The 2004 MetroBots Four-legged League Team

Vanessa Frias-Martinez[†] Rachel Goldman[†]
Marek Marcinkiewicz[‡] Simon Parsons[‡,§]
Elizabeth Sklar[†]

[†]Department of Computer Science
Columbia University
1214 Amsterdam Avenue
New York, NY 11025
vf2001@columbia.edu
rg2020@barnard.edu
sklar@cs.columbia.edu

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

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

Abstract

This paper describes the MetroBots Four-legged league team that participated in RoboCup 2004. The MetroBots team was formed in September 2002 and so this was our second 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 Historical Background

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 at CMU in April/May[2]. The further development of the team

[1]In alphabetical order.
[2]http://www.americanopen03.org/

following the American Open in 2002, has been the work of the Columbia and CUNY members.

In 2003, in addition to the American Open, MetroBots took part in RoboCup 2003 in Padua[3] in July. In 2004, MetroBots took part in both the 2004 US Open at the University of New Orleans[4] in April, and RoboCup 2004 in Lisbon in July[5].

2 Overview

In 2003, we struggled, as any legged-league team that starts from scratch does, to produce a team that was able to play a minimal game of soccer. Although we managed to win three of the eight games we played in RoboCup competitions, this was more a reflection of the robustness of the simple "run and kick" playing strategy we adopted and the weakness of our opponents than a demonstration of the skill with which our team could play soccer. In particular our 2003 team was unable to localise, so any meaningful coordination was impossible, and inter robot communication was limited to exchanging roles depending on which robot thought it was closest to the ball. We were moderately successful in 2003 because our robots were good at seeing the ball, moving directly to it, and kicking it in the rough direction of the oppsoing goal (though we did not localize, we were able to establish the rough orientation of the robot from the markers it could see).

By the time of the competitions in 2004, our robots were able to localise, and, as a result we were able to have much more effective coordination. However, this advance was not reflected in the results we obtained—we did not win a competitive game in 2004. This was largely, we believe, due to our decision to continue using the ERS-210 robots (a decision that reflected our desire to work more on coordination rather than spend our limited resources porting our code to the ERS-7s). As is clear from most of the games played in 2004, the ERS-7 simply overpowers the 210—right from the kickoff a team composed of 210s is, metaphorically, on the back foot since they simply cannot get to the ball as quickly. Furthermore, the additional lighting that the ERS-7s required gave us huge vision problems (which, of course, were not revealed until the competitions)—we found many spurious identifications of the ball which frequently misled our robots during games exactly because of the strategy that we had adopted for the robots the previous year (and which we continued to use). In addition, our approach to localization proved to be less robust than we would have liked, and this rather negated any benefit it brought to the team.

Of course, had both our motion and our vision been stronger to begin with, it is quit possible that we would have overcome these handicaps.

The rest of the paper is structured as follows. Section 3 gives a brief overview of the operation of the code (in terms of the capabilities of the MetroBots robots

[3]`http://www.robocup.org/games/03Padova/317.html`
[4]`http://www.cs.uno.edu/~usopen04/`
[5]`http://www.robocup2004.pt/`

in 2004), and Section 4 describes in some detail the code release that we had by the end of RoboCup 2004 in Lisbon. Section 5 then goes into some detail about the features of that code we consider more interesting (the localization code, the coordination mechanism, and the vision code), and Section 6 describes the tools we developed to support our development of the team—a tool for colour calibration and one for reporting the state of the robot. Finally, Section 7 sketches current research directions around the MetroBots team, Section 8 looks towards some future research directions, and Section 9 concludes.

3 Basic capabilities

A high level, and frank, assessment of the 2004 MetroBots code is as follows.

As discussed in more detail below, the team attempts to play in the standard manner, with a "defensive supporter" (DS) (to use the CMPack '02 nomclamenture) which guards the goal, a "primary attacker" (PA) which moves towards the ball and tries to kick it towards the oppsoing team's goal, and an "offensive supporter"(OS) which takes up position near the opposing team's goal.

