Rule-Based Behavior Prediction of Opponent Agents Using Robocup 3D Soccer Simulation League Logfiles

Asma Sanam Larik and Sajjad Haider

Artificial Intelligence Lab., Faculty of Computer Science,
Institute of Business Administration, Garden Road, Karachi-74400, Pakistan
asma.sanam@khi.iba.edu.pk, sahaider@iba.edu.pk

Abstract. Opponent modeling in games deals with analyzing opponents' behavior and devising a winning strategy. In this paper we present an approach to model low level behavior of individual agents using Robocup Soccer Simulation 3D environment. In 2D League, the primitive actions of agents such as Kick, Turn and Dash are known and high level behaviors are derived using these low level behaviors. In 3D League, however, the problem is complex as actions are to be inferred by observing the game. Our approach, thus, serves as a middle tier in which we learn agent behavior by means of manual data tagging by an expert and then use the rules generated by the PART algorithm to predict opponent behavior. A parser has been written for extracting data from 3D logfiles, thus making our approach generalized. Experimental results on around 6000 records of 3D league matches show very promising results.

Keywords: Robocup Soccer, PART algorithm, opponent modeling, machine learning.

1 Introduction

Behavior mining[1] of agents can lead us to some insight into how agents interact in a dynamic environment. RoboCup Soccer [2] is a research initiative that uses the game of soccer to advance research within artificial intelligence, cognitive robotics, multi-agent systems and other related fields. The RoboCup Soccer competition comprises of several robot leagues namely humanoid league, standard platform league, small-size league, middle-size league and simulation league. The simulation league consists of both 2D and 3D agents and it aims to simulate a robot soccer match. Matches are played in a client/ server environment. The platform provides a test bed to analyze behavior of agents and teams.

During a soccer game, we want to predict our opponent's strategy and then adjust our own accordingly. If we are able to analyze the logs of past games played by teams and distinguish their game play based on their behaviors then it can aid us in devising our own team strategy. Much work has been done within the Simulation 2D league

L. Iliadis et al. (Eds.): AIAI 2012, IFIP AICT 381, pp. 285–295, 2012.

due to the simplicity of available actions such as Kick, Turn and Dash. In Simulation 3D league, however, such primitive actions are not available, thus it becomes difficult to perceive high level behavior. This paper presents a novel approach that aims to predict the behavior of agents in Simulation 3D league. Our approach is the first attempt to connect low level primitive actions to high level behaviors; thus enabling us to discover strategic activity from raw data. Our goal is to develop a mechanism that discovers key behavior of an agent and translates multi-agent action sequences and observations into a rule based representation. The proposed approach is divided into three phases: data preparation, rule generation and prediction. The first phase comprises of extracting raw data from 3D simulation league log files. We have created a tool that performs post-hoc offline analysis of the past matches and extracts agent and ball locations. We also derive distances of the ball/agent and their velocity. The extracted data is manually tagged for behavior identification by an expert by pressing appropriate key as he/she observers the game. In this paper we have restricted our focus to learning the skills of an attacker. The skill set includes two actions: Approach Ball and Dribble. The extracted and (manually) tagged data is then utilized in the second phase for learning rules. The rules learn particular behaviors depending upon the features used in the training dataset. Once rules have been learnt, these rules are applied to predict the behavior of opponent player in the third phase.

The rest of the paper is organized as follows. Section 2 provides a brief overview of RoboCup Soccer Simulation League. Section 3 provides a literature survey of the past efforts in opponent modeling. The proposed approach is explained in detail in Section 4 while Section 5 outlines the design of experiment and results. Finally, Section 6 concludes the paper and provides future research directions.

2 Robocup Soccer Simulation Leagues

RoboCup Soccer Simulation League provides a multi-agent system in which teams of autonomous agents play soccer in a simulated environment. All agents can move and act independently as long as they comply with the soccer rules. Agents can also have limited communication among each other. The league is further classified into 2D and 3D leagues. In Simulation 2D league, shown in Figure 1a, two teams of eleven autonomous wheeled robots play soccer in a two-dimensional virtual soccer stadium. The agents can perform low level actions such as turn, kick or dash to influence the environment. Simulation 3D league, on the other hand, adds the complexity of locomotion and localization and hence a lot of research has been focused on handling these issues. In Simulation 3D, shown in Figure 1b, two teams of 9 humanoid robots play soccer in a simulated environment. Unlike 2D, here a robot can only perceive objects that are in their own field of vision.

