Robot Navigation Using Integrated Retrieval of Behaviors and Routes

Susan Eileen Fox and **Peter Anderson-Sprecher**
Macalester College
1600 Grand Avenue
Saint Paul, MN 55105
fox@macalester.edu and pandersonspr@macalester.edu

Abstract

RUPART[1] is a hybrid robot control system for navigating a real-world, academic building. Hybrid robot control systems provide robust low-level navigation together with strategic planning abilities. While a hybrid system may produce better overall performance, it is often a complex system that must balance reactive with deliberative tasks, and where learning and adaptation are more difficult to achieve. RUPART addresses the issues of complexity, balance, and learning by using a single case-based-reasoning (CBR) system to store and retrieve cases for both its reactive and deliberative systems. At this stage, the CBR system contains behavior cases, which determine the reactive actions of the robot, and route cases, which determine strategic plans for navigating to a goal location. RUPART will be extended in the future to use CBR to manage multiple goals and higher-level strategic decision-making. This first-stage RUPART system retrieves and applies cases that are suitable to its needs: behavior cases to control its reactive behaviors, route cases when planning to reach a new goal. The resulting system can learn new route and behaviors, without the complexity of multiple reasoning and learning algorithms usually entailed by a hybrid control system.

Introduction

Autonomous robots operating in the real world face multiple challenges at the same time. At one level, the robot must move around in a world that changes rapidly, dealing with inaccurate sensor data. At the same time, most robot applications require strategic planning as well: determining where to move and planning how to get there, and managing multiple, possibly conflicting goals. Hybrid robot control systems use separate modules to manage the robot's reactive tasks and to manage its strategic planning. Each module may use a completely different AI technique.

Hybrid systems are complex, both in design and in operation. System designers must decide what tasks each module will undertake, how the modules interact with each other, and how to balance the needs each module has for computing time, sensor resources, and control of the robot. Such decisions are often made ad hoc, and are typically fixed and unchangeable. Incorporating learning into a hybrid system may require separate learning algorithms for each module. The RUPART project suggests that we can simplify the design of a hybrid system by using a single reasoning and learning system to serve both reactive and deliberative tasks. RUPART can adjust between its tasks dynamically, as its goals and the world around it change.

RUPART is a hybrid robot control system for a straightforward task: a delivery robot navigating an academic building, the Olin-Rice Hall of Science. This environment is somewhat complex, as it contains offices, labs, classrooms, and open spaces, as well as hallways of varying dimensions. It is also a highly dynamic domain, used throughout the day by a variety of people. Human obstacles are common, as are carts and boxes blocking hallways. RUPART's delivery task requires both good reactive behavior, and good strategic planning. Currently in its first phase, the system receives one goal at a time; it plans a route to reach the goal, and executes the plan. Ultimately, RUPART will be extended to manage multiple goals, enriching its task and behavior. This extension does not significantly change the system's architecture, which is largely complete at this point. Work on RUPART appeared in the past (Fox 2001; 2000); the project was set aside for several years and restarted from scratch with the work appearing here

RUPART addresses the issue of hybrid system complexity by using a single case-based reasoning system to index, store, retrieve, and learn both behavior and route cases. The retrieved cases feed into separate behavior-cased control and route-planning systems. However, the kind of case retrieved determines whether new behaviors or new strategic route-planning is needed. A single case learning mechanism allows both reactive and deliberative modules to adapt to the environment they experience.

We first provide some background into robot control methods, particular those using CBR. Next we describe the integrated architecture and its underlying CBR system, and report some systematic tests of the current phase. From these tests we conclude that RUPART's architecture does support adaptive and responsive robot navigation. Extensions to RUPART planned in the second phase are outlined.

[1]Robot Uniting Plan-Ahead and Reactive Tasks

Background

Reactive robot control and deliberative planning were once approaches at odds with one another. Increasingly, they are used in combination with one another to produce systems that can both respond to the world quickly and respond strategically. Deliberative systems typically involve extensive AI reasoning, complex internal models, and a high level of abstraction. Reactive systems, on the other hand, focus on quick decision-making and the emergence of intelligent moment-to-moment behavior from collections of simple behaviors. In many ways deliberative and reactive systems have complementary strengths and weaknesses. Hybrid systems combine separate modules for deliberative and reactive tasks, intending to exhibit the best of both approaches. Such systems have been applied to a variety of complex robotics tasks, including office navigation and RoboCup (Simmons *et al.* 1997; Ferrein, Fritz, & Lakemeyer 2005).

