Agent-Based Control In A Global-Vision Robotic Soccer Team

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

Robotic soccer is a highly complex domain that has become a significant challenge problem in both mobile robotics and artificial intelligence. Two well-known annual competitions, ROBOCUP*and* FIRA*, allow teams to compete in a number of different leagues distinguished by robot size and hardware restrictions. Because the domain is based on teamwork in a real-time environment, we have found agent-based control of individual robots to be a very convenient approach to designing a robotic soccer team. While we have worked with both local vision and global vision robots in the past, our most recent work has been with global vision. This paper describes the nature of this robotic domain, its suitability to agent-based methodologies, some of our particular motivations, and our use of agent-based control to deal with the difficult problems it encompasses.*

1. Introduction

Robotic soccer is a highly complex domain that has become a significant challenge problem in both mobile robotics and artificial intelligence. This challenge problem has been a tremendously useful to researchers in mobile robotics in terms of providing a common ground for research in vision, control, and individual and team behaviour, among many other areas. Moreover, the use of a standard domain also affords the opportunity for competition in order to judge the suitability of various proposed solutions. The two major competition organizations, ROBOCUP and FIRA, both divide teams into leagues based on the size and structure (e.g. legged vs. wheeled) of the robots themselves, and whether vision is provided globally or locally. While we have had local vision teams in the past [5], the majority of our experience lies in the small-sized (F180) league, which typically uses global vision.

Like other researchers, our attraction to this domain stems from a desire to do grounded research in problems that are nicely exemplified in robotic soccer - primarily sensing, control, planning, and teamwork, in addition to the logistics of getting solutions to all of these problems operating in concert. From the standpoint of competing in robotic soccer competitions, however, our motivations differ from some others in two important ways. First, as artificial intelligence researchers, we primarily want to emphasize sophistication in the AI components of our work. We do this in part by avoiding the use of specialized hardware for the robotic soccer domains. A typical small-sized team has robots that have omni-directional drives, dribble bars and powerful kicking mechanisms. While such additions can result in a very powerful team, a significant amount of that power comes from those hardware additions themselves, and this is less interesting from an AI standpoint than a team that functions through strong intellect. Further, the solutions we develop that do not employ specialized hardware can always benefit through the addition of such hardware later on, while the reverse may not be the case. We thus limit ourselves to basic mobile platforms without dribble bars or kickers. Our robots must pass the ball in a more traditional fashion, being unable to kick it the entire length of the field in one shot, while the lack of a dribble bar means that the robots must rely more on being at the right place at the right time, and on the skills of teammates to move the ball.

Second, we have a strong motivation to use this environment as part of an educational process [10]. Our teams are programmed to a large degree by students learning about robotics, and we try each year to improve what we do as an educational resource for future students. This further emphasizes a focus on AI over specialized hardware, but also places important economic constraints on hardware: we have to be able to afford enough to supply a significant-sized group of students. A typical Small-size team has robots that cost approximately US$3,000. Added to this is the cost of high quality video cameras, and other expenses that typically drive the cost for a team to around US$20,000–US$30,000.

There are also additional motivations that the physical grounding and competitive nature of robotic soccer place on the design of a team: the hardware must be portable enough to travel to competitions (preferably distributed throughout checked luggage of team members in order to reduce cost), must be modular enough that it can be dismantled and set up on the other side of the world in a reasonable amount of time, and should be both robust enough to survive typical transportation situations as well as common enough that replacement parts could be obtained almost anywhere.

This paper describes our robotic soccer team, the Little Black Devils , and in particular the agent-based control of robots that is central to our approach. The Little Black Devils (named after the Royal Winnipeg Rifles) are notable for a number of reasons, including their affordability (the entire team cost less than US$1000 in total), their reliance on infrared rather than radio communication, and their emphasis on artificial intelligence over specialized hardware.

2. Overview of Approach

We satisfy the motivations outlined in the previous section through the use of the Lego Mindstorms RCX as the computational platform for the robots themselves. These are readily available, inexpensive, easily replaced, and reasonably robust. While these do not have a high computational ability, the fact that this is a global vision team means that little on-board computation is required. Agent control programs as well as vision software run on external machines, and the decisions of those agents are relayed to the robots themselves. Local computation is restricted to the lowest level control (i.e. turning motors on and off to orient and move the robot), and is handled by a customized version of BrickOS (an open-source alternative operating system for the Lego Mindstorms RCX).

