Logic Based Hybrid Decision System for a Multi-robot Team

Vasco Pires, Miguel Arroz, Luis Custódio
Institute for Systems and Robotics
Instituto Superior Técnico, Av. Rovisco Pais, 1049-001 Lisboa, Portugal
{mbsa,vmicp}@rnl.ist.utl.pt, lmmc@isr.ist.utl.pt

Abstract. Nowadays, we can see an increasing amount of robotic systems for purposes of continuously growing complexity. Some of these applications require, not just a single robot, but a group or team of robots that must work together to accomplish the goal. This raises some problems related to the coordination of several robots that must be handled accordingly.

Robotics has improved in many ways. Mechanics are becoming faster, more precise, reliable and durable. Electronics provides greater autonomy, reliable sensors, and faster processing. Artificial vision endows robots with the ability to detect objects in the world by just *looking* at them with video cameras, with no special extra sensors. But, even with all this great new technology, a big problem remains: how to *decide* what the robot should do? A robotic arm can pick up a glass of water without spilling it, but *when* should it do that? *Why* should it do that? Will picking up the glass help to accomplish the goal?

This is why the link between Robotics and Artificial Intelligence (AI) is becoming more relevant every day. Robotics provides the means to interact with the real world. AI allows a robot to decide over that world.

In this paper a decision system for a team constituted by several robots is described. The system uses logic to choose the most suitable action in order to accomplish a pre-determined goal. It also supports cooperation to better achieve the goal.

1 Introduction

Most robots rely mainly on reactive decision systems. Those systems are usually based on simple and less flexible tools, like state machines or decision trees. Reactive systems are conceptually simple, but involve some serious practical problems. For example, the complexity of a state machine may grow rapidly for relatively simple behaviours. This will lead to big, hard to read and modify, and prone to error state machines. If we consider a multi-robot system the problems related with behaviour modeling with state machines turn out to be much worse. In this kind of system, if a reactive approach is used for decision-making, everything, including coordination among robots, ought to be pre-defined. So a robot can not anticipate the consequences of its actions neither the consequences of its team mates actions. A system with multiple robots using reactive decision systems behave much more like a bunch of individuals than a team. Robots using this kind of tools usually show very primitive behaviors, and are not able to accomplish non-trivial goals on complex, dynamic, and incomplete domains. On the other side, we have deliberative systems, which make decisions based on more

sophisticated tools, like a logical engine. These systems have many advantages over reactive ones. They have more expressive power, in the sense that they allow us to model things about the world that reactive systems cannot. A deliberative system can decide based upon past experience and by predicting future consequences of its actions. The system may even act and decide in a way that was not predicted by the designers, but that is actually valid and efficient in order to achieve the goal.

Of course, this advantage does not come for free. Deliberative systems are computationally heavy. A logical engine may need hours to make a single decision. CPU and memory usage cannot be underestimated for this kind of decision systems, even more in real-time domains. A robot cannot be stopped for hours (or even minutes, and, sometimes, not even seconds) to *think* what to do next. It must be permanently making decisions, and acting accordingly.

How can we take advantage of both kinds of systems? How can we make intelligent decisions, and, at the same time, guarantee that the robot will act in real time? A possible solution is an hybrid architecture [2]. With this architecture, we have both systems working in parallel, independently. The system uses, normally, the decisions made by the deliberative component. But, when the deliberative component takes too long to decide or an unexpected event happens, it uses the decision made by the reactive component. That decision is not as good as the one provided by the deliberative component, but it is better than doing nothing.

In this paper a decision system for a team constituted by several robots is described. The system uses logic to choose the most suitable action in order to accomplish a pre-determined goal. The paper structure is as follows: in section 2 the RoboCup domain is described; section 3 presents the proposed architecture and a detailed description of its components, and in section 4 we present some results and, in section 5, some conclusions and future work.

2 RoboCup Domain

Our application domain is the middle-size RoboCup soccer league[1]. A middle-size league team is constituted by four robots (generally, three field players and one goal-keeper). The robots diameter is about 40 cm, and the game is played on an (approximately) 10x5 meters field. The robots may communicate via Wireless Ethernet, and may also send information to a computer (external to the game) for monitoring and debugging purposes. According to the RoboCup rules, the external computer may send data or commands to robots, but the fundamental law of our team is that robots should be completely autonomous. So, our robotic team only relies on information gathered from on-board sensors.