Hybrid robot control raises some new issues. Such systems are often more complex than a single-approach architecture, as they typically use different mechanisms for low-level decision-making versus high-level decision-making. Determining the right balance between reacting and deliberating is often solved on a case-by-case basis. To incorporate learning in a hybrid system may require separate learning techniques for each module. The RUPART project will simplify a hybrid system's architecture by using a single underlying mechanism: case-based reasoning. RUPART's case memory contains both reactive cases and deliberative cases, accessed using the same indices and retrieval. RUPART uses case-based learning for both reactive and deliberative components, and automatically finds a balance between reacting and deliberating.

Case-based reasoning is an interesting approach for unifying the levels of a hybrid robot system, because it has been successfully applied to both deliberative and reactive tasks. High-level planning has been a common application domain for CBR systems for a long time. Route planning, specifically, can be done well using CBR (Fox 1995; McGinty & Smyth 2001; Goel *et al.* 1991; Haigh & Veloso 1995). Reactive robot control systems have also be implemented using CBR to store and reuse behaviors and the scenarios in which they apply (Ram *et al.* 1992; Ram & Santamaria 1997; Chagas & Hallam 1998).

Supic and Ribaric (2001) suggest a hybrid system called SCBR for autonomous navigation. SCBR uses CBR at both deliberative and reactive levels. In SCBR, route cases contain a sequence of "situations," which may be viewed as locations along with how to behave in those locations. Thus, behavior is intrinsically tied to specific locations. By contrast, in RUPART, behavior selection and route selection are essentially independent of one another. Also, Supic and Ribaric also propose two separate CBR systems with separate case memories and indexing/retrieval methods. RUPART uses a single CBR system with a single indexing and retrieval method for all cases.

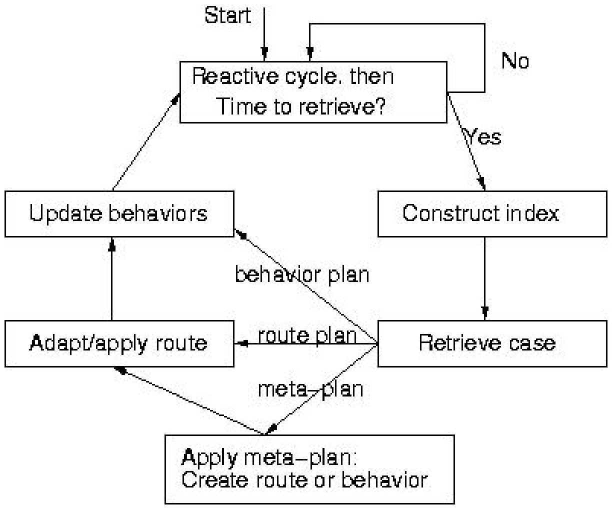

Figure 1: RUPART's control system: RUPART runs in reactive mode until a retrieval is triggered. It then retrieves and applies a case, having effects that depend on the case type. New cases may be generated and stored, enabling learning.

The RUPART System

Through RUPART we explore case-based-reasoning as a mechanism for integrating and balancing reactive control with more deliberative control. RUPART controls a robot navigating the halls of the Olin-Rice science building at Macalester College. Navigation in an office or classroom setting is well-understood in the robotics community: RUPART investigates alternative methods of controlling a robot in such an environment, and methods for incorporating learning into such a robot. RUPART's environment is one with many challenges to a robot: it is dynamic, with many mobile and stationary obstacles to manage, it is essentially unmodified to suit the robot's needs. Currently RUPART's localization method requires the placement of occasional landmarks: in future phases we intend to phase these landmarks out, using case-based localization in its place.

