HELIOS2008 Team Description

Hidehisa Akiyama, Hiroki Shimora, and Itsuki Noda

Information Technology Research Institute,
National Institute of Advanced Industrial Science and Technology
Ibaraki, Japan
{hidehisa.akiyama,h.shimora,I.Noda}@aist.go.jp

Abstract. HELIOS2008 is a 2D soccer simulation team which has been participating in the RoboCup competition since 2000. This paper describes the overview and the research focus of HELIOS2008. Our main goal is to develop a more realistic simulated soccer team. In particular, we are interested in applying a human's training operations without programming knowledge into the agent's decision making. We propose a novel positioning mechanism, which utilizes the Delaunay Triangulation and can be adjusted by human's intuitive operations using GUI tools.

1 Introduction

HELIOS2008 is a 2D soccer simulation team which has been participating in the RoboCup competition since 2000. Our former team name is TokyoTechSFC, that was ranked 3rd in RoboCup2005 and ranked 4th in RoboCup2006. HELIOS2007 was ranked 3rd in RoboCup2007 again. Our code is written by C++ and implemented from scratch without the source code of any other simulated soccer teams. We are developing several tool programs that helps us to develop a team and to make a experiment environment, and almost all of them have already been available in public.

Our main goal is to develop a more realistic simulated soccer team. In particular, we are interested in applying a human's training operations without programming knowledge into the agent's decision making. In the last few years, we mainly concentrated on the training of agent positioning behavior using a human's instruction. In order to realize this, we applied a novel positioning mechanism which utilizes Delaunay Triangulation. Moreover, in this year, we extended our model such that multi-dimensional input can be accepted.

The setup of this paper is as follows. In section 2, we will introduce the overview of our programs. In section 3, we will describe our basic positioning mechanism. In section 4, we will describe the extended model of our positioning mechanism. And in section 5, we will end with a conclusion.

2 Available Programs

We are developing three program packages as an open source project:

- librcsc: a base library for the simulated soccer agent and related tools, licensed under LGPL. librcsc can be used as the framework for the simulated soccer team. HELIOS2008 also uses librcsc.
- soccerwindow2: a viewer program for the soccer simulation, licenced under GPL. The soccerwindow2 package contains one more component, fedit, that is the editing tool to compose the team formation intuitively. soccerwindow2 is developed using Qt[7], which is a cross-platform application development library. Qt enables us to use soccerwindow2 in several platforms.
- agent2d: a sample agent programs that can work as a simple simulated soccer team, licensed under GPL. A simple behavior is implemented, but that behavior is more complicated than UvA base code[8]. The team strategy is still simple, but the performance is rather good. agent2d can be used as a good start point for new teams.

These packages have already been available at SourceForge.jp[1]. We hope that our programs help a new team to participate the RoboCup event and to start a research of the multiagent systems using the RoboCup soccer simulator[6, 1].

2.1 fedit

fedit is a tool for editing the team formation. This tool enables us to intuitively compose desired example position data. These data can be used as the training data for our positioning mechanism described in section 3 and 4. Figure 1 shows the screenshot of fedit. fedit can visualize the training process and enables us to edit the training data easily.

3 Triangulation based Positioning Mechanism

We have proposed a function approximation model that utilizes Delaunay Triangulation and linear interpolation algorithm[2, 3]. We call this model as *basic model*. In the basic model, we divide a input space into several triangles according to given training data. These triangles determine an effective region for each data. Therefore, it is easy to understand where each data has its influence.

3.1 Delaunay Triangulation

Delaunay Triangulation[5] is one of the method to triangulate the plane region based on the given point set. Delaunay Triangulation for a set P of points in the plane is a triangulation $DT(P)$ such that no point in P is inside the circumcircle of any triangle in $DT(P)$. Delaunay Triangulation maximize the minimum angle of all the angles of the triangles in the triangulation. So, we can get the most stable triangles from Delaunay Triangulation. Figure 2(a) shows the example of Delaunay Triangulation. In this figure, Voronoi Diagram is also shown. There is a duality between Voronoi Diagram and Delaunay Triangulation.

