AN AGENT-BASED APPROACH TO INTRODUCTORY ROBOTICS USING ROBOTIC SOCCER

John Anderson and Jacky Baltes
Autonomous Agents Laboratory
Department of Computer Science, University of Manitoba
Winnipeg, Manitoba, R3T 2N2, Canada
andersj,jacky@cs.umanitoba.ca

Abstract

Introducing robotics to undergraduates is a challenging problem from an educational standpoint. The overlap to numerous areas of applied computer science, and the number of difficult sub-problems that must be dealt with in any non-trivial robotics application (e.g. vision, path planning, real-time computing) easily overwhelms students at the undergraduate level. We have been employing robotic soccer as a vehicle to motivate undergraduate students in robotics, in order to provide students with an interesting domain that embodies significant research challenges in artificial intelligence. In order to manage the complexity of the domain, we employ software agents in several key components in our approach, and as a methodology for student implementations. This paper describes our strategies for introducing the elements of soccer-playing to undergraduate robotics students, and the agent-based approach we employ.

Key Words: Agents, Robotic Soccer, Behaviour-Based Agents, Robotics and Education

1. Introduction

Introducing robotics to undergraduates is a challenging endeavor. While there are many commercially available robotic kits targeting non-experts and even children, many of the more interesting aspects of the area (e.g. vision, control, navigation) require significantly more complex problems than these kits are typically capable of dealing with. Introducing robotics to undergraduates using a real-world problem brings the benefit of seeing the many areas of computer science and engineering to which robotics relates brought together, and gives an appreciation of the numerous problems that must be dealt with in any non-trivial robotics application. However, this brings with it an enormous pedagogical challenge: how can the complexity of a real-world problem be managed in such a way to expose students to it gradually? This has a host of related issues, from managing physical infrastructure,

to ensuring that topics can be focussed on in isolation while still eventually completing the entire complex application.

We have been using robotic soccer as a vehicle for introducing undergraduates to robotics. This is a highly complex domain that is increasingly viewed as a standard challenge problem in robotics and artificial intelligence (AI). From a research standpoint, this allows researchers to have a common grounding for vision and control, and a common set of conventions by which to judge performance. From an educational standpoint, it is wonderfully motivating for students, and also provides a context to which they can relate physically, allowing them to more easily visualize the problems a robot encounters. The use of this domain also provides motivation for future work in that there are numerous competitions students can attend should they wish to pursue their efforts further.

Robotic soccer has proven to be an important challenge domain for research. Like many others, we were initially attracted to the domain as a vehicle for pursuing grounded research where all aspects of a problem had to be considered, from sensing and control to planning and multi-agent interaction. Beyond this, robotic soccer has also proven in our experience to be a very useful educational environment as well, in that it allows the grounded introduction of important problems in AI, real-time systems, and software engineering [1]. Students are highly motivated by the action associated with soccer and the ease with which the physical nature of the domain is understood. The fact that the domain is at the forefront of modern AI research also makes students feel that they are part of a larger effort, and we find students are strongly research-motivated by their exposure to the problem.

The course which embodies the approach described here is a one-term (three month) classroom and laboratory-based course. Most students are in their fourth year, but we typically have several advanced third-year students as well. The course is presented as part of a program in computer science, and while there are normally several students from electrical and computer engineering, the course must be designed for those with some sophistication in programming and appreciation of systems issues, but little experience with electronic circuits and hardware design, and no prior exposure to control theory. The short time duration of the course adds to difficulty of properly managing the gradual introduction of robotics sophistication, and also places some limits on the goals that can be expected in the code that students generate. It is not possible, for example, to produce a competitive robotic soccer team given the introductory nature of this course. However, it is perfectly reasonable to expect to produce a team that can play soccer and demonstrate soccer skills. In the past, a second three month of work on the students' own time has resulted in a team with skill sufficient to qualify for international competitions such as RoboCup [2] or FIRA [3], but this depends on the particular students' abilities and ability to commitment to further work.

The number of students that can ultimately be supported is limited by the number of robots available and the size of the laboratory facilities. We have typically had 15-20 students divided ideally into groups of 4. Smaller groups are discouraged because the programming involved can become overwhelming, and because of the advantage of a good mix of skills between group members. With groups much larger than this, on the other hand, we find that all students are not exposed to all aspects of the problem.

In order to use robotic soccer for undergraduates, several interacting problems must be managed by the instructors. First, the hardware employed must be inexpensive enough to

afford a sufficient number of robots, and must be supportable by the students themselves after a short learning curve. The latter is required in order that students can work on their own time if they wish without the need for constant laboratory support.

The first step we take to manage complexity is through the use of global vision (that is, employing an external camera as a vision source for all robots, rather than using individual cameras). The use of an external vision system allows the issues surrounding computer vision to be demonstrated at a more abstract level, and removes a significant amount of code (and low-level course material) that would otherwise be overwhelming to undergraduates. It also makes the hardware required significantly less expensive and sophisticated, since each robot does not have to carry enough computer power to perform real-time vision. Similar decisions must also be made for other sub-problems that, while important, overcomplicate the introduction of robotics to inexperienced students.

We aim the materials for this course (other than general laboratory computers) to cost less than US$1000 in total, which is a reasonable expense for most universities. We have in the past brought teams to RoboCup that were based on a similar budget, a formidable task given that a typical RoboCup small-sized league team employs robots that cost US$3-5,000 each. Working with a limited budget requires some sacrifice in hardware performance: such inexpensive platforms will not have the omnidirectional drives, dribble bars, or the powerful kicking mechanisms that are typical of RoboCup robots. However, such mechanisms are not necessary from an educational standpoint, and can detract from a focus on AI. It is interesting to watch a robot perform feats such as propelling a ball hard enough to score a goal from any point on the field, for example, but such a skill does not enhance a focus on strong path planning or good teamwork. Equally, human soccer would not be especially interesting if players had such overwhelming physical skills. Limiting robots to more basic physical abilities forces a focus on building systems that rely on better intellectual skills, which is our main goal as AI researchers [1]. This is the main reason we tend to focus on hardware parsimony both in education and in our research.

In attempting to manage the complexity inherent in introducing undergraduates to robotics using a domain such as robotic soccer, we have found the use of software agents to be of great assistance. We employ software agents not only in the physical infrastructure supporting robots, but as a development medium to allow the gradual introduction of material and the associated implementation of that material. This paper describes our overall approach to introducing robotics using this methodology. We begin by describing the overall architecture employed and the role of software agents in this approach, along with typical choices for hardware and other infrastructure. We then describe our strategy for guiding students through the complexity of robotic soccer, through the development of increasingly complex software agents to control mobile robots as students expand their knowledge and experience.