The current phase of RUPART's development focuses on strategic route-planning and reliable navigation to a chosen point. Future phases will include goal management and reasoning about priorities.

RUPART Architecture

The RUPART system has two main levels, a basic operating level and a case-based retrieval level (illustrated in Figure 1). At its lowest level, it continually executes a reactive update cycle that receives sensor data and selects actions in response. Each update cycle takes a fraction of a second. The action chosen is governed by the currently active set of behaviors, and the route plan steps that guide those behaviors toward a goal.

During each update cycle, RUPART considers whether or not to retrieve new "guidance" from its case memory. A variety of high level and low level factors go into that decision, all reflect a sense of the world changing. Triggers include unexpected landmarks, achieving a goal location, or merely the passage of time.

The retrieval cycle forms RUPART's second level. During the retrieval cycle, an index is created that captures the robot's current state and its current view of its world. The index contains both low level and high level aspects of the robot's state: current sonar values, the average of the past five sonar readings, the current forward and turn velocities, the robot's estimated location, its current short-term goal, and its current long-term goal.

The case-based retriever finds the most similar case in the case memory: similarity is a weighted sum of distance measures for each index feature. Distances between locations are calculated by a straight-line metric. Some features may be missing from an index: in particular, if there is no current route plan in place, then there will be no short-term goal, either.

The initial case memory we have tested contains twenty different route plans, and nine different behavior cases; a very small set. Ultimately, the case memory may contain other kinds of cases, including *meta-plans* that direct RUPART to create new routes or new behaviors from scratch. Upon retrieving a behavior case, RUPART updates the behavior-based control system without disturbing the current route or goals. A retrieved route plan is adapted to fit the current situation, and then replaces the old route, and the robot's short and long-term goals.

Implementation Details

RUPART runs on a Pioneer 2 DX robot that has a PTZ camera as well as front and rear sonar sensors. All computation takes place through an on-board Linux computer. Robot control is implemented using the Python Robotics (Pyro) system (Blank *et al.* 2003). Pyro provides a interface for control programs written in Python, as well as a number of useful libraries, including a behavior-based control module that RUPART uses. RUPART uses Pyro's behavior-based control architecture, while providing the behaviors through its CBR system. Pyro, in turn, controls the robot through ARIA (ActivMedia Robotics Interface for Applications), ActivMedia's native library for the Pioneer robots (ActivMedia Robotics 2005). ACTS, also provided by ActivMedia, is a color-tracking system used by RUPART for its landmark-based localization system.

Originally, we intended to use a Monte Carlo localization module, SONARNL, provided by ActivMedia to work with sonar-based sensor data. In early tests it proved highly successful for the robot's environment. However, we were unable to integrate SONARNL with the Pyro system, and developed a simplistic, vision-based localization algorithm instead. Current work is underway on a Monte Carlo localization module of our own.

Behavior-based control in RUPART

The Pyro behavior-based system used by RUPART allows two levels of specification: behaviors and states. A behavior is simply a piece of code for some specific purpose. Multiple behaviors may be combined together in a state, and may be weighted according to their priorities. If a state is currently

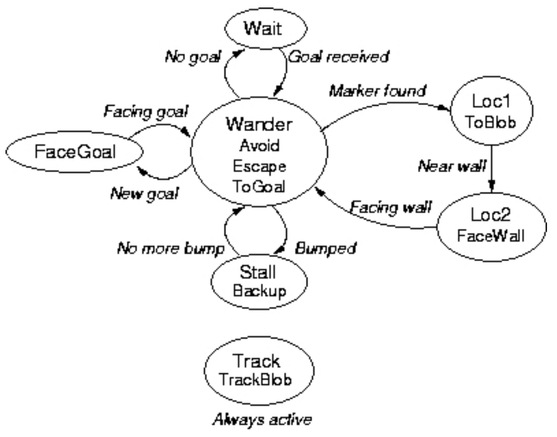

Figure 2: RUPART's behavior-based control system, consisting of states and associated behaviors.

active, then its behaviors each produce a recommended action by the robot. These actions are combined together using fuzzy logic. At the higher level, states may be connected like a finite-state machine, so that the robot can shift from one state to another based on some environmental triggers. Active states control the robot; multiple states may be active at one time.