A high-level view of the entire system implementing the Little Black Devils is depicted in Figure 1. At the heart of the system is a set of agents which control the robots on the field, and which will be described in Section 3. The remainder of the system consists of the robots themselves, and facilities for providing visual information about the world to the agents and allowing the agents to affect the environment through the robots on the field.

To provide vision for the agents, a single video source (we use a standard consumer-grade camcorder) serves as input to a computer with a video capture device (e.g. tv tuner card). This machine runs our vision server software, known as DORAEMON [1]. DORAEMON includes real-time camera calibration, color calibration, and object tracking components. More significantly, DORAEMON also has the ability to calibrate the geometry of the scene from any view, meaning that it is not necessary to have the camera mounted directly overhead relative to the playing field, nor is it necessary to add a wide-angle lens to the camera. We currently use coloured markers on top of the robots, but we have recently developed a pattern-recognition process that allows the system to track objects without such markers by recognizing the images of the robots themselves [7]. We are currently working on improving the accuracy of that process in order to be able to rely on it in future competitions.

DORAEMON transmits the position, orientation and velocity of all objects that were found over Ethernet, which is the basis for providing visual information to the agents. The messages are transmitted in ASCII via UDP broadcast in a single package, an example of which is shown in Figure 2.

The first line of the message contains the number of objects being tracked, the frame number, and the time difference between this message and the previous one, allowing an agent receiving this package to determine if frames were dropped. This is followed by a line of information about the coordinates of the camera in terms of this coordinate system, which is intended for use in distributed or stereoscopic applications. Following that is a line for each object being tracked, consisting of the type (0=robot, 1=ball) and name of the object, whether the object was recognized in the image or was predicted based on previous motion, the X, Y, and Z coordinates of the object (in mm), the orientation of the object in radians, and the velocity of the object in X and Y directions. The camera and computational hardware currently provide approximately 30 frames per second of data in this format, allowing the agents to formulate a picture of the world they affect.

The agents themselves run on a network of Linux-based workstations. The ideal situation is one machine per agent, but multiple agents per machine can be used (with corresponding loss of computational resources per agent). One of the nice features of dealing with independent agents rather than a single global controller in a competitive situation is that as many machines as are available at the time can be used to host the agents, and there is a natural distribution of resources (as opposed to trying to partition a global controller into distributed processes). In fact, since the other two software components (the video and command servers) are also running on Linux machines on the same network, it is also possible to run agents on these machines as well if a situation arose where machines were limited. Each agent hosted on these machines picks up the network packets from the vision server, and independently (and asynchronously) communicate its decisions for the actions of the particular robot it controls.

The robots themselves are built using standard Lego pieces around a Lego Mindstorms RCX brick, and are marked with coloured disks to allow easier recognition by the vision server (see Figure 3). This brick controls two motors that allow a robot to change its orientation and move forward and backward. We employed two different robots, shown in Figure 3: a wheeled robot most commonly used as a striker, and a treaded robot most commonly used as a goalkeeper. The RCX brick receives information via infrared broadcast. Because the infrared is broadcast to all robots, current instructions for all robots are relayed together. This