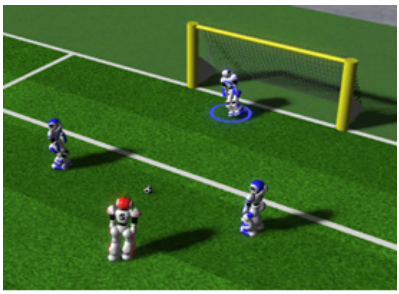

Fig. 1. a) Robocup Soccer Simulation Field 3D

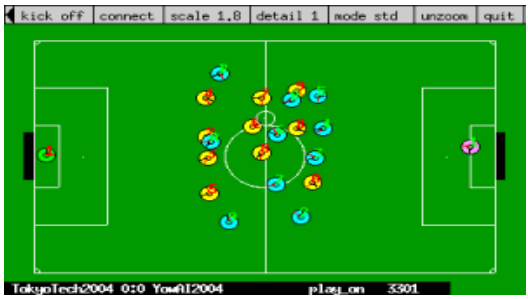

Fig. 1. b) Robocup Soccer Simulation Field 2D

3 Related Work

For our work we have studied papers contributed in three domains namely data extraction from log files, opponent modeling and papers on Robocup simulated coaching competitions. Although the area of data extraction in 2D RoboCup simulations has been extensively investigated, there is no work reported for 3D simulation environments. Within the 2D data extraction domain, T. Nakashima and H.Ishibuchi[3] have tried to mimic dribble trajectory of a player by taking snap shot of log files in which the player is actually performing the dribble action. The dribble intervals extracted create a set of training patterns for the neural network. T.Nakashima et al [4], have used offline learning for pass prediction behavior and predicted the new position of opponent player by training a neural network. Both the approaches are related to ours however we are using rule based learning for behavior prediction while they create neural network for this task. Within the opponent modeling domain, Ball and Wyeth[6] used Robocup small sized league to predict opponent behavior. Agapito et al.[7] presented OMBO, an opponent modeling approach based on observations. They first build a classifier to label opponent actions. In the next phase, they placed a dummy player in the field for the purpose of recording opponent actions and finally they predicted actions of opponent based on the training dataset. For the coaching competition, many simulated coaches have been presented; Fathzadeh et al.[8], Agapito et al.[9], Peter Stone at al[10][11] being some of them. The focus of these coaches is to learn normal base patterns of team plays and then predict the strategy with which the team is playing. The coach that recognizes more patterns wins. For this purpose they are using different data structures and pattern identification mechanisms. The coach advice is given in a particular language namely Clang[12].

4 Proposed Approach

This section presents our approach to discover and model low level behavior of opponent soccer agents. The approach is divided into the following three distinct phases namely:

- Extraction of information from 3D log files
- Rules learning
- Agent behavior prediction

In the first phase a parser is written that maintains a queue data structure that serves as a repository for storing noiseless contextual information. A log file is passed through the parser that extracts player's position and ball's position and populates it in separate lists. In the next step of this phase, behavior features, missing from the logfile, are incorporated by an expert. A simple application is written that helps an expert, visually watching the game, pressing some dedicated set of keys for recording the agent's behavior. This is essential for rules learning since the low level behavior (such as kicking, dribbling, etc.) is not recorded in the log files and is evident only to the individual watching the game. In the second phase we use the extracted (and tagged) data for learning a set of rules. Finally, in the third phase we parse an arbitrary log file (not in the training data set) and start predicting the agent behavior and verify visually whether the rules learnt are correct or not. For rule learning we have used the PART[13] algorithm. All phases are described in detail in the subsections below.

4.1 Extraction of Data from Logfiles

In Simulation 3D league, the server continuously records game state in a logfile during a match. The data from these logfiles can be extracted and analyzed to construct a model of opponent. The first phase focuses on data extraction from logfiles.