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

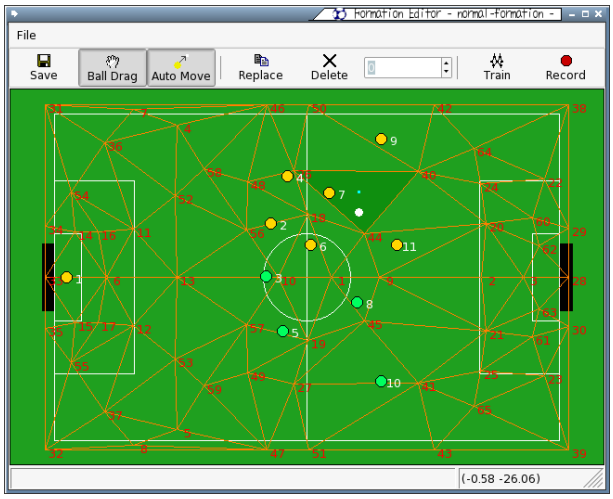


Fig. 1. The main window of fedit. The example training data set is shown in the window.

If the size of given points is more than 3, we can get a unique triangulation. There are several algorithms to calculate Delaunay Triangulation and the time complexity of the fastest one is $O(n \log n)$. Therefore, if the number of points is hundreds of levels, we can calculate Delaunay Triangulation in the real time. In librcsc, we implemented one of the fastest algorithm, the incremental algorithm[5].

In our method, the ball positions in training data are used as the vertices of triangles. Each vertex has the output value as the agent's move position for that vertex(=ball) position. When the ball is contained by one triangle, player agent's move position is calculated by interpolation algorithm described in next section.

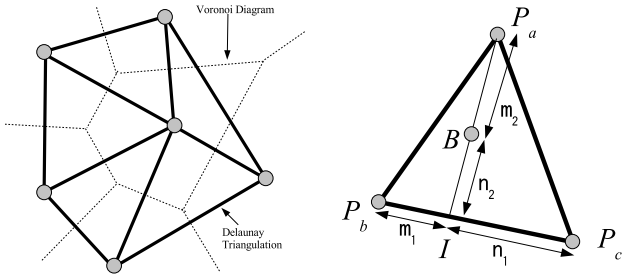
3.2 Linear Interpolation Algorithm

We use the simple linear interpolation algorithm to calculate the agent's move position. This algorithm is same as Gouraud shading algorithm[4]. Gouraud shading algorithm is a method used in computer graphics domain to simulate the differing effects of light and color across the surface of an object.

Figure 2(b) shows the process of Gouraud shading algorithm. The output values from vertices P_a, P_b and P_c are $O(P_a)$, $O(P_b)$ and $O(P_c)$ respectively. Now, we want to calculate $O(B)$, the output value of the point B contained by the triangle $P_aP_bP_c$. The algorithm is as follows:

1. Calculates I, the intersection point of the segment P_bP_c and the line P_aB.
2. The output value at I, $O(I)$, is calculated as:

$$O(I) = O(P_b) + (O(P_c) - O(P_b))\frac{m_1}{m_1 + n_1}$$



(a) Delaunay Triangulation. (b) Gouraud shading algorithm.

Fig. 2. Delaunay Triangulation and Linear Interpolation.

where $|\overrightarrow{P_bI}| = m_1$ and $|\overrightarrow{P_cI}| = n_1$.

3. $O(B)$ is calculated as:

$$O(B) = O(P_a) + (O(I) - O(P_a))\frac{m_2}{m_2 + n_2}$$

where $|\overrightarrow{P_aB}| = m_2$ and $|\overrightarrow{BI}| = n_2$.

3.3 Feature of Basic Model

The basic model can perform with light-weight because Delaunay Triangulation can be calculated with small computational cost. Also, the triangulation provides an unique separation of the input space for a given training data set. This feature is also useful because trainers need not care about the order to add training data when they are trying to modify previous data. It is also an important feature that basic model outputs continuous value, and it is easy to know visually which data are affected to the current output. Therefore, the trainers can tune training data intuitively.

On the other hand, the basic model has a drawback in the flexibility of input, that is, the dimension of the input space is supposed to be fixed and relatively small. Theoretically, it is possible to increase the number of dimension of the input space. However, in such case, it is difficult for human trainers to understand the meanings and relations of training data in the high-dimensional input space. And, if we increase the number of dimensions, we need to prepare enough number of training data to cover the high-dimensional spaces.

4 Extended Model

As described in the previous section, the basic model has a drawback when the input space consists of complex data like combinations of situations. Here, we introduce an extended model of the basic model.

