The MetroBots Four-legged League Team at RoboCup 2003

Vanessa Frias-Martinez[1], Marek Marcinkiewicz[2], Simon Parsons[3], and Elizabeth Sklar[1]

[1] Department of Computer Science
Columbia University
1214 Amsterdam Avenue, New York, NY 10027, USA
vf2001,sklar@cs.columbia.edu

[2] Department of Computer Science
Graduate Center, City University of New York
365 Fifth Avenue, New York, NY 10016
mom7@cs.columbia.edu

[3] Department of Computer and Information Science
Brooklyn College, City University of New York
2900 Bedford Avenue, Brooklyn, NY 11210, USA
parsons@sci.brooklyn.cuny.edu

Abstract. This paper describes the MetroBots Four-legged league team that participated in RoboCup 2003. The MetroBots team was formed in September 2002 and so this was our first year of competition. The paper gives some of the background to the team development, briefly describes the overall structure of the code, and gives some detail on those parts of the code that we were happy with. Finally, we describe ways in which we aim to develop the team in the future.

1 Introduction

The MetroBots four-legged league team was formed in September 2002 as a collaboration between[1] Michael Littman at Rutgers University, Simon Parsons at City University of New York (CUNY), and Elizabeth Sklar at Columbia University. Initially the team also included four Ph.D. students, Paul Batchis (Rutgers), Vanessa Frias-Martinez (Columbia), Dave LeRoux (Rutgers), and Marek Marcinkiewicz (CUNY), who participated fully in the project and some additional Rutgers students who had a more part-time role. This team was responsible for the initial development of the MetroBots team that participated in the 2003 American Open. The further development of the team following the American Open, and participation in RoboCup 2003, was the work of the four authors alone.

As with any new Legged league team, we struggled to get basic functionality working, and this was compounded by the small number of students involved and the fact that we were geographically distributed (though the groups from Columbia and CUNY have always met on a weekly basis). In particular we suffered in producing reliable vision,

[1] In alphabetical order.

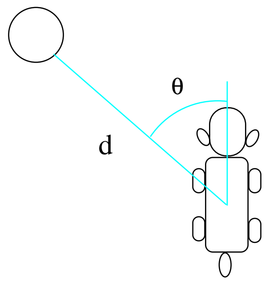

Fig. 1. The distance d and angle θ calculated by the Perception module.

and in being able to localize. In the end, we think we produced a fairly good solution to the vision problem, but localization remained beyond us, and this was reflected in our results in Padua[2].

That we spent all of our time in getting the robots to play a minimal game of soccer was particularly frustrating because we are less interested in developing robotic soccer players *per se* than in using robot soccer players as a testbed for our research in intelligent agents and multi-agent systems. We were unable to pursue this latter aim this past year. However, we see several ways in which we can carry out such research in the next year, and we sketch some of these below.

The rest of the paper is structured as follows. Section 2 describes the code release that we had by the end of RoboCup 2003 in Padua. Section 3 then goes into some detail about the features of that code we consider most interesting (the vision code, the way we do colour calibration, and the process of role assignment). Then, Section 4 discusses some of the ways in which we hope to use MetroBots as a testbed for our research in intelligent agents and multi-agent systems and Section 5 concludes.

2 The RoboCup 2003 code release

The version of the MetroBots code used at RoboCup[3] is structured as five modules, each of which is a C++ object:

1. Perception;
2. Localization;
3. Behaviour;
4. Communication; and
5. Control.

These provide the following functionality.

[2] Of the five group games that we played, we won only two, against Jolly Pochie from Kyushu University and Dynamo Pavlov from Uppsala University, both of which were also in their first year.

[3] Available from `http://agents.cs.columbia.edu/metrobots/`

Perception takes as input a frame from the on-board camera, identifies any interesting objects in that frame, and returns the relative locations of those objects with respect to the robot. The objects that we can currently detect are the ball, the goals, and the markers. The distance that is estimated is the distance from the centre of the robot body, and the angle that is estimated is the angle between the centreline of the robot and a line from the object to the centre of the robot body (see Figure 1).

