Jolly Pochie 2006 : Team Description Paper

Hayato Kobayashi[1], Akira Ishino[2], and Ayumi Shinohara[3]

[1] Department of Informatics, Kyushu University
h-koba@i.kyushu-u.ac.jp
[2] Office for Information of University Evaluation, Kyushu University
ishino@i.kyushu-u.ac.jp
[3] Graduate School of Information Science, Tohoku University
ayumi@ecei.tohoku.ac.jp

1 Introduction

The team "Jolly Pochie [dʒóli·pótʃi:]" has participated in RoboCup Four-Legged League since 2003. In the first two years, the team consisted of the faculty staff and graduate/undergraduate students of department of informatics, Kyushu University. Since 2005, it becomes a united team with Tohoku University.

Faculty members
Ayumi Shinohara, Akira Ishino
Student members
Hayato Kobayashi, Satoshi Abe, Akihiro Kamiya, Tsugutoyo Osaki, Tetsuro Okuyama, Shuhei Yanagimachi, Keisuke Oi, Wataru Matsubara, Takahito Sasaki, Tomoyuki Nakamura, Seiji Hirama, Eric Williams

Our research interests mainly include machine learning, machine discovery, data mining, image processing, string processing, software architecture, visualization, and so on. RoboCup is a suitable benchmark problem for these domains. Last year, we had utilized an scripting language in order to accelerate the development process, and developed a simulator which directly executes scripts on PC just as in the robots. In addition, we have developed a new localization technique for the ball location. Moreover, we have tried to apply an autonomous learning method into real robots in order to acquire ball trapping skills. This paper presents our current status of the development, mainly focused on the progress since the last year. Section 2 introduces our original framework. In Section 3, we give outline of our image processing system. Section 4 shows the ball localization techniques. We show how to learn ball trapping skills in Section 5.

2 Overview of Our Framework

Our framework [1] is mainly based on two techniques. One is the plug-in system, and the other is embedding a scripting language. The plug-in system helps the development by allowing several programmers to collaborate. We can simplify each task by splitting a huge, complex process into individual functions. Embedding a scripting language helps us eliminate the trial and error process and the

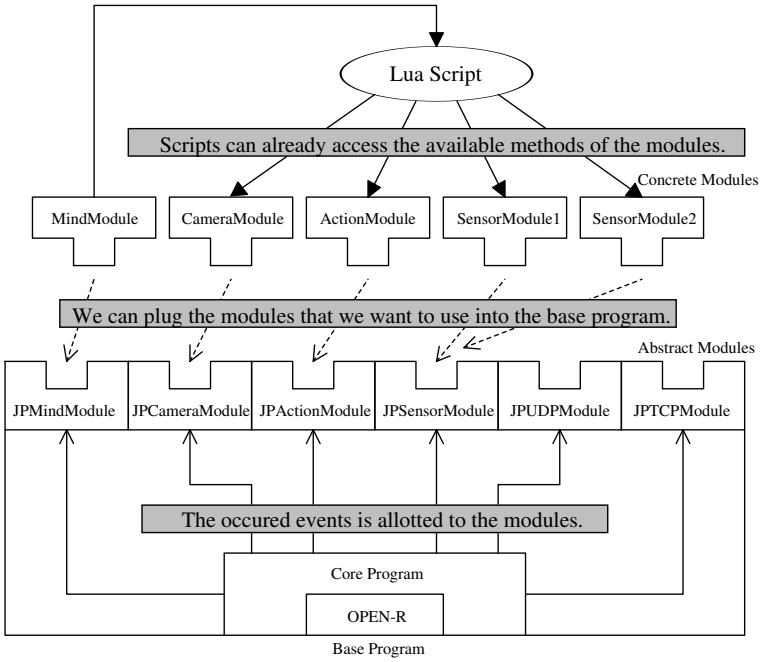

Fig. 1. The overview of our framework.

time it demanded. By making high-level scripts that do not need to recompile, our tasks can progress more efficiently.

