FUT-K Team Description Paper 2015

Masato Ishitaka, Yuya Kitajima, Kosuke Onda, Kento Ozaki,
Kazutomi Sugihara, and Teruya Yamanishi

Department of Management Information Science, Fukui University of Technology
Gakuen, Fukui 910–8505, Japan

Abstract. This paper describes the changes of the structure of the programing source and the kick motion followed walking without pause after approaching the ball for agents of FUT-K in the simulation league of RoboCup 3D Soccer. In addition, it is mentioned on our new kick motion over a long distance obtained using a method of Covariance Matrix Adaptation Evolution Strategy.

Key words: Program structure, Kick motion, Long Distance kick

1 Introduction

FUT-K that is mainly composed of undergraduate students of Fukui University of Technology in Japan has been organized since fall 2007. At the beginning of inauguration, we have participated in two leagues, namely one is RoboCup Soccer 3D Simulation, and another RoboCup Soccer Mixed Reality. Since the mixed reality league was withdrawn, we are concentrating operations on 3D simulation league at present.

The purposes of our team are to grow knowledge and experience of the computer language and the information science through applying themselves to RoboCup Soccer. Though almost members of our team are unskilled at programming yet, we believe that now our team is developing with getting advice from other teams.

We made six appearances in the world competition from RoboCup 2009 in Graz to RoboCup 2014 in João Pessoa, and could get to a lot of things about soccer strategies and techniques of the movements for humanoid robot as the 3D soccer agent from these competitions.

In this paper, we introduce our activities for developing the 3D soccer agent of this year as follows:

- Reconsideration of the overall program cord to make it easy to expand,
- Development of kick motion without pause after approaching the ball,
- Development of kick motion over the long distance.

The details are explained in the following sections.

2 Refactoring of Program Code on Agents

As the previous our program code on the agent has concentrated the functions at only one class, it is difficult to understand what kind of function is where in some class. We try to refactor our agent program by using Singleton pattern in order to make it easy to understand. The singleton is one of the design pattern that it restricts generation of the instance to be only one, and offers the global access to a calling from other classes[1]. For RoboCup Soccer 3D Simulation, the soccer agents decide the next behavior by basing information from the soccer server computer. So, arguments to turn over the object with the information from the soccer server computer to the member function of each class are needed when the information from the soccer server computer is required, if the singleton is not used. We modify the classes dealing with information used very much from the server to the singleton classes in oder to decrease such arguments. As result, we can obtain the information of the soccer server computer by using the static function in the singleton class without turning over the object between classes.

There are mainly two kinds of classes: one is the process on data from a soccer server, and other decides the information sending to a soccer server. We show classes processing the data receiving from the server and the data sending to the server in Tables 1 and 2, respectively.

In this section, we explain classes constructed by improving our program structure.

Table 1. The characteristic features of new classes processing data receiving from the soccer server in our program.

Class name	Characteristic features
RcssserverSocket	Connection of the server and sending and receiving of data.
Parser	Dividing data received from the server into character strings.
RawStorage	Normalization of the character strings divided by Parser class, namely numerical conversion, boolean transformation, and so on.
AgentState	Providing information of agent self.
WorldState	Providing information of the present game.
FieldState	Providing information of objects on the soccer field except for data of AgentState class.
StrategyInformation	Providing useful information on next actions of agents.

2.1 Class of data processing from sever

In our program, new classes have been constructed as routines processing data from the soccer server computer, which are named as “RcssserverSocket”, “Parser”,

Table 2. The characteristic features of new classes processing data sending to the soccer server in our program.

Class name	Characteristic features
Agent	Decision on basic motions of a agent.
Strategy	Decision on ways of walking on a agent.
JointController	Decision on angular speeds for joints of a agent.
Effector	Stocking the information sending to the server, and sending to the server.

"RawStorage", "AgentState", "WorldState", "FieldState", and "StrategyInformation". These classes are singleton ones except for RcssserverSocket class. Also, the process of data by these classes are imaged at Fig.1.