2. Overview of Approach

Students begin by getting acquainted with the laboratory equipment they will be working with. A number of different possibilities for suitable equipment exist that satisfy the motivations outlined in the previous section. It is possible to build simple differential drive-

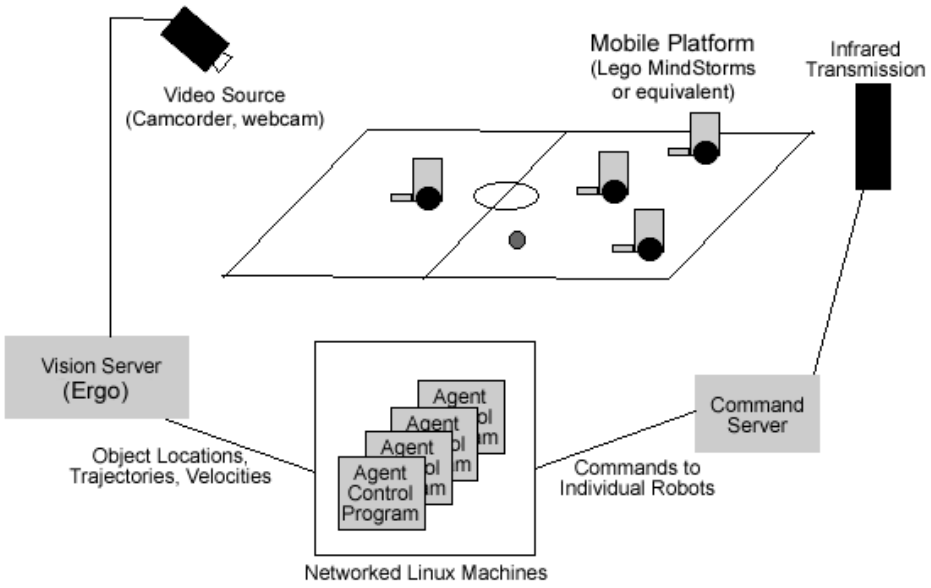


Figure 1: System Overview.

based platforms carrying small microcontrollers from scratch, and we have done this in the past [4], but in general it is simpler to work with commercially available hardware. The most well-known inexpensive platform suitable for this work is the Lego MindStorms RCX, while KAIMax's KRC controller provides a slightly more powerful platform. Each of these is essentially a microcontroller with modular motors and an external communication system. While some onboard computation is advantageous, it is also possible to employ infrared or radio based remote control (RC) toys for mobile platforms as well. While these are traditionally thought of as larger cars, the newer miniaturized toys (in the 1-3" range) have the advantage of allowing soccer play on a very small sized and portable field. Given the choices available in North America, the MindStorms platform is the easiest to begin with: these are readily available as well as robust (an issue that often arises with RC toys).

None of these platforms have significant local computing power (none in the case of an RC toy). Because we work with software agents controlling these mobile platforms, local computational power is not an issue, nor is there the necessity to continually write code on one platform, cross-compile it, and download it to a mobile platform repeatedly every time a change in robot behaviour is required. Software agents controlling the robots can consume as much computational power as an external computer can provide, allowing more sophisticated agents to be developed, and can be developed using the traditional tools and platforms that undergraduates are experienced with. In the case of Lego MindStorms, we restrict local computation to the lowest level control (turning motors on and off to orient and move the robot), and this is handled by a customized version of BrickOS (an open-source alternative operating system for the Lego MindStorms RCX).

Whatever mobile platform is chosen, these units fit into the overall system architecture depicted in Fig. 1. A set of software agents controlling the robots on the field lies at the

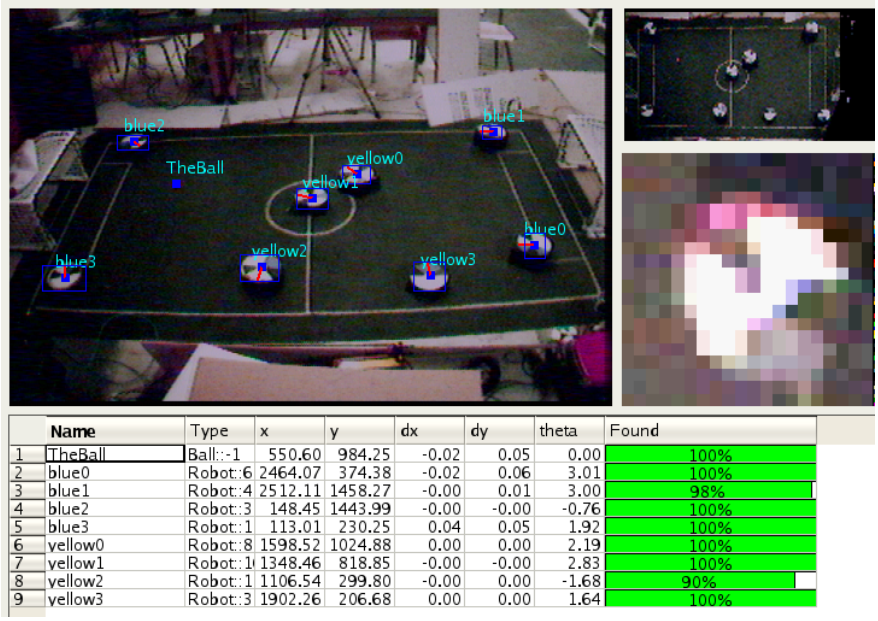
heart of the approach, with the other components used to either inform these agents or communicate their choices for action.

A global vision approach is employed, not only to alleviate the complexity of vision from the students' standpoint, but to better isolate vision from the rest of the system and simplify the agents controlling the robots. While local vision could certainly be used, it is much easier for students to be able to assume a standard set visual descriptions that will be supplied in an easily parsable format. Vision in our approach requires only a single video source (we currently use a commercial security camera that costs US$50), connected to a machine with a video capture device (tv tuner card). This machine runs our open-source vision server software, Ergo [5, 6]. The most significant feature of this software from the standpoint of undergraduate education is that it can be quickly calibrated to track objects from any camera angle. This brings great flexibility to the vision-processing aspects of this problem, because the camera does not have to be mounted perfectly overhead. It also means that a single camera can be used for both the tiny fields that would be used by miniature RC platforms, as well as those required by larger robots.