The PA, which is unchanged from last year, operates a state machine, but one that is purely behavior-based in the sense of [1]. The state it maintains is only intended to allow it to recall, for example, that it hasn't seen the ball for a few frames, and so should turn to look for it. There is no attempt to do any cognitive processing. Indeed, so intent is the PA on tracking the ball and heading towards it, that is make no attempt to check its location on the pitch, and frequently gets lost.

The OS and DS, in contrast, make extensive use of the ability to localize. Localization, described in more detail below, is a version of the Markov localization of [3, 7], and is sufficient to allow the OS and DS to move automatically (albeit too slowly for use during competitions) to the starting positions, and to move the robots into their role-positions during the game. The DS and OS then get involved in the game through role-switching. The nearest robot to the ball at any point in time takes on the role of PA, with the remaining two robots switching into the most appropriate other role—this is nicely illustrated in one of the videos at `http://agents.cs.columbia.edu/metrobots`. Furthermore, since the robots share information about the ball location, this switching will take place even when the robot that is to become PA cannot see the ball, though the failure of localization in much of our competitive play meant that this kind of ability was not seen much in anger.

4 The RoboCup 2004 code release

The version of the MetroBots code used at RoboCup[6] is structured as five modules, each of which is a C++ object:

1. Perception;

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

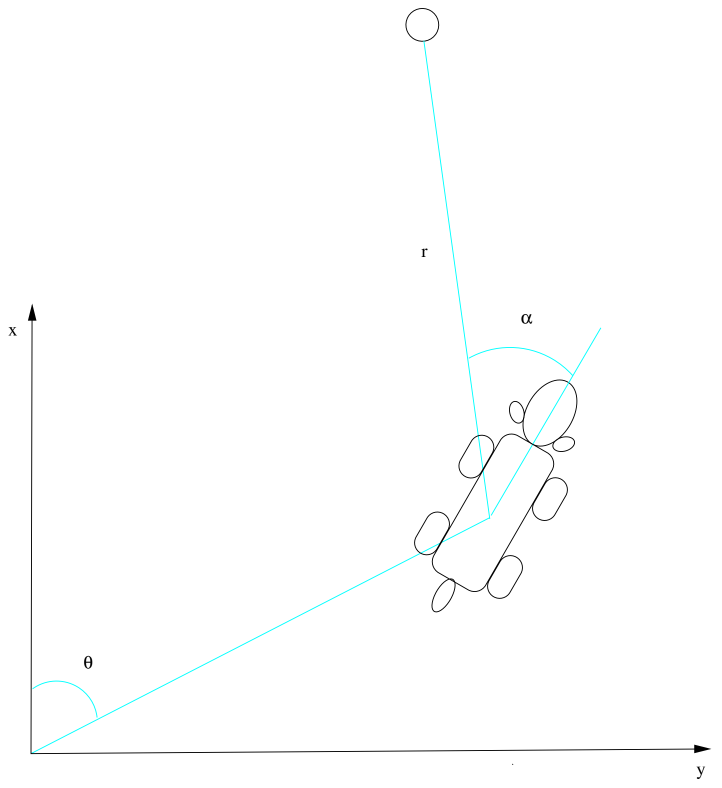


Figure 1: The orientation of the robot.

2. Localization;
3. Behaviour;
4. Communication;
5. Control.

These provide the following functionality.

Perception

The perception module takes as input a frame from the on-board camera, identifies objects within the 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. For the ball and the markers, he distance that is estimated is the distance r from the centre of the robot body, and the angle α 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). For the goal, just the identity of the goal is returned (we don't currently use the goals for localization, and indeed only use them when we are not localized to get an idea of which end of the pitch the robot is facing).

Figure 1 also shows the orientation of the robot with respect to the global coordinate system. $(0, 0)$ is at the marker to the left of the yellow goal (the left of the red goalkeeper that is), and x increases to the "north" and the blue goal, while y increases to the right of the red goalkeeper. The global heading of the robot θ is measured clockwise from the x axis.

Both the value of r and the value of α 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.

