STOCHASTIC DISCRETE EVENT MODEL OF A MULTI-ROBOT TEAM PLAYING AN ADVERSARIAL GAME

Bruno Damas [*,**] **Pedro Lima** [*]

[*] *Instituto de Sistemas e Robótica*
Instituto Superior Técnico
Av. Rovisco Pais, 1 – 1049-001 Lisboa, Portugal
Email: {bdamas,pal}@isr.ist.utl.pt
[**] *Escola Superior de Tecnologia*
Instituto Politécnico de Setúbal
Campus do IPS, Estefanilha, 2914-508 Setúbal, Portugal

Abstract: This paper introduces a method to model multi-robot teams using stochastic discrete event system techniques. The environment state space and robot behaviours are discretised and modelled by modular finite state automata (FSA). Then, all the FSA are composed to obtain the complete model of the team situated in its environment. Controllable and uncontrollable events are identified. Exponential distributions are assigned to the interevent times for uncontrollable events and stochastic dynamic programming is applied to the optimal selection of the controllable events. The method is illustrated by its application to a robotic football game. Simulation results are presented.

Keywords: Multi-Robot Systems, Discrete Event Systems, Dynamic Programming

1. INTRODUCTION

Finding the optimal sequence of actions a robotic agent should carry out in order to completely fulfil a given set of objectives, such as minimising a given cost function, or equivalently maximising an utility function, is a problem often not analytically solvable, due to partial and uncertain knowledge about the consequence of the robot actions over the surrounding environment. However, one can use an approximate world model, capturing its main features, and include uncertainty in such a model. In this paper, stochastic discrete event theory (Cassandras and Lafortune, 1999) is used for modelling and optimal decision making concerning a multi-robot team playing against an opponent team. The method is illustrated by an application to robotic football. This is a domain where probability distributions of event occurrences are strongly dependent on the continuous-valued state of the players and ball field positions and are not, in most cases, stationary with respect to time. Even so, the optimal solution for a given discretisation of this continuous state space, obtained by the use of an exponential timed stochastic model, may serve as a good guideline to design the agent decision scheme. Not many applications of discrete event systems to modelling and planning of robotic tasks have been described in the literature, and typically refer to the temporal specification, verification and code generation (Montano *et al.*, 2000), or coordination of sensing and control strategies (Kosecká, 1996), to name just a few. The work described here is rooted on previous work on the quantitative modelling of robotic tasks using Petri nets (check (Milutinovic and Lima, 2002) and the references therein).

2. FINITE STATE AUTOMATON MODEL

The robotic football game considered here consists of two players against two opponents. The field is divided in three distinct zones: *My Side*, *Their Side* and *Their Goal*. Each player can dribble, approach, face, shoot, cross, clear and move the ball around the field. Additionally, each player can have the ball under its control and may or may not see it at a given moment. The discretised environment states and the players behaviours are the discrete states of the finite state automaton (FSA) model of the game. Table 1 shows all the considered events, where pi means player i and opp$j$ means opponent j. More players could be considered, of course, and the field could also be discretised into a greater number of zones. Nevertheless, the options taken seem to be sufficient to illustrate the methods presented in this paper, while maintaining the resulting FSA small enough.

The model is mainly focused on the offensive capabilities of a football team — defensive abilities could certainly be included in this study, but the consequent automaton dimension would extent far beyond the scope of this work. Opponent capabilities are also extremely reduced in order to keep the model limited to a maximum number of a few thousand of different states. Basically, opponents just try to bring the ball to a field position as far as possible of their own goal. They are always in the field zone where the ball is currently standing, trying to get or to steal the ball when the ball is not on their team possession. The opponents model, despite its obvious simplicity (e.g., it does not take into account a possible score by the opponent team) is very challenging from the player point of view, as each opponent acts coherently in order to avoid the ball being near its goal. In fact, such opponents are harder to beat than other typical opponents, since they were designed to be near the ball all the time, while the considered team players are occasionally in a different field zone — sometimes even not being able to see the ball. Nonetheless, one must keep in mind that the methodology introduced here can be readily extended to more elaborated models of the football game.