Ergo is shown in operation in Fig. 2. The system interpolates the oblique image to reconstruct an overhead view (upper right), and uses a combination of motion-detection, position prediction between video frames, and a special marking pattern for robots to be able to track robots on the field (bottom) without requiring predefined colors. Additional details of the process by which this occurs may be found in [5]. Because colors do not have to be calibrated, the system is simpler to use and tolerates a wider range of lighting conditions. The system requires less recalibration over time, and is easily supported by the students themselves after a short time using the system. As can be seen in Fig. 2, the current identity, location, orientation, velocity, and confidence of tracked objects on the field is recorded, and this is transmitted in ASCII via UDP broadcast over Ethernet in a single package for each video frame processed. The information about each object can be received by individual software agents controlling the robots, potentially at various network locations, providing each agent with a description of the current state of play.

Software agents controlling the soccer playing robots are distributed across a network of clients (Linux workstations in our laboratory, but these can be running any OS capable of TCP/IP transmission). The extent of this network depends on the specific situation. In a classroom setting it may be amenable to have each agent run on an individual workstation, to allow students to create individuals that make up a team. In our situation, we normally have groups make a complete team, and have a team run on one or more workstations, while there may be situations in which only a single agent machine is available. This approach has been used for all of these (though a separate machine for the vision server is advisable due to the resources it consumes). One of the nice features of controlling robots with separate software agents is this natural distribution of computational resources. Each agent, irrespective of where it is hosted on the network, picks up the visual description packets broadcast by the vision server, and asynchronously acts on them.

Just as individual agents must perceive the world independently, some mechanism must also exist for these to commit to actions independently. Here there is a tighter connection to the physical platform chosen, because of restrictions that may exist on communication between the robotic platforms and host computers. If it is possible to have each robot



	Name	Type	x	y	dx	dy	theta	Found
1	TheBall	Ball::-1	550.60	984.25	-0.02	0.05	0.00	100%
2	blue0	Robot::6	2464.07	374.38	-0.02	0.06	3.01	100%
3	blue1	Robot::4	2512.11	1458.27	-0.00	0.01	3.00	98%
4	blue2	Robot::3	148.45	1443.99	-0.00	-0.00	-0.76	100%
5	blue3	Robot::1	113.01	230.25	0.04	0.05	1.92	100%
6	yellow0	Robot::8	1598.52	1024.88	0.00	0.00	2.19	100%
7	yellow1	Robot::1	1348.46	818.85	-0.00	-0.00	2.83	100%
8	yellow2	Robot::1	1106.54	299.80	-0.00	0.00	-1.68	90%
9	yellow3	Robot::3	1902.26	206.68	0.00	0.00	1.64	100%

Figure 2: Ergo in Operation.

operate on a different radio frequency, for example, N communications channels could be opened simultaneously for N robots. In many simple systems this is not easily done, however. Commercial RC toys tend to operate on only a small set of frequencies, while the IR communication used in Lego MindStorms employs one infrared signal for all robots. To deal with this, we use another software agent, the command server (Fig. 1). Agents individually commit to actions and send a description of this action out on the network, analogous to the operation of the vision server. The command server agent watches the network for these, intercepting such action descriptions and grouping them with those it has recently received from others. At regular intervals, these grouped commands can be broadcast via infrared to the robots on the field. Since agents controlling the robots are operating asynchronously, it is possible that during an interval an agent produces no command for its robot, in which case a no-op is generated so that each robot is represented in the transmission.

In the case of Lego MindStorms, this also requires the batched commands for all robots to be transmitted as frequently as possible over a slow (2400 baud) transmission channel. We thus attempt to limit the volume of information that is transmitted, by employing a 10-byte packet consisting of 8 bytes of data along with synchronization and checksum bytes. This will allow eight robots on the field (two teams of four) to receive one byte each. This can be adjusted upward and still maintain a reasonable broadcast frequency, but we have found one byte to be more than sufficient for the granularity of control possible with Lego MindStorms. Four bits can be used for orientation, allowing 16 different turn settings, and and a similar set of values for forward/reverse speeds.

When using Lego MindStorms, we have also introduced some additional custom-designed hardware for convenience. The standard IR towers, we have found, do not provide nearly the effective transmission distance that is claimed, especially under the bright lighting that

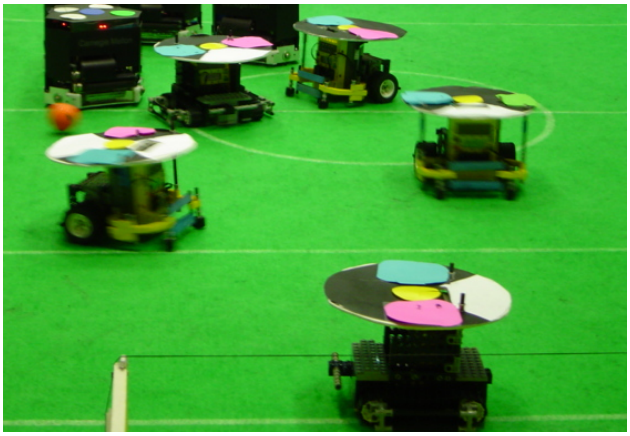


Figure 3: Examples of Lego MindStorms robots, RoboCup-2003, Padova

is typical of competition environments. We replace these with our own more powerful IR transmitter. While not strictly necessary, it does assist in avoiding "dead" areas on the field where robots cannot be reached.

One of the additional advantages of the MindStorms and KRC platforms is the ability to design various types of robots from the same basic pieces. As part of our course, students spend some of the early stages designing and building robots - a robot is typically produced quickly, and then altered as the idiosyncracies of control are better understood. The two types of designs that are generally arrived at are illustrated in Fig. 3: simple wheeled robots most that tend to be best employed as a striker, and treaded robots commonly employed as goalkeepers.

The system organization described here is intended to make it easy to manage the hardware and software necessary to support a robotic soccer team, and to provide reasonable constraints over what students must develop. We have previously advocated this general approach [7] and demonstrated its feasibility in the RoboCup E-League in 2004. The isolation of the video server and the use of a command server agent to collect asynchronous control agent commands for synchronous broadcast form a set of boundaries within which the students' work takes place. The students are responsible for designing software agents that perceive using the vision server, and act by communicating to the command server, and at the outset can consider the rest of the system as a black box.

At the same time, this is a sophisticated collection of hardware and software components, and the system as a whole cannot operate if any of these components is malfunctioning. There will always be physical issues with this approach (e.g. the video server losing an agent due to changing lighting changes) that are beyond the control of the students' software, and will require periodic recalibration, resetting, or repair. We have attempted in this design to produce an overall system that the students can maintain, for the most part, after some experience is gained. We find that after only a few weeks working with the equipment, the students can deal with the settings and calibration of the vision server, and can also fix minor problems with IR communication and other elements of the overall system themselves.