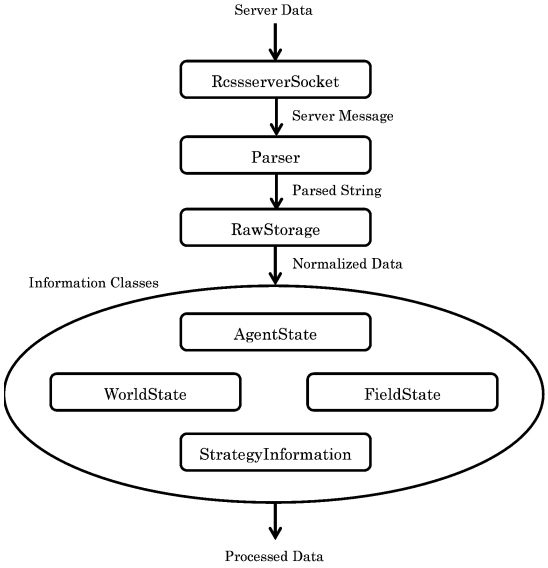

Fig. 1. The image on the processing data in new classes in our program.

We can take data processed on the soccer server by calling Information classes except for classes processing data receiving from the server. It is mentioned on Information classes by the next subsection for details.

2.2 Class of data sending to sever

Agent, Strategy, JointController, and Effector classes determine to select information in order to send the soccer server. Among these classes, JointController

class is only a singleton class. The relationship between each class is shown in Fig.2.

From Fig.2, one can see that Agent class plays a central role in updating Information classes by using JointClontroller, Effector, and Strategy classes. Agent class gives instructions to JointController class setting the angular speeds for joints of the agent, and Effector class gets the information on their angular speeds from JointController one and sends to the server. As results, agent class carries the role of determining processes except the agent's motions about walking, and strategy class deals with processes on walking motions of the agent.

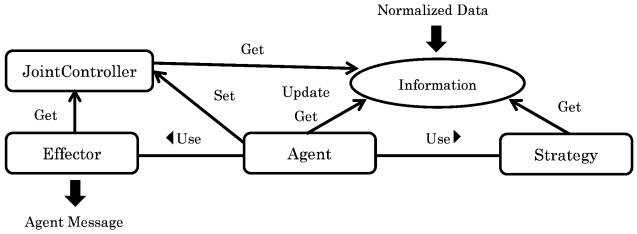

Fig. 2. The relationship between each class determining data sending to the server.

3 Creation of kick motion without pause

Since previous our TDP, namely FUT-K Team Description Paper 2014 in João Pessoa, has been discussed on the kick motion without pause after approaching the ball as future perspective, we attempt to develop a new kick motion with quick and stable. So, we impose the following rules in order to make the kick motion of the agent like persons approaching the ball and kicking:

1. Before kicking at the ball, the agent does not stop with feet together just front of the ball,
2. It is not reversed until the agent finishes kicking the ball.

Using ForceResistance Perceptor in the foot and adjusting the center of mass for the agent, we succeeded in making of the kick motion without pause by synchronizing the forward motion and the kick motion. The result is shown in Table 3.

From Table 3, we find that the agent can kick at the ball by quick kick motion without the rollover for five tries. However, the carry of the ball becomes short as the compensation of such quick and stable kick motion.

Table 3. Results of comparison of new kick motion with old one. Here the motion time means the time until the ball is kicked after the pivot leg of the agent lands

	New kick motion			Old kick motion		
	Carry of the ball [m]	Motion time [s]	Rollover	Carry of the ball [m]	Motion time [s]	Rollover
1st try	3.08	0.40	No	6.57	2.34	Yes
2nd try	2.14	0.36	No	6.48	2.34	Yes
3rd try	2.51	0.40	No	5.99	2.34	Yes
4th try	2.56	0.36	No	6.46	2.34	Yes
5th try	2.68	0.42	No	6.48	2.34	Yes
Ave.	2.54	0.39	No	6.40	2.34	Yes

4 Creation of kick motion over the long distance

At Table 3 in previous section, the kick motion before development flies the ball for only 6.4 meters. So we attempt to develop the kick motion over the long distance, where it does not care that the agent does stop with feet together just front of the ball before kicking at the ball. In this section, the algorithm of the kick motion over the long distance is explained, and the way of search on parameters required in the algorithm is discussed.

