ZJUNlict
Extended TDP for RoboCup 2015

Chuan Li, Wenjian Tang, Lisen Jin
Yangsheng Ye, Xiaoxing chen, Lingyun chen and Rong Xiong

National Laboratory of Industrial Control Technology
Zhejiang University
Zheda Road No.38,Hangzhou
Zhejiang Province,P.R.China
rxiong@iipc.zju.edu.cn
http://www.nlict.zju.edu.cn/ssl/WelcomePage.html

Abstract. ZJUNlict have participated in Robocup for about ten years since 2004. In this paper, we summarizes the details of ZJUNlict robot soccer system we have made in recent years. we will emphasize the main ideas of designing in the robots' hardware and our software systems. Also we will share our tips on some special problems.

1 Introduction

Our team is an open project supported by the National Lab. of Industrial Control Technology in Zhejiang University, China. We have started since 2003 and participated in RoboCup 2004-2014. The competition and communication in RoboCup games benefit us a lot. In 2007-2008 RoboCup, we were one of the top four teams in the league. We also won the first place in Robocup China Open in 2006-2008 and 2011. We won the first price in 2013 and 2014, which is a great excitation to us. And we incorporate what we have done in recent years to this paper.

Our Team members come from several different colleges, so each member can contribute more to our project and do more efficient job.

2 Hardware

2.1 Mechanical Improvement

According to last years fabrication and assembly process, we slightly modify some components to reduce the interference, such as the lower plate and dribbling structure frame. In the past years, linear open-loop motion in y axis direction wasnt accurate. After mass distribution analysis, we decide to add clump weight in front of the robot. So we can balance the mass distribution specifically on every robot to improve robots kinematic accuracy. In the meantime, we can change the mass damper to test the ball handling stability. It is shown in Figure.1.

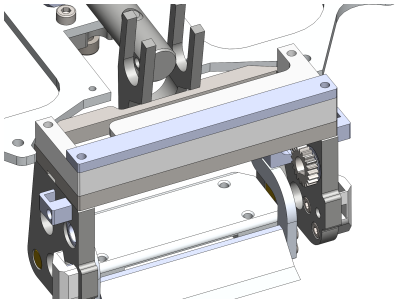


Fig. 1. The clump weight of the robot

2.2 Independent Brushless Motor Drive Circuit

The drive circuits of 5 brushless motors in old edition were placed in a common layer of PCB. But it took us lots of time to detect the error or to find the damaged component when the whole module didnt work. Thus, firstly we separate each three-phase full-bridge drive circuit and place them on a tiny piece of PCB with an interface. That makes the board easily to be pulled out from the base board and changed. Secondly we add an LED to each board so that once the fuse on the board burned, we can notice directly. Thirdly is the phase current detecting function. We replaced every 2-channel voltage comparator with a 4-channel voltage comparator to reduce the quantity of components.

2.3 Frequency Test and Labview Tools Improvement

In order to inspect the communication capabilities of robot more precisely and conveniently, we improved the Labview tool and the five communication check modes, which are listed below.

I. Robot Receive Mode

II. Robot Send Mode

III. Robot Receive and Send Mode

IV. Transmitter Receive Mode

V. Action Mode

Considering that actions of robot may exercise an influence over the communication of robot, we add action model specially. In Action Mode, robot are asked to do several action and we test its communication simultaneously, so we can acquire its real package lose rate in the contest. With the help of the improved communication-test software, we could test every parts of the communication system and acquire more precise package lose rate.

Besides, we add the frequency test function into Labview to simplify the operation. Thus, we can easily press a button to replace the complex operation of serial ports debugging tools.

2.4 Improvement of offline-test-mode

In order to improve the convenience of testing our robots, we modify former offline-test-mode. To be more specific, we need to test our robots to confirm they can communicate well with transmitter in certain frequency without operating the computer to send the packet. So we add offline communication tests as a subsidiary of offline-test-mode. In this mode, robots can automatically communicate with the computer and tell us the test result through the Led.

3 Intelligent Control System

3.1 Hierarchical Architecture of Strategy

The software architecture of the intelligent control system is shown as Fig.2. It is the central module for planning and coordination among robots in both attack and defense modes. The whole system is composed of the World Model, the Decision Module and the Control Module [1].