3. Agent-Based Control

From the students' point of view, developing a robotic soccer team involves designing a set of interacting agents that perceive the field via the vision server, and act by commanding individual robots on the field. The course begins by preparing them to be able to create the most basic features of such agents. We have never employed a textbook, as we have yet to find one that is strongly suited to this approach: the many good overview texts, such as Murphy's [8], tend to cover too little of the implementation details to be useful for actually building robot controllers as opposed to understanding the principles behind them. Instead, we prepare students by giving them original research papers describing the techniques they are employing. These are not always an easy read for undergraduates, and so extra work must go into highlighting or otherwise pointing out the particular elements that are most important to what they are currently working on. While this is still a challenge for the students, we have not found that it is beyond the capacity of undergraduates.

Robotic soccer is interesting precisely because it is a problem where there is a significant range of granularity that must be dealt with, from fine-grained motor control to high level path planning and multi-agent interaction. In order to deal with this breadth, the course must proceed so that the most basic facilities are understood (and developed) first, with later topics building on what the students have previously learned. We have found the concept of a software agent to be ideal in managing this complexity, in that very simple agents can be developed, and the facilities those agents provide subsumed by more complex agents.

While covering basic control approaches (e.g. PID controllers) in class, students apply these ideas by considering the actions their robots will be expected to perform at the lowest level (e.g. forward and backward motion, turns), and how this information could be conveyed to their robots via the command server. Each group is expected to design its own encoding scheme (such as that described previously) to fit these commands into the one byte they are allotted. Using this scheme, they implement a basic interface that allows them to show the position of their robot on the field by plotting vision server information, and control their robot remotely without being able to see the field itself. This gets students thinking about the concept of parsimony (in this case, in both robot design and the encoding of basic operations), as well as interfacing their code with the vision and command servers.

From the standpoint of good design, this encoding scheme represents the first choice students must make that impacts future design choices. We emphasize that previous decisions can be changed at any time, and such changes do occur regularly. Emphasizing the impact that students' choices have on their later code helps the students focus on the overall design process, and ultimately serves as an excellent lesson in practical software engineering. This implementation (like the rest of the implementation work in this course) is performed in C and C++ under Linux, using whatever tools the groups desire.

In completing this assignment, the students implement the infrastructure necessary to interface a controller agent (themselves in this case) with robots on the field, and gain enough of an understanding of the overall architecture depicted in Fig. 1 to begin designing their own agents, while at the same time, covering enough control material in class that they can begin work on a simple autonomous controller.

In the remaining course work, students develop increasingly sophisticated software agents

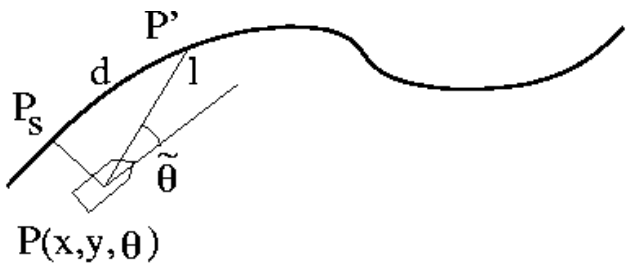


Figure 4: A look-ahead controller.

that will use their encoding mechanism to drive a robot and ultimately get a team of robots to play soccer. We divide the sophistication of such agents into four levels of abstraction. At the most basic level, agents must be able to follow a path to a specified goal, embodying basic motion control and goal-directed behaviour. Above this, agents must be able to generate paths to goals in order to be able to follow them. The basic concepts of soccer must be built into agents so that they can form appropriate goals, and finally, the agents must be able to function coherently as a team. The following sections detail the agents that students develop to satisfy these levels of abstractions, and the educational process framing this development.

4. Motion Control

The motion control approaches covered in lectures include Balluchi's sliding mode controller [9] and Egerstedt's look ahead controller [10]. To embody these in an agent architecture, students are introduced to the basic elements of reactive agents and the ideas behind behaviour-based approaches [11]. The students' first autonomous agent is a look-ahead path tracking agent that will allow the robot to follow a pre-specified path on the field. In order to represent a path, points on the playing field are marked (literally, with markers that can be picked up by the vision server and reported to agents), and the agents are responsible for drawing an imaginary path between these markers and issuing appropriate motor control commands to follow the path, using their own position as reported by the video server for feedback. The look-ahead controller required is illustrated in Fig. 4. Given the agent's current position $\mathbf{P} = (x, y, \theta)$, the agent calculates the closest point on the path $\mathbf{P_S}$. Given its current velocity and distance to the path, the agent calculates an approach point $\mathbf{P'}$ – that is, a point projected ahead of the robot by looking ahead distance d. The look-ahead distance d is determined by the velocity as well as the distance of the agent to the path.

The approach line l, that is the line from the agent to the approach point $\mathbf{P'}$ is used as reference and the orientation error $\tilde{\theta}$ of the robot with respect to the approach line l is calculated. Given the orientation error $\tilde{\theta}$, a steering angle is computed. Finally, the calculated steering angle is used to determine the desired velocity of the robot.

This approach to path tracking has proven to be robust and efficient, embodies many of the basic concepts behind path tracking, and is reasonably straightforward to implement. It is also easy for students to tune the controller by varying the lookahead distance and gain

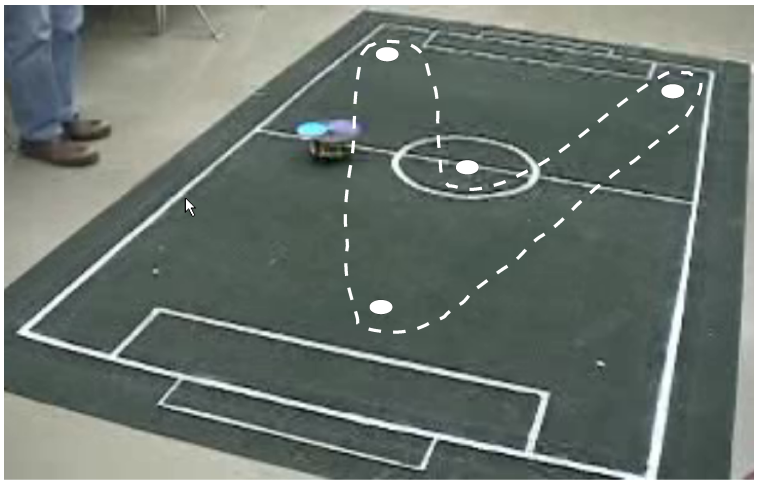


Figure 5: Video capture of a robot following an (imaginary) path based on field features.

on the steering angle control. This allows them to easily change the behaviour of the agent in future to follow a path more or less closely (e.g. when moving through a tight spot vs. through open space). That is, it forms a solid basis for higher-level agent control.

