PizzaTower: Pisa's RoboCup team for Melbourne 2000

Riccardo Bernardini, Antonio Cisternino, Tommaso Schiavinotto,
Maria Simi, Alessandro Tommasi, Cesare Zavattari
Dipartimento di Informatica, Università di Pisa
{rbernard, cisterni, schiavin, simi, tommasi, zavatt}@di.unipi.it

Abstract

Our team for the international RoboCup challenge in Melbourne 2000 is based on a layered architecture. The architecture is "vertical" in that it is only the bottom layer which outputs RoboCup actions, but its responsiveness is guaranteed by an interruption/failure notification system. Layers run concurrently as separated threads. Perceptions are abstracted and filtered by a memory module which is adaptive and predictive, in that it builds a model of the behavior of the players. This allows layers to perform high-level queries about the world status.

1 The Architecture

To avoid mixing up RoboCup-specific solutions and general AI techniques, which could be exported and reused, we made an effort to abstract the world, hiding most of the problems related to speed, communication, perception parsing and the like.

The design of the agent structure is based on the modules shown in Figure 1.

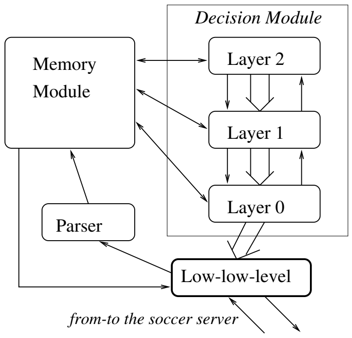

Figure 1: **the structure of a client with 3 layers**

Communication with the soccerserver is entirely filtered by the "low-low-level", which also synchronizes with it to make sure instructions are sent when the server expects them.

Perceptions, as soon as they are received, are passed from the low-low-level to the parser, implemented as a library which parses the perceptions in an efficient and versatile way using a finite state automaton. The parser triggers the appropriate memory update functions.

The decision module is structured in layers. Layers have an interface with the memory module, from which they obtain an enhanced and versatile view of the world status.

2 The Decision Module

2.1 The Layered Structure

The structure of an agent's decision module is an evolution of the one presented at Stockholm '99 by Cisternino and Simi for the IALP team [1]. The "intelligence" is split among hierarchical layers, which represent different levels of abstraction of the problem.

Conceptually, each level "solves", or implements, a set of actions, using as primitives the actions implemented by the underlying layers. The lowest level outputs basic RoboCup actions such as `kick`, `dash`, and so on, which need no further expansion. It implements basic skills, such as "head to point (x, y)", "find the ball", or "kick towards point (x, y)". The higher the layer, the more complex the skills that are implemented, like "tackle", "dribble", "evade your opponent", "pass", "take role R in scheme S" etc.

We can picture the layer structure as a stack of layers communicating via queues of directives which connect a layer to the underlying one. Each layer interprets the instruction it reads from the queue and generates a set of instructions to enqueue to the level underneath. The whole system is therefore organized as a *vertical* architecture [2], in that instructions flow from the top layer all the way down: no layer but the bottom one communicates with the soccerserver.

A strict top-down communication, however, is hardly an effective one. The world evolves in an unpredictable way; since a high-level action is expanded into possibly many primitive actions, which can take a long time to be completed, the world might change state so as to make it impossible or useless to complete the action. A strictly vertical architecture assumes the world to be "continuous". Top layers must assume that the world will be in a state where the actions they planned still make sense when those reach the bottom layer and are finally executed. In the case of RoboCup, the "continuity" is guaranteed by the physical laws that rule the world, but this does not ensure that a plan will still be worthy if performed with a delay.

A layer is not concerned with how the actions it enqueues are carried out, so it cannot judge the practical feasibility of the current plan. Failure must therefore be notified to the layer above by the layer which cannot find a suitable plan to implement an action. The layer receiving a failure from the layer underneath can bounce the failure up or handle it generating a different plan.

On the other hand, a layer which expands an action in several simpler actions is not concerned with the plan of the layers above it. It is therefore impossible for it to decide whether the action it is trying to expand is useful or not. This decision belongs to the layer which originated the action in the context of its plan. When a layer detects a change in the state of the world which makes the directive it enqueued to the layer underneath useless, it must interrupt it and replan. Interruption is usually forwarded down to the bottom layer, since replanning makes it useless to keep performing old actions.

This interruption and failure mechanism ensures that the layers are always running useful and feasible plans. This mechanism, provided that the layers are fast enough in computing a plan, guarantees the reactivity of the agent.

The communication interface between two subsequent layers is depicted in Figure 2.