Figure 1 shows some features of our framework. This is the programming procedure in our framework. We first separately create each low-level module. Then, we select some modules that we want to use, and plug the selected modules into the base program. For example, we would select a vision module, a localization module, a motion module, and so on, to make a robot that plays soccer. The robot program is generated easily by an automation tool for this procedure. We can build a binary program by compiling this robot program. Finally, we can write scripts for performing high-level processes by using the methods defined in the modules. In order to replace certain modules with others, once again, we only need to select modules, generate a program, and compile it. Nevertheless, we need not rewrite the script.

2.1 Plug-in System

Our plug-in system consists of a base program and individual modules. The base program is the foundation of this system and common to all the robot programs. Concretely speaking, it is composed of classes wrapping the OPEN-R SDK. The most important class in the base program is a class `JPObject`, which inherits the class `OObject` in OPEN-R SDK. We create a robot program for AIBO by

Table 1. The specifications of special functions.

Module	Special Function	When is the function called?
`JPCameraModule`	`cameraNotify()`	Every 40 ms in sync with the CCD-camera
`JPMindModule`	`mindNotify()`	The same as `cameraNotify()`
`JPActionModule`	`actionNotify()`	When a set of joint angles are achieved
`JPSensorModule`	`sensorNotify()`	When sensor data is detected
`JPUDPModule`	`udpNotify()`	When UDP data is received
`JPTCPModule`	`tcpNotify()`	When TCP data is received

making a subclass of the class `OObject`, *i.e.* the class `JPObject` is the core of our robot programs. The class `JPObject` registers the instances of the constructed modules, and calls the modules every time certain events occur.

In order to create modules, we only need to make a subclass of *abstract classes* (*e.g.* `JPCameraModule` for processing camera images, `JPActionModule` for calculating joint angles during motion, `JPSensorModule` for managing information from sensors, and `JPMindModule` for developing strategies). The abstract classes have various *special functions* that are called when the class `JPObject` receives certain events. Table 1 shows the specifications of the special functions. For instance, the class `JPCameraModule` has the function `cameraNotify()` called every 40 ms in sync with the frame rate of the CCD-camera. That means, to get an image from the CCD-camera, we only need to make a subclass of the class `JPCameraModule`. In the same way, we can easily create various other modules. By selecting certain modules, we can make various robot programs not only for soccer players, but also for other events, such as the open challenge.

2.2 Embedding Lua

We embedded a scripting language, Lua [2] so that mind modules creating strategies can call Lua scripts and so that Lua scripts can call methods in other C++ modules. In a mind module, the Lua function `mindNotify()` is called in the C++ member function `mindNotify()`. We can quite easily call a Lua function by using Luabind [3], which is a library that lets us intuitively create bindings between C++ and Lua.

In order for Lua scripts to call methods in other C++ modules, it is necessary to bind the modules on the C++ side. This means we must register an instance of the modules, as well as information of the classes and member functions. Using Luabind, we can also easily bind C++ modules *in functions*. When embedding scripting languages (*e.g.* Lua, Python, and Perl), we typically must define global wrapper functions for functions that we want to bind. This means we rewrite the bindings whenever we exchange modules. However, Luabind can cut out this annoying task because it is implemented utilizing template meta programming.

Luabind can also register information regarding class inheritance. We need not bind in the class `AdvancedExampleModule` inheriting the class `ExampleModule`. Therefore, we need not rewrite our scripts even if we exchange these modules.

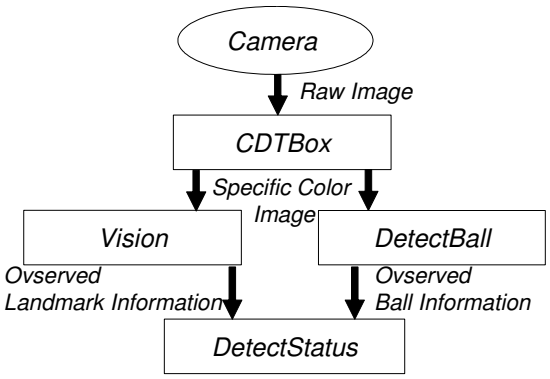

Fig. 2. The flowchart of Image Processing

The same is true for compatible modules in terms of bindings (*e.g.* `ExampleModule2` having the same methods that `ExampleModule` has).