An important component in our approach is motivation and performance comparison through open competition. As each agent is completed, student groups present their agent and a competition ensues that allows everyone to see how the design decisions and implementation choices of each group fare against one another. In this challenge, the path tracking agents are tested in turn on a field marked by spots that form an irregularly shaped track, as shown in Fig. 5. Each trial consists of a timed run over five laps of the track, with penalties each time one of the spots forming the corners of the track is not traversed. This provides a quantitative basis for performance comparison of implementations, and also allows students to see the results of tuning the controllers in different ways. It serves to emphasize the concept that robots must operate and be judged in the settings for which they are designed, usually by a specific deadline. It also ultimately demonstrates that parsimony is a worthwhile goal, since students that are tempted to build sophisticated variations on these controllers often do no better than a well-tuned basic controller.

We later cover more sophisticated controllers (fuzzy logic [12] and machine learning approaches [13, 14]), and students are free to use these more advanced techniques as a basis for more sophisticated agents, or those they employed here. Students typically experiment extensively with their controllers as their agents become more sophisticated, and often end in employing some form of hybrid control (e.g. a fuzzy logic controller providing input to a PID controller).

5. Path Planning and Real-Time Obstacle Avoidance

Having built a functioning path following agent, students can now begin considering the factors involved in getting to desirable places on the field: constructing useful paths and dealing

with the fact that obstacles may arise while following these. This is a much more sophisticated problem than they have been exposed to thus far, but students have the confidence to tackle it as a result of the working agents they have already constructed.

We consider obstacle avoidance and path planning as two different but interacting issues. The basics of dynamic obstacle avoidance are covered in lectures while students are building their path following agents. Given that we are using a behaviour-based approach, the concept of obstacles provides a nice example of more complex behaviour-based activity: following a path while avoiding obstacles through the use of potential fields [11]. Having already built a working path tracking agent, the concept that their controller values must be influenced not only by where the lookahead value, but through a combination of the potential fields generated by any obstacles, is a very natural one. Students begin by integrating first static and then dynamic obstacles into their existing path tracker. Since students already use feedback from the vision server to determine their robot's position on the field, recognizing obstacles (paper markers placed on the field) via the vision server and avoiding them is a natural extension. Static obstacles are placed before the robot begins moving, while dynamic obstacles are implemented by simply moving the paper circles around the robot's path while the robot traverses the field, to mimic the movements of teammates and opponents that will be considered in future.

While this development ensues, lectures cover path planning mechanisms. We introduce path planning as primarily the decision of how to functionally represent the space around the robot, with each representation leading naturally to methods for creating plans. We begin with gradient-descent methods, because of their simplicity and direct connection to prior material on obstacle avoidance. We then cover skeletonization methods, focussing on visibility graphs and Voronoi diagrams [15], where obstacles to be avoided form points on the vertices of a graph whose edges can form path segments. Following this, we cover cell decomposition methods, including quad-tree decomposition and binary space partitioning [16, 17]. Cell decomposition methods are similar in that they recursively divide space into regular units, represent these internally as a tree with adjacency links, with cells labelled free, blocked, or a mixture of both. Recursive partitioning stops when there are no mixed units or some minimum cell size is reached. Path planning in these approaches is then demonstrated as a traversal of this tree structure to link open areas until a path is created.

In addition to these, we also cover a cell decomposition method of our own, Flexible Binary Space Partitioning [18], which involves using binary space partitioning without the assumption that each new decomposition must result in regular-sized cells. The entropy of the space that would result is used to choose partition points in this approach, resulting in a more effective spatial representation. Covering this method is not intended to showcase our research, but to get students thinking of these algorithms as tools that can be modified and improved.

In implementing a path planner, students must first consider the details of the world model from which their agents will operate. This is a simpler problem to consider under global vision, because the vision server identifies objects of interest across the entire field. Like any other sensor, however, the vision server produces some noise, forcing students to intelligently process messages the vision server transmits. For example, if a robot is occluded by the angle of the camera, and temporarily lost from the vision server, their world model

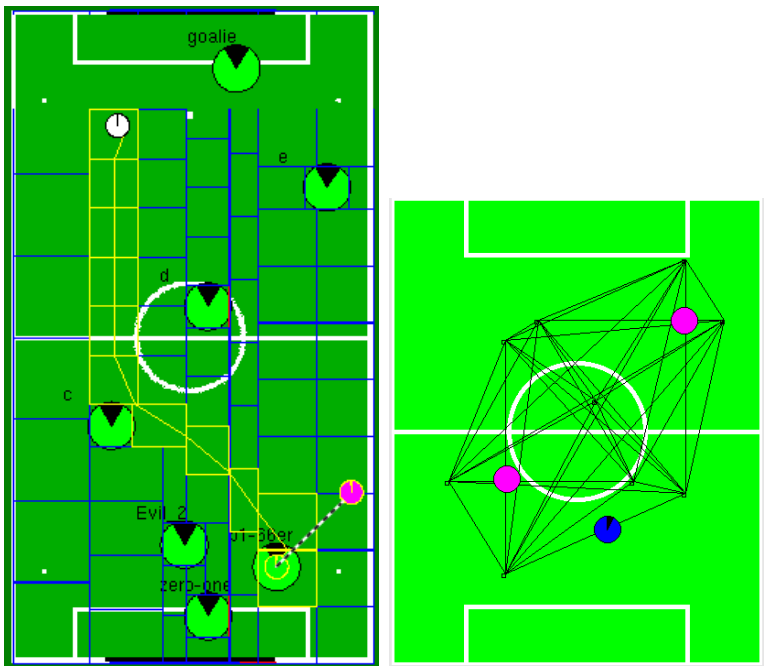


Figure 6: Path planning using using Flexible Binary Space Partitioning (left, with a dashed line emanating from the robot illustrating the influence of the recognized obstacles' potential fields), and using a Voronoi diagram (right).

simply cannot have the agent cease to exist.

After considering a world model, groups are free to choose any of these approaches to performing path planning. The widely different implementations that result have an interesting effect on the learning process. Students become much more motivated about comparing their work to others as a result of these differences, and begin to focus on the trade-offs they see between approaches in terms of portions of the problem that are simpler and more difficult, and more and less effective in practice. For example, while the complex linking that is involved in most cell decomposition mechanisms is initially frowned upon, the value that can be achieved through better paths in the end has led students to change their focus to these approaches in mid-implementation.

