# HELIOS2011 Team Description

Hidehisa Akiyama, Hiroki Shimora, Tomoharu Nakashima, Yosuke Narimoto, and Tomohiko Okayama

Fukuoka University, AIST, Osaka Prefecture University, Japan
{hidehisa.akiyama,h.shimora}@aist.go.jp,
{nakashi@,narimoto@ci.,tomohiko.okayama@ci.}cs.osakafu-u.ac.jp

**Abstract.** HELIOS2011 is a 2D soccer simulation team which has been participating in the RoboCup competition since 2000. This paper describes the overview and the current effort of HELIOS2011. We recently focus on an improvement of soccer skills based on probability estimation and an online tactics planning. We show the result of preliminary experiments of each research topic.

## 1 Introduction

HELIOS2011 is a simulation soccer team for the RoboCup soccer 2D simulator(RCSS)[1][1]. The team has been participating in the RoboCup competition since 2000 and reached 1st place in RoboCup2010. We recently focus on an improvement of soccer skills based on probability estimation and an online tactics planning. In this paper, we briefly introduce these research topics and show the result of preliminary experiments.

## 2 Released Software

We are developing several program packages as an open source software. All software are written in Standard C++ and support POSIX, or use Qt[2]. Now, we are mainly maintaining following software packages:

- librcsc: a base library for RCSS, which can be used as a framework for a simulated soccer team.
- agent2d: a sample team program using librcsc. agent2d can work as a simple simulated soccer team. Newbies can use agent2d as a start point for developing their own team.
- soccerwindow2: a viewer program for RCSS. soccerwindow2 can work as a monitor client, a log player and a visual debugger.
- fedit2: a formation editor for agent2d. fedit2 enables us to design a team formation using human's intuitive operations.

[1] The RoboCup Soccer Simulator: http://sserver.sourceforge.net/
[2] A cross-platform application and UI framework: http://qt.nokia.com/

These software are implemented from scratch without any source code of other simulated soccer teams. But, several idea were inspired from other teams' released code, such as CMUnited[2], YowAI[3], TsinguAeolus[4], UvA Trilearn[5] and Brainstormers[6]. Thanks for their contributions.

These software packages have already been freely available[3]. We hope that our released software help new teams to participate RoboCup events and to start a research of multiagent systems.

## 3 Improved Decision Making of Ball Intercept based on Probability Estimation

### 3.1 Overview

One of the most important skills in developing soccer agents is to intercept the ball. In this skill the agent has to maneuver itself to the (normally moving) ball in order for the agent to be kickable the ball. This skill can be divided into three subtasks:

Task 1: Estimating the cycles necessary to intercept the ball.
Task 2: Decision making if the agent should move to the ball or not.
Task 3: Maneuvering the agent to the ball.

In Task 1, from the sensed status of the field, the agent estimates the number of cycles required for it to complete the ball intercept. We call this intercept cycle. The agent makes a decision of whether it executes the ball intercept or not based on the information of the field including the intercept cycle in Task 2. In Task 3, if the agent decides to go for intercept, it selects the most appropriate action such as turn and dash in order to intercept the ball.

The above three tasks are closely related and sometimes cannot be clearly divided. Since Task 1 and Task 3 are implemented in the *librcsc* library, we just employ these functions and will not modify them at all. Regarding Task 2, its implementation is left for the developer of agents for the teams that use *librcsc*. For example in agent2d, Task 2 is solved by a combined conditions about the field. An example case is if there is not any kickable player in the team and the agent thinks it can intercept the ball faster than any opponent agents, it decides to intercept the ball and selects an appropriate action accordingly. However, the decision making by agent2d is not very accurate: the agent fails to intercept the ball even if it makes the decision of intercept.

This year we focus on Task 2 and develop a more accurate decision making system for this task. In our decision making system, first the probability of successful ball intercept is estimated based on the current field status. ¿From the estimated probability, the decision on whether to intercept the ball or not is made.

[3] http://sourceforge.jp/projects/rctools/

### 3.2 Taking Statistics on the Ball Intercept

The estimation of the probability of successful ball intercept is performed from the experience of the agent. That is, the agent estimates the probability using the statistics on the results of its previous ball-intercept trials.