2.3 Specification of Our Robot Scripts

After embedding Lua, high-level processes can be described as scripts. In this accomplished framework we can create robot scripts with simple rules that everyone easily understand. The function `init()`, where we can initialize variables, is called only once at the beginning. The function `mindNotify()` is called by the member function `mindNotify()` in the mind module. That is to say, it is called every 40 ms.

3 Image Processing

In RoboCup, image processing is one of the most difficult problems. Last year, our robots consumed most of the time for image processing. This year, we reconstructed our image processing system at first, and introduced new approaches. Our image processing system consists of three modules, CDTBox, Vision and DetectBall. CDTBox module converts original 24-bit colors into 8 specific colors. Vision module recognizes landmark objects, and DetectBall module recognize the orange ball. Figure 2 shows the flowchart of image processing.

We developed a tool for color classification (show Figure 3). The usage of this tool is as follows. First, choose a color in the group (Red Ellipse in Figure 3) and click a point in images on the tool (Yellow Boxes). Then the color of the point is learned. Eight specific color, red, blue, yellow, cyan, pink, green, white, and orange, are learned. Those are the color of uniform of robots, goal, pole, field, line and ball. Colors that does not relate with a game is learned as negative samples.

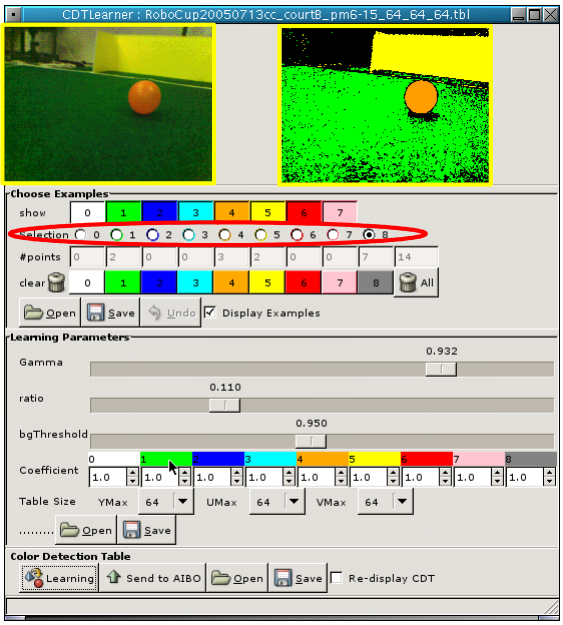

Fig. 3. Color Table Making Tool

4 Ball Localization using Monte-Carlo Method

We proposed a technique which estimates the position and velocity of a moving ball based on the Monte-Carlo localization [4]. The aim is to calculate the accurate position and velocity of the ball from a series of input images. We tried three variations of the method, in order to estimate both the position and velocity of the moving ball. (1) Each sample holds both the position and velocity, but is updated according to only the information of the position. (2) Each sample holds both the position and velocity, and is updated according to the information of both the position and velocity. (3) Two kinds of samples are considered: one for the position, and the other for the velocity, which are updated separately. Among them, the third method performed the best in our experiments. We will briefly summarize the third method below.

The idea is to split the samples into two categories, one for the positions $\langle \boldsymbol{p}_i, s_i \rangle$, and the other for the velocities $\langle \boldsymbol{v}_i, s_i \rangle$. The score of $\boldsymbol{p}$ is updated in step $8 \sim 11$ of *PositionUpdate* while that of $\boldsymbol{v}$ in step $2 \sim 5$ of *VelocityUpdate* independently, in Figure 6.

The line *MonteCarlo* (*two sets*) in Figure 5 shows the results. The estimated positions fits the pathway of the rolling ball, and the predicted trajectory when ball was out of sight is also as we expected. The effectiveness of it is comparable to the Kalman filter method in Figure 4.