This is also the first time that an agent is directly subsuming previous work, in that the planned paths must also be followed, and so students also see the effect of good (or poor) choices they made earlier, a vital lesson from a software engineering perspective. They also extend previous agent work in adapting their earlier interface to display the field and the obstacles on it, as well as the current path. The left side of Fig. 6, for example, shows the output of a flexible binary space partitioning path planning agent, while the right illustrates an agent using a Voronoi diagram, where vertices are chosen by taking the midpoint between each visible obstacle and the wall(s) near it in order that a resulting paths do not stray too close to obstacles.

After students are satisfied with basic path planning, the problem of dynamic path planning is considered. This in part involves integrating their previous work on obstacle avoidance

(the influence of obstacle avoidance on a planned path is shown in the left side of Fig. 6), but also forces students to consider re-planning. While the mechanics of this are simple (calling the planner again from the current position), students quickly recognize that this will need to be done often, and consumes enough resources that it cannot simply be run constantly. Students thus begin to consider how different a field must be in order to make re-planning necessary.

This completes the third assignment, which culminates in another competition. This consists of a race of five passes from one goal to the other across the field scattered randomly with a number of marked obstacles, a specific number of which are moved randomly after each pass. The nature of this challenge forces the use of re-planning while keeping the demands on the planner simple: a plan must be re-created only once per pass to reflect the obstacle changes. Students recognize, however, that their planners will ultimately be called much more often, as a fast moving ball rapidly changes the state of the game and with it the agents' goals.

6. Demonstrating Soccer Skills

To this point, we have implemented no functionality particular to playing soccer. Indeed, students now have a very versatile agent that could be adapted to numerous environments, all involving putting basic skills together and giving the agent a reason to be at particular places under different circumstances. In order to move to a soccer-playing agent, students must consider how to put the abilities their agent has to use in particular context. This inevitably involves including transitions from one type of behaviour to another (e.g. moving in for a goal kick by getting behind the ball and then projecting it), which in turn necessitates consideration of more sophisticated agent architectures.

To support this, lectures focus on covering a range of agent architectures. We focus on those that are likely to be useful in a complex and fast moving application like soccer: the subsumption architecture [19], as well as behaviour trees and other forms of networked behaviours [11, 20, 21]). In the interest of breadth, we also cover some architectures that students are less likely to make use of in this setting, such as BDI architectures [22].

The goal of the next assignment is to build two more sophisticated agents: one that will play as a soccer striker, and the other that will play as a goalkeeper. Here students begin to see the differences between architectures as they appear in publication and as they used in practice: they learn that architectures are always described as very elegant and carefully arranged, but that problems tend to arise when following any rigid architectural standard in the real world. Because no real-world problem ever perfectly fits one architecture, students also learn more about design trade-offs, as they must consider violating the principles of an architecture in order to get something to work better or more easily, as well as the impact of this on maintenance and future development.

Because of the extensive work required to make large-scale architectural changes after code has been written, students are told that the agents they are developing at this stage are likely too complex to easily be implemented using a *pure* subsumption-based architecture, and are pointed toward other forms of connected hierarchical behaviours. Many groups

use behaviour trees directly, but other forms of behaviour assemblages adapted from those described in [11] are also seen.

Whatever their architectural choice, students begin by considering the low level behaviours of each agent, and from these the kinds of perceptions the agents are interested in. It is easy for students to see that a goalkeeper is interested the proximity of the ball and agents that are closely placed to it, for example. From there, students consider how those should lead to good choices for actions, and how these behaviours need to be stimulated by perceptions (e.g. for example, a striker, upon seeing the ball, likely wants to get behind it to capture it). Methods of encoding these in the architecture of their choice and linking these to path planning and following are then considered.-

Similarly, students must consider obvious goal-directed behaviours by laying out what kinds of goals an agent should be considering, and forming a second layer to encompass a context from which lower-level behaviours can be invoked. For example, areas of the field that are considered useful for scoring goals, and contexts under which an agent should perform a goal kick or pass the ball to another player. These two layers are simple, but can capture many of the basic soccer skills while serving as an example of how assemblages of behaviours can be put together.

The concept of re-planning also comes into play more directly here, since new goals (adopted by goal directed behaviours or perceptions) lead to changes in already-planned paths. Students must begin to consider the price of significant re-planning, and ultimately some constraints on adopting new goals arise in light of this. The initial response is typically to decide some suitable frequency with which behaviours are re-evaluated in their architecture. This is sufficient at this stage, but must be re-considered when playing a real soccer game.

Since each group has created both a goalkeeper and a striker, we demonstrate the abilities of these agents competitively by demonstrating the skills necessary to play one-on-one soccer. We employ three different events of increasing complexity. First, each team's striker attempts to score as often as possible on an empty net over a specific time, with the ball returned to a random place on the field after each goal. This demonstrates basic coordination skills (getting behind the ball to capture it, moving with it to a feasible shooting position and attempting to score). This is then run with a goalkeeper.

The third challenge involves passing: two strikers attempt to pass the ball back and forth between them as many times as possible within a specific time period (where a successful pass is one where the ball stops within a given diameter of the receiving robot). Since agents do not yet explicitly interact, this challenge serves to stimulate students' thinking about the requirements for successful multi-agent interaction.

These three events are some of the same challenges used by organizations such as RoboCup and FIRA to evaluate the basic skills of submitted competition entries. While they do not in themselves demonstrate a successful soccer team, they form a strong foundation for one. Conversely, if these skills are not performed well, there is little hope that team organization alone will be enough to produce reasonable play.

7. A Soccer Team

The final stage involves moving beyond individuals by extending the behavioural architecture implemented thus far to the collective behaviours of an entire team. As part of this, students typically re-evaluate all their earlier decisions, since all of these will directly affect the ability to play good soccer. Changes at this stage are often made to the basic low-level skills just completed (since something as simple as poor agent placement behind the ball when attempting kicks, for example, can have a significant effect), as well as to their lower level path planning and following implementations.

The first extension necessary for team play is greater sophistication in an agent's representation of the world. Since the world changes rapidly as the game unfolds, students must consider some type of confidence-based model of the positions of the objects around them. Other factors must also additionally be tracked for good play – which player is closest to the ball, the number of players on each team in various areas of the field, etc. – some of which are realized in early planning stages at this level, while others are discovered as a result of faulty play.