4.1.1 Data Format in Logfiles

Messages from server to agent and vice versa use S-Expression. The basic idea of S-Expression is that they are simple and are best known for their use in the Lisp family of programming languages. An advantage of using S-exp over other data formats is that it provides an easy to parse and compact syntax that to some extent is also readable by human for debugging purpose. Figure 2 shows an excerpt from a 3D logfile.

```
((FieldLength 21)(FieldWidth 14)(FieldHeight 40)(GoalWidth 2.1)(GoalDepth 0.6)(GoalHeight 0.8)(BorderSize 0)
((time 0))(RDS 0 1)((nd(nd))(nd(nd))(nd(nd StaticMesh (setVisible 1)))(nd(nd StaticMesh (setVisible 1)))(nd(
((FieldLength 21)(FieldWidth 14)(FieldHeight 40)(GoalWidth 2.1)(GoalDepth 0.6)(GoalHeight 0.8)(BorderSize 0)
((time 0))(RDS 0 1)((nd(nd))(nd(nd))(nd(nd StaticMesh))(nd(nd StaticMesh))(nd(nd(nd StaticMesh))(nd)(nd)(nd)
((time 0))(RDS 0 1)((nd(nd))(nd(nd))(nd(nd StaticMesh))(nd(nd StaticMesh))(nd(nd(nd StaticMesh))(nd)(nd)(nd)
```

Fig. 2. Sample Logfile Header

The messages recorded in the log are environment information messages, game state messages, Ruby scene graph header and scene graph contents. The scene graph is a structure that arranges logical and spatial representation of a graphical scene. In Simspark [14], the scene graph is a tree with a root node defined to be at the origin. Each node has one or more children. The nodes are further classified as base node, transformation nodes, geometry nodes, static mesh nodes, light nodes, etc. A header

expression is sent initially that contains information regarding environment, game state and scene graph contents. These variables are stored in the form of nodes with opening and closing brackets distinguishing among them. The initial header contains information messages including field length, field width, field height, goal width, goal depth, goal height, border size, free kick distance, wait before kickoff time, radium of the agent, ball radius, ball mass, , play mode (such as goal kick, play on, side kick, etc.), time, score, , SLT (single linear transform), light nodes, TRF (transformation matrices), etc. A new header is sent whenever the scene changes. For instance, loading of a player, removal of a player from field, etc., result in the transmission of a new header. However, if only the player or ball changes its position (and orientation) then no header is resent. Instead, only minor modifications in the nodes are represented by another s-expression that contains the time stamp and the change information that has been modified as evident in Figure 2. RDS attribute indicates that the scene has changed partially and only few nodes have changed. As a result, we traverse the entire S-Expression to find the node that has changed and update our values accordingly.

4.1.2 Parser for Data Extraction

In order to analyze the log data we first need a parser that can traverse each and every S-Expression and identify nodes for ball and players. The main challenges in writing a parser are: Extraction of node information from the header, Computing position of players and ball from transformation nodes, Extraction of updated nodes from the timestamp information

4.1.2.1 Extraction of Node Information from Header. To extract required information, we traverse the entire S-Expression, break the header information into tokens information and store tokens in a list $\top$. During this tokenization process, if we encounter any node **"nd"** we store its index in the list **I**. After the completion of tokenization process we populate the list of nodes **N** and players **P** respectively using the indexes stored in the list **I**.

4.1.2.2 Computing Positions of Players and Ball. The major challenge in extracting position information lies in the fact that in 3D this data is stored in the form of transformation matrix specified by token **"TRF".** A transformation matrix **T** is a 4*4 matrix defined as:

$$\mathbf{T}=\begin{bmatrix} nx & ox & ax & px \\ ny & oy & ay & py \\ nz & oz & az & pz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Where: $\vec{n}$,$\vec{o}$ and $\vec{a}$ vectors represent orientation information,
$\vec{p}$ represents the position information in x, y and z space.

In a Scene Graph there are a number of frames so a particular node can have many transformation matrices. To obtain correct position information we need to multiply all the specified 4*4 transformation matrices as demonstrated using the following

example. Let **"nd"** be a ball node containing the image information **"soccerball.obj"** and let **"TRF"** be its transformations and **SLT** be the single linear transformation. We get the the following information from the logfile:

(nd TRF (SLT 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1))(nd TRF (SLT 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0.0402764 1)(nd StaticMesh (setVisible 1) (load models/soccerball.obj)
In the above expression, we have two matrices T_1 and T_2:

$$\mathbf{T_1}=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix} \qquad \mathbf{T_2}=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0.0402764\\ 0 & 0 & 0 & 1\end{bmatrix}$$

When we multiply the two matrices we get the desired matrix and from that we can extract the position vector $\overleftarrow{p}$ **with** $(p_x, p_y, p_z)= (0, 0, 0.0402764)$.