4.1 Simple algorithm of kick motion over the long distance

We attempt to create the kick motion over the long distance by setting the target angle, the angular speed and the time to move the joint. Then, we can a select relative speed or a fixed speed for the angular speed; the angular speed is decided by the difference between the target angle and the present angle for the relative speed, the angular speed is not changed for the fixed speed. This algorithm can be written in a XML form, and is described as shown in Fig. 3, for example.

```
<sequence name="sample">
  <motion name="right_up" time="50">
    <move id="hj1" degree="120.0" speed="0.075" type="0"/>
    <move id="hj2" degree="30.0" speed="0.075" type="0"/>
  </motion>
  <motion name="left" time="100">
    <move id="hj1" degree="-120.0" speed="1.0" type="1"/>
  </motion>
</sequence>
```

Fig. 3. Simple algorithm of new kick motion over the long distance written by XML.

The algorithm described by XML in Fig. 3 means that Type 0 is setting the relative speed, or Type 1 does the fixed speed. First of all, for 50 frames, the

joint “hj1” moves to 120 degrees with the relative speed, and “hj2” moves to 30 degrees with one. After that, for 100 frames, “hj1” moves to -120 degrees with the fixed speed, and “hj2” stops.

By derive the algorithm on new kick motion over the long distance, we have to generate parameter from stopping of the agent to a close of raising of the leg. Fortunately, because it could be fixed from stopping of the agent to the place where the leg is lifted a little, the parameter could be reduced. After the result of the trial and error, the amount of the parameters becomes to be 60, which consist of the target angles and the angular speed.

4.2 Parameter search by CMA-ES

According to an idea[2], we use Covariance Matrix Adaptation Evolution Strategy (CMA-ES) method for parameter search. CMA-ES is the optimization algorithm which applies a dispersion covariance line to an evolution strategy. The fitness function is defined as follows:

$$fit_{LocX} = \begin{cases} ballLocationX & \text{for success ,} \\ -1 & \text{otherwise .} \end{cases} \tag{1}$$

In case of success, the agent raises one leg and the ball flies to the front, and the fitness value is -1, otherwise. The X-Location of the ball, not the distance of the ball, is used because we want the agent to shoot the ball in the direction of forward, exactly. By taking 200 as the population size, the average and the maximum change performed repeatedly 400 times is shown on Fig. 4.

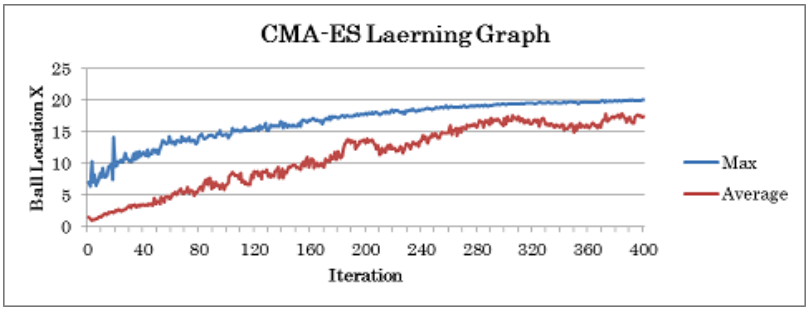

Fig. 4. The result of kick motion over the long distance by CMA-ES method.

This result is similar to UTAustinvilla’s CME-ES learning curve[2]. At 400 times, the maximum of the fitness value becomes about 20 m.

5 Conclusions and Future Works

By refactoring our program, the movement of a agent has not be improved, but we expected that bugs was able to be suppressed during our developing agent and that the efficiency of coding was increased.

The kick motion of the agent without pause was convenient because it was put in a goalpost from the nearby distance. At present, as it is not introduced such kick motion into the agent, we guess that the score rate of our team is risen. In future works, we will attempt to improve that distance because the flying distance of the ball is still short for this kick motion.

Also, we succeeded in development of a long distance kick. By means of this, all the agents were able to shoot into goalpost from the far distance. But if it was not the right location, the agent failed in a kick. Even if there are some errors, we want the agents to succeed in the long distance kicking.

Acknowledgements

This work is supported in part by research grants from Fukui University of Technology.

References

1. Yoshihara, H.: Sec. II Singleton in Reverse dictionary on design patterns (in Japanese), `http://www.nulab.co.jp/designPatterns/designPatterns2/designPatterns2-1.html` (2015).
2. Depinet, M., MacAlpine, P., and Stone, P.: Keyframe Sampling, Optimization, and Behavior Integration: Towards Long-Distance Kicking in the RoboCup 3D Simulation League, RoboCup2014, Robot Soccer World Cup XVIII, Lecture Notes in Artificial Intelligence, Springer Verlag, Berlin (2015).