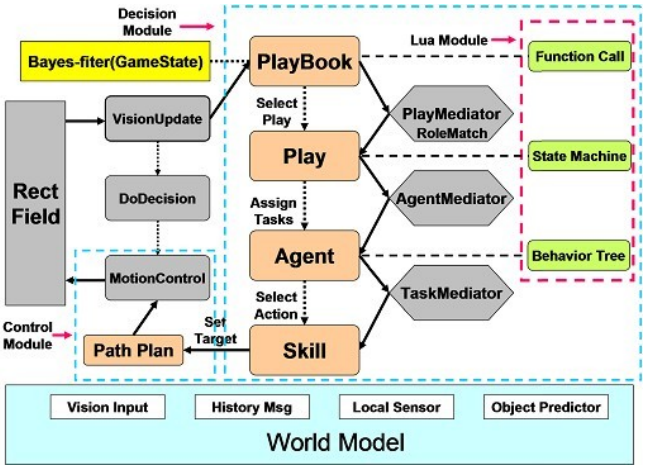


Fig. 2. Software architecture

The Decision Module is designed in a hierarchical structure, which is consist of the PlayBook module [2], the Play module, the Agent module and the Skill module. The PlayBook module selects the appropriate play by using a Bayesian filter to evaluate the game status, as described in section 3.1.1. The Play module focuses on coordination between teammates and is organized by a Finite State Machine (FSM), as described in section 3.1.2. The Agent module emphasizes on the planning skills of the single robot with the assigned tasks from the play. The Agent module selects its behaviors from Behavior Tree (BT), as described in section 3.1.3. The Skill module is a direct interface of the Decision Module

with the Control Module. The Skill module generates target point and selects trajectory generation method, as described in section 3.1.4.

The Control Module is responsible for the path planning and trajectory generation. It traditionally uses the Rapidly-exploring Random Trees (RRT's) algorithm to find a feasible path and Bangbang-based algorithm to solve two-boundary trajectory planning.

Finally, the World Model provides all the information of the match. The History Message records all the decisions from the Decision Module. Besides, the Vision Input means the original vision messages from two cameras, the Local Sensor Message is uploaded from the robots via wireless. The Object Predictor algorithm mainly focuses on our robots' real velocity information in a time-delayed system, and a Kalman Filter method is appropriate for this situation. Thus, the closed-loop system adjusts all robots' behavior according to the change of the environment in real time.

Game State Evaluation A basic problem in SSL is when to attack and when to defend, which means we should evaluate the game status based on observed information. But it's very complicated due to the complexity of game situation. If only the observation is taken into account, the evaluation result will be easily impacted by momentary sensing error, such as ball missing and robot misidentification. Imagining an evaluator only considering ball's position, then a wrong location of the ball would lead to a huge impact on the choice between attack and defense.

Thus, we propose a new method based on the Bayesian Theory [4], which evaluates the game state by combining the observed information and the historical strategy. The new evaluator of the game status is shown as Fig. 3,

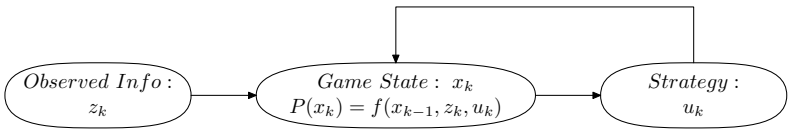


Fig. 3. Bayes-based evaluation method

where z_k is determined by the observation, u_k denotes the attribute of current play script, which can be discretized into values such as 0(attack), 1(stalemate), 2(defense), the game state x_k is calculated using a Bayesian filter method. Fig. 4 depicts the basic Bayesian filter algorithm in pseudo code.

Fig. 5 gives a practical application of Bayes-based filter in the system. We define three states for the game and there are corresponding plays for each state. The selection and switch of the plays is up to the result of the filter. Furthermore, there also exists a score-evaluating mechanism between the plays with the same attribute, this mechanism is realized by the PlayBook module.

The proposed method has two main features:

Algorithm Bayesian filter p(x_k,u_k,z_k)
for all x_k **do**
$\overline{p}(x_k) = \sum_{x_k} p(x_k|u_k, x_{k-1})p(x_{k-1})$
$p(x_k) = \eta p(z_k|x_k,)\overline{p}(x_k)$
end for

Fig. 4. General algorithm for Bayesian filter

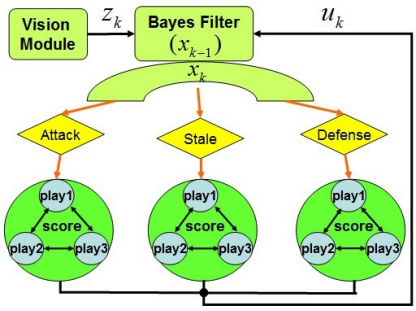


Fig. 5. Bayes Processing Flow Chart

- **More stable:** Compared with the evaluation method that only considers observation, our evaluator gives a more appropriate analysis of the game state and helps to reduce the perturbation of the strategy and to strengthen the continuity of the strategy.
- **More flexible:** The values of prior probabilities $p(x_k|u_k, x_{k-1})$ and $p(z_k|x_k)$ is predefined in code according to different team's characteristics, making it more convenient and flexible to configure the attacking and defending strategy.