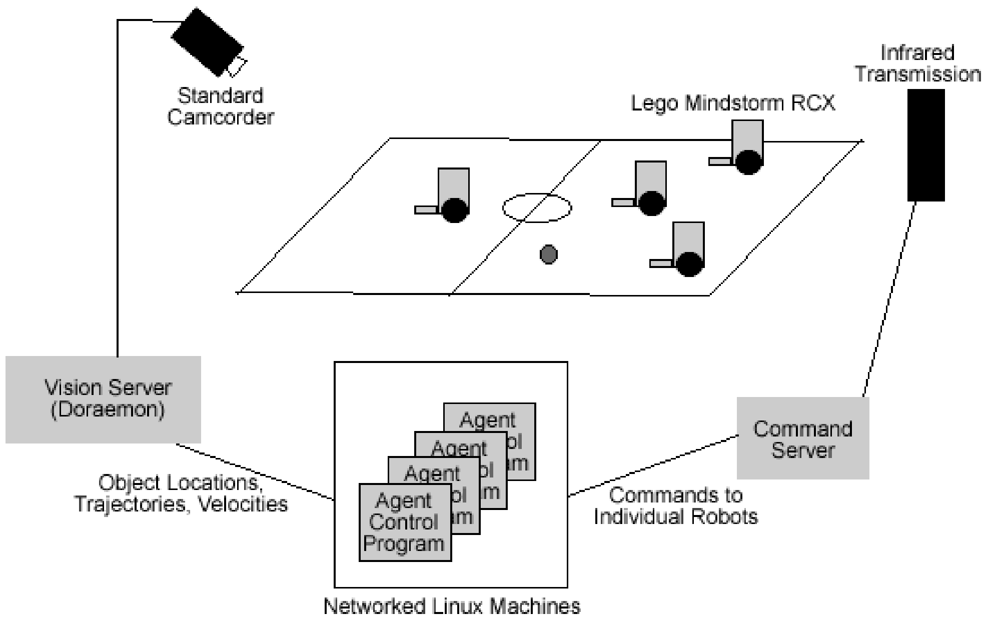


Figure 1: System Overview

```
9 1073821804 0.00139797
-76.3836 -1820.48 2356.39
1 ball NoFnd 971.056 840.711 35 0 -2.31625 58.1464
0 b0 Found 1185.59 309.187 100 0.0596532 499.282 285.083
0 b1 Found 1158.59 308.198 100 0.0596532 499.282 285.083
0 b2 Found 1086.95 309.187 100 0.0596532 499.282 285.083
0 b3 Found 1185.59 309.187 100 0.0596532 499.282 285.083
0 y0 Found 989.95 304.588 100 -0.10185 413.528 -1.08564
0 y1 NoFound 1189.95 304.588 100 -0.10185 413.528 -1.08564
0 y2 Found 1189.95 304.588 100 -0.10185 413.528 -1.08564
0 y3 Found 1189.95 304.588 100 -0.10185 413.528 -1.08564
```

Figure 2: Sample output message from DORAEMON.

requires the use of a command server running on a Linux system. This command server groups the commands to individual robots into a data packet that describes commands to all robots (possibly *no-op* commands if an agent has not made a decision for any action since the last command message was issued) and then broadcasts that packet using infrared transmission hardware.

The infrared transmission used by the RCX allows for 2400 baud, a very limited rate of communication. In order to allow the maximum number of messages to be sent per unit time (which directly affects how often a robot can change its activities), it is therefore important to limit the volume of information that is sent to the robots themselves. We use a ten-byte packet (a stop and start byte and eight bytes of data). This is intended to allow for a one-byte command to each of eight players on the field (allowing us to play four-on-four games with both teams using the same IR communication hardware). While it is possible to control only four robots and use two bytes each, there is a significant amount of information at this scale that can be encoded into a single byte. For example, out of eight bits, four can be used for orientation, allowing 16 different turning values, and a similar set of values for foward/backward motion from the other four bits.

The only non-standard hardware we employ is a custom set of infrared broadcasting hardware. While using infrared to communicate with robots is advantageous in a competitive setting because of the significant interference that the presence of teams generates, some difficulties had to be overcome. The infrared towers that are provided by

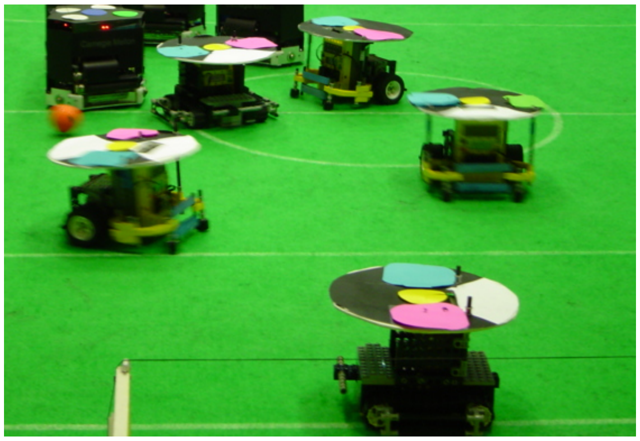


Figure 3: Team in play.