The domain is very incomplete: it is really hard for a robot to have a complete representation of the world, because sensors are not perfect and have limited range. The domain is also very dynamic: all robots are moving, the ball keeps moving around the field, so there are a lot of things happening at the same time.

All these characteristics make the middle-size RoboCup league a very interesting and challenging domain for AI. In the next section we shall describe a functional decision-making architecture developed for tackling some of these problems. It is being applied and tested on a soccer robotic team composed of four Nomadic SuperScout II robots.[2]

[1] http://www.robocup.org

[2] http://socrob.isr.ist.utl.pt

3 Architecture

The hybrid architecture developed for our team, that enables us to use deliberation and reactiveness, is presented in Fig. 1.

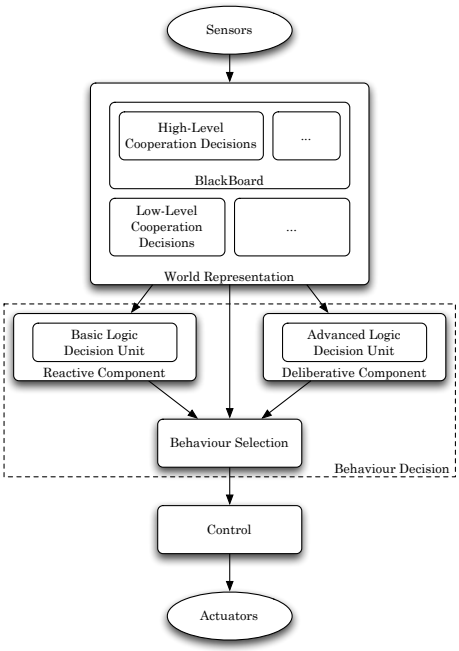

Figure 1: Hybrid Architecture.

This architecture is composed by several components, from which the most important ones are: World Representation, Reactive Component, Deliberative Component and Behavior Selection.

3.1 Behaviour Decision

This is the most important set of components for our work, where the robot behaviour is decided. It is easy to find a close resemblance with the generic hybrid architecture described in [2], with Reactive and Deliberative Components generating decisions that will be processed by the Behaviour Selecion component. The behaviors are then executed by the Control component.

3.1.1 Reactive Component

The reactive component has two main purposes: to guarantee a quick, real-time decision to be executed by the robot, and to react to unexpected events. For example, imagine one wants to take a glass of water to a friend. Her brain generates a plan (going to the kitchen, opening the cabinet, picking up a glass, filling it up with water, etc). If an unexpected event occurs, like dropping the glass, she does not wait for a new plan... she just tries to catch the falling glass [8]. On our domain, such things may happen very often: if a robot has the ball and plans to

take it to the goal, and score, it needs to react to an unexpected event (like an opposite robot taking the ball away from it, the robot loosing the ball, or bumping against another robot).

Until the work presented here, decision-making within our robotic soccer team was performed by a reactive system implemented as a state machine. This solution worked fine for a reduced number of behaviours and simple behaviour switching. When we started to design and implement more behaviours and, consequently, more sophisticated behaviour switching, we soon realized that state machines were not an adequate tool for our problem. State machines have a very serious practical problem: complexity. If you try to model a complex behaviour with a state machine, you will quickly get dozens or even hundreds of states, with a large number of arcs among them. It is easy to make errors (like deadlocks), and hard to read, understand and modify a state machine that big.

The first step of this work was to replace the state machine by a logic-based decision system, called *Basic Logic Decision Unit* (BLDU). This component, supported on first-order logic statements, allows us to easily model the reactive behaviour of the robot. The BLDU runs in a cyclic manner, and at each iteration it executes some operations related to the decision process and the world modeling. First, it broadcasts the status of the robot to its teammates. This includes the player role and the play mode (playing, paused, going to start position, etc). Then, it checks if it needs or wants to change its role. Finally, it decides the next behaviour to be executed (going to some place on the field, taking the ball to goal, etc).

