Sharif Arvand Simulation Team

J. Habibi, M. Arasteh, P. Chubak, M. S. Javadi,
A. Heydarnoori, H. Soroush, and H. Younesi

Computer Engineering Department
Sharif University of Technology
habibi@sharif.edu
{arasteh,chubak,sjavadi,noori,soroush,younesi}@ce.sharif.edu

1 Introduction

Like most teams SharifArvand consists of three parts: WorldModel, skills, and strategy. Duty of the WorldModel is getting received messages from the Soccer Sever (SServer[1]), creating a virtual model of the game field and updating the information about each object. Other layers of the program will take the necessary information for playing soccer from the WorldModel in a convenient format. Also any command sent to the server should be reported to the WorldModel. Other parts of the player program are implemented in a multilevel manner. In other words, in the lowest level there are simple skills which only use the WorldModel for undertaking their duties, whereas more complex skills in higher levels use the WorldModel and the lower level skills to achieve their goal. Finally, a strategy uses these skills for playing a soccer game. So, a strategy is the highest level skill. Therefore, a strategy should create, control and destroy skills to play a successful soccer game.

2 WorldModel

The WorldModel is to know all the information about all the existing objects in the soccer field. For representing the amount of error for each data in the WorldModel, each information unit has a probability density function to show its correctness.

2.1 Information Related to Other Objects

To find information regarding other objects, WorldModel can use one or more of the methods explained below:

– Finding this information directly from the received visual information.

– Simulating the motion of the objects (players and the ball) in the case that server has not sent information about those objects, the information is not complete or it is not exactly correct.

– Receiving a message by a player from a teammate. Because sending messages should be done via the WorldModel, and the WorldModel uses the extra empty part of the message to inform other players its own information, when the WorldModel receives a message, it can update some of its information according to that message.

– For the objects that their types are not known because of the long distance, such as players that their uniform number or their team is unknown, the World-Model uses Finding the Maximum Weighted Matching algorithm[3] to assign each data item to an object (of course it considers and computes the correctness probability in this case too).
Weighted Matching acts on a weighted complete bipartite graph[3] like G with parts X and Y such that $|X| \leq |Y|$, and gives a perfect matching with maximum weight. A perfect matching is a matching (a set of none neighborhood edges) that covers all vertices of part X . For our purpose, those objects that have been seen in previous (current) cycles creates vertices of Part X (resp. Y), and weight of edges are calculated by considering distance between objects and specified parameters of objects. for example if P_1 denotes a player with number 4 and unspecified team number who can be seen in the current cycle and P_2 denotes a player with number 4 and team number 1 who is seen in the past cycles, P_1 and P_2 can be identical if distance of them does not exceed a limit. So weight of edge connecting them will be computed based on their distance.

2.2 Positioning

We have used an algorithm which acts well in a noisy environment. This method can be used in every environment with arbitrary noise distribution for example in the Middle Size League.

Consider a robot has been located in an environment in which are some fixed objects called flags(let N f to be the number of them). This robot observes these flags with some noise. Received Visual information by the robot includes Distance(i) and Angle(i) which are respectively, distance and angle with respect to flag i.

For each type of noise one can obtain $P_i(x, y) = f(Distance(i), Angle(i), x, y)$ which is the probability density function of existence in $position(x, y)$ concern ing to received information of flag i. Now, we decide to obtain $P(x, y)$ which is the probability density function of existence in $position(x, y)$ concerning to all received visual information. This function is obtained from P_is. We could not find a precise formula for $P(x, y)$, but, we suggest a formula which works well practically, however, we explain the reason of using this formula below (h and w are the height and width of the soccer field). Suggested formula:

$$P(x, y) = \frac{\prod_{i=1}^{N} P_i(x, y)}{\int_{-\frac{h}{2}}^{\frac{h}{2}} \int_{-\frac{w}{2}}^{\frac{w}{2}} \prod_{i=1}^{N_f} P_i(x, y)dxdy}$$

$P(x, y)$ must at least have following properties, all of which the above suggestion satisfies:

– $\int_{-\frac{h}{2}}^{\frac{h}{2}} \int_{-\frac{w}{2}}^{\frac{w}{2}} \prod_{i=1}^{N_f} P_i(x,y)dxdy = 1$ (It is trivial that each P_i must also have this property).

– $\forall i,x,y : P_i(x,y) = 0 \Rightarrow (P(x,y) = 0)$

– $\forall i,x,y : P_i(x,y) = 1 \Rightarrow (P(x,y) = 1)$ (It is worth to mention that if we have $P_{i1}(x,y) = 0$ and $P_{i2}(x,y) = 1$ for a (x,y), then we call them inconsistent and in this case the suggested formula is ambiguous ($\frac{0}{0}$).)