Lego are meant for limited distances in normal indoor environments. The playing fields involved in competition are larger, and the intense lighting interferes with the infrared transmission as well. While it is possible to use a number of Lego IR towers, difficulties arise in that delays between towers cause inconsistency in information (and interference between signals). These problems required the development of an infrared tower that mimics the behaviour of that provided by Lego, but had many more (32 as opposed to 1) infrared transmitters.

This completes an overview of the set of components required to run the Little Black Devils. As can be seen, this is a sophisticated set of hardware and software elements, and each part must interact with all the others in an appropriate manner for the system to operate. It is also a problem where there is a range of granularity that must be dealt with, from fine-grained motor control to high level path planning and team-level interaction. This is the major reason why agent-based control is so useful here, and in many other areas where robots operate: agent-based software controllers can deal with higher-level functionality, possibly on external systems that are more powerful than what a mobile robot could move (or would be affordable for a mobile robot), while lower-level functionality can be handled on the robots themselves.

Having seen that the agents themselves are central to this methodology, we now turn to examining the functionality of the agents in our approach.

3. Agent-Based Control

The agents we use to control the robots in the Little Black Devils are entirely custom-coded in C++: that is, we currently use no agent-based toolkits, and all facilities are written from scratch except for obvious components like device drivers, gui libraries and the like.

Part of the rationale for avoiding the use of agent toolkits is to avoid being tied year to year to a particular technological viewpoint, and part also lies in the educational approach. We want students to learn about how robot control operates and also how agents operate: they thus code everything from low-level robot control code to basic agent facilities, in addition to higher-level functionality.

The functionality of our agents is divided into three levels based on the level of abstraction from basic motion: (1) at the lowest level, agents must be able to follow specified paths (to get behind a ball, move toward a goal, block another player, and so on); (2) in order to follow these paths, agents must be able to generate them in response to particular goals, i.e. perform path planning; (3) finally, agents must have individual and team-oriented strategies that create goals for the agent (places the agent desires to be and courses of action the agent desires to take). The following subsections describe each of these levels in turn.

3.1. Path Tracking Control

Our agents follow a behaviour-based [3] approach. At the lowest level in our agent architecture are behaviours that allow an agent to follow a given path efficiently. In the robotic soccer domain, it is advantageous to move to desired positions (e.g., a position to receive a ball, a position to intercept an opponent) as quickly as possible. We therefore limit ourselves to the generation of paths with implicit maximum speed trajectories instead of the more complex problem of trajectory generation.

We have tried many different path tracking controllers in our research, including Balluchi's sliding mode controller [4], a fuzzy logic controller [9], Egerstedt's look ahead controller [11], and a reinforcement learning controller [8]. A combination of a fuzzy logic controller using Egerstedt's lookahead formulation of the path tracking problem has proven to be the most robust, efficient, and versatile method for path tracking control.

The algorithm and representation are illustrated in Figure 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 $\mathbf{P}'$, i.e., 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 itself by being robust and efficient. It is furthermore easy to change the

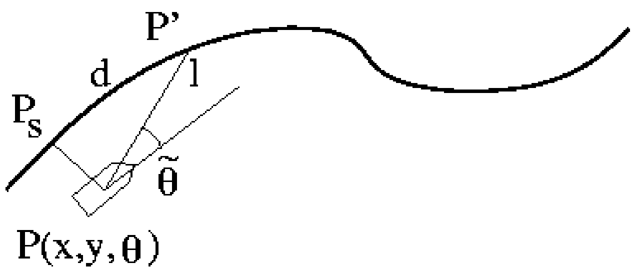


Figure 4: Path Tracking Control

behaviour of the agent to follow a path more closely (e.g. when moving through a tight spot) or fast and loose (when in more open spaces) by varying the lookahead distance and gain on the steering angle control.

3.2. Path Planning

Accurate path planning is both a difficult problem and a vital part of mobile robotics. While it is possible to go by orientation and simply move to the goal when one has the ball, for example, an intelligent agent should be able to be aware of its surroundings and plan paths to avoid obstacles (opponents), keep close to desirable places (teammates), and minimize travel to a goal.

