press, New York, 1997.
3. R. Simmons et al., "Coordinated Deployment of Multiple, Heterogeneous Robots," *Proc. 2000 IEEE/RSJ Int'l Conf. Intelligent Robots and Systems*, IEEE Press, Piscataway, N.J., 2000.
4. D. McDermott, "A Reactive Plan Language," tech. report YALEU/DCS/RR-864, Yale Univ., New Haven, Conn., 1991.
5. M. Beetz and H. Grosskreutz, "Probabilistic Hybrid Action Models for Predicting Concurrent Percept-Driven Robot Behavior," *Proc. 5th Int'l Conf. AI Planning Systems*, AAAI Press, Menlo Park, Calif., 2000, pp. 42–61.
6. D. McDermott, "Transformational Planning of Reactive Behavior," tech. report YALEU/DCS/RR-941, Yale Univ., New Haven, Conn., 1992.
7. D. Schulz, "Template-Based State Estimation of Dynamic Objects," *Proc. EUROBOT Workshop*, IEEE CS Press, Los Alamitos, Calif., 1999.
8. M. Beetz and T. Belker, "Environment and Task Adaptation for Robotic Agents," *Proc. 14th European Conf. Artificial Intelligence* (ECAI-2000), IOS Press, Amsterdam, 2000, pp. 648–652.
9. M. Beetz et al., "Transparent, Flexible, and Resource-Adaptive Image Processing for Autonomous Service Robots," *Proc. 13th European Conf. Artificial Intelligence* (ECAI-98), IOS Press, Amsterdam, 1998, pp. 632–636.
10. W. Burgard et al., "Experiences with an Interactive Museum Tour-Guide Robot," *Artificial Intelligence*, vol. 114, no. 1–2, 1999, pp.3–55.
11. D. Fox et al., "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots," *Proc. 16th Nat'l Conf. Artificial Intelligence*, AAAI Press, Cambridge, Mass., 1999, pp. 343–349.
12. T. Arbuckle and M. Beetz, "Extensible, Runtime-Configurable Image Processing on Robots-the Recipe System," *Proc. 1999 IEEE/RSJ Int'l Conf. Intelligent Robots and Systems*, IEEE Press, Piscataway, N.J., 1999.
13. D. Hähnel, W. Burgard, and G. Lakemeyer, "Golex-Bridging the Gap between Logic GOLOG and a Real Robot," *Proc. 22nd German Conf. Artificial Intelligence* (KI 98), Springer, Heidelberg, 1998.
14. M. Beetz, "Concurrent Reactive Plans: Anticipating and Forestalling Execution Failures," *Lecture Notes in Artificial Intelligence*, vol. LNAI 1772, Springer-Verlag, Heidelberg, Germany, 2000.

The Authors

Michael Beetz is a lecturer in Munich University of Technology's Department of Computer Science. He is also the principal investigator for several nationally funded research projects, including plan-based control in distributed supply chain management, learning robot action plans, and autonomous robotic soccer. His research interests include AI, plan-based control of autonomous agents, and intelligent autonomous robotics. He received his PhD in computer science from Yale University and his venia legendi from the University of Bonn in 2001. Contact him at the Computer Science Dept. IX, Munich Univ. of Technology, Orleansstr. 34, D-81667 Munich, Germany; beetzm@in.tum.de; www9.in.tum.de/~beetzm.

Tom Arbuckle is a member of the University of Bonn team creating ABIS, a fully automated system identifying bee species from images of living bees' wings. He previously worked in the EU Virgo project and the development of Recipe, a system for performing image processing on autonomous robots. He received a BSc Honours (1st class) in chemical physics and a PhD in computer algebra studies of wave propagation in nonlinear media from Glasgow University. Contact him at the Institute fuer Informatik III, Universitaet Bonn, Roemerstr. 164, D-53117 Bonn, Germany; arbuckle@cs.uni-bonn.de; www.cs.uni-bonn.de/~arbuckle.

Thorsten Belker works as a research scientist at the Intelligent Autonomous Systems Group, University of Bonn. He also studies philosophy and computer science at the University of Bonn and at the University of Warwick. His research interests include AI and machine learning with a focus on robot learning. He received his diploma in computer science from the University of Bonn. Contact him at the Dept. of Computer Science III, Univ. of Bonn, Roemerstrasse 164, D-53117 Bonn, Germany; belker@cs.uni-bonn.de.

Armin B. Cremers is a full professor of computer science and chair of the University of Bonn's Department of Computer Science III. His research interests include database and information systems, autonomous mobile robots, and AI. Contact him at the Dept. of Computer Science III, Univ. of Bonn, Roemerstr. 164, D-53117 Bonn, Germany; abc@cs.uni-bonn.de.

Dirk Schulz is a PhD student in the University of Bonn's Department of Computer Science III. His main research interests are service robotics, probabilistic state estimation, and Internet-based telepresence systems. Contact him at the Dept. of Computer Science III, Univ. of Bonn, Roemerstr. 164, D-53117 Bonn, Germany; schulz@cs.uni-bonn.de.

Maren Bennewitz is a research scientist in Department of Computer Science at the University of Freiburg and is also involved in Carnegie Mellon University's Nursebot Project. Her research interests include mobile robot navigation and multirobot coordination in dynamic environments. She received her diploma degree in computer science from the University of Bonn. Contact her at the Institut für Informatik, Albert-Ludwigs-Universität Freiburg, Georges-Köhler-Allee, Geb. 079, D-79110 Freiburg, Germany; maren@informatik.uni-freiburg.de.

Wolfram Burgard is an associate professor for autonomous intelligent systems in the University of Freiburg's Department of Computer Science and an adjunct faculty member at the Carnegie Mellon University Center of Automated Learning and Discovery. His research focuses on autonomous mobile robots and AI applications, including probabilistic techniques for robot navigation and control. He received his PhD in computer science from the University of Bonn. Contact him at the Institut für Informatik, Albert-Ludwigs-Universität Freiburg, Georges-Köhler-Allee, Geb. 079, D-79110 Freiburg, Germany; burgard@informatik.uni-freiburg.de; www.informatik.uni-freiburg.de/~burgard.

Dirk Hähnel is a research scientist in the University of Freiburg's Department of Computer Science. His current interests are learning compact 3D maps for mobile robots and model reconstruction from surface measurements. He received his diploma in computer science from the University of Bonn. Contact him at the Institut für Informatik, Albert-Ludwigs-Universität Freiburg, Georges-Köhler-Allee, Geb. 079, D-79110 Freiburg, Germany; haehnel@informatik.uni-freiburg.de.

Dieter Fox is an assistant professor of computer science at the University of Washington. His research interests are AI, probabilistic state estimation for mobile robotics, and multirobot collaboration. Together with colleagues, he recently introduced particle filters as a powerful tool for state estimation in mobile robotics. He received the 2000 European Artificial Intelligence Dissertation Award and an NSF Career award. He received his PhD from the University of Bonn. Contact him at the Dept. of Computer Science, Univ. of Washington, Box 352350 Seattle, WA 98195-2350; fox@cs.washington.edu.

Henrik Grosskreutz is a PhD student in the Graduate School of Informatics and Engineering at RWTH Aachen. His research interests include cognitive robotics with a focus on reasoning with continuous change and probabilistic effects. He received a MS in computer science from the University of Bonn. Contact him at the Dept. of Computer Science V, Aachen Univ. of Technology, 52056 Aachen, Germany; grosskreutz@informatik.rwth-aachen.de.