Both these values are currently likely to be inaccurate in the general case (though measurement suggests they are not that inaccurate in the cases we have tested) because of the way they are measured. Distance is calculated by counting the number of pixels in an object and the angle is estimated to the centroid of a block of the right coloured pixels. Both these calculations can be distorted when the object is occluded by another robot.

Localization does not currently do anything useful. We did not get the robots to localize, and so this module just passes the information it receives from Perception on to Behaviour and Control.

Behaviour currently makes all the decisions about what the robot should do. Based on information about the relative location of the ball, goals and markers, the Behaviour code makes appropriate calls to the code that makes the robot move. We did not write this motion code—instead it is the motion-controlling code from CMPack '02 which we used, precompiled, from the Walkbench code repository maintained at `www.aibo-pet.com`.

The Behaviour code is structured around the idea of roles—Goalkeeper, Primary Attacker, Offensive Supporter and Defensive Supporter—and for each role it provides a set of behaviours. Some of these are specific to a particular role, for example, the Goalkeeper has a role to walk to the ball, kick it, and then return to the goal, and the Primary Attacker has a behaviour to walk to the ball and to kick it in the direction of the opponent's goal. Other behaviours are available to all roles, such as the behaviour to turn the head of the robot looking for the ball.

Since some of these behaviours require knowledge of the robot's location on the pitch, and since this information cannot currently be supplied by Localization, the Behaviour code also works out the orientation of the robot on the pitch (in other words which end and which side of the pitch the robot is facing).

Communication handles incoming messages from the Game Controller, as well as communication with the other robots on the team. Inter-robot communication allows the robots to coordinate themselves, albeit in a rather limited way.

Control provides the overall coordination between robots on the team. While Communication provides the message passing, Control decides what messages to pass, and translates incoming messages into what the robots actually do. The basic mechanism by which Control changes the behaviour of the robots is by having them switch role (all robots except the Goalkeeper change role dynamically through the course of a game) in the way described in Section 3.3 below.

3 Interesting features

Since so much of our work this year was reinventing the "basic soccer" wheel, it is arguable whether any of the code described in the previous section is really interesting. However, there are three areas that we are happy with, and since these constitute our main achievements, we will describe them in a little more detail. They are:

– Object detection
– Colour calibration; and
– Role assignment

Each is covered in one of the following sections.

3.1 Object detection

The way that we detect objects is very simple, but also seems to be effective. Though we have not tested it against any alternative methods, reviewing the video footage of our games in Padua suggests that it is a robust approach.

We analyse a frame by looking in turn at each pixel and establishing whether the RGB values[4] of that pixel mean we should identify it as being one of five colours—pink, orange, blue, green or yellow. (We compare each of the R, G and B values with a range corresponding to the calibrated values for the five colours. If a pixel, for example, has R, G and B values that fall between the maximum and minimum R, G and B values for orange pixels, that pixel is classified as orange.) The result of this processing is a 2D matrix where each element is an indication of the colour of the corresponding pixel (most of which are none of the colours we detect and so remain unlabelled).

We then scan through this matrix twice. On the first pass we consolidate existing groups of pixels of the same colour. This involves, for example, identifying pixels that are not already part of a group but should be (because they are above or below a group for example), and merging two adjacent groups of the same colour. On the second pass we then merge groups of the same colour that are sufficiently close into "blobs". If the number of pixels in a given type of blob is above a threshold, we consider that we have seen the relevant object (for example any orange blob with more than 10 pixels is considered to be the ball). We can then estimate how far we are from the object by looking at the number of pixels that we see.

As already mentioned, this is crude but effective.

For objects like the ball and the two goals, this is all that we need to do, but for markers we need a little more processing. We identify an object as a marker by the presence of a pink blob, but obviously need to establish what marker it is as well. This is done by looking above and below the pink blob to see whether there is any recognisable colour in that location. If there is, then we positively identify the marker and perform the same crude "pixel counting" distance calculation. If there is no colour, we assume that either the pink blob is a false positive, or that it is at the edge of the frame and so

[4] As described below in Section 3.2, we use RGB values rather than YUV values, translating the YUV reading received from the camera into an RGB reading before we start the object detection.

is unidentifiable. We also use the presence of a pink blob to determine that patches of blue or yellow adjacent to the blob cannot be a goal.