Play and Finite State Machine Each play represents a fixed team plan which consists of many parts, such as applicable conditions, evaluating score, roles, finite state machine and role tasks, based on CMU's Skill, Tactics and Plays architecture [2]. The plays can be considered as a coach in the soccer game, who assigns different roles to robots at different states.

For a play script, the basic problem is "What should be done by Whom at What time". So we develop a novel FSM-based Role Match mechanism. Traditionally, a state node is described by the execution and switch condition [2]. We coupled a role match item in each state node to this basis. The new framework is shown as Fig.6, a role match item comprises several matching groups with priority, the priority determines the groups' execution order. Our goal is to find an optimal solution for every group by Munkres assignment algorithm [3], using the square of the distance between the current positions of the robots and expected roles' target positions as the cost function. In the implementation, we

just consider five roles matching in the scripts, because the goalie corresponds a fixed robot by default.

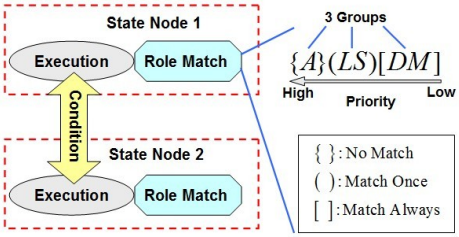
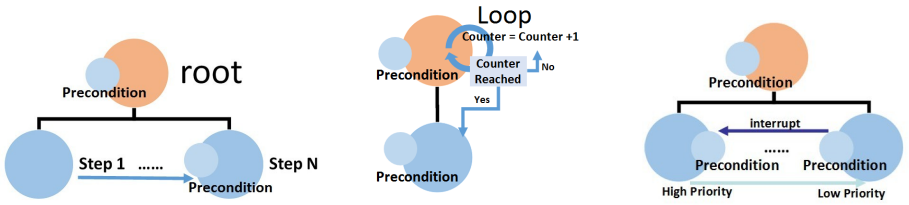


Fig. 6. FSM-based Role Match Framework. A role match item's syntax is described on the right: the letter in the string is short for the role's name in the state node(A-Assister, L-Leader, S-Special, D-Defender, M-Middle). We offer three modes for matching, *No Match* means to keep the role as same as last state, *Match Once* means match the role once stepping into this state, and *Match Always* means do role match every cycle. If a robot is missing, then ignoring the last role in the item and *No Match* group always has a higher priority.

Agent and Behavior Tree Agent-level is responsible for the behavior planning such as manipulating ball to proper region, passing ball to a teammate and scrambling ball from the opponent, when a play-level decision is made. The main work of agent-level is to select a proper skill and the best target for the executor in each cycle. In order to make the behavior selection more intelligent and human-like, we build a behavior tree, which consists of a set of nodes including control nodes and behavior nodes.

– **Control Nodes** are utilized to set the logic of how different behavior nodes should be connected. In our agent behavior tree system, there are mainly three types of control nodes as shown in Fig.7. The first type is a sequence node that ensures all of its child nodes will be executed in a deterministic order if the precondition of the children node is satisfied. For example, as shown in Fig.7(a), when the precondition of the root node is satisfied, the step 1 node will be executed. After the step 1 node has been executed and the precondition of the step 2 node is satisfied, the step 2 node will be executed. This rule will last until the last child node is executed. The second type is a loop node which works like a counter. The execution of its child node depends on two conditions: one is that the precondition of this node is satisfied; the other is that the control node should have been executed for a predefined time period as illustrated in Fig.7(b). The third type is a priority selector node which executes its children nodes actions according to their priority. When the preconditions of a node with the higher priority have been satisfied, the existing node will be interrupted and the executing of the

node with the higher priority will be executed, see Fig.7(c). All the nodes are associated with external preconditions that must be satisfied before the executing of the node. These preconditions should be updated before the update of the behavior tree.



(a) (b) (c)

Fig. 7. Three types of control node: (a)sequence node; (b)loop node; (c)priority selector node

– **Behavior Nodes** are simply a set of atomic skills such as going to a specific position. They are all leaf nodes of a behavior tree and saved in an atom skill factory. They are also associated with preconditions that should be satisfied before executing.

Skill Skill is a set of basic knowledge of actions different robots can perform, such as how to move to a point, how to get the ball and kick. Some skill will generate a next target point, which will be passed to the navigation module for path planning and trajectory generation. Some skill will generate the speed trajectory for some special behavior such as pulling the ball from an opponent front. Parameter tuning is always an important work for skill's performance in the competition.

3.2 Defense Strategy in AI System

In this section, we mainly introduce our defense which is a vital part for competition and a superiority of our team. We only lost one point in the RoboCup 2014 tournament, which verifies the effectiveness of our strategy. Our defense is called Close-Marking Defense.