In order to take the statistics, a number of practice matches were made. In the matches, ten agents excluding the goalie monitor their intercept cycle using *selfReachCycle* function of *librcsc*. The agents check if the intercept cycle is correct or not when it is less than or equal to 15. In this procedure, the current field status was also recorded. The field status in this article is specified as the relative position and velocity of the ball to the agent. Thus, the field status is represented as a four-dimensional real vector.

The practice matches were made against 18 teams (17 teams that participated in RoboCup 2010 and agent2d-3.0.0 (librcsc4.0.0)). Since there were some problems in starting teams, KickOffTUG and WrightEagle were not used in this procedure.

### 3.3 Estimating Probability of Successful Ball Intercept

In this subsection, the procedure to estimate the probability of successful ball intercept from the statistics taken is explained.

As explained in the procedure of taking statistics (Subsection 3.2), the intercept cycle is recorded with the field status and the result of ball intercept trial. The data set was divided into a small number of subsets using $k$-means clustering algorithm. The distance between two field statuses is defined as the Euclidean distance between the corresponding four-dimensional real vectors.

Now let us assume that there is some field status (i.e., we have the information on the relative position and velocity of the ball). First the most probable cluster is identified that is the nearest cluster to the field status. Then the number of successful ball-intercept trials are counted for each intercept cycle in the identified cluster. The probability of successful ball intercept is estimated as the accumulated number of successful trials within the current intercept cycle divided by the total number of trials in the cluster that the current field status belongs to.

In this article, if the estimated probability of successful ball intercept is larger than a pre-specified threshold (0.5 in the next subsection), the agent decides to intercept the ball.

### 3.4 Performance Evaluation

We examined the performance of the decision making system that is described in the last subsection through a series of computational experiments. In the experiments, the decision making system was implemented in the offensive-half agent of HELIOS2010. The other agents are not modified at all. The modified team played the original HELIOS2010 and agent2d-3.0.0 (librcsc4.0.0) ten times.

The original HELIOS2010 also played the two teams as well. During the matches, the successful ball-intercept trials by the modified agents were monitored.

The above experiments were performed with different numbers of clusters for the $k$-means algorithm. The experimental results are summarized in Table 1. Table 1 shows only the experimental results where the modified and original HELIOS2010 played the original HELIOS2010. The second row shows the performance of the original HELIOS2010, and the other rows the performance of the new decision making system with different number of clusters. ¿From this table, it can be seen that the agents intercept the ball more accurately than the original team. Even if the collected data was not divided into subsets (i.e., when the number of clusters is just one), the successful rate of ball intercept is larger than the original agents. The same conclusion was drawn in the other experiments against agent2d-3.0.0.

**Table 1.** Results of ball intercept trials (against HELIOS2010).

| # of clusters | Successful trials | Failed trials | Rates |
|---|---|---|---|
| Original (N/A) | 279 | 531 | 0.34 |
| 1 | 248 | 192 | 0.56 |
| 2 | 280 | 186 | 0.60 |
| 3 | 272 | 184 | 0.60 |
| 4 | 269 | 159 | 0.63 |
| 5 | 236 | 158 | 0.60 |
| 6 | 232 | 151 | 0.61 |
| 7 | 271 | 164 | 0.62 |
| 8 | 258 | 169 | 0.60 |
| 9 | 276 | 172 | 0.61 |
| 10 | 293 | 167 | 0.64 |

## 4 Online Tactics Planning

We propose a framework to search a sequence of action using game tree search methodology in order to realize more flexible decision making. It is important to realize a cooperative behavior among several teammate agents to achieve a better team performance. During such a cooperative behavior, involved agents have to share the target state and each agent needs to perform some planning to achieve the target state. In this paper, we call such a cooperative behavior tactics. Our framework enables agents to generate and evaluate tactics as a sequence of action online.

### 4.1 Framework to Search Action Sequence

The proposed framework can generate and evaluate a sequence of action (such as pass, dribble and shoot) online. Figure 1 shows an example of action sequence.