3.2 Colour calibration

When we first started work on the AIBOs, our object recognition code used YUV values. However, we had no means to easily calibrate YUV ranges for the colours we were detecting, something that we needed quite badly given that we were constantly moving the robots between different laboratories. In addition, the lighting in the space we were using at Columbia changes fairly frequently (because one wall of the space is lined with windows, and there is no way to block the light from them), meaning that we needed a quick means of doing calibration. Using RGB values made it possible to develop a nice tool for rapid colour calibration and allows us to calibrate from scratch in around 30 minutes.

The tool takes as input eight images, and displays these in two groups side by side, each group containing all eight images. The user selects which colour they want to calibrate, and identifies a region of that colour in one of the images in the left-hand group. The maximum and minimum R, G and B values of pixels in that area define the colour in question, and the calibration tool then identifes all pixels that fall within those combined ranges in the images in the right-hand group. Thus one gets instant feedback on false positive and false negative readings and can adjust the ranges accordingly. This adjustment can be done either by selecting further regions on the lefthand set of images, or by manually changing the RGB values (the tool provides sliders for this purpose).

Once all colours have been calibrated (and the results of all the calibrations can be displayed by the tool simultaneously), the tool will write a text file that includes the maximum and minimum RGB values of all the colours. This is then included in the code that is placed on the AIBO memory stick and is read by the Perception code. Note that the calibration tool is currently set to handle six colours—pink, yellow, green, blue and orange plus a second "shadow orange". This latter colour is the colour of the ball when in the shadow of the head of the AIBO. We found that if you try to establish the RGB range for the ball without the shadowed region, the robot can fail to see the ball when it is close and lighting is overhead, and if you try to establish ranges for orange that include the shadowed region then "orange" overlaps with other colours. Thus the only answer is to detect two different colour ranges and merge them when doing object detection.

3.3 Role assignment

Since one of our main interests outside RoboCup is in developing multi-agent coordination techniques, we were very keen to build in some kind of coordination. Sadly, without localization our efforts in this direction were very limited, but we still managed to develop an approach that was effective in preventing the robots from crowding around the ball. (We also found that having all the robots rush to the ball, as they do without coordination working, was an effective tactic against some teams and so turning coordination off could be advantageous.) Again the approach we took was simple but effective.

The coordination is based around the roles used by the CMPack '02 team. We start the game with each robot designated as "primary attacker", "offensive supporter". "defensive supporter" and "goalkeeper". The role of goalkeeper is fixed throughout the game. The other three roles, however, are more fluid. The robots signal one another constantly, via the TCPGateway, broadcasting their estimate of the distance between themselves and the ball. Based on the various estimates, the robot that is closest to the ball takes on the role of primary attacker, proceeds towards the ball and tries to kick it towards the opposing team's goal. The offensive supporter also moves towards the ball, but stops before it gets too close, and the defensive supporter waits until the ball approaches it (before switching to a primary attacker mode).

Without effective localization, the only real effect of this coordination is to prevent the robots interfering with one another, and it achieves this reasonably well. Much of the tuning that we carried out during the RoboCup competition was to perfect the changeover in roles, minimising the occasions on which the robots hesitate while deciding which should go and kick the ball, thereby giving a robot from the opposing team an opportunity to steal it.

In order to ensure that there is always one primary attacker (since having no robot playing this role would be disastrous), all the robots (other than the goalkeeper) default to being primary attacker if communication breaks down.

4 Future directions

In this section we describe our research agenda for the next year as regards the MetroBots team. The work we intend to do can be considered as falling into two broad areas—work on individual intelligent agents and work on multi-agent systems.

4.1 Decision making and agent models

One major aim we have within the arena of individual intelligent agents is to integrate work on decision making under uncertainty, which has led to models such as partially observable Markov decision processes (POMDPs) [11], with work on agent architectures and to test the integration on the AIBO robots.