A flow chart of Close-Marking Defense is shown as Fig.8. We get the information about all opponent robot and calculate feature value. According to the attribute value we will match the role for every opponent robot,such as leader, passer, etc. Then attack array is set up to describe the robot in order and design defense strategy at last.

Feature Value We calculate all feature values for every opponent robot according to vision messages. Feature values, which is used to estimate the threatening

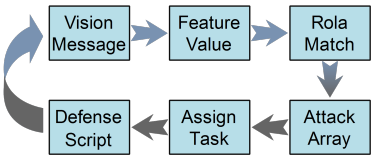


Fig. 8. Defend flow chart

level, reflects the state of opponent robots. We have more than a dozen of feature, some are simple value which can be calculated by vision message according to some obvious geometrical relationship, just like the distance between robot and ball, the distance between opponent robot and our goal, shoot angle and so on. The other are some complex feature value which can be calculated by simple feature value, just like Touch Ball value(the ability to receive ball and shoot), Chase Ball value(the ability to chase all and shoot), Pass Ball value(the ability to chase and shoot). For example, Touch Ball value of an opponent robot depends on the shoot angle of the opponent robot, whether other robots block in the pass line, the distance between the robot and our goal. All These features will be used in the next step opponent robots role matching.

Opponent Robots Role Match Different roles have different priorities. A default role group includes:

- **receiver** is an offensive role which receives the ball and finishes shooting.
- **leader** is an offensive role which controls the ball.
- **attacker** is an offensive role which is always ready for the attack.
- **defender** is a defensive role which takes part in defense strategy.
- **goalie** is the goalie.

The role matched degree is calculated according to the features of robot. For example, when we judge whether the robot is a receiver, we will use three feature value: Touch Value(the ability to receive ball and shoot), Chase Value(the ability to chase all and shoot), Receive Angle(the angle between ball receiving and ball shooting). Matched degree will be calculated for each robot in priority decreasing order.

Attack Array Attack array is a list of opponent robots. In attack array, We generate the attack array according to the match result. Now every opponent robot has their role and the matched degree for this role. We can compare the role priority and matched degree to set up attack array. But we don't need to mark for certain roles, such as goalie, opponent blocker in kick off area, which should be ignored.

Design Defend Script We design our defence script in the form of a Finite State Machine[6]. The task is assigned in a state and can receive a parameter

which is a number representing the order in attack array. So the robot will defend the corresponding opponent in attack array.

There is an example to demonstrate our defense. Fig.9 shows the task assignment in the defense script. Defend Kick is a task defending the opponent robot which takes the free kick. We can see marking task receiving a parameter which is just an order in the attack array.

A simulation is shown as Fig.10. Yellow team is the attack side. There are five attackers, four are of attacker role, and one is of leader role. Their defense superiority order is 4-3-1-5-2.

```
["DefendState"] = {
        Leader   = task.defendKick(),
        Special  = task.marking("First"),
        Middle   = task.marking("Second"),
        Defender = task.marking("Third"),
        Assister = task.marking("Fourth"),
        Goalie   = task.goalie(),
}
```

Fig. 9. a part of defense script

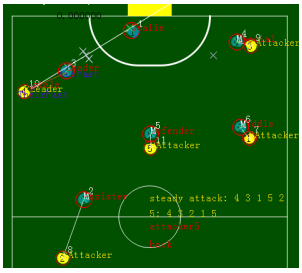


Fig. 10. defense simulation

4 Conclusion

Owing to our all team member hard work, we can obtain this result. If the above information is useful to some new participating teams, or can contribute to the small size league community, we will be very honor. We are also looking forward to share experiences with other great teams around the world.

References

1. Yonghai Wu, Penghui Yin, Yue Zhao and Yichao Mao, Rong Xiong: ZJUNlict Team Description Paper for RoboCup2012. In: RoboCup 2012 (2012)
2. B. Browning, J. Bruce, M. Bowling and M. Veloso: STP: Skills, tactics and plays for multi-robot control in adversarial environments. In: J. Syst. Control Eng., vol. 219, no.I1, pp. 33-52 (2005)
3. J. Munkres: Algorithms for the assignment and transportation problems. In: Journal of the Society for Industrial & Applied Mathematics 5.1: pp. 32-38. (March 1957)
4. S. Thrun, W. Burgard, and D. Fox: Probabilistic robotics. Vol. 1. Cambridge: MIT press (2005)
5. R. Ierusalimschy. Programming in Lua. Lua.org, 2nd edition (2006)
6. Yonghai Wu, Yue Zhao, Rong Xiong, ZJUNlict Team Description Paper for RoboCup2013 (2013)