

# Strategic behaviour-based reasoning with dynamic, partial information

James Westendorp<sup>†\*</sup>

Paul Scerri<sup>‡</sup>

Lawrence Cavedon<sup>†</sup>

<sup>†</sup>Dept. of Computer Science, R.M.I.T.  
GPO Box 2476V, Melbourne 3001, Australia  
{jhw, cavedon}@cs.rmit.edu.au

<sup>‡</sup> Dept. of Computer Science, Linköping University  
S-581 83 Linköping, Sweden  
pausc@ida.liu.se

**Abstract.** Behaviour-based agents rely on current perceptions for behaviour selection. However, this can cause problems in dynamic worlds where only partial information is available at any point in time, leading to unstable oscillation between behaviours. While retaining previous perceptual information can alleviate this problem, augmenting the behaviour selection mechanism with a classical approach to belief maintenance is not appropriate in truly dynamic environments, since the environment can change without the agent being aware of it. Using ideas from dynamic belief networks, we augment behaviour selection with information which is temporally degraded: i.e. information is retained from previous percepts but loses certainty, and influence on behaviour, as time passes. We have implemented this approach in the RoboCup simulation domain, using a hierarchical behaviour-based architecture: higher-level behaviours provide the agent with the capability to reason about strategy, but also require non-local information about the environment. We describe a series of controlled experiments which demonstrate that this general-purpose approach to information management addresses the specific problem of unstable behaviour oscillation, while also improving other aspects of the agent's performance.

## 1 Introduction

Complex simulated worlds provide important test-beds for autonomous agent systems, and the RoboCup soccer simulator [5] is becoming increasingly popular as an interesting domain for testing simulated agents. An important aspect of the RoboCup domain is that agents must combine fast reactive behaviour with the ability to perform long-term strategic reasoning. While *behaviour-based* architectures have been advocated as an approach suitable to satisfying both requirements [2], the problem of managing behaviour interactions so as to engineer long-term strategic behaviour can be extremely difficult [7].

We have addressed the problem by designing a hierarchical behaviour-based architecture, where higher-level behaviours control the strategic behaviour of

---

\* James Westendorp and Paul Scerri are both currently full-time students.

the agent.<sup>2</sup> These strategy-level behaviours activate only a select set of lower-level behaviours at any given instance. For example, the strategy-level *defend* behaviour will activate a low-level behaviour such as *pass* but not the *shoot-at-goal* behaviour. The final selected action executed by the agent will depend on the current situation it finds itself in: perceptual input is used to “excite” the various activated behaviours, and the one deemed most appropriate is selected for execution.

Strategy-level behaviours are similarly influenced by the use of perceptual input. However, the perceptual input appropriate to such behaviours is often more abstract and of a “non-local” nature. For example, the *defend* strategy may be chosen by an agent when the opposition is in possession of the ball and the opposition has more players in the attacking part of the field than the agent’s team has defenders. In RoboCup, however, only partial information about the world is present at any time: the simulated vision is limited by both an angular field of view and also distance from objects. This means, for example, that the agent may be unable to determine the identity of players in its defensive half to decide whether its team is outnumbered or not. Since behaviours are directly tied to perceptual input, this can lead to problems: the agent may find itself oscillating between behaviours in an unstable way. For example, a behaviour we observed was that a defensive agent would move towards its defensive half to check whether its team was outnumbered, discover that it wasn’t, move back towards a more attacking position, then immediately move back to check whether it was outnumbered in defence, and so on.

Rather than devising a special-purpose approach to the problem, such as adding a significant “inertial” component to the behaviour selection, we decided to modify the behaviour selection process to take account of previous perceptual input. However, given the dynamic nature of the RoboCup domain—in particular, the environment changes without the agent necessarily being aware of the change—older retained information is less reliable in its accuracy, and must have correspondingly less influence on the selection of behaviours. Hence, classical approaches to belief maintenance were inappropriate. The approach we were used was loosely based on ideas from *dynamic belief networks* [4, 8]. In particular, perceptual information is retained over time, but becomes less certain the older it is, leading to a notion of *temporal decay*.