The distance calculations are more accurate than they were last year because of a change in the way that we calibrate and compute distance. The 2003 code assumed a linear relationship between the number of pixels seen and the distance of the object, and calibration was carried out at a single distance. Clearly one would expect this to be wrong—perceived area, which is proportional to the number of pixels, has a square relationship with distance. However, given that we were not trying to localise in 2003, this did not matter. This year distance was much more important, and so we now calibrate by taking several distance measurements, and fitting a curve to them. The resulting function is then used to establish the distance of objects perceived (this is the same mechanism as suggested in [29], developed developed in ignorance of the former work).

Localization

The localization module carries out a simple form of Markov localization [3, 7], with a grid that has 14×9 cells in the x/y plane and divides the absolute heading into eight equal slices. This corresponds to a cell size of 300×300mm, and an angular resolution of 45 degrees. The update algorithm is that suggested by [7]. Sensor noise is captured by applying a 2D guassian distribution to the values of d and θ returned by the Perception module, and noise in the motion model is captured by a 3D guassian applied to a noiseless motion prediction which itself is derived from the instructions sent to the motion module. This motion model is pre-compiled, when the robot boots up it a table is constructed that predicts the effect of every possible motion in every possible location on the pitch.

When stationary, the robot is able to localize fairly reliably, and we were able to use it to incorporate some basic intelligence into the behavior when the robot was moving—being able to move to the initial positions, being able to avoid tresspassing into the home goal area, and picking an appropriate kick for the rough location on the pitch.

The localization is not, however, performed with sufficient resolution (as is clear when considering the level of resolution suggested in [7] that it is any help to the goalkeeper (in fact using the localization in conjunction with the goalkeeper turned out to be detrimental). In addition, to be sure of localizing (as at the start of the game) the robots need to move very slowly—moving with speed they can easily become disorientated.

Behavior

As in 2003, the behavior module makes all the decisions about what the robot should do, making apprioriate calls to the motion code. Our code for 2004 used CMU's code for 2003 which we compiled separately from our source code and linked in. This gave us a wider range of possible motions than we had used in 2003, and we made use of a number of these new motion options.

The Behaviour code is still structured around the idea of roles—Goalkeeper, Primary Attacker (PA), Offensive Supporter (OS) and Defensive Supporter (DS)—and for each role it provides a set of behaviours. As mentioned above, the PA and goalkeeper roles were virtually unchanged from last year and continued to use the basic state machines we used then. The goalkeeper makes no attempt to localize. It just tries to stay within the its own goal, while moving to position itself between the ball and the goalline, and lunging if it detects the ball is within a certain distance. The PA heads for the ball and kicks it towards the opposing team's goal, using what information it can glean from localization to pick an appropriate kick (it kicks the ball harder when further from the opposing team's goal, and, when the kick allows, aims to kick the ball towards the center of the pitch).

Given even the coarse-grained localization we were running, it was possible to implement much more effective DS and OS roles than in 2003. The DS was programmed to head to a spot in front of the goal and in the middle of the pitch (in the y direction), to face the opposing teams's goal, and to look for the ball. The OS does much the same, but takes up position near to the opposing team's goal. Neither role attempts to move towards the ball, or kick it—the idea is that the role allocation would switch the robot to be the PA were it ever to become close to the ball.

The OS and DS roles were stateless, behavior-based (and essentially reactive), an experiment to see if this approach worked better than the state-machine approach. The stateless approach trades the complexity of figuring out what the robot does in a particular situation (which requires tracing a set of state transitions to figure out what state the robot is in) for the complexity of setting up the mutually exclusive conditions under which particular actions are undertaken. The conclusion was that it is helpful to maintain some state information, though maintaining some of the reactive structure is attractive.

In addition, all roles made use of localization information to try to avoid penalties for trespassing into their own goal area, by simply turning away when they detected that they were too close. This worked reasonably well for such a simple strategy.

Communication

The communication modeule is the same as in 2003. It 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. As in 2003 communication uses the TCPGateway.

Control

The control module provides the overall coordination between robots on the team, and exchange of information between them. 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 5.2 below, and the robots also exchange information on their locations and the locations of the ball.

5 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.

5.1 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[7] 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.

[7]As described below in Section 6.1, 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.