Our agents maintain a global world model of all interesting objects in the vicinity of the robot (a natural consequence of global vision). The world model is limited, however, in the sense that the vision server broadcasts inherently noisy data to the agents. The agents interpret this message and create a world model which includes confidence values and best estimate of an objects position, orientation, and velocity.

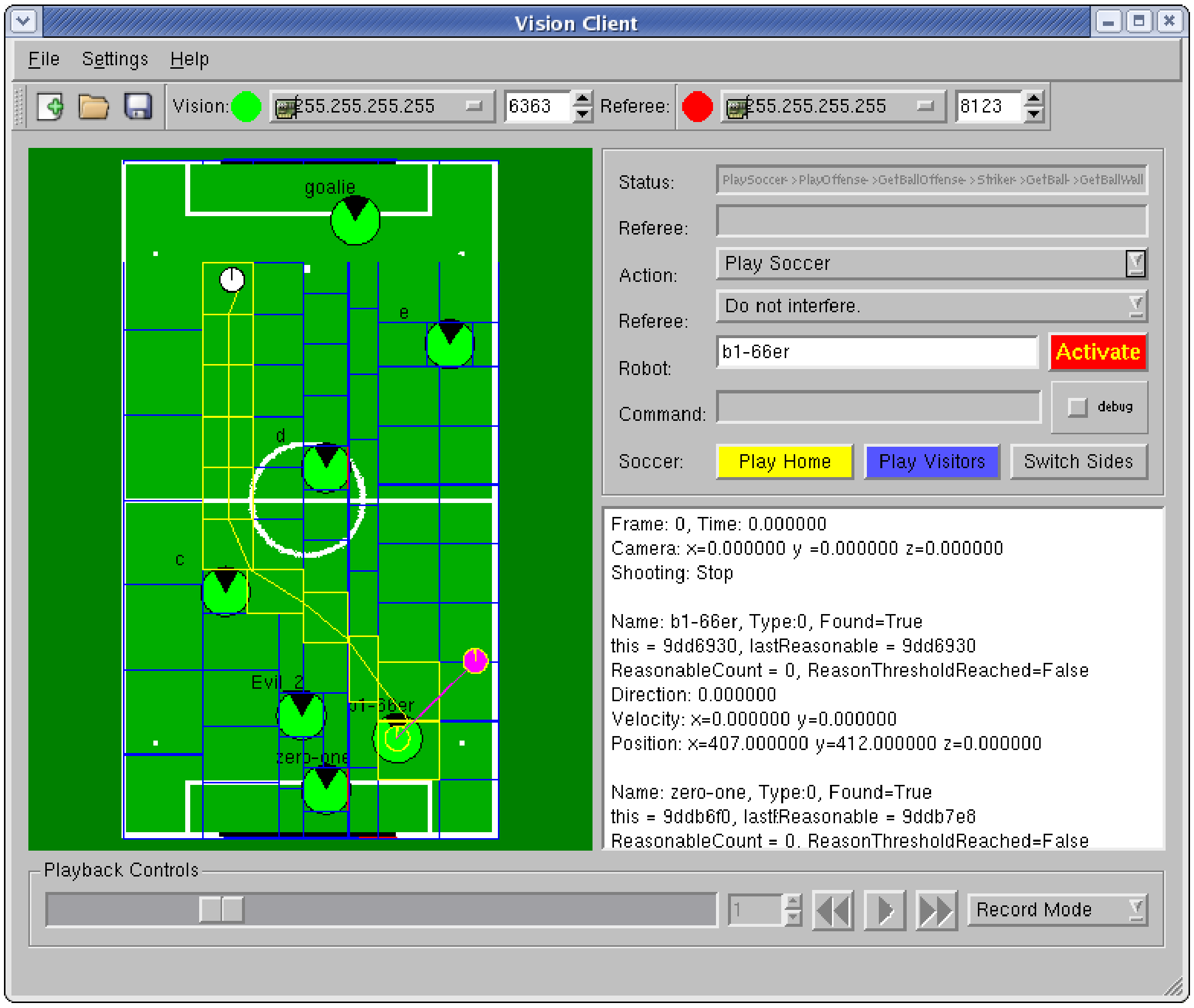
This world model serves as the basis for path planning through cell decomposition. Cell decomposition methods, which include Quad-tree decomposition, oct-tree decomposition and binary space partitioning (e.g. [13, 14]), construct a representation of the environment for path planning by recursively decomposing the known or detectable spatial area into partitions of regular size. Each approach begins with a single large cell, and labels that cell as either blocked (i.e. entirely filled by an obstacle), free (completely open space), or mixed (at least part of the cell is occupied by an obstacle). Any cell that is mixed must be recursively broken down by sub-partitioning it (using four sub-partitions in the quad-tree approach, eight in the oct-tree approach, and two in binary space partitioning), resulting in a tree structure. Any cell that can be labelled as blocked or free at any point forms a leaf node in that tree structure. As this tree is built, adjacency links must be maintained between branches of the tree in order to facilitate path planning. Path planning can then occur by traversing this structure in order to find a low-cost path from the robot's current location to its goal.