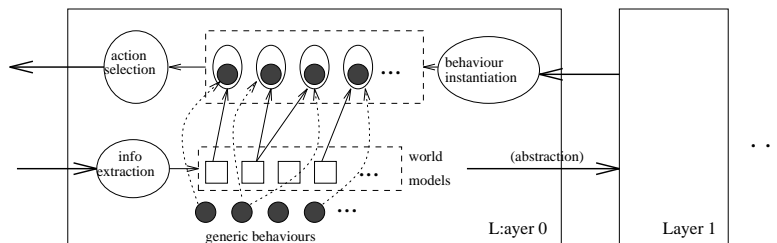
We have implemented a behaviour-based architecture, including the temporal decay aspect, in the RoboCup domain and performed a series of controlled experiments to test the effects of adding the temporal decay. We found that allowing temporally decayed information to influence behaviour selection significantly improved the performance of the system, both with respect to reducing the instability of the oscillation between behaviours, as well as certain aspects of the agents’ play.

---

<sup>2</sup> Hierarchical behaviour-based architectures have also been proposed by Tyrrell [10] and Blumberg [1].

## 2 A behaviour-based architecture for RoboCup

In this section, we describe the hierarchical behaviour-based architecture for the agent, minus the temporal decay component. The architecture uses a multi-layered approach, in which each layer is effectively an independent behaviour-based system, as shown in Figure 1. Each layer has its own set of behaviours, and all the layers run asynchronously. This was designed to simplify the management of behaviour interaction, as each layer contains only a small number of active behaviours at any given point. Behaviour selection is performed on the basis of level of activation (c.f. [6]), a main factor of which is perceptual input.



**Fig. 1.** The layered design of the behaviour-based system.

As shown in the figure, there are a number of *generic* behaviours, each of which controls a “type” of action (e.g. *kick-to-a-point*). Instantiating a behaviour involves selecting a generic behaviour and possibly supplying a parameter to it (e.g. the pair of coordinates of the point to kick to). This is described in more detail below.

### 2.1 General architecture

The multi-layer architecture consists of increasingly abstract behaviours as we move up the hierarchy. The lowest layer interacts directly with the RoboCup server, receiving percepts from and sending commands to the server. Each higher layer affects the layer directly below it, by instantiating new sets of behaviours for the lower layers. Behaviour instantiation is discussed in more detail below. Each layer also contains a control loop and a world model (discussed later). The control loop is responsible for selecting the most applicable behaviour and executing it, and also for telling the world model when to update itself.<sup>3</sup>

In the lowest layer, execution of a behaviour involves sending a command to the server, while at higher levels it causes a new set of behaviours to be instantiated in the layer immediately below. This results in changing sets of

<sup>3</sup> While this aspect of the system seems to contravene aspects of behaviour-based systems, it is appropriate for the RoboCup simulation environment, which is basically a discrete-event simulation system.

active behaviours throughout the life of the agent, which allows us to have a smaller set of behaviours for each level rather than having a single huge set of possible behaviours active at any given time in each layer. This reduces the problem of managing interactions between behaviours in such a way to produce the desired behaviour of the agent. It also allows for an easier way of providing strategic behaviour, as each strategy can instantiate a specific set of behaviours which are relevant to it.

At present there are three layers in the system. These are the *action*, *strategy* and *style* layers. The most abstract layer is the style layer. It contains behaviours related to the state of the game, such as *normalPlay*, *ourKickOff* and *theirCorner*. Each of these behaviours instantiates a new set of strategy level behaviours appropriate to the game state. For example *ourCorner* instantiates *attack* and *takeCorner*, while the style layer behaviour *theirKickOff* instantiates *defend* as the new behaviour in the strategy layer.

The strategy layer contains behaviours related to field-level strategies, which determine the player's overall position on the field. It contains behaviours such as *attack*, *defend* and *chase*, each of which creates a different set of lower behaviours in the action layer which are appropriate to the given strategy. For instance, *attack* will instantiate a kick for goal behaviour, *defend* won't.

The action layer is the lowest layer, and is the simplest both in behaviours and actions. It contains behaviours related to the performing of a single action. These behaviours roughly translate into one or two server commands. For example, the *moveTo* behaviour sends a *Turn* command followed by a *Dash* command to the server, while *kickTo* just sends a *Kick* command (with appropriate parameters).