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 2DD 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. 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 basic mechanism is the same crude but effective approach we took in 2003, and for 2004 we augmented this with some simple knowledge-based segmentation.

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 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.

We do something similar for the ball. Any orange blob with more than 10 pixels is considered to be a candidate for the ball. But we also know that a ball is round, so long thin (in any orientation) orange blobs are discarded, as are any blobs that are too high up in the image, or blobs that are found to be adjacent yellow blobs. This latter filtering was necessary because of the problems we had with lighting this year which often mean that areas of the goal overlapped with the RGB ranges of the ball. While one might imagine that this kind of filtering would cause us problems when spotting the ball just in front of the yellow goal, this was not the case (possibly because we never managed to get in a scoring position).

5.2 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. In 2004 we made some progress in this direction.

We started from the basic mechanism that we had implemented in 2003. In that approach, 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 by the end of RoboCup 2003, the robots were tuned to achieve this reasonably well, 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.

With the ability to localize, more could be achieved. The first step (and the only one that was implemented on the robots in time for RoboCup 2004) was to pass more information between the robots while keeping the same role allocation mechanism as we used in 2003. Under this scheme, each robot now passes its own location and a global estimate of the position of the ball rather than just its distance to the ball. assimilating this information allows a robot to know its distance from the ball even when it cannot see the ball itself, and when the robots are well loclaised, this improves their performance—for example allowing a robot to know that it should turn and become primary attacker because the ball is just behind it.

We have also experimented in simulation with some additional approaches to coordination, and these are described in more detail below (Section 7).

5.3 Localization

As mentioned above, the 2004 MetroBots team incorporated a form of Markov localization [3, 7]. This code localizes against a 300×300 grid where the origin is in the corner with the yellow/pink (ie yellow at the top, pink underneath) marker. Since the 2004 pitch measured 4200mm by 2700mm, this is a 14 grid, and the cells are numbered 0 to 13 in the x direction (positive x direction also being known as north) and 0 to 8 in the y direction (east). The heading of the robot is quantised into 8 segments, numbered 0 to 7, and oriented such that the midpoint of 0 is due north (so that when the robot is pointing north, the orientation doesn't keep flipping from one value to another) and with the number increasing clockwise.

This is the internal representation used within the localization module. The values returned by localization are x and y values in millimetres, and heading in radians. The values returned are the midpoints of the corresponding cells, so that the heading, for example, is the angle θ in Figure 1

The algorithm that implements localization uses the usual:

```
**if** new observation $o_{t+1}$ is obtained.
    **for** every location $x^i$ in the grid at $t$
```

```
        $\mathsf{Bel}(x^i_{t+1}) := \Pr(o_{t+1} \mid x^i_{t+1}).\mathsf{Bel}(x^i_t)$
    normalize $\mathsf{Bel}(x_{t+1})$.

if new odometry information $a_t$ is obtained.
    for every location $x^i_{t+1}$ in the grid at $t$
        $\mathsf{Bel}(x^i_{t+1}) := \sum_j \Pr(x^i_{t+1} \mid x^j_t, a_t).\mathsf{Bel}(x^j_t)$
```

but our implementation does not perform these steps at different times. Instead both updates occur together when a marker is observed—that is first we update with the last motion command, and then we update the probability of every cell in the grid with the probability that the robot is in that location given the latest observation.

As mentioned above, we use a simple motion model which assumes that the robot will always move from the cell it was in to one an adjacent one in the direction of motion (both for translation and rotation). Though this model is then randomized—in the sense that we assign a probability to both the cell that this model predicts that the robot will move to plus the surrounding cells—in retrospect it is probably too simple a model. As a result we have spent time between RoboCup 2004 and the writing of this report extending the localization code.