Since so much of good behaviour selection involves the appropriate recognition of context, students are shown how context can be used in behaviour trees. They are shown, for example, that in principle, every agent could have a tree that begins with a behaviour to evaluate the field situation, and choose offensive or defensive behaviours based on one's current field position. This is not enough to play good soccer, but it does give the students a foothold into a complex collection of behaviours. Students quickly develop heuristics to more subtly evaluate the state of the game, and extend such trees to more detailed levels: adding the roles that need to be filled by various agents for each of these, for example and the ways those roles can be achieved. This results in structures similar to that shown in Fig. 7.

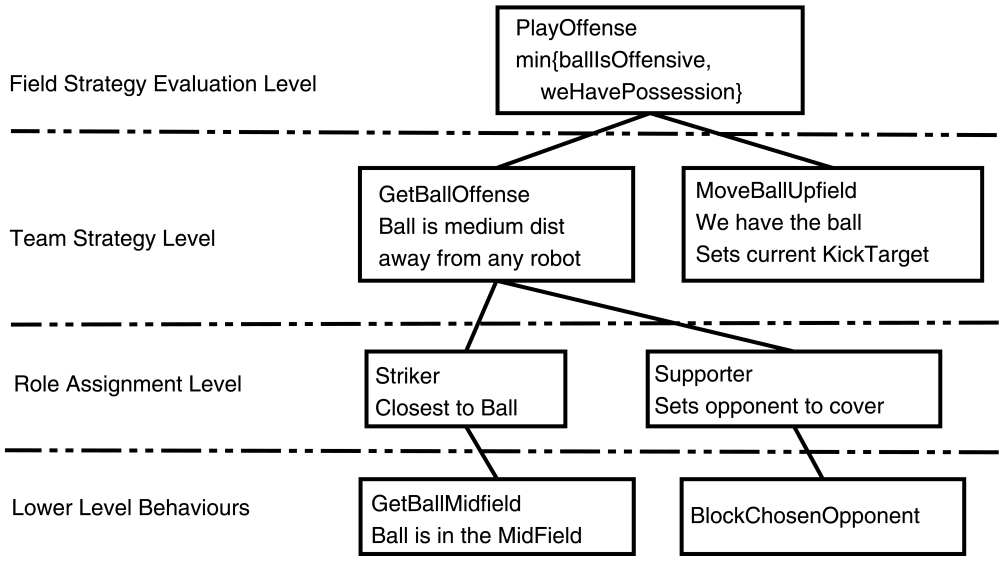


Figure 7: Subset of a behaviour tree.

This particular branch illustrates a structure for recognizing the need for an offensive play,

the definition of one such play, down through role assignment to lower-level behaviours. In practice this is often split to further levels with different strategies for any given adopted role. We have found that this structuring is especially convenient for making code changes quickly, which is advantageous in the final stages before competition. This degree of adaptability is also of great importance to the educational approach: the quicker students can adapt their code to new ideas, the more they can focus on the ideas themselves rather than the low-level coding involved.

It is common at this stage for students to propose forms of multi-agent interaction that would ultimately be much more complex than are needed. To counter this, we ask that students first begin by designing their team of agents in isolation, with only an understanding of what others are likely to do given the nature of soccer. Assuming they can develop a behaviour tree that lets each agent recognize what the team as a whole should be doing, it is possible to develop heuristics for making one's own choices correctly based only on visual perception and the agent's world model. For example, consider a team moving from defensive to offensive play: the closest player to the ball can attempt to capture it, while the remainder attempting to block an opposing player closest to them. If all agents have similar knowledge and perceptions, each can select its own role correctly without having to ask what others intend. Once students see a number of such situations, they also realize that significant amounts of everyday interaction can be handled by shared understanding rather than explicit communication.

While this is a minimalist approach, it can be used to implement fairly intricate collective behaviours. For example, students have found that the two-wheeled bodies of Lego MindStorms robots are limiting for strikers, in that they cannot move directly sideways to catch a rebounding ball if a goal shot is blocked by the goalkeeper. To counter this, a set of behaviours allowing two strikers to move together and avoid the need for direct sideways movement was designed. One striker moves for a shot on goal, while a second moves into a likely position to intercept a rebound. Once this play emerges as the most desirable (again, by both agents independently based on similar perceptions), the agent to initially shoot the ball is understood by both agents based on a weighting of distance to the ball and the degree to which each agent is blocked by others.

Relying on shared understanding in an uncertain domain can also result in misinterpretation. Two agents may both choose to block an opponent that they are equidistant from, for example, or differences in world models might make one agent consider a different play to be most appropriate. The use of global vision makes differing world models less frequent, but the imprecise nature of heuristics does leave open situations such as these. This can be handled in part by re-evaluating the behaviour tree regularly, since if the same decision were made a second later, the likelihood of two agents being equidistant from the same opponent is unlikely.

This re-evaluation of the current behavior was previously considered in the last agent in terms of limiting re-planning. Here a bound on how often behaviours are re-considered is also important, since constant change in the game can leave the agent continually re-planning if no attempt is made to deal with this. Students also see through the performance of their agents at this stage that other factors must be considered as well. For example, it is possible for the choice of appropriate behaviour to oscillate when behaviours are invoked with very

Figure 8: Soccer competition using 2" IR controlled tanks on a tabletop field.

similar strengths. If two possible plays can be considered approximately equal in quality, for example, re-evaluation over time may lead an agent to switch back and forth between them, accomplishing nothing. In these cases, some commitment to an overall choice must be built in, and changes made if no progress is shown. Similar hysteresis factors are found by the students in many places, and changes made to the allowable transitions between behaviours as a result.

After exploring coordinated behaviour through shared understanding, students are allowed to employ communication between agents to allow more explicit cooperation if they so desire. The agent-based approach allows for easy communication between software agents running on the network, and simple communication is both easy to add using standard text-based mechanisms.

The completion of this implementation (students typically have 20-30 behaviours structured using a behaviour tree or some other mechanism) leads to the final round-robin soccer competition, which is the capstone of the course. The field size used is proportional to the size of the robots (Fig. 5 shows the standard RoboCup E-League field size for Lego MindStorms robots, which conform to the RoboCup small size robot standards, while Fig. 8 shows a smaller field used for 2" toy tanks). Given the effort students have put into this and the friendly competitions that emerge throughout the term, this final competition is often very driven, and all the students are justifiably proud of their designs and the effort they have put into their teams.

At this point, there may be students interested in taking their working soccer teams and making them more competitive. If enough interest is generated, it may be possible to spend additional time addressing a team's weaknesses and then enter a competition. While qualification for international competitions has become very competitive, there are more regional competitions such as the as the U.S. Open, and students can also organize their own local competitions. We strongly encourage the latter, as we believe that the development of more competitions at the local level can be of great benefit to science and engineering education.