As things stand, decision making models have largely been developed for rather simple tasks (for example [13] where the case study is a one-on-one soccer game played in a small grid-world), and the techniques do not scale for real-time, complex applications like the RoboCup task. In contrast, techniques from the multi-agent systems world, like the belief/desire/intention (BDI) model, have been designed to be scaleable, but do not deal well with the uncertainties of interactions in real, physical environments. Our previous work has investigated the use of BDI models in robotics [16] and on the integration between BDI models and POMDPs [26]. The AIBOs seem to be an ideal platform on which to carry out further research on this topic.

4.2 Improving decision making over time

Our second aim in the area of individual agents is to use the AIBOs in developing and testing new approaches to learning behaviours from experience (reinforcement learning and evolutionary learning). The majority of work in reinforcement learning [12] has assumed that decision makers have perfect knowledge about the state of the environment. While this assumption can be approximately satisfied in some domains (like the small robot league with its overhead camera), it ignores some of the most important and interesting aspects of cognition, namely learning and reasoning based on uncertain observations. There is some reinforcement learning work that uses the POMDP framework, which can capture sensor-oriented decision making, but sensors are typically assumed to return fewer than 6 bits of information per time-step.

Both evolutionary and reinforcement learning are traditionally difficult on real robots because the number of iterations it takes for a learning algorithm to converge is typically longer than the battery life of a real robot. In addition, evolutionary learning in simulation has traditionally been a problem for robotics, because the behaviours learned in simulation do not transfer well onto robots due to the uncertainties in the environment which the simulator does not model. Evolutionary learning has the additional problem of needing sometimes large populations of agents to learn from, and these cannot be run on an on-board processor. We plan to take advantage of the wireless connection with the AIBO to feed real-time learning engines that run in parallel with the AIBO. As the AIBO experiences the world, it sends environmental parameters to the learning engines. As the learning engines progress, they will send improved behaviours to the AIBO.

4.3 Dialogues for robot coordination

When robots communicate during a soccer game, it can be helpful to have them share the reasons behind their actions [24], but the limited bandwidth for communication means that such explanation should be restricted to relevant information. This is exactly the kind of *argumentation-based* communication that is increasingly being applied to the design of agent communications languages and frameworks, for example in the work of Dignum and colleagues [7]; Grosz and Kraus [9]; Reed [23]; and Sycara [29]. Our recent work in this area has been to identify how this kind of approach can be used to carry out a range of common dialogues, including those to elicit and exchange information, and to determine the properties of such dialogues [18, 19].

In the context of MetroBots, our aim is to use argumentation-based dialogues to improve communication between robots. However, we are not suggesting equipping the robots with the ability to engage each other in logic-driven dialogue during a game. Instead, our aim is to use the kind of dialogue systems we have explored as a specification for the communication components of the robots, allowing the kinds of guarantee we can obtain for these systems—about the desirable outcomes of the dialogues for instance—to be carried over to the dialogues between robots.

4.4 Market-based coordination

As described above, one of the major tasks that the robots on a RoboCup team have to deal with, is deciding which robots will undertake which roles. Deciding the allocation of roles to robots is a resource allocation problem, and we intend to investigate the use of market-based programming mechanisms [30] for this problem. To do this, we will build on our ongoing work on evolving auction mechanisms [22, 20, 21].

In this work, we have been using genetic programming to evolve both strategies for buyers and sellers—that is functions that determine what offers to buy and sell agents should make—and strategies for the auctioneer—that is what functions auctioneers should use to determine the price of goods given what the buyers and sellers offer. Our work so far demonstrates that such an approach can generate efficient auctions [20] and can come up with sensible pricing rules [21], and we are continuing to develop tools to evolve other parts of auctions. This line of work is also capable of discovering new kinds of auction mechanisms [5, 4] which have not previously been studied.