Now, we will describe the usage of $P(x,y)$ for robot positioning. Suppose the approximation of robot positioning be Δ. Then, the best position for the robot is the position with the most probability. In other words, it is the position that $\int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} \int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} P(x+tx, y+ty)dt_x dt_y$ takes its maximum value in it.

since the computing of $\int_{-\frac{h}{2}}^{\frac{h}{2}} \int_{-\frac{w}{2}}^{\frac{w}{2}} \prod_{i=1}^{N_f} P_i(x,y)dxdy = 1$ is time-consuming and also this expression is only a constant coefficient for $P(x,y)$, to find the position of the robot, one can find a (x,y) in which $\int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} \int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} P(x+tx, y+ty)dt_x dt_y$ takes its maximum value. (We have used the simpler formula for this purpose: $\prod_{i=1}^{N_f} \int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} \int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} P(x+tx, y+ty)dt_x dt_y$

Finding the maximum point is a slow algorithm in general case. The reason is that this function must be computed in so many points. Hence, we used some methods which does not consider all points. Hill Climbing[6] is one of the methods which we used in this case. In this method, we start with one point and we go to a better point while it is possible, and so this algorithm finishes when there isn't any better point in neighborhood of the current point.

3 communication

Arvand agents communicate with each other frequently. In fact anytime they feel that they have enough new information about their miniworld or other agents decisions (like passing or speeding down because of their low stamina) they will say a message including newest data that they have seen or heard. The only remaining problem is interpreting "having enough new information". Because saying a useless message will cause in preventing some teammates from hearing more useful messages. We used some statistical methods to determine when one must say a message. It is expected that each agent say about one message in each 3 cycles.

4 Skills

It is obvious that for achieving the main goal in the game there are many smaller and local goals that should be achieved. Therefore, skills are implemented for performing these intermediate duties. An important point that should be taken into consideration for implementing the skills is that at any time when a skill is running, we should be able to stop the skill and to activate another skill. All in all, we decided to implement skills in the manner that each skill is a class derived from the main class, TSkill. Also each skill should fulfill at least the main function 'TSkillState TSkill::NextCommand(TCommand &)'. This means that a skill will not send a command to the server itself, but instead it should

just return one command in each call. So a skill is similar to a finite automaton that in each call (step) changes . Of course this automaton can dynamically be changed. Thus, a complete execution of a skill is calling the NextCommand function of the skill successively and sending the commands to the WorldModel until the skill approximately or completely achieves its goal. So we can stop the current skill whenever we want and create a new skill and make it to be the current skill.

5 Strategies

As we said above, a strategy is a high level skill that uses other skills to play a soccer game. In order to do this, we can use many different algorithms. So we may have various strategies. A player can change his strategy in two ways. First, it can decide to change his strategy himself, corresponding to the state of the match. Another way is that a certain player analyzes the match and decides which strategy should be used and informs the other players of the decision. Of course, in the second one, online coach is the best choice. We preferred to use the second method for changing strategies. One of the advantages of the second approach is that all players will change their strategies simultaneously.

6 Team Development

6.1 Tactic Designer One noticable thing in designing the strategy of this game is that in various cases we can perform the decisions by a simple statemachine or a set of conditions actions. For instance, suppose the player who has the ball in his kickable area. He knows that if he shoots, there will be a high probability of scoring a goal. so it will be reasonable to shoot. Direct implementation of this statemachine in the source code, in addition to its difficulty and errorproneness will result in a nonflexible code and so production of more and different tactics will be harder. In developing the SharifArvand team we have used a tool called TacticDe signer. Using its visual user interface it is possible to design the statemachine part of the tactic according to the team's available skills. It is important to know that in the TacticDesigner we have provided the ability of using some parameters which their value is not static and it will be computed by the executive engine during the game.

References

1. E. Corten, K. Dorer, F. Heintz, K. Kostiadis, I. Noda, J. Riekki, P. Riley, P.Stone, T. Yeap, Soccerserver Manual
2. S. Y. Kung, Digital Neural Networks, PTR Prentice Hall, 1993

3. D. West, Introduction to Graph Theory, Prentice Hall, 1996
4. Tom M. Mitchell, Machine Learning, McGrawHill, 1997
5. A.Barrand E. Feigenbaum, The Handbook of Artificial Intelligence, W.Kaufmann, San Mateo, CA, 1981