Each layer also contains its own simple world model which contains information relevant to the behaviours in that layer. RoboCup percepts are discrete, not continuous, and all the layers of the agent run asynchronously, hence the percepts need to be processed and stored. Also, since the RoboCup server accepts actions at a faster rate than it provides percepts, the world information for the actions between perceptual cycles needs to be stored. Without temporal decay, the system uses a simple world model to store the information—each time a layer cycles through its control loop, it extracts appropriate information from the percept, and uses this to create a new world model for each layer. The information stored in higher-level layers is also increasingly abstract. For example, in the action layer, information about player positions is stored, while in the strategy layer, only information about the number of players in various zones of the field is stored.

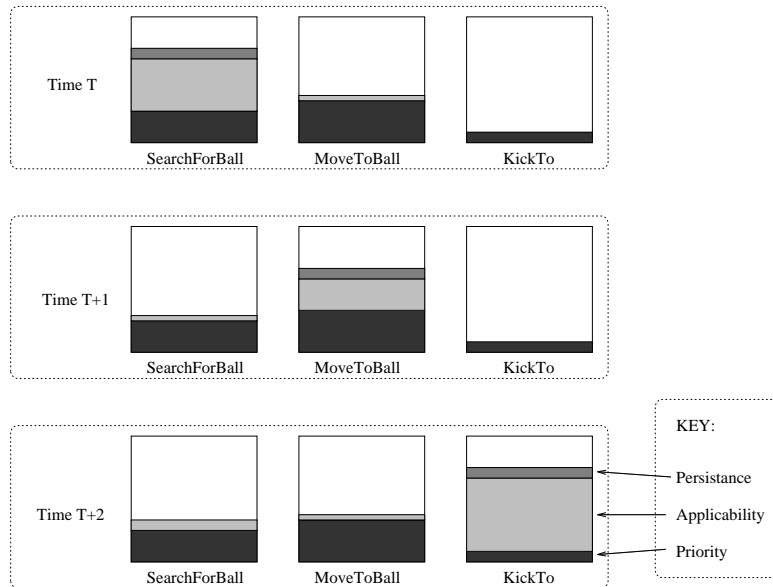
## 2.2 Behaviour selection

A behaviour consists of two parts, a *generic* part and a *parameter* part. The generic part provides the functionality of the behaviour, while the parameter gives specific values to each instance of the behaviour. For example, each *moveTo* behaviour causes the player to move, however the location the player moves to is determined by the parameters. This allows re-use of behaviour code, and also allows multiple instantiations of the same generic behaviour to be active at

once—e.g. there could be two different *moveTo* behaviours which have different position parameters active at one time.

Behaviour selection is based on the activation levels of the behaviours in a layer. The behaviour with the highest activation is selected and executed. Each behaviour’s activation level consists of three values which are added together: *applicability*, *priority* and *persistence*.<sup>4</sup> Applicability is a measure of how useful a behaviour is at any time. This depends on the current state of the world, and can change with each percept. Applicability is the means by which high reactivity is achieved, as a behaviour with low applicability in one turn may suddenly become highly applicable in the next as the world changes. For instance, *kickTo* may have a low applicability when the ball is not within kicking distance of the agent. However, if, on the next turn the ball moves near the player, the applicability increases, reflecting the increased usefulness of the behaviour.

Priority is a measure of how desirable the behaviour is overall. It is a constant value and remains fixed over the instantiation of a behaviour. Priorities allow flexibility in modelling the agent’s characteristics. For example, a player can be made more likely to be a defender than an attacker simply by modifying the priorities of *attack* and *defend*. Persistence is a small value added to a behaviour’s activation which decreases over time, as the behaviour remains the most active. This is intended to reduce oscillation in player behaviours, by making the currently active behaviour slightly more preferable.



**Fig. 2.** Example behaviour activations for a player over several turns

<sup>4</sup> A possible avenue for further experimentation is to explore more complex functions of combination.

An example of behaviour activation is illustrated in Figure 2. The situation is one where the ball is not initially visible. Of the various active behaviours, *kickTo* and *moveToBall* have little or no applicability since they require the ball to be seen. *searchForBall* has a high applicability and is the most activated behaviour, so it is executed this turn. At the next point, the ball is seen and the applicability of *searchForBall* drops. However, the ball is not within kicking distance, so the applicability of *kickTo* remains low, and *moveToBall* has a higher applicability. At time  $t + 2$ , the player has moved within kicking distance, and *kickTo* becomes the most highly applicable and a corresponding *Kick* command is sent to the RoboCup server for execution.