Although market-based mechanisms like auctions can be applied to resource allocation problems (as argued in [6]), it is not obvious at first sight how auctions might be used in robot soccer. However, consider a role, such as primary attacker, being a scarce resource which can be allocated to exactly one robot. If no robots are allocated this resource, then the team suffers since no robot will try to move to the ball. If several robots are allocated this resource, then the team will suffer as they interfere with one another. So auctioning, in some form, the right to take a role can be a useful mechanism. Indeed, the mechanism we use now can be considered such an auction—each robot offers its distance to the ball as the "payment" it requires to undertake the role, and the lowest offer wins. What we aim to do is to use our auction evolution tools to investigate whether there are "coordination" auctions that are more effective at allocating roles than the techniques that are already in use.

4.5 Engineering good protocols

Once we have established coordination protocols, whether by evolution or from argumentation-based specifications, we want to ensure that the protocols are sound. By that we mean that we need to ensure that the protocols do not lead to deadlock—leaving the robots unable to coordinate and thus unable to play properly—or in a situation where resources are overcommited—and, for example, several robots have taken on the same role. One way to check protocols to ensure that this does not happen is through the use of model checking [3, 10].

Our previous work [31] extended the SPIN model checker to work for programming languages in which one might specify agents, and this has subsequently been extended [2] to allow the agents to be specified in an even richer agent programming language which includes the kind of constructs we will need to use to communicate coordination information between agents. Our aim is to take this work, and use it to check our coordination protocols. Clearly, we will already have some guarantees about the protocols from the work described above. For simple protocols we can prove, as in [18, 19], the validity of the protocols, and the evolutionary process gives some guarantees about the protocols we evolve. However, for protocols more complex than those we can handle analytically, we believe that model checking can give us better guarantees than the evolutionary process alone.

5 Summary

This paper has described the MetroBots Four-legged league team that took part in RoboCup 2003 in Padua. The paper gives some of the background to the development and provides an overview of the code. It also provides some detail on three parts of the code—object detection, colour calibration and role assignment—that we believe function effectively. Finally, the paper sketches some of our plans for future research.

References

1. L. Amgoud, N. Maudet, and S. Parsons. An argumentation-based semantics for agent communication languages. In *Proc. 15th European Conf. on AI*, 2002.
2. R. Bordini, M. Fisher, C. Pardavilla, W. Visser, and M. Wooldridge. Model checking AgentSpeak. In Jeffrey S. Rosenschein and Michael Wooldridge, editors, *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems*, New York, USA, 2003. ACM Press.
3. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking*. MIT Press, Cambridge, MA, 2000.
4. D. Cliff. Evolution of market mechanism through a continuous space of auction-types. Technical Report HPL-2001-326, HP Labs, 2001.
5. D. Cliff. Evolutionary optimization of parameter sets for adaptive software-agent traders in continuous double auction markets. Technical Report HPL-2001-99, HP Labs, 2001.
6. D. Cliff and J. Bruten. Minimal-intelligence agents for bargaining behaviours in market-based environments. Technical Report HP-97-91, Hewlett-Packard Research Laboratories, Bristol, England, 1997.
7. F. Dignum, B. Dunin-Kęplicz, and R. Verbrugge. Agent theory for team formation by dialogue. In C. Castelfranchi and Y. Lespérance, editors, *Intelligent Agents VII*, pages 141–156, Berlin, Germany, 2001. Springer.
8. F. Dignum, B. Dunin-Kęplicz, and R. Verbrugge. Creating collective intention through dialogue. *Logic Journal of the IGPL*, 9(2):305–319, 2001.
9. B. J. Grosz and S. Kraus. The evolution of SharedPlans. In M. J. Wooldridge and A. Rao, editors, *Foundations of Rational Agency*, volume 14 of *Applied Logic*. Kluwer, The Netherlands, 1999.
10. G. Holzmann. *Design and validation of computer protocols*. Prentice Hall, Hemel Hempstead, 1991.
11. Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998.
12. Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.
13. Michael L. Littman. Value-function reinforcement learning in Markov games. *Cognitive Systems Research*, 2(1):55–66, 2001.
14. P. McBurney. *Rational Interaction*. PhD thesis, Department of Computer Science, University of Liverpool, 2002.
15. S. Parsons and N. R. Jennings. Negotiation through argumentation — a preliminary report. In *Proc. 2nd Intern. Conf. on Multi-Agent Systems*, pages 267–274, 1996.
16. S. Parsons, O. Pettersson, A. Saffiotti, and M. Wooldridge. Intention reconsideration in theory and practice. In W. Horn, editor, *Proceedings of the 14th European Conference on Artificial Intelligence (ECAI-2000)*. John Wiley, 2000.
17. S. Parsons, C. Sierra, and N. R. Jennings. Agents that reason and negotiate by arguing. *Logic and Computation*, 8(3):261–292, 1998.
18. S. Parsons, M. Wooldridge, and L. Amgoud. An analysis of formal interagent dialogues. In C. Castelfranchi and W. L. Johnson, editors, *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems*, New York, USA, 2002. ACM Press.