RUPART uses multiple states, and states with multiple behaviors. Figure 2 illustrates RUPART's behavior-based system. The Track state and its behavior move the camera to look for markers. This state is always active and operates in parallel with the motion control states. The primary motion-control state is Wander, which combines together three different behaviors. Currently, behavior cases specify the parameters of the Wander state: different velocities and weights for its behaviors. The other states manage unusual circumstances that occur for only brief periods of time: having no goal, having a brand-new goal, bumping some object in the world, or localizing when a marker is detected.

The Wander state blends three behaviors: Avoid, Escape, and ToGoal. Avoid controls primary translation, as well as basic obstacle avoidance. It moves forward and straight when obstacles are absent, slows and turns when they are present. Escape turns toward perceived gaps in sonar; useful for entering doorways and other narrow spaces, as well as escaping local minima generated by cluttered areas. ToGoal rotates robot to face the current short-term goal. The strength of the rotation is inversely proportional to the robot's estimated distance from the goal.

A single state was chosen for basic behavior because it allows flexibility in the behaviors activated by the CBR system, while keeping the initial behavior cases very simple. In the next phase, we will begin experimenting to broaden the range of behavior cases, intending to reduce the non-CBR portions of the system. We have begun work on case-based learning of behavior cases.

Localization

RUPART uses a simple, landmark-based localization scheme, integrated with the behavior-based system. Landmarks are brightly colored pieces of paper posted on walls. The exact position of each landmark is known to the system, and each is placed near a known location the robot uses for navigation.

Except when localization it triggered, the robot updates its estimated location using dead reckoning. Localization occurs whenever the `Track` state of the behavior-based system detects a nearby marker. At that time, the `Loc1` and `Loc2` states take over to localize the robot, putting all other actions, including goal-seeking, on hold.

During localization, the robot first moves close to the marker, in order to get close enough to the wall for accurate sonar readings. It then turns itself perpendicular to the marker's wall, while keeping its camera fixed on the detected marker. The robot's location is then calculated based on the sonar-measured distance to the wall, and the angle of the camera toward the marker.

Markers are not placed at every known location the robot has. Instead, markers are placed near intersections of hallways, where hallways empty into open spaces, and halfway along the longer hallways the robot may traverse. The robot determines from its estimated location which marker is closest, and localizes based on that assumption.

If all goes well, the robot localizes only when a short-term goal location is reached. If an unexpected marker is detected, RUPART guesses that the marker is the next one it expects to see, and erases its current route plan. This triggers a case retrieval, which typically generates a new route plan, starting at its newly localized position.

RUPART's localization is its weakest point: too few landmarks and RUPART gets lots between them; too many landmarks and its progress is slowed and it risks misidentifying a landmark. Improvements to localization are ongoing at the present time, incorporating Monte Carlo localization with the landmark-based system currently in place.

Deliberative planning in RUPART

The behavior-based control system provides robust short-term action by the robot, including seeking a particular location in the world, so long as it is within "view" (i.e., not separated from the robot by a wall). The goal of the deliberative planner, then, is to set out a sequence of locations that lead the behavior-based system toward the planner's ultimate goal location. An appropriate analogy is setting out a series of dog treats in order to entice a dog where you want it to go. As the robot reaches each short-term goal, the next goal in the sequence is automatically put into place.

A route is just a series of locations, where adjacent locations lie within sight of each other. We have fixed certain "meaningful" locations in the world and have created an adjacency graph to represent these locations. Figure 3 shows RUPART's map of the world, including the set of known locations and the adjacency graph that uses them. The locations chosen for this graph are those that the robot might reasonably expect to receive as goal locations: outside and inside of important rooms, and key positions in hallways and open spaces.