4.1.2.3 Extraction of Updated Nodes from the Timestamp Information Sent. When we receive a timestamp and RDS node, this means that there has been a partial change in the scene and position/orientation of one or more nodes. In the S-Expression, the node structure is preserved and only the changed nodes contain the new transformation matrix. Thus, during the update process all node indexes remain the same and the node which contains some new **"SLT"** information is updated. A formal extraction process of the current and the previous subsection is described in detail in Table1.

Table 1. Algorithm

Data Extraction Algorithm

```
Read a new line from .logfile
 While (not end of file)
  Begin
     Parser algorithm
     Read Another line
  End
Write to csv file
```

Parser Algortihm:

Let $\mathbf{T}=\{t_1,t_2,\ldots..,t_n\}$ be a list of tokens

Let $\mathbf{P}=\{p_1, p_2,\ldots\ldots p_n\}$ be a list of players in the game
Let $\mathbf{I}=\{i_1,i_2,\ldots i_n\}$ be a list of node indexes
Let $\mathbf{N}=\{ n_1,n_2,\ldots n_n \}$ be list of nodes
where each node n_i contains the following attributes:

```
ni={ pos = node position, // 3*1 vector denoting (x,y,z)  coordinates
     name = node name,
     id = node id,
     mat= denoting 4*4 transformation matrix,
     type= type of node e.g SLT, SMN, StaticMesh, Light etc
     child= list of child nodes
    }
```

```
Begin
 1. Initialization Process: Set T= P = N = Null denoting Empty Lists
 2. Tokenization of file
       Foreach (element e in file separated by space)
          Begin
              T. Add(e)   // Add to tokens list
              If (e is a node) then
                I. Add(e)  // Add to node indexes list
           End
 3. If (RSG= true)  // The scene has changed totally
     Begin
                Foreach (element e in T )
                  Begin
                       If (e is a node) then
                         Create n_i  with attributes initialized
                      If (e is a SLT node) then
                         Compute transformation matrix of node
                       N.Add(n_i)
                    If (e is BallNode ball node ) then
                         Preserve its index in I
                     If (e is a PlayerNode) then
                         Preserve its index in I
                End
                 Add these nodes in I into player list P
                 Parse first player to identify team and player Id
                 Foreach  ( player p_i in P)
                    Compute their positions in the field
                 Compute the position of ball in the field
      End
 4. if (RDS= true) // the scene has partially changed
      Begin
         Foreach (node n_i in N)
           Update postion of n_i
         Foreach (index i in I)
            Recompute node indexes
         Foreach (player p_i in P)
            Recompute player positions in the field
        Recompute the position of ball in the field
      End
```

4.1.3 Preprocessing of Extracted Data and Derivation of Attributes

Given a logfile, our parser is able to extract positions of all the eighteen players of both the teams and the ball position. This extracted data is in the form of a csv file. Although the parser extracts data for all the players but for the scope of this paper we have limited our focus to the generation of data related to only one player, that is, opponent's attacker. We have used Roboviz [15] application for logfile generation with a single player, that is, the attacker taking the ball towards the goal. The parser extracts the following features:

- Timestamp
- Position of the opponent attacker
- Position of the ball

In addition to the extracted data, we derive the following attributes from the raw data:

- Distance travelled by ball (DTB)
- Distance travelled by player (DTP)
- Distance between player and ball at time t (DBPBat t)
- Distance between player and ball at time t-1 (DBPBat t-1)

4.1.4 Behavior Identification

As mentioned previously, logfiles do not provide us any details about the primitive actions executed by a player. In 3D we only learn about specific motion of hinge joints but the information is not enough to categorize this data into behaviors such as kick, approach ball, dribble, clear ball, etc. This information needs to be inferred by observing an agent's behavior. The information is also needed to label each record extracted in the previous step and to learn a classifier. To obtain this data we have written a piece of code that helps an expert, observing a game, to tell the behavior that he/she is observing. By manually pressing a key, an specific behavior performed by the robot agent a particular timestamp can be recorded. For the scope of this work, we have limited our attention to two behaviors: Dribble and Approach Ball.

- *Dribble*: is an event in which the ball and player both are moving and they are at a considerably shorter distance from each other.
- *Approach Ball*: In this event, the ball is stationary and at a farther distance, and the player is moving towards the ball.

Thus, we tag/label data with the help of expert. This behavior information is then merged with the previous features. The combined data can then used to generate rules in the next step.