In our approach we use a new form of binary space partitioning which we term *Flexible Binary Space Partitioning* [6]. Instead of consistently partitioning the space into components of precise sizes (i.e. dividing in half for binary space partitioning), this approach attempts to make a logical choice for a partitioning point in any partitioning decision. An entropy-based measurement is employed to examine the potential information gain that would be made by choosing a particular point to further divide the space, and the point with the largest information gain is chosen. Once the space is partitioned, path planning ensues based on this decomposition.

In our implementation the internal states of individual agents can be displayed, and an example of this is shown in Figure 5. The results of flexible binary space partitioning are shown in dark and light lines overlaying the field (where light lines show the components that are actually part of a final path), along with the computed path from the selected agent's current location to its desired goal (the white marker close to the opponents' goal, showing a desired position slightly behind the ball and oriented toward the goal. The ball in this case is underneath the space occupied by the white marker). In this particular case, all the other agents on the field have been made opponents rather than teammates, but are considered as obstacles as far as path planning itself is concerned. This path then forms the basis for the path following behaviour described in Section 3.1.

In this domain, planning a path and then following that path without deviation cannot be the basis of a successful agent on its own. In a fast-moving dynamic domain such as soccer, an agent must deal with the fact that the path will often be invalidated as a result of agents moving.

Our path planning uses a local obstacle avoidance method based on potential fields [3]. While traversing a path, the agent stays clear of obstacles since virtual forces emanating from other objects push the robot away from such obstacles. Thus obstacles that were not seen by the vision server when a path was planned can still be accounted for during path following, and the type of obstacle can also be considered. For example, in Figure 5, the small light circle to the upper right of the agent performing path planning shows the desired immediate direction of travel based on a potential field that takes into account the obstacles around the agent in addition to the path. In this case, the two robots near the agent planning the path are opponents, and their presence is indicating a direction of travel away from where a purely obstacle-based path would indicate. Assuming the world stayed in the state it is in in the snapshot, the agent would move away from those opponents to a safer distance and would then turn toward the path that was originally intended. Similarly, the presence of opponents would also cause the agent to stray from the planned path at future points.



Status: PlaySoccer->PlayOffense->GetBallOffense->Striker->GetBall->GetBallWall
Referee:
Action: Play Soccer
Referee: Do not interfere.
Robot: b1-66er
Command:

```
Frame: 0, Time: 0.000000
Camera: x=0.000000 y =0.000000 z=0.000000
Shooting: Stop

Name: b1-66er, Type:0, Found=True
this = 9dd6930, lastReasonable = 9dd6930
ReasonableCount = 0, ReasonThresholdReached=False
Direction: 0.000000
Velocity: x=0.000000 y=0.000000
Position: x=407.000000 y=412.000000 z=0.000000

Name: zero-one, Type:0, Found=True
this = 9ddb6f0, lastReasonable = 9ddb7e8
ReasonableCount = 0, ReasonThresholdReached=False
```

Figure 5: Path planning using Flexible Binary Space Partitioning

It is of course also possible that the environment changes so drastically that the current behaviour (i.e., current goal and path to achieve this goal) is inappropriate and should be aborted completely. For example, a team has to switch quickly from offensive to defensive play because possession of the ball has changed. This decision is left to the upper, more knowledge-intensive layers of our architecture instead of the path planning layer. The nature of these upper-level individual and team behaviours are described in the following subsection.