### 2.3 Engineering strategic behaviour

The hierarchical approach to the behaviour-based architecture facilitated simplified engineering of strategic behaviour. In particular, the particular strategy currently being pursued by the agent can be selected independently of the selection of the action-level behaviour. In a “flat” behaviour-based architecture, the system designer must identify the different influences, both from the current strategy as well as the perceptual inputs, on each individual behaviour and weight these inputs in such a way that leads to the appropriate low-level behaviour being selected. This becomes increasingly complex for agents which may pursue multiple “goals” simultaneously and is a recognised problem for such architectures [1, 7].

By allowing a strategy-level behaviour to instantiate multiple behaviours at the action level, the ability to react at the lowest level is not compromised: for example, the agent can switch between action-level behaviours without needing to rethink its strategy if it needs to in order to react to a change in the world.

## 3 Extending the world models temporally

The basic agent system as described above was sometimes observed to have a problem with unstable oscillation between behaviours, particularly in the strategy layer. One possible approach to rectifying the problem was to simply increase the *persistence* factor of the overall activation. However, this was seen as ignoring the root of the problem, which we conjectured to be the lack of any retention of information over time. Behaviour selection is based on activation level, which is in turn based on the current world state. As players only receive a partial view of the world at any one time, the applicability of behaviours was not based on the actual world state, but only on the visible part of the world. This was noticed most obviously in the strategy layer, where the applicabilities of *attack* and *defend* are based on the spread of players across the entire field. For example, a player standing in defence sees that its team outnumbers the opposition in defence, so the applicability of *defend* drops, and *attack* becomes more highly activated. However, as the player moves into attack, it can no longer see the players in defence, and so *defend* becomes more activated. The player returns

to defence, and the cycle continues. Using persistence slowed the rate of oscillation but did not solve the problem, as oscillation is the symptom of a greater problem, which is the incomplete picture of the world which the player has.

To address this problem, the player needs a more complete picture of the world on which to base its action selection. This is most obviously built up by retaining perceptual information across time. However, RoboCup is a truly dynamic environment in that the agent is not necessarily informed of change. Hence, any retained perceptual information must decrease in its certainty as time passes. To model this, we use ideas from dynamic belief networks.

### 3.1 Dynamic belief networks and temporal decay

To model beliefs where reliability can change over time we use the notion of *temporal decay* of beliefs. This concept is adapted from *dynamic belief networks* [4, 8] which are themselves a type of *Bayesian* network to which has been added a temporal component. A dynamic belief network has many states, one for each time slice, and each state can itself be a Bayesian network.<sup>5</sup> Each state is connected to the previous state and receives the previous world state as evidence for the new state, along with the current perceptual information.

This passing of information forward through time requires a *temporal decay function*, which models the information’s loss of reliability over time as we move forward through time. Of course, it may be that the information loses no value over time, in which case it would be carried across with probability 1, but this is an unlikely situation in a dynamic world. We experimented with the use of different temporal decay functions. The simplest function involved linear decay over time: i.e. if  $t$  is the number of time-points since a percept was observed, then that percept’s reliability is given by  $f(t) = 1 - \frac{t}{5}$ .

A more accurate model of the reliability of old information was obtained by playing 10 games involving the original teams, logging player and ball movement. An analysis of the data led to an exponential decay function:  $f(t) = 1 - \frac{2^t}{10}$ . Of course, this function was constructed based on the movements of our original team only (and would therefore not necessarily be accurate for predicting reliability of information relating to other teams); however, we assumed that (in the absence of better models of prediction) this function would suffice for decay of all information for the purposes of our experiments.

As each percept is received, the appropriate information is extracted, “time-stamped”, and added to the world models. Old perceptual information is discarded when its reliability drops below a threshold lower bound.<sup>6</sup> For reasons of efficiency, little *data reconciliation* is performed—i.e. there is little checking that an object in the current percept is the same object in a previous percept. The only such reconciliation is when a player can be unambiguously identified

---

<sup>5</sup> Although we do not provide the full power of a dynamic belief network (i.e. each state is not a full Bayesian network), the architecture is designed in such a way as to leave open the possibility of such an extension.

<sup>6</sup> An alternative option was to keep a fixed-size “window” of states [4].