Locations in RUPART are described using either the name of a known location (e.g., `AtriumSW`, `HomeLab`, etc.) or

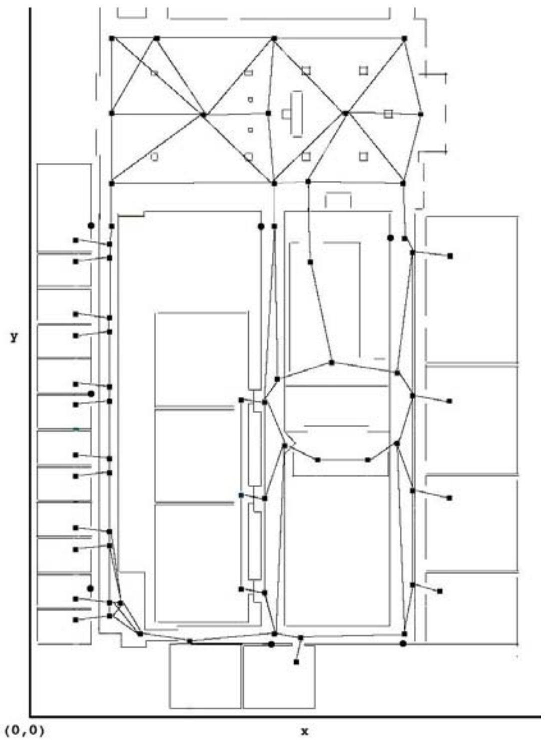

Figure 3: RUPART's map of its world, including known, named locations (squares) and the adjacency graph over them. Localization markers are shown as dots.

an x-y coordinate pair. The origin of the coordinate system is at the southeast corner of the building and, due to the building's shape, is actually outside the building itself. Locations are in meters from the origin; the x dimension runs north-south and the y dimension runs east-west.

Routes combine both kinds of location descriptions to form a sequence of locations on the map. Initially, the case memory is seeded with routes that contain only known locations, but as the system learns new routes coordinate locations become integrated.

Route planning takes place whenever the case-based system is triggered and a route case is retrieved. This occurs when the robot has a new goal to reach, when it finds itself at an unexpected location, or if it fails to reach a required short-term goal location. Route cases are indexed by the robot's state, just as behavior cases are. Adaptation is based on the difference between current start and goal, and retrieved start and goal.

Adaptation of a route takes place at the front and/or the end of the route. Since adaptation at the end of a route is entirely symmetric to adaptation at the front, we will only discuss front-end adaptation. Figure 4 illustrates the adaptation process at both ends. If the current starting location does not match the retrieved starting location, then a supporting graph-based path planner is used to generate a path from the current start to the retrieved start. This route will contain only known locations, except its endpoints, which are the current and retrieved starting locations.

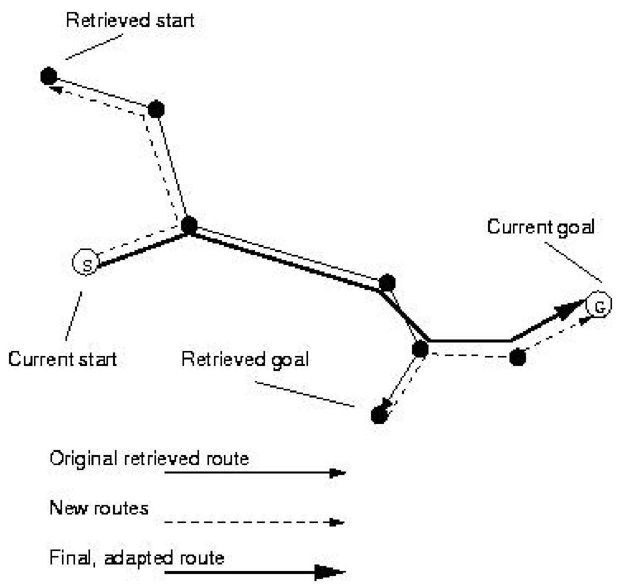

Figure 4: An example of route adaptation: new routes are generated from current start to retrieved start and from retrieved goal to current goal. Then overlaps between retrieved steps and new steps are eliminated.

The new route is then integrated with the front of the retrieved route. A simplistic integration would simply append the steps from the new route to the front of the retrieved route. This, however, can quickly lead to very poor routes. If the start and retrieved locations are close to each other, as they should be, then there is a very good chance that the new route overlaps with the retrieved route in some way (see Figure 4 for an example). RUPART's integration process finds the point at which the two routes begin to overlap, and appends the first part of the new route to the remainder of the retrieved route. The same process is repeated symmetrically to adapt the end of the route.