3.3. Individual and Team Behaviour

Given agents that have the ability to plan paths and execute them in the presence of noise and dynamic obstacles, we have the basis for performing interesting individual and team behaviour. The most abstract level of our agents defines what individuals are interested in and how they operate as a team.

This layer implements the behaviour manager. Each possible high-level behaviour of an agent (e.g., intercept ball, intercept opponent, shoot on goal, move to rebound position) has an applicability and a reward function associated with it. The applicability function is a heuristic estimate of how well the current world model matches the behaviour. In other words, it can be interpreted as an estimate of how likely it is that the behaviour will succeed in the current world model. The reward function is an estimate of how much the robot's position would be improved if the behaviour succeeds. For example, blocking a shot on goal has a reward of +0.80, whereas scoring a goal for our team has a reward of +1.00. So our team is inherently biased towards a more aggressive strategy.

There are also additional perceptual behaviours. For example, there is a general perception strategy that calculates an offensive and defensive strategy estimate given the global view of the field. This estimate is based on the position of the ball, the number of opponent agents in the agent's own half of the field, the open area of our goal and the open area of the opponents goal. The output of the perception behaviours are evaluated by the task manager first and are available to the applicability and reward functions of the task-specific behaviours.

The behaviour manager is also responsible for selecting the strategy with the highest weighted average of applicability and reward. The selected behaviour will then be activated.

The behaviour manager calculates the applicability and reward for all behaviours. Large changes in the environment may then lead to the behaviour manager selecting a different behaviour with a different associated goal and path plan. Furthermore, the behaviour manager may also terminate a behaviour which is not making progress. The switching of behaviours is constrained by a hysteresis function, which limits the behaviour manager from switching behaviours too often or too quickly. This way, the agent deals with large changes in the highly dynamic domain of robot soccer.

As an example of team-level behaviours for a particular agent, consider the striker agent (the wheeled agents in Figure 3). Since the two-wheeled body does not permit the strikers to move sideways, striker agents implement a "cycling" behaviour. One striker moves in for a shot on goal, while another striker moves into a position to wait for a rebound or to shoot at the goal next. Since we want to avoid having both strikers try to shoot at the goal at once, they need to communicate their intention. This communication is currently implicit in the set of behaviours.

Two behaviours are responsible for the cycling behaviour. The SHOOT-ON-GOAL behaviour calculates the open area of the goal as well as the distance of the agent to the ball. If there is no other agent that is in a better shooting position, the SHOOT-ON-GOAL behaviour returns a high applicability, otherwise a lower one. Similarly, the POSITION-FOR-REBOUND behaviour also estimates the probability of a successful goal shot, but also expects an agent that is in a better shooting position. In this case, it returns a high applicability value. After the first striker shoots at the goal, the world state will change and the striker waiting for the rebound will then start its attack run, since no other agent will be in a better position and SHOOT-ON-GOAL will have a higher applicability then POSITION-FOR-REBOUND.

This scheme is augmented by: (a) the use of a hysteresis function as described above, in order to avoid oscillating between switching from goal shot to rebound, and (b) a time horizon scheme. The striker agent has only a limited time to show progress. If the goal shooter does not make progress towards the goal shot (because for example, it is being blocked by an opposing robot), the rebound striker will attempt a goal shot. Once the rebound striker is closer to the ball than the goal shot striker, their roles will change and the goal shot striker will become the rebound striker and will attempt to move toward the rebound position.

We augment this scheme with explicit communication between the agents. Explicit communication can easily be added to the behaviour manager by extending the current set of applicability and reward functions for our agents. For example, the applicability of POSITION-FOR-REBOUND may be further increased if a message from the other agent was received that explicitly states that the other agent is executing the SHOOT-ON-GOAL behaviour. This is similar to players shouting messages to one other (e.g., "I've got the ball").