(e.g. by shirt-number), or when the object has moved only minimally between percepts and no other similar object is nearby. This means that some objects may influence behaviour selection more than once, via presence in the current percept and in previous percepts. However, we believe the experiments described below indicate that this is not a serious problem in the current implementation.

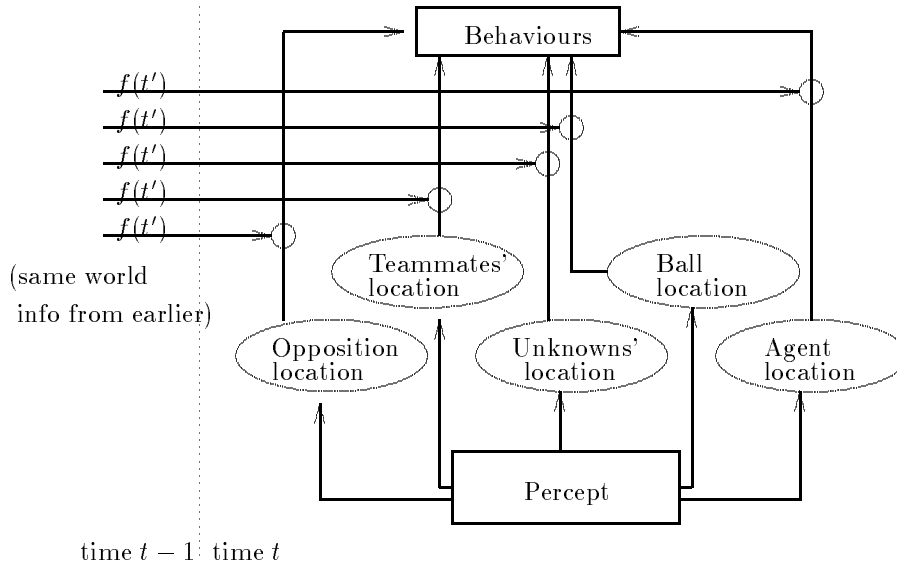
### 3.2 Influencing the behaviours

We extended the behaviour-based architecture described in the previous section so as to retain perceptual information over time, and to allow such information to influence the applicability of behaviours, although in a temporally degraded way—i.e. the older the perceptual information, the less influence it has on the applicability of the behaviours. This modification was made at each layer in the system: the process of abstraction of information from the current percept is performed independently at each layer (rather than abstracting information for the strategy layer from the world models of the action layer), avoiding problems with duplication of information at higher layers.

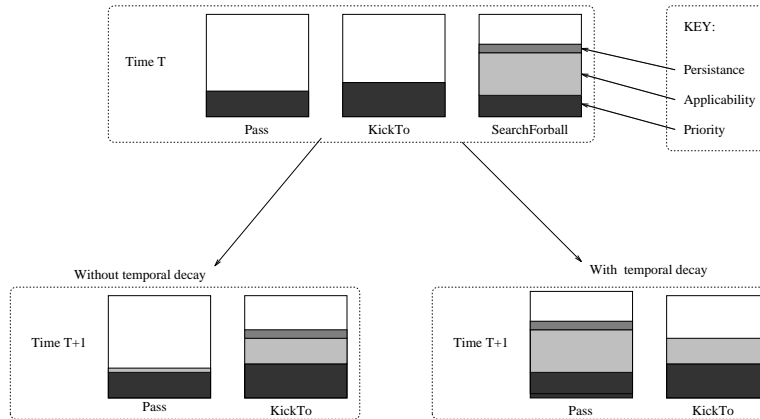
The effect of the retained information on behaviours is straight-forward: retained perceptual information influences the applicability of the relevant behaviours, after being appropriately decayed. For example, the behaviour to Pass to a position  $x$  may be applicable at time  $t$  because of some percept at some previous time which showed a teammate at  $x$ . The teammate may not have been seen at time  $t$  (e.g. if the player was facing a different way), but the retained perceptual information will still influence the Pass behaviour with strength  $f(t')$ , where  $f$  is the decay function and  $t'$  is the number of percepts received since the teammate was seen. Similarly, in the strategy layer, previously perceived information about the number of teammates in the defensive part of the field should keep the applicability of the *attack* behaviour high enough to prevent oscillations to *defend* when the player is too far upfield to perceive the identity of players. Figure 3 illustrates this organisation—in general, information retained from earlier than the immediately preceding time slice will also influence the selection of behaviour.

