A Multi-layered Planning Architecture for Soccer Agent

Ransui Iso (*ransui@ina-lab.ise.aoyama.ac.jp*)
Hiroshige Inazumi (*hiro@ina-lab.ise.aoyama.ac.jp*)

Dept. of Industrial and Systems Engineering
Aoyama Gakuin University.

Abstract

Based on manual simulation experiments, we propose a type of agent as a Multi-Layerd Planning(MLP) Architecture to identify the game situation and create action policy. As a team, we will arrange the various type of agent according to game strategy, and realize semi-cooperative Multiagent model with minimum amount of communication.

1 Introduction

Simulated Robotic Soccer is a typical and interesting AI domain, especially Multiagent domain. RoboCup97 will include its tournament using the Soccer Server system. Soccer Server captures enough real-world complexities to be a very challenging domain.

This simulator is realistic in the following way;

1. players' vision is limited,
2. players can communicate by posting to a blackboard that is visible to all players,
3. each player is controlled by a separate process,
4. players have limited stamina,
5. actuators and sensors are noisy,
6. dynamics and kinematics are modelled, and
7. play occurs in real time: the agents must react to their sensory inputs at roughly the same speed as human or robotic succer players.

The simulator, acting as a server, provides a domain and supports users whi wish to build their own agents.

We firstly developed several type of field viewer and manual simulation tools, based on data only from soccer server, like Video Games, that is limited area of soccer field, we play soccer game with those tools. We have investigated which kind of data are inportant and useful, how human beings make decisions about action rules, and so on.

We consider, agents must have a following ability.

1. Quick desition to dynamic environment
2. Self learing to low-level skill and hi-level strategy
3. Easy programming to build a agent

We made 2 types of experiment agents before RoboCup97. Each agents are contains following ideas.

1.1 View-Information-Oriented Model

In view-information-oriented model, the main purpose is to reinforce abilities of agent as soccer players. This model consists of two shemes. i.e., analyzing scheme and reasoning scheme. Analyzing scheme works as a filter from view data given by soccer server to useful information of objectsm for the agent. In Reasoming scheme, each agent bridges time-gap of view data from soccer server, estimates position-information, and decides suitable action by using huristic rules satisfying robustness. Although any agent dosen't always calculate the parameters of estimated position and action exactly.

1.2 Q-Learning Model with Multi-Demensional State-Transition

Daiji Tamagwa who is one of our teammate, makes a agent that uses Q-Learning Model with Multi-Dementional State-Transition(MDST) [1]. MDST provides state-transition that sparated by object status in soccer field that is called "Sub-State-Transition" . This architecture provided quick decition and self learning, but program and data structures are very complexly.

2 Multi-layered Planning Architecture

2.1 Environment Modeler

Environment Modeler, as a preliminaru module for planning, makes ObjectList structure from object information from Soccer Server. ObjectList holds information of objects to which other modules easily made reference.

2.2 Planner

Planning module, as core of MLP, reads the object list and creates command text sequence that can be sent to Soccer-Server directly (see fig1).

We construct our agent under Multi-Layerd Planning (MLP) Architecture. MLP is made up by following function layers.

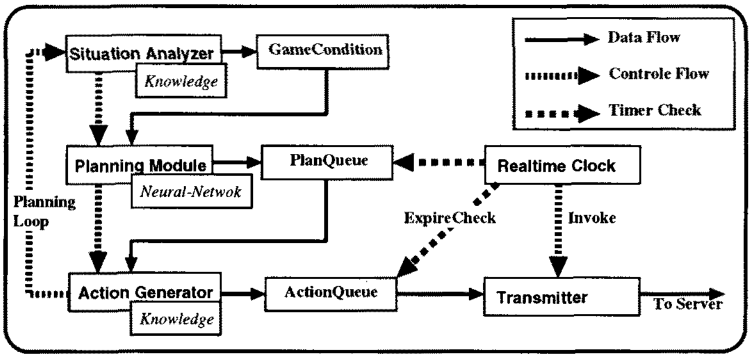

Fig. 1.

Situation Analyzer : Situation Analyzer genetates identifier of game condition from ObjectList. It has simple rule database for analyzing game situation. It contains some rule pair of predicate for object condition and identifier of game situation.

Planner : Planner layer is main fuction of MLP. Planner generating action-policy that made up by action-style, and derection of sending a ball. Action-style describes what's agent must to do. The value could be assigned one of 3 attributes, such as "Attack", "Deffence" and "Normal". Those 2 elements of action-polycy generated by perceptron type Neural-Network(NN) that implemented in the Planner.