We attempt to be realistic and have this communication be at the software agent level: that is, the agent program for a striker robot can send a message to the agent program for another striker robot, asking what it is doing or intends to do. This agent communication is implemented with standard text-based transmission over the network. It is also possible for indirect communication to occur, in that a robot could move or signal on a field that provides visual information for the control program of another agent to pick up. Since the robots have no appendages or any sensing other than vision, this would have to be in the form of a movement pattern, which would result in extremely slow communication, but would be possible.

4. Conclusion

This paper has described the agent-based controllers used in the Little Black Devils, an F180 league robotic soccer team. Agent-based control is an important part of fulfilling our two motivations: to explore artificial intelligent techniques to solve problems in this domain as opposed to relying on specialized hardware, and to provide an educational environment for students. The agent control programs are heterogeneous, as the robots themselves are heterogeneous.

We believe this is a good example of the kind of problem where agents and robots operate well together. Most robots (save the most expensive available) manage only very limited computational resources locally, but potentially have access to significant networked computer resources. This situation will become more prevalent in future, and swarms of very small robots become more widely affordable, while at the same time, applications for internet-based control of robots become more broad (e.g. [12]). In such situations, agents are an ideal mechanism for handling high-level individual and team control from low-level robotic control.

As stated in Section 1, we also have a strong motivation from the standpoint of educating students. Here agents also excel as a tool for making this manageable from a student standpoint, by providing a level of abstraction and allowing the students to separate things they need to be concerned with in terms of low-level robotic control and the the higher-level intelligent facilities that make use of such control. The facilities outlined in Section 2 form the basis of our approach to the design of a new ROBOCUP league suitable for undergraduates, the ULeague [2]. This has been incorporated into the design of the new E-League that will be run at the 2004 ROBOCUP competition.

Our own students are strongly encouraged to use the agent-based approach in their work in robot control for soccer. In a course setting, we begin by having students install firmware (customized BrickOS) for the RCX, and then design an encoding pattern that their robot will use for the interpretation of commands sent by the command controller. The students then do an assignment that provides the rudiments of agent-based control by writing an interface that allows the students to control the robots manually through a graphical user interface for robot control and a visualization of the data from the vision server. At this point, the students can fill the roll of the agent based controller by guiding the agent by hand, and get a feel for the kinds of problems that a controller for such a robot encounters. From here, students gradually design an agent controller through a series of assignments that allow them to a) follow a static path allowing a basic race track; b) plan paths to goals and re-plan based on changes in the environment through moving obstacles; and c) define the skills for a striker and goaltender agent that make use of the facilities provided by a) and b). Note that this follows the same progression for a soccer agent laid out in Section 3.

In terms of future work with the agent model, we are not only improving the various aspects of our software agents, but are working on a paper describing the educational process involved in using agents to teach students how to control robots in a classroom setting. We hope that this may evolve into a textbook suitable for use in an advanced undergraduate class (we have used this approach with both graduate and undergraduate students).

There are also ways in which the agents used here can be extended. For example, agents are currently not mobile: they are started on a particular machine and remain on that machine until they are deactivated. While mobility in a software sense is a common part of agent technology, it is not strongly needed in this particular application: A machine's load does not drastically change from point to point during a soccer game. If machines were also used for other purposes, or if a machine going down during the game would not already be a fairly catastrophic event, having agent code be able to migrate from machine to machine might be more useful.

We expect to continue to employ the agent approach described here in future, both from an educational standpoint and as a basic design mechanism for robotic soccer teams.

5. References

[1] John Anderson and Jacky Baltes. *The Doraemon User's Guide*. Department of Computer Science, University of Manitoba, Winnipeg, Canada, August 2002. http://robocup-video.sourceforge.net.

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

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

[4] 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.

[5] 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.

[6] 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.

[7] Jacky Baltes and John Anderson. Identifying robots through behavioral analysis. In *Proceedings of the Second International Conference on Computational Intelligence, Robotics, and Autonomous Systems*, Singapore, 2003.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] C. Saona-Vazquez, I. Navazo, and P. Brunet. The visibility octree: A data structure for 3d navigation. *Computer & Graphics*, 23(8):635–644, 1999.

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