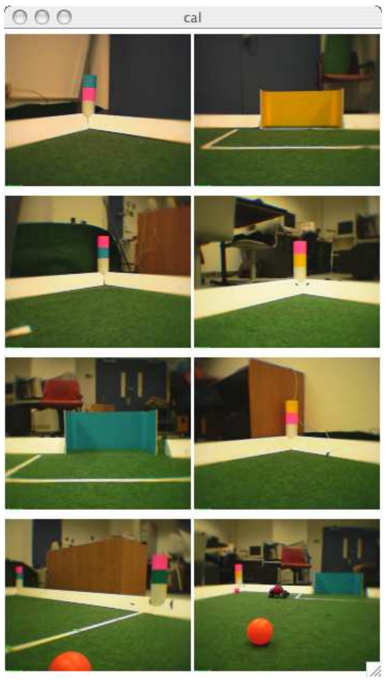
6 Tools

This section describes the two utilities that we have developed as part of our work on MetroBots.

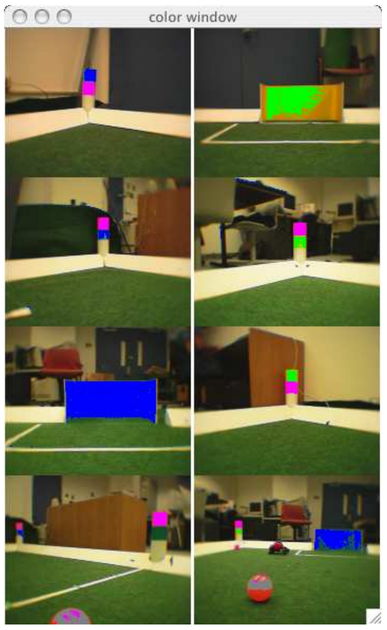
6.1 Colour calibration

When we first started work on the AIBOs, our object recognition code used YUV values (since that is the raw format the camera returns). 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 camera images, and displays these in two groups side by side, each group containing all eight images. The camera images are shown in figure 2(a). 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 Red, Green and Blue values of pixels in that area define the colour in question, and the calibration tool then highlights all pixels that fall within those combined ranges in the images in the right-hand group, as shown in figure 2(b). (Note that for visual clarity, the colors we use to highlight



(a) raw image

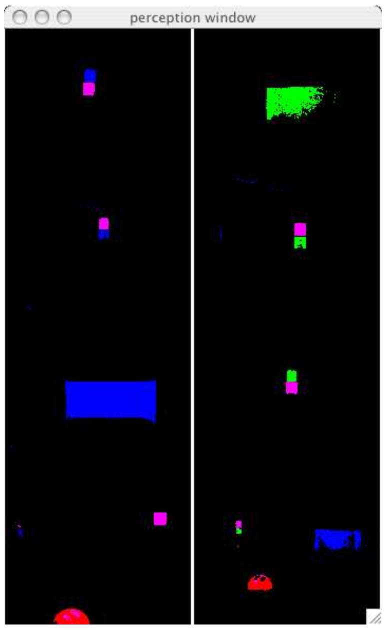


(b) highlighted colour ranges

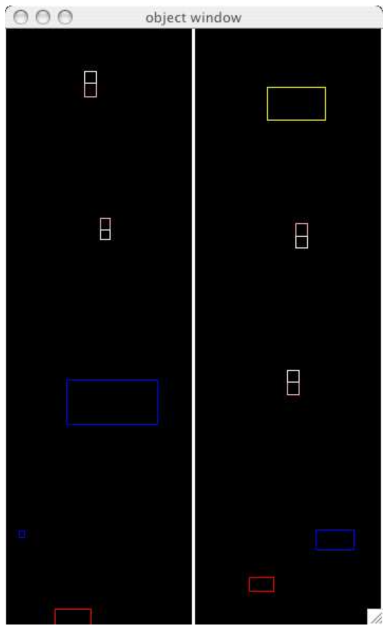
Figure 2: The colour calibration tool (part 1).

are not necessarily the same as the calibration colors; e.g., we use light green to highlight yellow.) 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).