From a practical standpoint, there is only a nondeterministic *shot* event: its division into *shot_b*, *shot_m* and *shot_g* (bad, medium or good shot) will be explained in Section 3. Note also that events *see_ball* and *dont_see_ball* are also split into three distinct events. This allows the use of context dependent variable transition rates, as explained later in this section.

Given a set of environment states, player behaviours and events, it is not a trivial task to combine them in order to get an automaton representing the desired system. Even using simplifications as those assumed in this work, a robotic football game model can easily reach a dimension of a few thousand different states. Building such an automaton from scratch is therefore out of question. The approach taken is

- to model the environment dynamics by one FSA per environment resource (e.g., the ball, the player position), where the FSA states represent environment discretised states;
- to model each player behaviour by an FSA, whose states represent the available behaviours;
- to compose the above smaller automata to model the complete robotic football game;
- to model constraints on the composition mentioned in the previous item by further composing the resulting FSA with automata representing the constraints.

For each of the above models, events are associated to the transitions between states. One may consider *uncontrollable* events, which occur without the players being able to disable them, and *controllable* events, which can be enabled by the players, representing actions that trigger appropriate behaviours. The uncertainty on the occurrence of the uncontrollable behaviours and the cost function to be minimised determine the optimal sequence of controllable events, corresponding to the optimal player behaviour. In the next sub sections we describe the approach taken to model the different sub-systems and the complete system.

Controllable events	Uncontrollable events
stop_pi	see_ball1_pi
move_my_side_pi	see_ball2_pi
move_their_side_pi	see_ball3_pi
move_their_goal_pi	dont_see_ball1_pi
get_ball_pi	dont_see_ball2_pi
face_ball_pi	dont_see_ball3_pi
shot_b_pi	got_ball_pi
shot_m_pi	lost_ball_pi
shot_g_pi	stole_ball_pi
center_pi	approached_ball_pi
clear_pi	reached_my_side_pi
dribble_my_side_pi	reached_their_side_pi
dribble_their_side_pi	reached_their_goal_pi
dribble_their_goal_pi	dribbled_my_side_pi
	dribbled_their_side_pi
	dribbled_their_goal_pi
	dribbled_my_side_opp$j$
	dribbled_their_side_opp$j$
	got_ball_opp$j$
	lost_ball_opp$j$
	stole_ball_opp$j$

Table 1. Events used in the robotic football model.

2.1 Ball Position

The ball can be placed in any of the previously presented field zones as a consequence of each player actions. The opponent players will only be allowed to dribble the ball in their opposite goal direction. Both players of the considered team, on the other side, can also shoot the ball, clear it in the goal direction or cross the ball to the middle of the field. Fig. 1 shows the automaton $Ball_{pos}$ (ball position). Note that $Ball_{pos}$ is non-deterministic due to the different consequences a shot can have in a given state. Although conceptually other events like *cross* or *clear* could be also non-deterministic, the option taken is sufficient to introduce a new factor of richness into the system. The state *Scored* is the only marked state of this automaton, as expected.

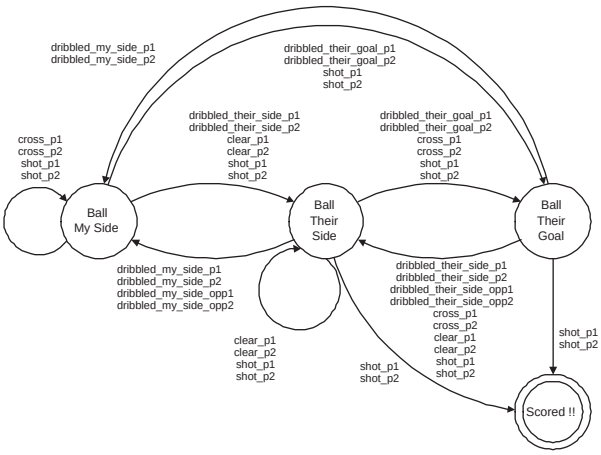

Fig. 1. $Ball_{pos}$, the ball position model.