The decision process is based on a set of Prolog[3] rules, constituted by a set of preconditions, that must all be true in order for that rule to be applied. The set of rules of the BLDU has a pre-defined order, in the sense that the first rule following that order that has all pre-conditions true will be the one used to make a decision (the consequence of the rule). A pre-condition is anything that can be represented by a logical formula, like having the ball, or being inside a pre-determined area of the field. In Fig. 2 there are two examples of rules used in BLDU. The first one is related with the defender role. This rule tells the control component to move the robot to a specific position on the field (given by X, Y, Theta) and it is applied if the robot is playing, it sees the ball but does not have it, the ball is outside the defenders zone, the robot is not near the position where he wants to go (a simple hysteresis), and there is not another robot in that position.[4] The second rule is related to the attacker, and tells the control component to score if the robot is playing, if the ball is in the attacker zone, if the robot can see the opposite goal, if it sees and have the ball, and if it is near the goal. This way, it is very easy to design new behaviors, simply by adding new rules to the system. The designer just needs to be careful with the rules precedences. This is clearly an easier system to work with, compared to the state machine.

One good example of the simplicity and power of the BLDU is the management of player roles. We defined three roles for our players: attacker, defender and full player. An attacker robot remains on the attacker midfield and tries to receive or get the ball, take it to the goal and score. A defender keeps itself between the ball and its goal, trying to intercept a possible shoot. Also, it tries to get the ball if the ball enters the defensive midfield. The full player role is used when, by some reason, there is only one player on the field. This player simply tries to get the ball, wherever it is, take it to the goal and score. It would be very hard to model

[3]The choice of the Prolog Language with it's backtracking inference system was based on the following facts: there are more start states than goal states; the branching factor is smaller if the search starts by the goal; and because the reasoning is triggered by a query and not by the appearance of new facts. All this makes backward inference more suitable for our application than forward inference.

[4]defenderTrackBall is a function that calculates a position between the ball and the robots goal.

```
basicBehaviour( goGeneric, Generic, X, Y,
Theta, defender) :-
  currentMode(play),
  visionSeeball,
  \+ hasBall,
  ballOutDefenderZone(now),
  plGetGenericConst(Generic),
  defenderTrackBall(X, Y, Theta, now),
  \+ xyInSideMargin(X, Y),
  \+ plRobotInPosition(X,Y).
```

```
basicBehaviour( score, 0, 0.0, 0.0,
0.0, attacker) :-
  currentMode( play ),
  \+ ballOutAttackerZone(now),
  \+ stateFinished( score ),
  visionSeeOthergoal(true),
  visionSeeball,
  hasBall,
  nearGoal.
```

Figure 2: Two example rules of the Basic Logic Decision Unit.

all these behaviors with a state machine, but it would be even harder to switch roles in real time, like BLDU does. This dynamic role switching is also an example of the the kind of cooperation we intend to develop with our architecture.

As stated before, the robot keeps checking if it needs or wants to switch roles. The need comes from two situations: if a robot stops playing (for instance, due to a referee decision, or a software crash), or if another robot decides to change its role. In the latter case, it may be necessary to switch roles in order to keep the strategy of the team. For example, if the team is playing with 2 attackers and one defender, and if the defender decides to switch to attacker, one of the attackers (and only one) will switch to defender, and take its teammate position. But when the robot decides it wants to change roles? Imagine a defender in a situation that the ball enters the defensive midfield. It will try to approach it, and take it to the opponent goal. But, trying to score a goal is an attacker's task, and, moreover, it would leave the team with no one protecting the goal. So, one of the teammates will switch to defender and go back in the field, protecting the goal again. The old defender may stay an attacker after a possible shoot to the opposite goal, it doesn't need to return to its old position and loose time and battery power.

3.1.2 Deliberative Component

The *Advanced Logic Based Unit* is the next step of our work. The ALBU is based on situational calculus [7][4], and on a planning system. The goal of this component is to determine plans (sequences of behaviours) that allow the team to achieve something (like scoring on the opposite goal). Situational calculus is an extension to first-order logic, specially developed to handle dynamic worlds. The changes in the world are the results of actions, that have pre-conditions and effects.