The underlying path planner uses an "informed" variation on Dijkstra's algorithm to generate a path through the graph of known locations. If the start or end locations provided to it are coordinate locations, then it first determines the closest known locations, and then plans from those points instead.

Determining the closest known location to a coordinate is not done using a straight-line distance metric. The planner has a map that consists of the graph of known locations, and their associated coordinates in the "real" map. It does not have information about walls or other obstacles in the world. Going by straight-line distance, it might select a closest known location that lay on the far side of a wall. This problem was avoided by associating "regions of influence" with those known locations that were within a room or surrounded by other sorts of obstacles. Any coordinate point within a region of influence is automatically assumed to be closer to the region's known location than to any other location in the graph.

The deliberative planner learns new routes by storing the results of its adaptation. We analyzed the effect of this learning and report the results below.

Results

RUPART, in its current form, is able to navigate reliably from a starting point to locations halfway across the length of its domain. Given goals that are further away, limitations of current localization often cause the robot to lose its way. More importantly than its surface behavior, the underlying integrated hybrid architecture performs at a high level, even given the tiny initial case memory we have tested. As such, RUPART demonstrates the feasibility of case-based reasoning for creating an adaptive robot controller of less complexity than the typical hybrid controller.

We systematically analyzed two aspects of the RUPART system: the quality of adapted route plans and the quality and consistency of cases retrieved over a range of test runs. The analysis indicates the importance of a larger case memory than RUPART has been tested with, but also demonstrates great consistency among the cases retrieved during multiple test runs.

Route plan quality

The initial case memory contains twenty route cases. These cases were generated using the underlying graph-based path planner, with randomly-chosen start and goal locations. We knew the case memory was likely inadequate, but used it to explore the limits of the case-based route planner.

The quality of cases produced by the case-based planner was tested by retrieving and adapting routes from a single starting point (the robot's home lab, for simplicity) to 100 randomly generated goal locations. These routes were compared against optimal routes we computed by hand. The robot's navigation component was disabled for this test, only the routes were generated. This test was repeated with and without case learning. From 100 random locations, 56 to 58 unique goals were created. Without exception, the retrieved case was judged to be the closest case in memory to the given problem.

Without case learning, optimal or near-optimal results occurred 88% of the time (51 of 58 tests). However, 12% of the routes generated were highly inefficient, moving from one hallway to another through one classroom, then back to the first hallway through another. All but one of the seven poor routes were based on a single case in memory that was simply not a very close match to the problem for which it was used.

With case learning enabled, 56 unique goal locations were generated from 100 random tests. As might be expected, the holes in the initial case memory were amplified when learning was included: poor quality cases were generated and stored in the case memory. Still 86% of the routes generated were optimal or nearly optimal.

We compared the work done during adaptation with and without case learning, with unsurprising results. We examined the number of plan steps that were generated in order to adapt a retrieved route, and found that 70% of the adapted cases with learning required no more than 2 additional plan steps, while only 45% of the no-learning cases could be adapted with 2 or fewer additional steps.

Case retrieval quality

To examine the quality of retrieval of both route and behavior cases, we ran multiple test runs to three different, achievable goal locations. These tests used the full system, including real world navigation by the robot. In each case, we generated five successful runs to each location. The analysis of retrieval quality is necessarily subjective: each case retrieved was labeled as optimal, near-optimal, fair, or poor, based on where the robot was when the case was retrieved.

Current and average sonar readings inject quite a bit of variability into the retrieval indices in RUPART. We expected to see quite a bit of variability in the cases retrieved. While some variability existed, the results were surprisingly consistent across multiple runs. At each point in the navigation process to a particular goal, two or three behavior cases predominate.

Across all fifteen runs, 58% of the behavior cases retrieved were judged to be "optimal," 27% were "near-optimal." Even the "fair" choices had reasonable explanations: for example, RUPART retrieved cases from cluttered environments when the robot was too close to a wall. Because of this, no retrievals were judged to be poor. The performance of the system under the merely fair cases remained robust, as no case controlled the robot for more than a few seconds without being re-evaluated.