2.2 Player and Opponent Ball Possession

Fig. 2 shows the ball possession model for player 1 and opponent 1. Player 2 and opponent 2 models are similar to these. A player or opponent, as can be seen in Fig. 2, will control the ball if it gets the ball or steals it from other player; it will lose this control if it loses the ball or if the ball is stolen by another player (note that the self loop on state *Don't have ball P1* is necessary in order to allow player 2 stealing the ball from opponent 1, for example). A controlled player can additionally lose this ball possession if it crosses, clears or shoots the ball.

2.3 Player Vision

Vision is modelled in a very simple way: a player either is seeing the ball or not seeing it. Opponents are assumed to be always watching the ball.

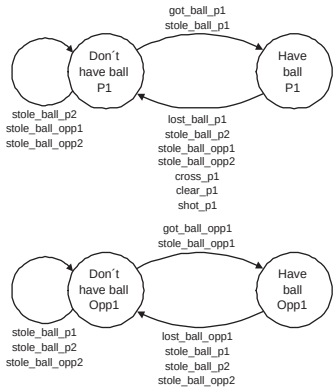

Fig. 2. $P1_{ball}$, player 1 ball possession model, and $Opp1_{ball}$, opponent 1 ball possession model.

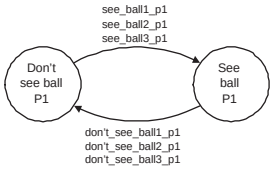

Fig. 3. $P1_{vis}$, player 1 vision model.

2.4 Player Position

The actions of the players also have an effect on their positions. Automaton $P1_{pos}$ corresponds to the different position states player 1 can assume as a result of dribbling or moving in the field. This automaton effectively maps the field of play, indicating, for a given position, the adjacent places where the player can go.

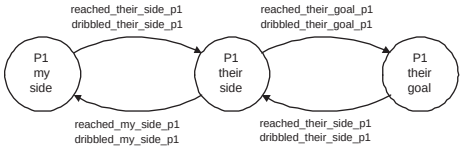

Fig. 4. $P1_{pos}$, player 1 field position model.

2.5 Ball Position Dependence

A few events cannot be easily included in the previous automaton, $P1_{pos}$. An example is the event *approached_ball*, although it does effectively change the player position. In fact, *approached_ball* needs a *context*, *i.e.*, the ball position is also needed to unequivocally determine the state that event leads into. Event *got_ball* needs a context too, since a player can only get the ball if it is in the same place the ball is. Since finite state automata are not context dependent, the only solution to implement such an event is to build an automaton that accounts for both the ball and player positions. This is accomplished by completing the parallel composition between $Ball_{pos}$ and $P1_{pos}$ with these contextual events. Besides *approached_ball* and *got_ball*, *see_ball* and

don't_see_ball are also dependent on both the ball and player positions: intuitively, if a player is near the ball it has more chances of seeing it than a player very far from it. In Section 3, the interevent occurrence times will be modelled stochastically as having an exponential distribution. However, no existing FSA formalism considers a variable exponential rate for interevent times, i.e., each event must be assigned a constant, state independent transition rate. Yet, suppose there are n different *see_ball* events, *i.e.*, "sub-events" see_ball_1, see_ball_2, ..., see_ball_n, behaving each one as the original *see_ball* event. The transition rate of *see_ball* can then be clearly controlled setting the number of active "sub-events" at each state. Enabling a large number of these "sub-events" at a given state effectively increases the *see_ball* event transition rate, since the sum of independent *Poisson* random variables is also *Poisson*, with transition rate equal to the sum of the rates of these variables. On the other hand, if one intends to make the ball hard to be seen at a given state, then a small number of "these sub-events" should be enabled. The same mechanism applies to the *don't_see_ball* event. Fig. 5 shows $P1_{b+p}$, the application of these context dependent events to the parallel composition of player 1 and ball position resulting automata. Note that some other events do not appear in the resulting automaton in order to improve readability.