4.2 Rule Generation

The csv file generated at the end of previous phase would be utilized in creation of rule base for agent behavior identification. The stronger the rule base, the greater would be the accuracy with which we would be able to predict agent behavior. Numerous algorithms that efficiently search large databases for rules have previously been developed. We use standard software named WEKA[16] and utilize PART[13] algorithm for rule generation. The reason for selection of this particular algorithm is that it generates rules in the form of decision tree thus making them quite understandable.

4.3 Agent Behavior Prediction

Once the rules have been learned, we can use them to predict the behavior of a player. To test the accuracy of the proposed approach, we record the logfile of a new game using RoboViz[15]. We use the parser to extract raw data. The behavior information would be tagged manually by visual verification. This combined data would serve as a test data.

5 Design of Experiment and Results

In our experiments, we placed a goalie of our team (Karachi Koalas) in the field whose sole purpose is to help us in recording the data. We ran attacker from another team whose job is to take the ball towards our goal. It must be mentioned that our goalie does not interfere in this process as it is primarily standing there as an observer. An expert watches the game and records key presses referring to events such as **"D"** for Dribble and **"A"** for Approach Ball. If the expert feels that some erroneous activity is being performed, for example, the agent has fallen, the agent is unable to locate the ball, etc. then the expert presses "E" to eliminate these erroneous episodes during data preprocessing phase. In a similar manner, we ran ten different games with five teams and recorded the attacker behavior of each and every team. In the next step, the logfiles were processed by the parser. The parser extracted the timestamp, opponent attacker's position and ball's position. The behavior information is merged with this data. The behavior became the class variable that we are going to predict. To obtain better rules, a pre-processing of data was performed as follows:

- Erroneous episodes such as getting up from back, falling, back walk recorded in the log and behavior files were removed
- All the pre-kickoff records were removed.
- To reduce the number of records, instances in which the play was at a stop or a player was idle were also eliminated.
- It was also observed that an event takes multiple cycles to execute thus the transactions were divided by 10. This made sure that very similar records do not disturb the correct identification of event.

We used feature selection algorithms provided by Weka[16] and after several experiments, the following attributes were retained: ballX, ballY, playerX, playerY, DBPBat t (Distance between player and ball at time t), DTP (Distance travelled by Player) and DTB(Distance travelled by Ball) respectively. Table 2 shows some tuples from the csv file.

Table 2. Demonstration of some training instances

Time	ballX	ballY	playerX	player	DBPBatt	DTB	DTP	Behavior
6.26	5.369	1.505	0.687	0.300	4.834	0.002	0.009	ApproachBall
6.46	5.425	1.488	0.843	0.327	4.727	0.020	0.033	ApproachBall
11.66	5.577	1.520	4.009	0.890	1.690	0.001	0.003	Dribble
11.86	5.580	1.492	4.139	0.922	1.550	0.006	0.033	Dribble

Next, we used the PART[13] algorithm for rules generation from approx 6000 tuples pertaining to ten games played by five different teams. Its configuration was set to a confidence factor of 0.25 and support of 30 rules after successive experimentations by varying these values. Some of the learned rules are shown below:

Rule 1:

If (DBPBat t > 1.615657 AND ballX > 5.354948 AND
ballX <= 5.757669) then ApproachBall

Rule 2:

If (DTP > 0.000391 AND ballY <= 0.148716 AND
ballY > -0.859947 AND ballX > 3.953067 AND
playerY > -1.286594 AND playerY <= -0.0246) then DribbleTowardsGoal

The rules proved to be 84% accurate when we used training data. The results obtained on training data are described in Table 3. Next we used training instances for a single game of approx 1000 records and obtained an accuracy of 80.4%. The results on test data are shown in Table 4.