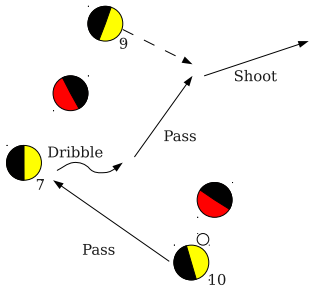


**Fig. 1.** Example of action sequence generated by player 10.

In the figure, player 10 predicts the state after four ball handling actions. At first, player 10 pass the ball to player 7. Player 7 receives the ball and dribbles forward, then passes it to player 9. Finally, player 9 receives the ball and shoot it to the goal. In the last state, the ball is within the goal.

This framework has a mechanism to generate and evaluate action sequence, that consists of two components, ActionGenerator and FieldEvaluator. When users develop a team using this framework, they have to prepare these two components.

**ActionGenerator** ActionGenerator generates several action-state pairs from one input state. When an successful action is generated from some state, the result state after the action is also estimated. If an action-state pair is successfully generated, it is queued in the game tree. ActionGenerator can generate new action-state pair from a queued action-state pair. If such an action-state pair is generated, it can be appended to the input one. The result list can be used as a new action sequence. This procedure can be repeated unless the new action-state pair is not generated, the length of action sequence is more than the threshold value, or the number of traversed node is more than the threshold value.

**FieldEvaluator** FieldEvaluator evaluates a generated action-state pair or entire action sequence. The returned value is a real number. Player agents choose the best action sequence based on this value. The evaluated value is also used as a heuristic value to prune the game tree.

### 4.2 Current Implementation

We have already implemented several ActionGenerator such as PassGenerator, DribbleGenerator, and so on. However, currently, our framework only search the kick action sequence. This is because it is difficult to generate the positioning action and to operate them efficiently. This is one of the main issue of our future development.

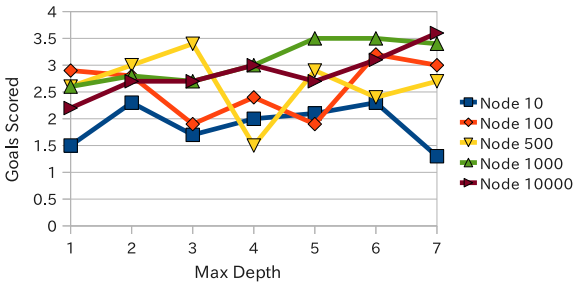


**Fig. 2.** Result of the Effect of Action Sequence Search.

Recently, we set the maximum depth of tree to 4 and set the maximum number of traversed node to 500. In order to evaluate the effect of these parameters, we performed some experiments. Figure 2 shows one of the results. In this experiment, we performed 10 matches for each setting against agent2d-3.0.0 without heterogeneous players. The vertical axis means the average goals scored. The horizontal axis means the maximum depth of game tree. Each line means the difference of the maximum number of traversed node. This result shows that the deeper search may improve the team performance if the number of traversed node is enough. However, of course, there is a trade-off between the computational cost and the team performance.

## 5 Conclusion

This paper described the research focus and the current effort of HELIOS2011. We applied a probability estimation to improve the ball intercept skill and applied a game tree search methodology for online tactics planning. We performed some experiments to show the effectiveness of our approach.

## References

1. Noda, I., Matsubara, H.: Soccer server and researches on multi-agent systems. In Kitano, H., ed.: Proceedings of IROS-96 Workshop on RoboCup. (Nov. 1996) 1–7
2. Stone, P., Riley, P., Veloso, M.: The CMUnited-99 champion simulator team. In Veloso, M., Pagello, E., Kitano, H., eds.: RoboCup-99: Robot Soccer World Cup III. Volume 1856 of Lecture Notes in Artificial Intelligence. Springer Verlag, Berlin (2000) 35–48
3. Nakayama, K., Ako, T., Suzuki, T., Takeuchi, I.: Team YowAI-2002. RoboCup 2002: Robot Soccer World Cup VI (2002)
4. Yao, J., Chen, J., Cai, Y., Li, S.: Architecture of tsinghuaeolus. In: RoboCup 2001: Robot Soccer World Cup V, Springer-Verlag (2002) 491–494
5. UvA Trilearn 2003. http://www.science.uva.nl/~jellekok/robocup/2003/
6. Brainstormers 2d: Brainstormers Public Source Code Release. http://www.ni.uos.de/index.php?id=1023