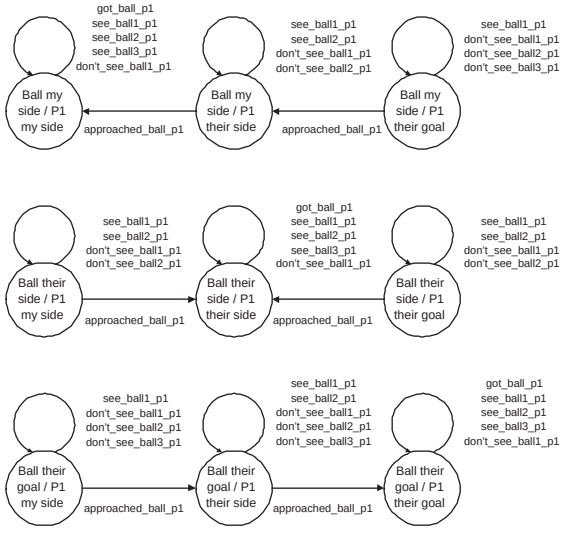

Fig. 5. $P1_{b+p}$, player 1 and ball positions dependencies (partial picture).

2.6 Player Behaviour

Each player is always performing a behaviour from a pre-specified set of different behaviours. Fig. 6 presents that set of behaviours, showing not only the events that start a given behaviour, but also the events that cause each behaviour to cease. Note that event *stolen_ball_p1* does not exist: it is only an abbreviation to *stole_ball_p2, stole_ball_opp1,stole_ball_opp2*. Obviously some of these behaviours need some preconditions in order to be executed: a player cannot dribble the ball if it does not have it. Such preconditions are imposed by the constraining automata presented in the next subsections. Note again that self-loops in some states represent event occurrences that do not change the FSA player behaviour but are relevant for its composition with the other FSA.

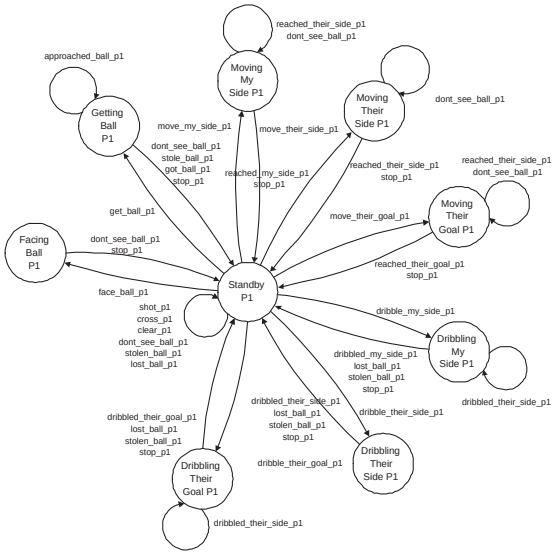

Fig. 6. $P1_{bhvr}$, player 1 behaviour model.

Parallel composition of previous automata generates an automaton that typically has a significant number of illegal states. Since it is not reasonable to examine each state individually to assert its admissibility, a set of additional automata are created to model the dependencies between the previous automata. This procedure is described in the following subsections.

2.7 Vision, Position and Ball Possession Constraints

Some events are only acceptable at a specific player vision state. Starting *Getting Ball* and *Facing Ball* behaviours is only allowed if the player effectively sees the ball. In the same way, a player can only get or steal the ball if it can see the ball. Automaton $P1'_{vis}$, the model of player 1 vision dependencies shown in Fig. 7(a), is in fact a "richer" version of $P1_{vis}$.

Additionally, the only restriction imposed to the player field movement is the impossibility of dribbling or moving to a field position where the player already is. Automata $P1_{pos}$ thus becomes $P1'_{pos}$, presented in Fig. 7(b). In this way only movements to different positions of the field are allowed.

A player cannot lose the ball from sight if it controls the ball. Getting or facing the ball when

the ball already is controlled by the player is also impossible. Automaton $P1'_{ball}$, shown in Fig. 7(c), takes care of these additional constraints. Note also that the main difference between dribbling and moving is that the former is carried out when a player has the ball, while the later is accomplished when the player does not control the ball. (Since opponents are only allowed to dribble the ball, the only change in $Opp1_{ball}$, not shown here, is the inclusion of events *dribbled_my_side_opp1* and *dribbled_their_side_opp1* as self loop transitions in state *Have ball Opp1*.)