The development of the Advanced Logic Based Unit will be made in Golog [6], a language built on top of Prolog with the purpose of programming intelligent robot behaviour. This language allows us to use situation calculus in order to produce plans. We will study two forms of team planning: distributed planning and multi-agent planning. The first uses the resources of every robot to build a *cluster*, used to produce a plan. The second solution, more elegant, considers every robot as an individual, that proposes a plan. The final plan is built after some negotiation among all the robots.

3.1.3 Behavior Selection

Behaviour Selection (BS) will choose between the decisions produced by DC and RC. It is also the component that handles a plan execution and checks if it is still valid. If a plan is no longer valid (due to an action pre-condition being no longer true), it will discard the plan and use the reactive decision. This way, the robot may actually react to an unexpected event.

3.2 World Representation

The World Representation Component (WRC) is responsible to build a world model using sensorial data. From the sensory inputs and the static information about the game, the WRC builds the game model, that consists of basic information, like ball position and players postures, and advanced information, such as cooperative decisions. The variables used to define the world model are stored in a Blackboard. The Blackboard is a data pool accessible by several components, used to share data and exchange messages among them. Traditional blackboards are implemented by shared memories and daemons that awake in response to events such as the update of some particular data slot, so as to inform the components requiring that data updated. Some data of the blackboard is *local*, meaning that the associated information is only relevant for that robot, but other is *global*, so its update must be broadcasted to the other teammates (e.g., the ball position) [5].

The cooperation is divided in a high-level cooperation and a low-level cooperation. The data related to the former is stored in the Blackboard, and consists of *Group-Level* and *Team-Level Tactics*, that can be viewed as analogues of the coach's directives in real soccer. The *Group Level Tactics* defines tactical parameters for the different player groups: defense, midfield and attack. For instance, a good defensive tactic is to form a defensive line with the goalkeeper to block all paths to our goal. The *Team-Level Tactics* set general tactical conditions of the whole team. Parameters as basic formation, e.g. 2 defenders - 1 attacker, if we are in a defensive play, or 1 defender - 2 attackers, if we are in an offensive play. With our decision system, we may switch roles in real time, doing a better teamwork than just having fixed roles. The low-level cooperation data is outside the blackboard because it is a commitment between the robots that are involved in a cooperative action, e.g. when a robot tells another teammate to move to a certain position, in order to be able to receive a pass.

The remaining question is how the world representation is related with a knowledge base used by the inference engine. In fact they are very different in the sense that the world representation keeps numerical values, like ball position and team mates position, contrariwise the logic knowledge base contains beliefs that the robot (agent) has about the world. Each time the world representation is updated, the beliefs should be changed accordingly. We have to maintain a link between the world representation values and the knowledge base.

4 Results

In order to compare the BLDU with the state machine, we performed the following test: the robot starts facing the opposite goal, near the penalty mark. The robot must go back to the middle of the field, get the ball, return to the opposite goal and score. We applied this test ten times using each system (logic based decision system and the original state-machine decision system). The results are on Table 1.

Table 1: Average results for the tests using both decision systems.

Decision System	Ball Losses	Path Length (m)	Time (s)	Ball Out
Logic	0.8	9.28	30.5	0.2
State Machine	0.7	9.40	33	0.1

Not surprisingly the results obtained are similar for both systems, since the control unit is the same. For the robot performance on this test, the control unit is much more important than which tool is used for behavior switching. The logic-based system has showed to be fast enough (even a little faster than the state machine) to handle the low-level control problems, like behavior switching, with the right timings. So, we concluded that the BLDU may replace the state machine without negative consequences, and leave room to many improvements like the one described next.

Another result obtained within this work is the importance of playing as a team. Figure 3.a and 3.b shows the difference of playing with roles and without roles. The robots using the logic architecture maintain a formation in the field. In Figure 3.a, while the black robot takes the ball to the goal, the grey one stays at the defence between the ball and his goal in order to protect it. On the other hand, Figure 3.b shows the behavior without using roles. The black robot takes the ball to the opponent's goal and the grey robot gets lost in the field. Using a dynamic role change we get a team formation during the game. Figure 4 shows a dynamic change during the game, when the black robot (defender) catches the ball at the defence and takes it to the opponents goal, the grey robot (attacker) switches to defender to replace his teammate.