This year, we expanded on our colour calibration tool to include output from our perception module. We realized that calibration is not just about finding the right colour ranges, but also about providing flexibility within those ranges when looking for certain types of objects. For example, the top of the ball is very often mistaken for yellow because of lighting conditions. We created two more windows, demonstrating two of the processing steps that are done on the AIBO with the calibrated colors. The first processing step is to identify "blobs" in the image, groups of connected pixels that match the calibrated colors; and then we filter out blobs that do not meet certain criteria (for example, blobs that are too small and blobs that have highly elongated shapes; the edge of the field, e.g., can give a long, thin line that is 1-pixel wide of falsely identified yellow pixels). This "blob perception" step is illustrated in figure 3(c). Then we need to correlate blobs with objects. Figure 3(d) shows how we have identified goals, markers and the ball within the sample images.



(c) blobs perceived



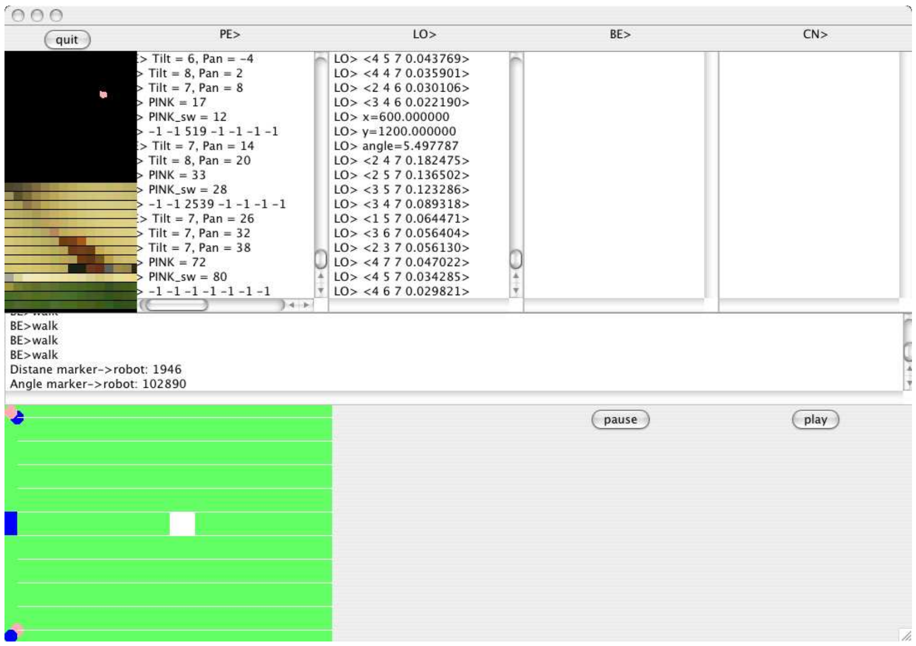
(d) objects identified

Figure 3: The colour calibration tool (part 2).

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 five colours—pink, yellow, 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.

6.2 The monitor

To help in debugging code, we developed a simple monitoring tool for viewing output from the robot. This exploits the `OSYSPRINT` command that OPEN-R provides as a means of sending output to port 59000, and a screenshot is provided in Figure 4. The tool is written in Java, and works by opening a Telnet connection to the robot in question, essentially using the usual:



```
> Tilt = 6, Pan = -4
> Tilt = 8, Pan = 2
> Tilt = 7, Pan = 8
> PINK = 17
> PINK_sw = 12
> -1 -1 519 -1 -1 -1 -1
> Tilt = 7, Pan = 14
> Tilt = 8, Pan = 20
> PINK = 33
> PINK_sw = 28
> -1 -1 2539 -1 -1 -1 -1
> Tilt = 7, Pan = 26
> Tilt = 7, Pan = 32
> Tilt = 7, Pan = 38
> PINK = 72
> PINK_sw = 80
> -1 -1 -1 -1 -1 -1 -1
```

```
LO> <4 5 7 0.043769>
LO> <4 4 7 0.035901>
LO> <2 4 6 0.030106>
LO> <3 4 6 0.022190>
LO> x=600.000000
LO> y=1200.000000
LO> angle=5.497787
LO> <2 4 7 0.182475>
LO> <2 5 7 0.136502>
LO> <3 5 7 0.123286>
LO> <3 4 7 0.089318>
LO> <1 5 7 0.064471>
LO> <3 6 7 0.056404>
LO> <2 3 7 0.056130>
LO> <4 7 7 0.047022>
LO> <4 5 7 0.034285>
LO> <4 6 7 0.029821>
```

```
BE>walk
BE>walk
BE>walk
Distane marker->robot: 1946
Angle marker->robot: 102890
```

Figure 4: The monitor.

```
telnet <ip address> -r 59000
```

but within a Java program. Then information that is output using `OSYSPRINT`s is collected and displayed by the monitor.