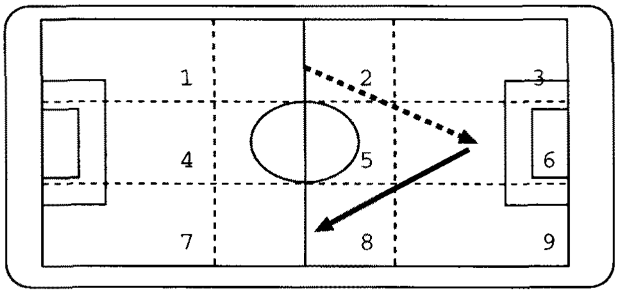

Fig. 2.

We devided soccer field to express ball position (see fig2). NN in planner takes ball transition data that made up a pair of past ball position and present ball position, and outputs pair of action-policy and derection of sending a ball.

For example. Ball moves from field-section 2 to 6, NN takes a pair of ball position, such as (2 6). then NN outputs a pair of action-policy and derection of sending a ball, such as ("Deffence" 8).

Now, NN is teached how to decition to make a correct action-policy by our hand, not learning itself.

Action Generator : Action generator creates command-text sequence from a pair of action-policy , derection of sending a ball which generated by planner, and ObjectList that generated by Environment Modeler. The command-text can be send to Soccer-Server directly.

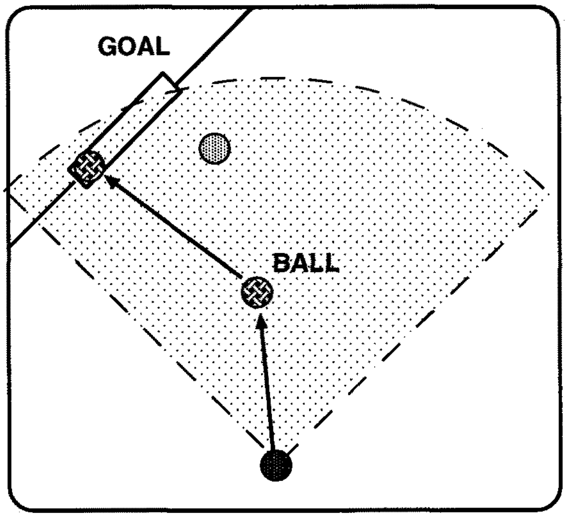

Fig. 3.

Action generater references Oject-List and Action-policy to make a command-text sequence. For example, see upper figure. A Agent position is front of a goal, and he can see a ball that is free. If planner made a acton-policy that say "Get a goal" such as "(4 G)"[1], and send a action generator. Action generator searchs a "Get a goal" pattern in action list that made up by skelton of command sequence. Agent uses following skelton of command sequence for "Get a Goal".

```
if distance(BALL) $\leq 2.0$
    if distance(GOAL) $\geq 50$
        power = 100.0
    if distance(GOAL) $\leq 50$
        power = distance(GOAL) $\times$ 2
    kick(power, derection(GOAL))
    done
```

[1] Symbol 'G' is a Goal symbol"

```
if distance(BALL) $>$ 2.0
    turn(derection(BALL))
    if (distance(BALL) $\leq$ 3.0
        power = distance(BALL) / 3.0
        dash(power)
        done
    if (distance(BALL) $>$ 3.0)
        n = distance(BALL) / 3.0
        $\sum_{i=1}^{n}$ dash(100)
        done
```

In this case, agent made a following command sequence. (Ball position is (-5.0 15.0), Goal position is (-20.0 30))

1. turn(-5.0)
2. dash(100)
3. dash(100)
4. dash(100)
5. dash(100)
6. dash(100)
7. kick(-10.0, 100)

Real-Time Clock : Real-Time clock module controls all of the layers. When agent wants to send any command-text to Soccer-Server, it must have some interval time to send, because SoccerServer can execute only one command in 0.1 second time slice. Therefore agent must send to one command every 0.1 second time slice. Real-Time Clock also provides expire time for old plan. If agent have old plan that can not apply to new game situation , agent must creates new plan for new game situation.

3 Discussion

The MLP is useful architecture to make agent that lives in complexly environment. We consider MLP has 3 features.

First, MLP is made up some modules. Each modules are independence by others. This mean we can make modules separately, and we can implement different methods each modules. now we implimented rule-based decition in SituationAnalyer, and Neural-Network in Planner.

Second, All functions module in MLP can run separately. For example, If planner-module is too hevy to run in machine that you use, you can run a planner-module in faster machine, and you can use network connection for exchange information between modules.

Finally, MLP containg Real-Time Clock module. That module provides time information to modules. Modules can trace environment transition and use past time information effective. now we use time information to timing of sending command and expire a plan.

References

1. Daiji Tamagawa, "Reinforced Learning in Dymamic Environment with Multi-Dimentional State-Trasition" *Aoyama Gakuin University, Graduation thesis 1996*