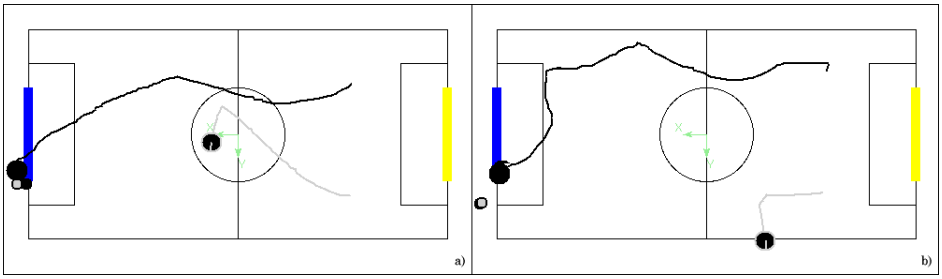

Figure 3: Robots behaviour with the new architecture (a) and the old one (b).

5 Conclusions and Future Work

The work done so far on the BLDU revealed how fast and simple is to model a robot high-level behaviour using a logic based system. With a few lines of code (no more than a printed page), we could make the robot behave as if it was being controlled by the state machine, even a little better. We created a reactive decision system with practical results way better than the state machine, while keeping the system simple, easy to modify and improve, contrariwise to what happens with the state machine. Also, we have concluded that the cooperation among

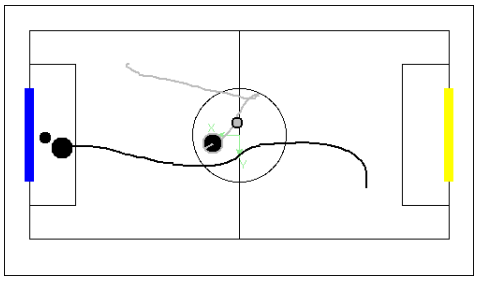

Figure 4: Dynamic role change.

the robots has showed itself very useful, delegating different tasks to each robot and keeping them working together to achieve the goal more easily.

Future work includes the development of the deliberative component of our architecture based on situational calculus. We will also improve the world representation by translating the perceived world status to a symbolic representation with the right characteristics needed to build versatile inference system.

Acknowledgement

This work was partially supported by the project ”RESCUE-Cooperative Navigation for Rescue Robots, FCT SRI/32546/99-00”.

References

[1] R. Reiter, Knowledge In Action, MIT Press, Cambridge, Massachusetts London, England 2001.

[2] M. Wooldridge, An Introduction to MultiAgent Systems, John Wiley & Sons, 2002.

[3] F.Dylla and A. Ferrein and G. Lakemeyer, Acting and Deliberating using Golog in Robotic Soccer - A Hybrid Architecture, Aachen University of Technology 52056 Aachen, Germany.

[4] C. Boutilier and R. Reiter and S. Thrun, Desicion-theoretic, high-level agent programming in the situation calculus, in AAAI’2000, 355 - 362.

[5] P. Lima, Current status of the SocRob Project, 2002.

[6] H. Levesque and R. Reiter and Y. Lesprance and F. Lin and R. Scherl, Golog: A logic programming language for dynamic domains. Journal of Logic Programming, 1997.

[7] J. McCarthy, Situations, Actions and Causal Laws, Technical report, Standford University, 1963. Reprinted in Semantic Information Processing (M. Minsky ed.), MIT Press, Cambridge, Mass., 1968, pages 410-417.

[8] R. Sadio and G. Tavares and R. Ventura and L. Custódio, An emotion-based agent architecture application with real robots, AAAI Fall Symposium, 2001.

[9] J. LeDoux, The Emotional Brain. Simon & Schuster, 1996.

[10] A. Kleiner and T. Buchhein, Simsrv, A RoboCup Simulator, Universities of Freiburg and Sttutgard, 2003.