4.1 Hierarchical Input Space

In the extended model, input space forms an hierarchical structure(figure 3(b)). In the basic model, a relation between input and output can be illustrated as figure 3(a), where the operation $interpolate(O_1, O_2, O_3)$ in the figure means a procedure shown in section 3.2. On the other hand, in the extended model, each point of the root input space, that is the same as the basic model, can have sub-spaces of input that consist of other dimensions/features of the situation. The sub-space of a certain point A is interpreted as follows: when the situation can match the condition specified by A, the output value can be modified and adjusted according to the situation in the sub-space. The relation between the root space and the sub-space can be applied recursively, so that the whole input space forms a tree structure.

Detailed definition of the hierarchical input space is as follows: Suppose that, a set of training data for an input space A is given. We can get an Delaunay Triangulation like figure 3(b), in which a_1, a_2, a_3 are points of given training data. We introduce a *don't-care* point for each root or sub space like $a_{dontcare}$ in the figure. The *don't-care* point is independent from the triangulation and does not affect to the triangulation. The *don't-care* point can have an output value that is used when its sub-space is not used for the output. In other words, the output value of the *don't-care* point $a_{dontcare}$ in the space A is used when the current input value does not include the dimensions of the sub-space A.

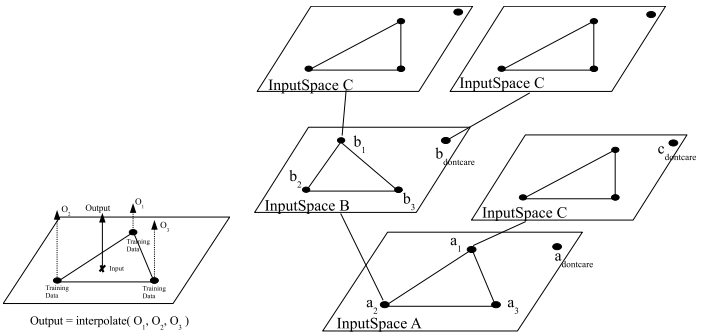
Each vertex can have its own sub-space which consists of examples of the sub-space. For example, the vertex a_1 and a_2 in figure 3(b) have a sub-space C and a sub-space B, respectively. Moreover, points b_1 and $b_{dontcare}$ in the sub-space B also have sub-space C. Note that three sub-spaces C belong to a_1, b_1 and $b_{dontcare}$ is independent with each other. The procedure to form sub-space can be applied recursively. As a result, the input space forms a n-branch tree with the root space A.

The merit of the tree-structured input space is that we can omit verbose dimensions of the situation in the given training data. This means that we can give training data roughly in the root space, and provides more detailed example in a certain area using additional dimensions to describe difference of the situations in the sub-space.

4.2 Output Procedure

When an input is given to the extended model, an output is calculated by the algorithm 4.1.

In this algorithm, the tree structure is traversed from the root space, and the summation of each sub-space is calculated. $Output(space, inputList)$ is a recursive function that traverses the tree structure according to input value $inputList$. $FindInput$ is a procedure to find a variable whose name is given in the second argument from the input list given in the first argument. $inputList$ stores the pair of the name of a sub-space and its variable. The calculation in a sub-space is the same as the interpolation procedure $Interpolate$ in the basic



(a) Input/Output in Basic Model.

(b) Input Space in Extended Model.

Fig. 3. Hierarchical Input Space.

model described in section 3.2, while an output for a vertex in the triangle that has a sub-space is determined by the calculation in the sub-space. *SpaceOutput* is a procedure to calculate output value of the sub-space, which adds the output value of each vertex and the output of its sub-space. *FindTriangle* is a procedure to find a triangle that includes *input* from an triangulation.

For example, the output of b_1 in figure 3(b) is the sum of the output value of an data at b_1 and the output of sub-space C. Using this value, the sub-space B returns its output to a_2