The influence of retained information can be illustrated by the earlier example of passing to a teammate that is not perceived in the current percept. In Figure 4, the situation on the lower left indicates where previous percepts are not retained and the applicability of the Pass behaviour is low and a KickTo behaviour (to a chosen position on the field) is executed. The situation on the lower right, on the other hand, illustrates how retaining the previous perceptual information increases the applicability of the Pass behaviour so that it is in fact selected.

Of course, as time passes, the reliability of the information about the presence of the teammate also drops, and the use of the temporal decay function causes the corresponding applicability of the Pass behaviour to also drop away (unless the teammate is perceived another time), leading to a point in the future where the KickTo behaviour is the more highly activated, and therefore selected for execution. Such behaviour was indeed observed when testing the system.



**Fig. 3.** Previous percepts influencing current behaviour selection



**Fig. 4.** Example behaviour activations with and without retention of previous percepts

### 3.3 Experimental evaluation

To evaluate the effect of allowing temporally decayed old information to affect behaviour selection, we have run a series of experiments.<sup>7</sup> The primary issue we were interested in was the stability of oscillation between behaviours: in particular, we expected to observe a given behaviour at the strategy level remaining

<sup>7</sup> Simulated domains are particularly conducive to such experiments, of course, allowing controlled environments and an opportunity to automatically gather a rich set of data.

active for more consecutive turns under the revised model. However, we also gathered data on other aspects relevant to the domain to test whether the revised agents performed better than the original ones.

The final list of test criteria for performance was the following:

- number of **goals scored**
- amount of game time (measured in turns) for which the team had **possession** of the ball
- the number of successful **passes** between teammates
- the **spreading** factor: i.e. whether players kept out of their teammates' way
- **stability** of strategy-level behaviours

As well as Stability, we were most particularly interested in the number of successful Passes and the Spreading of the players, since improvement on these criteria would indicate an improved "awareness" of other teammates, attributable to the improved perceptual processing.

We ran a series of controlled experiments, measuring the effect of allowing temporally decayed information to affect behaviour selection, with respect to the criteria listed above. The "control" team was one with no retained information: i.e. the decay function was set to 0. The control data was obtained by running 10 games of the control team playing against itself, with four players per team, giving data from 20 teams.<sup>8</sup>

Next, we obtained data on the teams which retained temporally decayed information. We ran 20 games each for the linear-decay function team against the control team, and for the exponential-decay function team against the control team. Again, in all cases there were four players per team.

The experimental results for the comparisons of linear-function team versus control team, exponential-function team versus control team, and linear-function team versus exponential-function team are shown in Tables 1, 2 and 3. The final column in each table shows the result of a *t-test*, a statistical test of significance appropriate for sample sizes of less than about 30.<sup>9</sup> For a standard *t-test*, a confidence factor (i.e. chance of error) of 0.05 or less is generally required to support a hypothesis; for a two-tailed *t-test* (which we have used), a level of 0.025 is usually required [3].

Measurable	Control Control		Linear Linear		t-test
	Mean	St Dev	Mean	St Dev	
Goals scored	1.85	1.31	1.85	1.31	0.500
Possession	335	143	312	116	0.287
Passes	16.6	6.37	20	5.4	0.061
Spreading	16.8	2.81	19.9	2.15	0.000
Stability	4.16	0.23	6.07	0.39	0.000

<sup>8</sup> Cohen [3, page116] suggests that, "If you cannot draw a reasonably certain conclusion from a sample size of 20 or 30, then perhaps there is no conclusion to be drawn".

<sup>9</sup> We used a two-sample, two-tailed *t-test* in each case, since we were comparing two samples with unknown direction of change of means.