Since the idea is to support general debugging, there is no specific set of messages that are collected and displayed. Instead, we use the convention that the first two characters of any message denote the module from which the message is sent followed by the > prompt, e.g., PE> denotes messages from Perception, LO> denotes messages from Localization, CN> denotes messages from Control and CO> denotes messages from Communication. Messages with these initial sequences are displayed in dedicated windows (which can be seen in Figure 4), and messages without these sequences are displayed in another window—the programmer can thus customise the output in whatever way is useful.

In addition, there are two further aspects to the display. We have found it useful to have the Perception module stream the camera images that it processes back to the monitor (especially when it is used in playback mode, see below), and so every image is transmitted using the same `OSYSPRINT` mechanism. To avoid overloading the connection, only every tenth pixel is sent, and images are sent one line at a time, with each line prefixed by VI>. This information appears in the window at top left of the monitor.

The other aspect of the display concerns localization. As for many teams, we have found it useful to have a way of reconciling information about where the robot's localization mechanism thinks the robot is, and where the robot really is on the pitch—this is easiest to do if the robot's internal estimate of its location is displayed on a diagram of the pitch. The monitor accepts messages of the format

```
XY> N <x1 y1 a1 p1> ... <xn yn an pn>
```

as indicating that N tuples will sent, each stating an x coordinate, a y coordinate, and an angle coordinate and a probability (where the x, y and angular coordinates use the coordinate system described in Section 5.3). These locations are then displayed on the map of the pitch in the lower left of the monitor.

The monitor also accepts input from a file. This allows us to use it as a mechanism for viewing logs of practice games—during play the incoming data is dumped to a file, and can then be played back at leisure, and paused when required.

7 Research results

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 [32] for this problem. To do this, we have been building on our ongoing work on evolving auction mechanisms [24, 21, 22, 23].

In this vein, we view the role assignment mechanism described above as a merket-based mechanism in the following way. 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.

We have extended this simple mechanism in two ways. First [10], we have examined more complex mechanisms. We have delineated a space of possible auction mechanisms in which robots can bid for multiple roles (so that, for instance, a robot can indicate it most wants to be primary attacker, but, if it can't be primary attacker wishes to be offesive supporter), and can pass a variety of information in addition to what roles they want. We then examined how several of these mechanisms worked using a simple simulator. The aim of this work was to investigate whether sharing more information was advantageous, and the results seem to support this. One interesting feature of this work is that it also suggests that the best mechanism (of those we examined) is one which allocates the same role to several robots, rather than restricting any role to only one robot as in the mechanism we started with.

Second [9], we have looked at the evolution of new kinds of mechanism. In contrast to the work described in [24, 21, 22, 23], which uses genetic programming and reinforcement learning, the work in [9] codes the auction mechanism as a genetic algorithm, and evaluates the auction by running games between teams that use the evolved mechanism and those that don't, with the fitness being related to the performance of the team rather than individual agents.

8 Future research 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. Some of this work is described in more detail in [8].

8.1 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) [13], 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 [17] 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 [18], on the integration between BDI models and POMDPs [27], and the theoretical and empirical relationships between them [28]. The AIBOs seem to be an ideal platform on which to carry out further research on this topic, and one where the tradeoffs apparent in [28] will be important.

8.2 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 [14] 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 have traditionally been considered to be difficult to carry out 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 (though this conventional wisdom has recently been challenged by Peter Stone's group, which has carried out a range of learning tasks on AIBOs [6, 15, 16, 30]). 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.

8.3 Dialogues for robot coordination

When robots communicate during a soccer game, it can be helpful to have them share the reasons behind their actions [26], 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 [5]; Grosz and Kraus [11]; Reed [25]; and Sycara [31]. 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 [19, 20].