Table 3. Results on training dataset

Behavior (class)	Instances (N)	Correctly classified (C)	Incorrectly classified (I)	Precision (P)	Recall (R)	F-measure (F)
Approach Ball	2456	1936	520	0.78	0.792	0.78
Dribble	3966	3459	507	0.87	0.869	0.869

Table 4. Results on test dataset

Behavior (class)	Instances (N)	Correctly classified (C)	Incorrectly classified (I)	Precision (P)	Recall (R)	F-measure (F)
Approach Ball	640	515	73	0.80	0.87	0.78
Dribble	423	298	125	0.70	0.80	0.746

6 Conclusion and Future Work

The paper presented an approach for rule based behavior classification of opponent agents in Robocup Soccer Simulation 3D environment. The behaviors classified are pertinent to the skills of the attacker namely Approach Ball and Dribble the ball towards goal. The proposed approach used a parser for positional data extraction and expert guidance for behavior identification and rule generation. The rules thus generated aided in predicting the behavior of an agent. The approach is first of its kind as it creates an opponent model based on 3D soccer simulation logfiles. The approach, being tested on approximately 6000 records, seems very promising and has generated good results. In the future we wish to extend this approach to classify if a team is playing in a defensive or an offensive manner. Similarly, it can also be used to distinguish between strong and weak teams. Furthermore, we can also learn the skills of goal keeper, defender and supporter using the same technique. In addition, currently the generated rules have crisp boundaries that somehow restrict the proposed approach. We aim to fuzzify the rules so that they become generalized and better readable by humans.

References

[1] Symeonidis, A., Mitkas, P.: A Methodology for Predicting Agent Behavior by the Use of Data Mining Techniques. In: Gorodetsky, V., Liu, J., Skormin, V.A. (eds.) AIS-ADM 2005. LNCS (LNAI), vol. 3505, pp. 161–174. Springer, Heidelberg (2005)

[2] Robocup official website, http://www.robocup.org

[3] Nakashima, T., Ishibuchi, H.: Mimicking Dribble Trajectories by Neural Networks for RoboCup Soccer Simulation. In: IEEE 22nd International Symposium on Intelligent Control, ISIC 2007, pp. 658–663. IEEE (2007)

[4] Nakashima, T., Uenishi, T., Narimoto, Y.: Off-line learning of soccer formations from game logs. In: World Automation Congress (WAC), pp. 1–6. IEEE (2010)

[5] Faria, B.M., Reis, L.P., Lau, N., Castillo, G.: Machine Learning algorithms applied to the classification of robotic soccer formations and opponent teams. In: 2010 IEEE Conference on Cybernetics and Intelligent Systems (CIS), pp. 344–349. IEEE (2010)

[6] Ball, D., Wyeth, G.: Classifying an opponents behavior in robot soccer. In: Proceedings of the Australasian Conference on Robotics and Automation, Australia (2003)

[7] Ledezma, A., Aler, R., Sanchis, A., Borrajo, D.: OMBO: An opponent modeling approach. AI Communications 22(1), 21–35 (2009)

[8] Fathzadeh, R., Mokhtari, V., Kangavari, M.R.: Opponent Provocation and Behavior Classification: A Machine Learning Approach. In: Visser, U., Ribeiro, F., Ohashi, T., Dellaert, F. (eds.) RoboCup 2007. LNCS (LNAI), vol. 5001, pp. 540–547. Springer, Heidelberg (2008)

[9] Iglesias, J.A., Ledezma, A., Sanchis, A.: CAOS Coach 2006 Simulation Team: An Opponent Modelling Approach. Computing and Informatics Journal 28(1), 57–80 (2009)

[10] Kuhlmann, G., Knox, W.B., Stone, P.: Know thine enemy: A champion RoboCup coach agent. In: Proceedings of the Twenty-First National Conference on Artificial Intelligence, pp. 1463–1468 (2006)

[11] Kuhlmann, G., Stone, P., Lallinger, J.: The UT Austin Villa 2003 Champion Simulator Coach: A Machine Learning Approach. In: Nardi, D., Riedmiller, M., Sammut, C., Santos-Victor, J. (eds.) RoboCup 2004. LNCS (LNAI), vol. 3276, pp. 636–644. Springer, Heidelberg (2005)

[12] Robocup Simulation Coach Competition, http://www.cs.utexas.edu/~ml/wasp/robocup-clang.html

[13] Eibe, F., Witten, I.H.: Generating Accurate Rule Sets without Global Optimization. In: Proceedings of the 15th International Conference on Machine Learning, San Francisco, USA (1998)

[14] Simspark, `http://simspark.sourceforge.net/wiki/index.php/Main_Page`

[15] RoboViz official webiste, `https://sites.google.com/site/umroboviz/usage/startup`

[16] Holmes, G., Donkin, A., Witten, I.H.: WEKA: A Machine Learning Workbench. In: Proceedings of Second Australia and New Zealand Conference on Intelligent Information Systems, Brisbane, Australia (1994)