Table 1: results for control team against linear-function team

Measurable	Control	Control	Expon	Expon	t-test
	Mean	St Dev	Mean	St Dev	
Goals scored	1.85	1.31	2.15	1.18	0.226
Possession	335	143	348	147	0.388
Passes	16.6	6.37	20.2	7.54	0.058
Spreading	16.8	2.81	19.3	3.08	0.005
Stability	4.16	0.23	5.47	0.25	0.000

Table 2: results for control team against exponential-function team

Measurable	Linear	Linear	Expon	Expon	t-test
	Mean	St Dev	Mean	St Dev	
Goals scored	1.85	1.31	2.15	1.18	0.226
Possession	312	116	348	147	0.194
Passes	20	5.4	20.2	7.54	0.387
Spreading	19.9	2.15	19.3	3.08	0.222
Stability	6.07	0.39	5.47	0.25	0.000

Table 3: results for linear-function team against exponential-function team

From the tables, we can see that both the linear- and exponential-function teams provide statistically significant improvements over the control team in the Stability of behaviours; interestingly, the linear-function team is a slight improvement on the exponential-function team. This indicates that the approach to behaviour stability via improved information processing is a sound one; however, it would seem that the choice of decay function used is not as important as the fact that one is used at all, since the carefully crafted exponential-function decay function performed no better than the simple linear decay function in the Stability (and not significantly better in other measurable, measurable) variables. Looking at the other significant behavioural improvements, there are improvements on all variables (except for Possession) over the control team. However, of these, only improved Spreading of players is statistically significant, although the increase in successful Passes is close to acceptable. These results suggest that the improved information processing had some benefit in the selection of appropriate behaviours. In particular, the simple model temporal decay does not lead to a *decrease* in performance, which is a possibility given the dynamic nature of the environment—e.g. if information is retained without decay of reliability, it may lead to inappropriate actions being executed, such as passing to a player that is no longer at the believed position.

Similar experiments were also performed with teams of various sizes, and it was found that the above improvements seemed to become more pronounced as team size grew, as would be expected since improved information management becomes more important when there are more objects. However, because of a lack of independence in the data (it was collected from opposing teams playing in the same game), we were hesitant to draw strong conclusions based on it.

## 4 Conclusion

We have implemented a behaviour-based system for the simulated RoboCup domain, using a hierarchically layered approach. This architecture was designed specifically to allow strategic behaviour to be more easily implemented in a behaviour-based system, alleviating the standard problem of managing the interaction of large numbers of behaviours.

An important extension to the basic architecture was to retain perceptual information across time, resolving the problem of having only partial information available in the percepts received from the RoboCup server. Because of the dynamic nature of the domain, information must be considered increasingly unreliable as it ages, even if the agent does not perceive anything that contradicts it. This was handled by allowing the influence of the information on behaviour selection to decay with time.

The improved information processing resulted in some definite improvements. In particular, unstable oscillations in behaviours were diminished and aspects of the agent's ability in the domain were also improved. Crucially, this was at no cost to the agent's ability to react quickly to the domain—an action is required from the agent by the RoboCup server each 0.1 of a second, and this requirement was always successfully met.

Further improvements to the system include experimentation to improve the behaviour selection function (e.g. different weightings on applicability) as well as more advanced information processing (e.g. better data reconciliation).

## References

1. B. Blumberg and T. Gaylean. Multi-level control of autonomous animated creatures for real-time virtual environments. In *Siggraph 95 Proceedings*, 1995.
2. R. Brooks. Intelligence without reason. In *International Joint Conference on Artificial Intelligence*, Sydney, 1991. (Computers and Thought lecture).
3. P. Cohen. *Empirical Methods for Artificial Intelligence*. MIT press, 1995.
4. T. L. Dean and M. P. Wellman. *Planning and Control*. Morgan Kaufman, San Mateo, CA, 1991.
5. H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa. Robocup: The robot world cup initiative. In *IJCAI-95 workshop on Entertainment and AI/A Life*, Montreal, 1995.
6. P. Maes. Situated agents can have goals. In P. Maes, editor, *Designing Autonomous Agents*. MIT Press, Cambridge, MA, 1990.
7. P. Maes. Modeling adaptive autonomous agents. *Artificial Life Journal*, 1, 1994.
8. A. Nicholson and J. Brady. Dynamic belief networks for discrete monitoring. *IEEE Systems, Man and Cybernetics*, 24(11):1593–1610, 1993.
9. S. Russell and P. Norvig. *Artificial Intelligence, A Modern Approach*. Prentice-Hall, 1995.
10. T. Tyrrell. *Computational mechanisms for action selection*. PhD thesis, Centre for Cognitive Science, University of Edinburgh, 1993.

This article was processed using the L<sup>A</sup>T<sub>E</sub>X macro package with LLNCS style