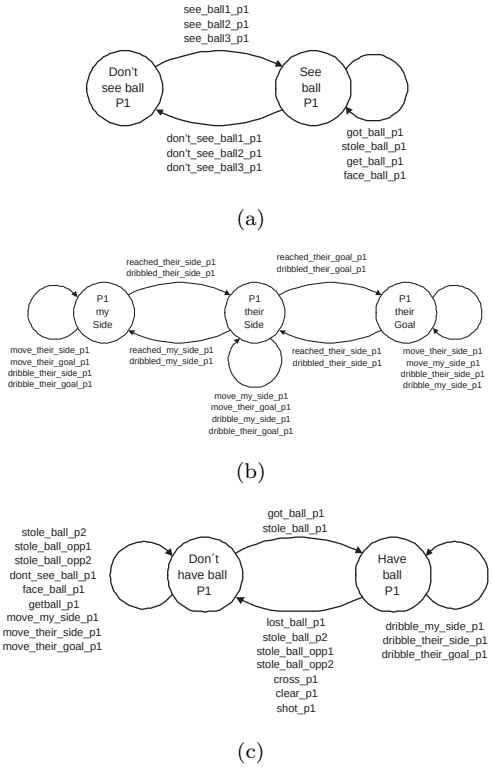

Fig. 7. $P1'_{vis}$, $P1'_{pos}$ and $P1'_{ball}$, respectively player 1 constrained vision, position and ball possession models.

2.8 Complete Model

After modelling all the "sub-automata" and the constraining automata, one can obtain the football game automaton, by a parallel composition between all those automata. After that the marked states are merged into a single marked state — the marked states correspond to a goal being scored by our team —, resulting in an automaton with 5617 different states.

3. OPTIMAL ACTION SELECTION

The principle of dynamic programming provides a method of obtaining the optimal sequence of actions that minimise a given cost function over a specified time interval (Bertsekas, 1987). This principle can readily be applied once an automaton representing the system under consideration is available. Since the ultimate goal of a robotic football game is scoring more goals than the opponent team, a behaviour selection policy that guarantees the minimum average time until the team scores is sought. This behaviour selection is cooperative, *i.e.*, the actions of the two players composing the team are considered as a whole, instead of referring to each player individually, and therefore an optimal sequence of action pairs (one per player) is to be obtained for the team.

A cost $C(j) = 0$ is assigned to every marked state j, while the other states get a cost $C(j) = 1$. The expected time spent at unmarked states by an automaton representing the match is then given by

$$E\left[\int_o^\infty C[X(t)]dt\right], \tag{1}$$

where $X(t)$ denotes the match state at time t. Furthermore, if the only marked states are those corresponding to a goal being scored and if there are no transitions from marked to unmarked states, then clearly (1) represents the desired expected time until the team scores. Equation (1) is appealing, since it corresponds to the total expected undiscounted cost criterion over an infinite horizon (Cassandras and Lafortune, 1999). Now suppose there is a set of actions the player can perform as soon as it reaches a state that affect not only the cost value at that state but also the transition probabilities leaving that state. The problem of interest is then determining a policy that minimises

$$E_\pi\left[\int_o^\infty C[X(t), u(t)]dt\right], \tag{2}$$

where $u(t)$ is a control action taken when a new state is entered that depends on that state and π is a given policy that determines the control action for each state. To do so the automata representing the game must be converted to a Continuous-Time Markov Chain (CTMC), assigning an exponential transition rate λ_i to each event i. Afterwards, this CTMC is converted into a Discrete-Time MC, applying an uniformisation method with uniform rate γ (Cassandras and Lafortune, 1999). Determining a policy to minimise (2) is equivalent to determining a policy that minimises

$$\frac{1}{\gamma} E_\pi\left[\sum_{k=0}^{\infty} C[X_k, u_k]\right]. \tag{3}$$

Since the cost is clearly bounded for every state and there is only a finite set of control actions U, then the minimum average time until a goal is scored is given by $T_{min} = \frac{1}{\gamma} \lim_{N\to\infty} V_N(i)$,

where i is the initial state and $V_N(i)$ is the N-step finite-horizon version of problem (3), defined recursively by

$$V_{k+1}(j) = \min_{u \in U_j} \left[C(j,u) + \sum_r p_{jr}(u) V_k(r) \right], \quad (4)$$

with $k = 0, ..., N-1$, and $V_0(j) = 0$ for all j. U_j is the set of admissible actions at state j and $p_{jr}(u)$ is the transition probability from state j to state r when control action u is applied. The best action to be performed at each state, in order to minimise T_{min}, is then equal to the action u that minimises (4).