Algorithm 4.1: CALCULATE($tree, inputList$)

procedure OUTPUT($space, inputList$)
 $input \leftarrow$ FINDINPUT($inputList, space.name$)
 if $input == NULL$
  then $\begin{cases} space \leftarrow space.dontcare.childSpace \\ \textbf{return } (\text{OUTPUT}(space, inputList)) \end{cases}$
  else remove $input$ from $inputList$
 SPACEOUTPUT($space, input$)

procedure SPACEOUTPUT($space, input$)
 $tri \leftarrow$ FINDTRIANGLE($space, input$)
 if $tri == NULL$
  then {**return** (0)
  else $\begin{cases} out_1 \leftarrow tri.v_1.output + \text{OUTPUT}(tri.v_1.childSpace, inputList) \\ out_2 \leftarrow tri.v_2.output + \text{OUTPUT}(tri.v_2.childSpace, inputList) \\ out_3 \leftarrow tri.v_3.output + \text{OUTPUT}(tr.v_3.childSpace, inputList) \\ \text{INTEROPOLATE}(out_1, out_2, out_3) \end{cases}$

main
 $space \leftarrow tree.rootSpace$
 return (OUTPUT($space, inputList$))

4.3 Procedure to Add New Data

When a new training data is given, the hierarchical input space is reformed by procedure shown in algorithm 4.2

In this procedure, *data* indicates a training data, in which *data.inputList* is a list of input variables. When some variables of a traversing sub-space *space* do not exist in *data.inputList*, the sub-space *space* is handled as *don't-care* and pickup *don't-care* point in the space. When *input* is included by *space*, then *input* moves from *data.inputList* to *data.removedInput*. This procedure terminates when *data.inputList* is empty. If *input* matches with a vertex in *space* the output of the vertex is overwritten. Otherwise, a new vertex is defined. *Calculate* provides difference of outputs between the parent vertex and its sub-space.

Algorithm 4.2: AddNewData($tree, data$)

procedure AddData($space, data$)
 $input \leftarrow$ FindInput($data.inputList, space.name$)
 if $input == NULL$
  then AddData($space.dontcare.childSpace, data$)
  else move *input* from *data.inputList* to *data.removedInput*
 if *inputListisempty*
  then
   $vertex \leftarrow$ a vertex that matches with *input*
   if $vertex == NULL$
    then $vertex \leftarrow$ a new vertex
   $vertex.input = input$
   $vertex.output = data.output -$ Calculate($tree, data.removedInput$)
  else
   $vertex \leftarrow$ a vertex that matches with *input*
   if $vertex == NULL$
    then
     $vertex \leftarrow$ a new vertex
     $tri \leftarrow$ FindTriangle($space, input$)
     $out_1 \leftarrow tri.v_1.output$
     $out_2 \leftarrow tri.v_2.output$
     $out_3 \leftarrow tri.v_3.output$
     $vertex.output$
     $=$ Interpolate(out_1, out_2, ou_t3)
   $vertex.input = input$
   AddData($vertex.child, data$)

main
 $space = tree.rootSpace$
 AddData($space, data$)

5 Conclusion and Future Directions

This paper described our simulated soccer team, HELIOS2008, and the related programs. And, we also described about our research focus and current sta-

tus. We propose an extended model for function approximation model to decide multi-agent/robot locations using hierarchical triangulation and linear interpolation methods. In this model, the tree style of input spaces enables us to reduce the number of training data that explain knowledge about combinational locations of multiple agents/robots.

While the quantitative evaluation have not done yet, we found that the proposed model may have high performance when we cannot prepare enough number of training data by human.

There still remain the following issues to apply more realistic tasks:

- Methodologies to manage training data.
- Formalization of proposed model with several team-work models.

References

1. The RoboCup Soccer Simulator. http://sserver.sourceforge.net/.
2. Hidehisa Akiyama, Daisuke Katagami, and Katsumi Nitta. Training of agent positioning using humans's instruction. volume 11, 2007.
3. Hidehisa Akiyama and Itsuki Noda. Multi-agent positioning mechanism in the dynamic environment. In *RoboCup 2007: Robot Soccer World Cup XI*, 2008. to appear.
4. Henri Gouraud. Continuous shading of curved surfaces. In Rosalee Wolfe, editor, *Seminal Graphics: Pioneering efforts that shaped the field*. ACM Press, 1998.
5. Marc van Kreveld Mark de Berg, Otfried Schwarzkopf and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer Verlag, second edition, 2000.
6. Itsuki Noda and Hitoshi Matsubara. Soccer server and researches on multi-agent systems. In Hiroaki Kitano, editor, *Proceedings of IROS-96 Workshop on RoboCup*, pages 1–7, Nov. 1996.
7. Qt: http://www.trolltech.com/products/qt/.
8. UvA Trilearn 2003 Base Source: http://www.science.uva.nl/~jellekok/robocup/2003/.