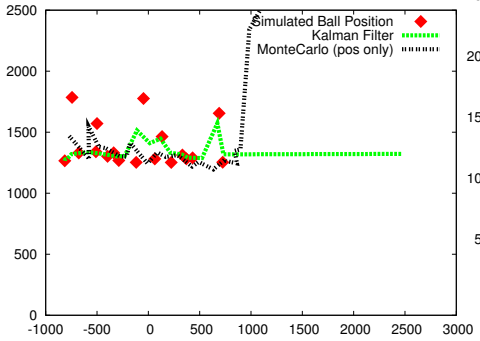

Fig. 4. The result of the simulation with Kalman filter and MonteCarlo (pos only)

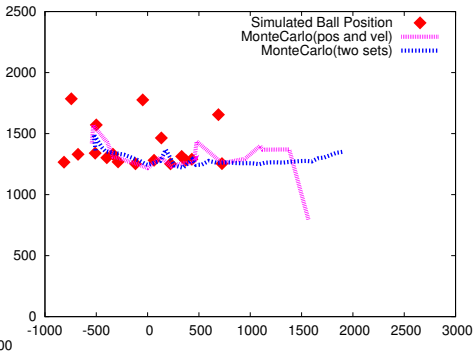

Fig. 5. The result of the simulation with MonteCarlo (pos and vel) and MonteCarlo (two sets)

4.1 Real-world Experiments

We also performed many experiments in the real-world environments, at which we used the real robot in the soccer field of RoboCup competitions. We show some of them in Fig. 7, where we compared the third method which we proposed with the Kalman filter method. The purpose was to evaluate the robustness of the methods against the obstacle and the change directions of the ball movement. In the left figure, the ball rolled from right to left while a small obstacle in front of the robot hides for some moments. In the right figure, the ball was kicked at $(400, 900)$ and went left, then it is rebounded twice at $(-220, 1400)$ and $(-50, 500)$, and disappeared from the view to the right. Mesh parts in the figures illustrates the visible area of the robot. From these experiments, we verified that the proposed method is robust against the frequent change of the directions, which is often the case in real plays.

5 Ball Trapping

Passing (including receiving a passed ball) is one of the most important skills in soccer and is actively studied in the simulation league. For several years, many studies [5, 6] have used the benchmark of good passing abilities, known as "keepaway soccer", in order to learn how a robot can best learn passing. However, it is difficult for robots to even control the ball in the real robot leagues. In addition, robots in the four-legged robot league have neither a wide view, high-performance camera, nor laser range finders. As is well known, they are not made for playing soccer. Quadrupedal locomotion alone can be a difficult enough challenge. Therefore, they must improve upon basic skills in order to solve these difficulties, all before pass-work learning can begin. We believe that basic skills should be learned by a real robot, because of the necessity of interaction with a real environment. Also, basic skills should be autonomously learned because

Algorithm BallMoteCarloLocalizationWithTwoSets
Input. Two sets of samples $POS = \{\langle \boldsymbol{p}_i, s_i \rangle\}$ and $VEL = \{\langle \boldsymbol{v}_i, s_i \rangle\}$, observed ball position $\boldsymbol{p}_o$, calculated ball velocity $\boldsymbol{v}_o$.
Output. estimated ball position $\boldsymbol{p}_e$ and velocity $\boldsymbol{v}_e$.

PositionUpdate(POS, $VEL, \boldsymbol{p}_o, \boldsymbol{v}_o$)

```
1   v_e := VelocityUpdate(VEL, p_o, v_o);
2   for i := 1 to n do begin
3     p_i := p_i + v_e;
4     if p_i is out of the field then
5       randomize(⟨p_i, s_i⟩)
6   end;
7   if p_o ≠ ε then
8     for i := 1 to n do begin
9       score := exp(−τ_p|p_i − p_o|);
10      s_i := max(s_i − maxdown_p,
                   min(s_i + maxup_p, score))
11    end;
12  p_e = 0;    w = 0;
13  avgScore := (1/n) Σ_{i=1}^{n} s_i;
14  for i := 1 to n do begin
15    if s_i < avgScore · random() then
16      randomize(⟨p_i, s_i⟩)
17    else if s_i > t_p then begin
18      p_e := p_e + s_i p_i;
19      w := w + s_i
20    end;
21  end;
22  p_e := p_e/w;
23  output p_e, v_e
```