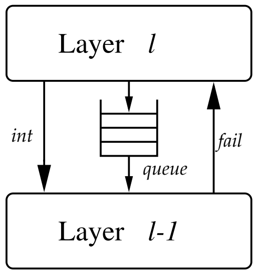

Figure 2: **interface between two layers**

2.2 Concurrency and Asynchronicity

It is likely that more abstract instructions take longer to expand, but they are not as time-critical as low level ones. Therefore, while the lowest layer must output basic RoboCup actions at each simulation cycle, the higher plans need not be as fast. Our approach is to run all the layers in parallel, dividing the simulation cycle in sub-cycles of computation, during which each layer proceeds with its state. The use of queues to communicate between layers gives higher layers more time to produce an action before the queue becomes empty.

Some degree of concurrency in the structure of an agent is not new to the RoboCup challenge but it is rarely as extreme as allowing the parallelization of virtually any skill level. In [3] it involves just the external *sense-think-act* cycle, divided in three threads, while in [4], although more similar to ours, concurrency is used to keep the bottom layer reactive, while doing higher-level reasoning above it. Our approach is more granular, making time-dependency of the tasks fade away with the height of the layer.

By introducing queues between layers, we aim at making the system as asynchronous as possible. Strict synchronicity is required only at the bottom layer: this is implemented using another queue from which an instruction is sent to the soccerserver at time intervals adjusted against the server running speed, as determined by `SENSE-BO-`

DY perceptions. This ensures that no cycle is lost, and that no instruction is sent before the previous one was received.

Concurrency is transparent to the layers, which need not be concerned with the time slice that is granted to them. This is done via thread programming, by a light thread library designed and implemented *ad-hoc*.

3 The Memory Module

3.1 The Representation Model

Representing the perceived information in a compact, yet efficient way is crucial to develop a client which goes beyond pure reactivity.

We have developed an abstraction at the memory representation level to provide each client with an effective snapshot of the world (the playing field). The representation is meant to be suitable for memory-level computations, such as prediction, remembering past positions, simple inductions over experiences and so on.

Positions are made absolute, so that each client can abstract from its position. To accomplish this, a flag and a line which are visible are used as navigation buoys, as described in [1]. Perception-related commands, such as TURN-NECK's are issued by this module to maximize the covering and precision of the snapshot.

We use a predictive and adaptive memory model to support high level decisions taken by the reasoning module during the game. Our model associates a bivariate Gaussian probability distribution to every mobile object in the field, representing the probability distribution of the object location.

This approach defines functions of the field to represent given properties like in reactive systems (for an example see [5]). However functions are used by the reasoning module to predict possible evolutions of the world like a typical deliberative agent would do. The shape of a bivariate Gaussian distribution is shown in figure 3

This distribution is obtained by combining two random variables X and Y that have a Gaussian distribution [6]. Five parameters fully describe an object: the two coordinates (x_0, y_0) which indicate the center of the distribution, the two variances σ_x and σ_y of the random variables and the angle θ by which the distribution is rotated.

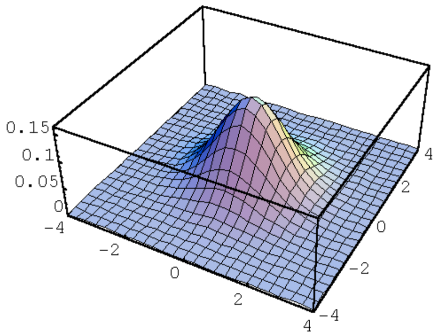

Figure 3: **bivariate Gaussian distribution**

A situation is defined as the set of the distributions assigned to the various objects, and is used by the agent to evaluate functions and predicates useful to take decisions. An example of situation is represented graphically in figure 4. The top two surfaces represent the distributions associated with the players of the two teams.

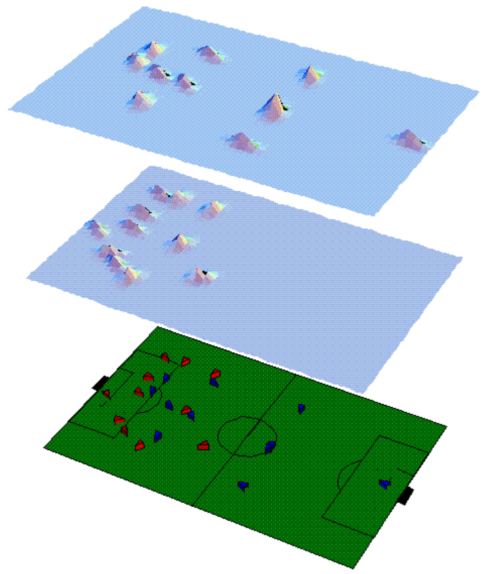

Figure 4: **snapshot of a player's knowledge of the world status**

An algebra to transform situations is used to predict evolutions of the current situation.