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.

8.4 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 [4, 12].

Our previous work [33] 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 [19, 20], 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.

9 Summary

This paper has described the MetroBots Four-legged league team that took part in RoboCup 2004 in Lisbon. The paper describes the overall structure of the code, what each module achieves, the aspects of our work that we think are most interesting, the tools we have developed, the RoboCup-related research work that we have carried out this year, and the research that we hope to get to next year.

The paper expands on the report we wrote in 2003, in the sense that it fully describes every aspect of this year's team. Rather than just describe the advances we have made since 2003, the covers those unchanged aspects of the 2003 work plus all the changes. The only parts of the description in this paper that is not in the description of the 2003 team are those things that have changed.

References

[1] R. Arkin. *Behavior-based robotics.* MIT Press, Cambridge, MA, 1999.

[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] W. Burgard, D. Fox, D. Hennig, and T. Schmidt. Estimating the absolute position of a mobile robot using position probability grids. In *Proceedings of the 14th National Conference on Artificial Intelligence*, 1996.

[4] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model checking.* MIT Press, Cambridge, MA, 2000.

[5] 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.

[6] P. Fidelman and P. Stone. Learning ball acquisition on a physical robot. In *2004 International Symposium on Robotics and Automation*, August 2004.

[7] D. Fox, W. Burgard, and S. Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391–427, 1999.

[8] V. Frias-Martinez, M. Marcinkiewicz, S. Parsons, and E. Sklar. Using multiagent coordination techniques in the robocup four-legged league. In *Proceedings of the AAAI Spring Symposium on Bridging the multiagent and multirobotic research gap*, Stanford, CA, 2004.

[9] V. Frias-Martinez and E. Sklar. A team-based co-evolutionary approach to multi agent learning. In *Proceedings of the Workshop on Learning and Evolution in Agent Based Systems*, New York, NY, 2004.

[10] V. Frias-Martinez, E. Sklar, and S. Parsons. Exploring auction mechanisms for role assignment in teams of autonomous robots. In *Proceedings of the RoboCup Symposium*, Lisbon, Portugal, 2004.

[11] 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.

[12] G. Holzmann. *Design and validation of computer protocols*. Prentice Hall, Hemel Hempstead, 1991.

[13] 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.

[14] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[15] N. Kohl and P. Stone. Machine learning for fast quadrupedal locomotion. In *The Nineteenth National Conference on Artificial Intelligence*, pages 611–616, July 2004.

[16] N. Kohl and P. Stone. Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2004.

[17] Michael L. Littman. Value-function reinforcement learning in Markov games. *Cognitive Systems Research*, 2(1):55–66, 2001.

[18] 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.

[19] 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.

[20] 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.

[21] 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*. Springer-Verlag, Berlin, Germany, 2002.

[22] 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.

[23] S. Phelps, S. Parsons, and P. McBurney. Automated trading agents versus virtual humans: an evolutionary game-theoretic comparison of two double-auction market designs. In P. Faratin and J. A. Rodriguez-Aguilar, editors, *Proceedings of Agent-Mediated E-Commerce VI: Theories for and Engineering of Distributed Mechanisms and Systems*, New Tork, NY, 2004.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] G. Simari and S. Parsons. On approximating the best decision for an autonomous agent. In S. Parsons and P. Gmytrasiewicz, editors, *Proceedings of the Workshop on Game Theoretic and Decision Theoretic Agents*, New Tork, NY, 2004.

[29] M. Sridharan, G. Kuhlmann, and P. Stone. Practical vision-based monte carlo localization on a legged robot. In *IEEE International Conference on Robotics and Automation*, April 2005. To appear.

[30] D. Stronger and P. Stone. Simultaneous calibration of action and sensor models on a mobile robot. In *IEEE International Conference on Robotics and Automation*, April 2005. To appear.

[31] K. Sycara. Argumentation: Planning other agents' plans. In *Proc. 11th Joint Conf. on AI*, pages 517–523, 1989.

[32] 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.

[33] 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.