VelocityUpdate($VEL, \boldsymbol{p}_o, \boldsymbol{v}_o$)

```
1   if p_o ≠ ε then
2     for i := 1 to m do begin
3       score_new := exp(−τ_v|v_i − v_o|);
4       s_i := max(s_i − maxdown_v,
                   min(s_i + maxup_v, score))
5     end;
6   v_e = 0;    w = 0;
7   avgScore := (1/m) Σ_{i=1}^{m} s_i;
8   for i := 1 to m do begin
9     if s_i < avgScore · random() then
10      randomize(⟨v_i, s_i⟩)
11    else if s_i > t_v then begin
12      v_e := v_e + s_i v_i;
13      w := w + s_i
14    end;
15  end;
16  v_e := v_e/w;
17  output v_e
```

Fig. 6. Procedure Ball Monte-Carlo Localization with two sets of samples

changes to an environment will always consume much of people's time and energy if the robot cannot adjust on its own.

There have been many studies conducted on the autonomous learning of quadrupedal locomotion, which is the most basic skill for every movement. However, the skills used to control the ball are often coded by hand and have not been studied as much as gait learning. We studied an autonomous learning method for ball trapping skills [7], where we restricted the model in one-dimension. We prepared some training equipment and then experiment with only one robot. The robot could use our method to acquire these necessary skills on its own, much in the same way that a human practicing against a wall can learn the proper movements and actions of soccer on his/her own. We also experimented with two robots, and our findings suggest that robots communicating between each other can learn more rapidly than those without any communication.

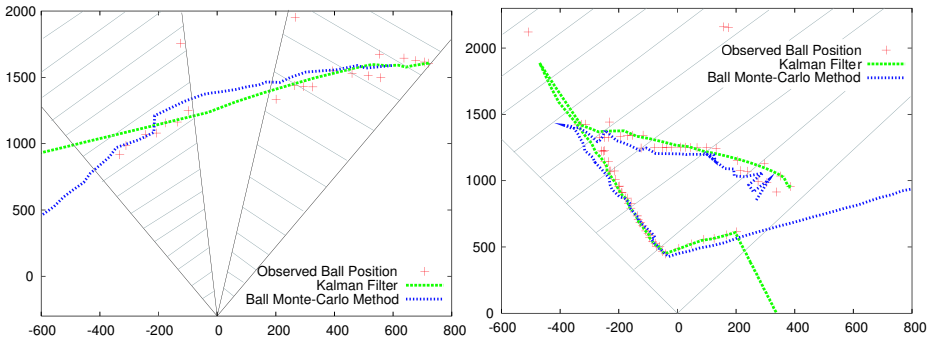

Fig. 7. Real world experiments. In the left situation, the ball rolled from right to left behind a small obstacle. In the right situation, the ball started at $(400, 900)$ and rebounded twice at $(-220, 1400)$ and $(-50, 500)$, and disappeared from the view to the right.

6 Concluding Remarks

This paper described the footstep of Jolly Pochie in this year. We developed a new localization technique for the ball location and suggested an autonomous learning method for ball trapping skills. We hope to use this trapping skills for actual games as well as the Passing Challenge.

References

1. Hayato Kobayashi, Akira Ishino, and Ayumi Shinohara. A framework for advanced robot programming in the RoboCup domain. In *Proc. Intelligent Autonomous Systems 9 (IAS-9)*, pages 660–667. IOS-Press, 2006.
2. The Programming Language Lua. `http://www.lua.org/`.
3. Luabind. `http://luabind.sourceforge.net/`.
4. Jun Inoue, Akira Ishino, and Ayumi Shinohara. Ball tracking with velocity based on monte-carlo localization. In *Proc. Intelligent Autonomous Systems 9 (IAS-9)*, pages 686–693. IOS-Press, 2006.
5. Peter Stone, Richard S. Sutton, and Gregory Kuhlmann. Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.
6. William H. Hsu, Scott J. Harmon, Edwin Rodriguez, and Christopher Zhong. Empirical comparison of incremental reuse strtegies in genetic programming for keep-away soccer. In *Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference*, 2004.
7. Hayato Kobayashi, Tsugutoyo Osaki, Eric Williams, Akira Ishino, and Ayumi Shinohara. Autonomous learning of ball trapping in the four-legged robot league. In *Proc. RoboCup International Symposium 2006*, LNCS. Springer-Verlag, 2007. to appear.