The operators affect the parameters that describe a Gaussian distribution. It is possible, for example, to *shift* the distribution by changing the coordinates (x_0, y_0). The shape of the distribution is controlled by operators that change the value of the two variances, σ_x and σ_y. The *unseen* operator increments both variances to estimate the location of objects that are not currently perceived. The *rotate* operator changes the orientation of the distribution.

We can update these parameters for each mobile object in the field in a very efficient way during the game, in order to update the representation of the current situation at each cycle of simulation. Moreover suitable operators can be used to predict future situations. For example the operator for shifting the distribution of a dynamic object, can be used to predict its future position, taking into account the current direction and speed of the object.

3.2 Learning the Operators

Once we have defined the operators that transform a situation, such as the shifting operator, our goal is to learn how these operators are to be applied in order to predict how the world evolves. In fact there are many ways to apply operators and only few of these are reasonable in a given situation: an object is subject to physical constraints (imposed by the server) and to convenience considerations.

We have identified three steps of increasing complexity for the task of learning how to apply operators to situations.

The first step is to learn common sense knowledge about simulated soccer: there are many facts that can be learned by observing typical game patterns. At the end of this step we expect our players be able to make a plausible simulation of the game starting from the current situation.

The basic algorithm used for learning a better behaved operator exploits playing experience from the logs. Starting from a known situation, it applies an operator obtaining a representation of the next situation. Then it compares the new situation to the one in the log to see how the game has actually evolved. The difference between the real situation and the predicted one is used to adjust the parameters.

The second step is to learn behaviors of a particular team in order to allow a finer prediction of opponent's actions. In this case the goal is to learn game patterns that are typical of that team. This is a form of opponent modeling.

Finally the third step is to adjust parameters during the game trying to adapt prediction to the effective strategy employed by the opponent team. Learning how a particular team behaves playing with our team makes our model adaptive. A possible approach for changing the parameters during the game can be the classification of the opponent's actions in a similar way as the CMUnited-99 does [7].

For the first two steps the learning process can be performed by using logs of games that are available on-line. Learning from the logs gives us the opportunity to observe a very large number of situations, and we have no time constraints on the learning algorithm used. Moreover it avoids the cost of generating learning examples and dependencies of the learned behavior on the timing of the particular network and workstations used.

The decision module of our agent uses the prediction module to simulate changes in the world; when the desired situation has been computed, appropriate predicates and functions are used to decide the player strategy and actions.

For instance, the decision module can ask to the prediction module the best direction to minimize the risk of losing the ball while heading for a point in the field. The prediction module uses its knowledge about the game to simulate the possible evolution; it then computes the probability for opponents to be in a given area and returns the area with the lowest value.

In order to do that we have found an exact and efficient formula to estimate the probability that an object is situated in a rectangular area by computing the integral of the distribution over the area.

Acknowledgement

We would like to thank Alessandro Sperduti, Paolo Ferragina and Francesco Romani for their technical advice.

References

[1] Cisternino A., Simi, M.: *Layered Reactive Planning in the IALP RoboCup team*, proceedings of the Third International Workshop on RoboCup, Stockholm, 1999.

[2] Brooks., R.A., *A Robust Layered Control System for a Mobile Robot*, A.I. Memo 864, Massachusetts Institute of Technology Artificial Intelligence Laboratory, 1985.

[3] Kostiadis, K., Hu, H.: *A Multi-threaded Approach to Simulated Soccer Agents for the RoboCup Competition*, proceedings of the Third International Workshop on RoboCup, Stockholm, 1999.

[4] Loureiro da Costa, A., Bittencourt, G.: *From a Concurrent Architecture to a Concurrent Autonomous Agents Architecture*, proceedings of the Third International Workshop on RoboCup, Stockholm, 1999.

[5] Riekki, J.: *Reactive Task Execution of a Mobile Robot*, Oulu University Press, Oulu, 1998.

[6] Papopulis, A.: *Probability, Random Variables, and Stochastic Processes*, Third Edition, McGraw-Hill, 1991.

[7] Stone, P., Veloso, M., Riley, P.: *The CMUnited-99 Simulator Team*, RoboCup-99 Team Descriptions, Simulation League, Coradeschi, S., Balch, T., Kraetzschmar, G., Stone, P., editors, Link oping University Press, 1999, 7-11.