19. S. Parsons, M. Wooldridge, and L. Amgoud. On the outcomes of formal interagent dialogues. In Jeffrey S. Rosenschein and Michael Wooldridge, editors, *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems*, New York, USA, 2003. ACM Press.
20. S. Phelps, P. Mcburney, S. Parsons, and E. Sklar. Co-evolutionary mechanism design: A preliminary report. In J. Padget, D. C. Parkes, N. M. Sadeh, O. Shehory, and W. E. Walsh, editors, *Agent-Mediated Electronic Commerce IV*. Spinger Verlag, Berlin, Germany, 2002.
21. S. Phelps, P. McBurney, E. Sklar, and S. Parsons. Using genetic programming to optimise pricing rules for a double auction market. In *Proceedings of the Workshop on Agents and the Semantic Web*, Pittsburgh, PA, 2003.
22. S. Phelps, S. Parsons, P. McBurney, and E. Sklar. Co-evolution of auction mechanisms and trading strategies: Towards a novel approach to microeconomic design. In *Proceedings of the 2nd Workshop on Evolutionary Computation and Multi-Agent Systems*, New York, NY, 2002.
23. C. Reed. Dialogue frames in agent communications. In Y. Demazeau, editor, *Proc. 3rd Intern. Conf. on Multi-Agent Systems*, pages 246–253. IEEE Press, 1998.
24. P. Riley, P. Stone, and M. Veloso. Layered disclosure: Revealing agents' internals. In C. Castelfranchi and Y. Lespérance, editors, *Intelligent Agents VII*, pages 61–72, Berlin, Germany, 2001. Springer.
25. M. Schroeder, D. A. Plewe, and A. Raab. Ultima ratio: should Hamlet kill Claudius. In *Proc. 2nd Intern. Conf. on Autonomous Agents*, pages 467–468, 1998.
26. M. C. Schut, M. Wooldridge, and S. Parsons. Reasoning about intentions in uncertain domains. In D. Dubois and H. Prade, editors, *Proceedings of European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, Toulouse, France, 2001.
27. M. P. Singh. Agent communication languages: Rethinking the principles. In *IEEE Computer 31*, pages 40–47, 1998.
28. M. P. Singh. A social semantics for agent communication languages. In *Proc. IJCAI'99 Workshop on Agent Communication Languages*, pages 75–88, 1999.
29. K. Sycara. Argumentation: Planning other agents' plans. In *Proc. 11th Joint Conf. on AI*, pages 517–523, 1989.
30. M. Wellman. A market-oriented programming environment and its application to distributed multicommodity flow problems. *Journal of Artificial Intelligence Research*, 1(1):1–23, 1993.
31. M. Wooldridge, M. Fisher, M-P. Huget, and S. Parsons. Model checking multi-agent systems with MABLE. In C. Castelfranchi and W. L. Johnson, editors, *Proceedings of the 1st International Joint Conference on Autonomous Agents and Multi-Agent Systems*, New York, USA, 2002. ACM Press.
32. M. J. Wooldridge. Semantic issues in the verification of agent communication languages. *J. Autonomous Agents and Multi-Agent Systems*, 3(1):9–31, 2000.