8. Discussion

In this paper we have described our educational strategies for employing robotic soccer to introduce undergraduates to robotics. As part of this we have emphasized the agent-based approach that we use to manage the infrastructure for robotic soccer and structure the development of robot controllers to manage the complexity from the students' perspective. We have employed these collectively for a number of years to teach students about robotics and AI, and have brought a number of competitive teams to both RoboCup and FIRA through this approach.

While we are emphasizing its use in undergraduate education, it is certainly possible to adapt this to other types of students. It is a simple matter employ this in graduate education by advancing the material covered and expecting a stronger background upon entry, and with care on the part of the instructors it could also be adapted to high school students. The inexpensive equipment we advocate is largely intended to emphasize more intelligent software, but also has the advantage of making this affordable to many high school programs. Similarly, the path tracking, path planning, and behaviour organization steps could be employed in many different domains other than soccer.

From the standpoint of educating students, introducing robotics through the development of increasingly-sophisticated software agents excels as an approach because it is able to make complex problems manageable. Complexity is managed by allowing earlier, simpler agents to give students a foothold on the next level of the problem. At the same time, the instructor can present material for future, harder problems by grounding these in terms of what the student has already implemented. The motivation provided by working, moving robots cannot be overstated: students can demonstrate their work right from the start, and gain confidence in their abilities through early successes.

The fact that later agents subsume and enhance the abilities of earlier agents also makes it possible for an instructor to provide some agents and concentrate their teaching efforts at a higher level. Students could begin with functional agents that can follow and plan paths, and study behavioural interaction and multi-agent systems from the standpoint of physical embodiment, for example.

The policy of allowing student groups to make their own choices is also important educationally, in that students are empowered but also have the responsibility of dealing with the consequences of their decisions, providing important lessons in software engineering that are difficult to teach in other courses.

From the standpoint of combining agent and robot technology, we believe this educational approach is also a good example of how these technologies can co-exist and enhance one another. Most inexpensive robots have such little local computational ability that controlling them via networked computers a logical approach, and at the same time, applications for the remote operation of robots have become broader in recent years, such as remote operation over the Internet (e.g. [23]), leading to more widespread use of these ideas. In all of these situations, agents are a proven and effective mechanism for separating high level individual and team control from the low-level actions of physical robots.

References

[1] Jacky Baltes, Elizabeth Sklar, and John Anderson. Teaching with robocup. In *Proceedings of the 2004 AAAI Spring Symposium on Accessible Hands-on AI and Robotics*, Stanford University, Stanford, CA, March 2004.

[2] Robocup official website. http://www.robocup.org.

[3] Federation of international robot-soccer association. http://www.fira.net.

[4] Jacky Baltes. 4 stooges. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup 2001: Robot Soccer World Cup V*, pages 559–562. Springer-Verlag, Berlin, 2002.

[5] Paul Furgale, Jacky Baltes, and John Anderson. Real-time vision-based pattern tracking without predefined colours. In *3rd Int. Conf. on Computational Intelligence, Robotics and Automation (CIRAS-05)*, Singapore, 2005. (submitted).

[6] Jacky Baltes. Open-source robocup vision server. http://robocup-video.sourceforge.net.

[7] John Anderson, Jacky Baltes, David Livingston, and Elizabeth Sklar. Toward an undergraduate league for robocup. In *Proceedings of the RoboCup Symposium*, 2003.

[8] Robin Murphy. *Introduction to AI Robotics*. MIT Press, Cambridge, MA, 2000.

[9] A. Balluchi, A. Bicchi, A. Balestrino, and G. Casalino. Path tracking control for Dubin's car. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Minneapolis, MN, April 1996.

[10] M. Egerstedt, X. Hu, and A. Stotsky. Control of a car-like robot using a dynamic model. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 3273–3278, Leuven, Belgium, 1998.

[11] Ronald C. Arkin. *Behavior-Based Robotics*. Cambridge: MIT Press, Cambridge, MA, 1998.

[12] Jacky Baltes and Robin Otte. A fuzzy logic controller for car-like mobile robots. In *Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Monterey, CA, November 1999.

[13] James S. Albus. A new approach to manipulator control: The cerebellar model articulation controller. *Journal of Dynamic Systems, Measurement, and Control*, 97:220–227, 1975.

[14] Jacky Baltes and Yuming Lin. Path-tracking control of non-holonomic car-like robots using reinforcement learning. In Manuela Veloso, Enrico Pagello, and Hiroaki Kitano, editors, *RoboCup-99: Robot Soccer World Cup III*, pages 162–173, New York, 2000. Springer-Verlag.

[15] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, 1991.

[16] Alexander Zelinsky. A mobile robot exploration algorithm. *IEEE Transactions of Robotics and Automation*, 8(6):707–717, December 1992.

[17] Z. Kedem H. Fuchs and B. Naylor. On visible surface generation by a priori tree structures. In *Proceedings of SIGGRAPH '80*, pages 124–133, 1980.

[18] Jacky Baltes and John Anderson. Flexible binary space partitioning for robotic rescue. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Las Vegas, October 2003.

[19] Rodney Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1), 1986.

[20] Esben H. Ostergaard. Evolving complex robot behavior. Master's thesis, Department of Computer Science, University of Aarhus, Aarhus, Denmark, May 2000.

[21] Dario Floreano. Evolutionary robotics in behavior engineering and artificial life. In Takashi Gomi, editor, *Evolutionary Robotics*, volume II. AAI Books, Ontario, Canada, 1998.

[22] Michael Bratman, David Israel, and Martha Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4(4):349–355, 1988.

[23] K. Goldberg, editor. *The Robot in the Garden: Telerobotics and Telepistemology in the Age of the Internet*. MIT Press Cambridge, MA, 2000.

Biographies

John Anderson is an Associate Professor in the Department of Computer Science at the University of Manitoba, where he also completed his Ph.D. research, and co-directs the Autonomous Agents Laboratory. His main research interests lie in autonomous agents, multi-agent systems, robotics, and the application of these to urban search and rescue and other complex problems. He is also interested in human-robot interaction and the use of robotics in education.

Jacky Baltes is an Associate Professor in the Department of Computer Science as well as the Department of Electrical and Computer Engineering at the University of Manitoba, and co-directs the Autonomous Agents Laboratory. He obtained his Ph.D. at the University of Calgary, and was previously a Senior Lecturer at the University of Auckland. His main research interests are robotics, computer vision, machine learning, and the application of these to complex problems, as well as the use of robotics in education.