Note that each action effectively leads to a state transition, rather than modifying directly the cost and/or the transition probabilities of that state. Assume $j' = f(j, s(u,j))$ is the new state, given by the automaton transition function, after applying the event sequence $s(u,j)$ at state j, generated by action u. Additionally, when dealing with non-deterministic events or non-deterministic actions (actions that can result in different sequences of events), an expected cost must be assumed. Eq (4) then becomes

$$V_{k+1}(j) = \min_{u \in U_j} \left[C(j') + \sum_r p_{j'r} V_k(r) \right], \quad (5)$$

with $k = 0, ..., N-1$, and $V_0(j) = 0$ for all j.

The main reason event *shot* was divided into *shot_b*, $shot_m$ and *shot_g* was the impossibility of performing a parallel composition between non-deterministic automata that share at least one event, as this can lead to a lost of synchronisation on that event. To overcome this problem it was decided to model the automaton as a deterministic one, creating events *shot_b*, $shot_m$ and *shot_g*. Randomness was then applied to action consequences: action *shoot* could randomly trigger one of the three mentioned events. Note that, in the end, there is no change to the automaton stochastic behaviour.

4. RESULTS AND CONCLUSIONS

To obtain the optimal set of actions exponential rates were assigned to each of the uncontrollable events, based on empirical considerations. The rates absolute value does not matter: what is important are the relative rates of different events, with high transition rates corresponding to short interevent times and low rates associated with long time intervals. These rates values were chosen to simulate a worst-case environment, where the ball is often lost and considerable time is taken to recover it back.

In order to obtain the optimal actions and respective state costs, equation (5) was successively applied until $\sum_j V_{k+1}(j) - \sum_j V_k(j) < 1 \times 10^{-5}$, this way assuring an accurate solution was obtained. Supposing the game starts with both players in the field zone denoted *My Side*, that the ball is in that zone too, that no player initially controls the ball neither sees it, then the minimum expected time until scoring a goal using such initial state is obtained at the 912^{th} iteration and is equal to $T_{min} = \frac{1}{\gamma} V_{912}(i) = 143.83$ seconds. 1000 experiences were then conducted using the obtained optimal policy on a simulator specifically developed in the context of this work. The observed average time until scoring a goal was 143.84 seconds, confirming the theoretical value. The player behaviour was the intuitively expected. It is curious to see how the optimal actions change when some of the event rates are modified. If dribbling the ball to the opposite goal becomes too slow, then the optimal action when the player has the ball is to shoot the ball or cross it to the opponents goal zone. However, if the probabilities of taking a good shot are lowered, then the player may want to dribble the ball again to the opposite goal.

This work represents preliminary steps on the modelling of robotic behaviour based on discrete event techniques, and the usage of such model for optimal decision making concerning the selection of the controllable events, therefore of the optimal behaviours given a state of the robot and its surrounding environment. Topics for future work will include the experimental determination of the interevent time rates by either Monte-Carlo "batch" methods of iterative methods, such as those based on Reinforcement Learning techniques (Sutton and Barto, 1999). The discretisation resolution of the environment should also be further studied as it exponentially affects the state space dimension.

REFERENCES

Bertsekas, D. (1987). *Dynamic Programming: Deterministic and Stochastic Models*. Prentice-Hall. Englewood Cliffs, NJ.

Cassandras, C. and S. Lafortune (1999). *Introduction to Discrete Event Systems*. Kluwer Academic Publishers.

Kosecká, J. (1996). A framework for modeling and verifying visually guided agents: Design, analysis and experiments.

Milutinovic, D. and P. Lima (2002). Petri net models of robotic tasks. In: *Proc. of IEEE 2002 Int. Conf. on Robotics and Automation (ICRA 2002)*.

Montano, L., F. García and J. Villarroel (2000). Using the time petri net formalism for specification, validation and code generation in robot-control applications. *The International Journal of Robotics Research* **19**(1), 59–76.

Sutton, R. and A. Barto (1999). *Reinforcement Learning - An Introduction*. The MIT Press.