Overall, the CBR system works well, given the limited size of our initial case memory. No tweaking of similarity measures was needed for RUPART to retrieve route cases when most appropriate, and behavior cases when most appropriate. The robot navigation is robust under the current behavior case retrieval, and flaws in either route or behavior cases can be addressed by starting from a more complete case memory.

Conclusions and Future Work

RUPART is a robot control architecture that is both hybrid and a single mechanism at the same time. Behavior-based control and strategic route planning work together in the current system, both controlled by a single underlying CBR system. The resulting system is less complex, balances itself between reactive and deliberative tasks, and incorporates learning through a single basic mechanism.

At the current time, we are working to improve and extend RUPART's case memory, both through more careful initial creation and through route and behavior learning. Improved localization will enable a broader range of tasks for RUPART. In the near future we will be extending the range of behaviors encoded in behavior cases, and reducing the non-CBR elements of the system. We plan to extend the system to include goal management and meta-plans for guiding the creating of cases. This extension will provide opportunities for additional kinds of cases, a richer set of indexing features, and a more challenging task for the robot.

Acknowledgments

Peter Anderson-Sprecher's participation in this project was funded by the Keck Student-Faculty Collaborative Research Program at Macalester College, during Summer, 2005.

References

ActivMedia Robotics, L. 2005. Robot software: Tools and applications. Retrieved, Nov. 18, 2005.

Blank, D.; Kumar, D.; Meeden, L.; and Yanco, H. 2003. Pyro: A python-based versatile programming environment for teaching robotics. *J. Educ. Resour. Comput.* 3(4):1–15.

Chagas, N. C., and Hallam, J. 1998. A learning mobile robot: Theory, simulation and practice. *Lecture Notes in Computer Science* 1545:142–154.

Ferrein, A.; Fritz, C.; and Lakemeyer, G. 2005. Using Golog for Deliberation and Team Coordination in Robotic Soccer. *KI Künstliche Intelligenz* (1).

Fox, S. 1995. *Introspective Reasoning for Case-Based Planning*. Ph.D. Dissertation, Indiana University, Computer Science Department. IUCS: Technical Report 462.

Fox, S. E. 2000. A unified cbr architecture for robot navigation. In *Advances in Case-Based Reasoning, Lecture Notes in Artificial Intelligence 1898*. EWCBR 2000.

Fox, S. E. 2001. Behavior retrieval for robot control in a unified cbr hybrid planner. In *Proceedings of the 14th International FLAIRS Conference*. Florida Artificial Intelligence Research Society.

Goel, A.; Callantine, T.; Shankar, M.; and Chandrasekaran, B. 1991. Representation, organization, and use of topographic models of physical spaces for route planning. In *Proceedings of the Seventh IEEE Conference on AI Applications*, 308–314. IEEE Computer Society Press.

Haigh, K., and Veloso, M. 1995. Route planning by analogy. In *Proceedings of the First International Conference on Case-Based Reasoning*. Sesimbra, Portugal: Springer Verlag.

McGinty, L., and Smyth, B. 2001. Collaborative case-based reasoning: Applications in personalised route planning. In Aha, D. W., and Watson, I., eds., *Case-Based Reasoning Research and Development (ICCBR-01)*, volume 2080 of *LNAI*, 362–376. Berlin: Springer.

Ram, A., and Santamaria, J. C. 1997. Continuous case-based reasoning. *Artificial Intelligence* 90(1-2):25–77.

Ram, A.; Arkin, R.; Moorman, K.; and Clark, R. 1992. Case-based reactive navigation: A case-based method for on-line selection and adaptation of reactive control parameters in autonomous robotic systems. Technical report, Georgia Institute of Technology, Atlanta, GA.

Simmons, R. G.; Goodwin, R.; Haigh, K. Z.; Koenig, S.; O'Sullivan, J.; and Veloso, M. M. 1997. Xavier: experience with a layered robot architecture. *SIGART Bull.* 8(1-4):22–33.