Jolly Pochie 2005 in the Four Legged Robot League

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

[1] Department of Informatics, Kyushu University
{j-inoue, 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

Abstract. Jolly Pochie participates for the third time in RoboCup Four-Legged League. This paper presents the main development of our team in 2005: adjusting gaits parameters by genetic algorithm, embedding a scripting language, and the development of a simulator, which directly executes the scripts.

1 Introduction

The team "Jolly Pochie [dʒóli·pótʃi:]" has participated in RoboCup Four-Legged League since 2003. In the last two years, the team consisted of the faculty staff and graduate/undergraduate students of department of informatics, Kyushu University [1]. This year it becomes a united team with Tohoku University.

Faculty members
Ayumi Shinohara, Akira Ishino
Graduate Students
Jun Inoue, Hayato Kobayashi, Narumichi Sakai, Kazuyuki Narisawa, Satoshi Abe, Akihiro Kamiya
Undergraduate Students
Tsugutoyo Osaki, Tetsuro Okuyama, Shuhei Yanagimachi, Yuki Matsumoto

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 three wins and two losses in the preliminary league. For this year, we utilized an embedded scripting language in order to accelerate the development process, and established a simulator which directly executes the script on PC just as in the robots. In addition, we are developing a new localization technique for the ball location.

Section 2 introduce our original framework. An overview of the designing motions and genetic algorithm are illustrated in Section 3. Section 4 shows the self and ball localization techniques. Section 5 gives outline of our vision system. The scripting language and simulator are described in Section 6. Section 7 concludes this paper.

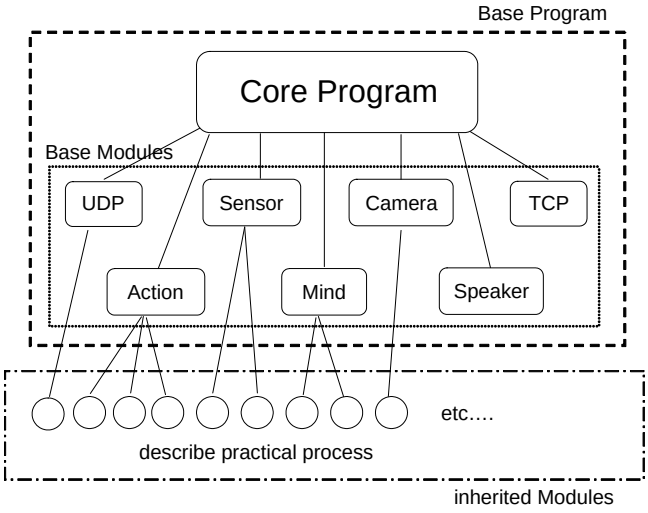

Fig. 1. Jolly Pochie Framework

2 Jolly Pochie Framework

Our team Jolly Pochie has developed an original framework. We created various modules which have individual functions based on it. Combining suitable modules, we can design various robots easily. By this framework, each programmer concentrates on developing each module separately. Fig 1 shows the outline of Jolly Pochie Framework.

The framework hides the original types and functions in OPEN-R [2], so that we can implement the modules using the standard C++ class libraries, and can test and debug the modules in ordinal environments. Each module calls no OPEN-R functions directly.

Each module inherits the base modules. There are many kinds of base modules, such as camera module which takes image processing, action module which computes joint angles in the motions, mind module which selects appropreate strategy. Especially, the mind module is a central part which controls all the other modules. Therefore we embedded the scripting language Lua mainly for the mind module. The mind module has a special function named `mindNotify` which is called every 40ms. Strategies are described in this function. For example, we implemeted the following procedures, receiving the data from the camera and sensors, and actuating the robot through action module, and so on. Moreover, a state transition model is introduced in the mind module, so that `mindNotify` calls different functions depending on the states. By embedding the scripting language Lua as shown in Section 6, we can also write the functions in Lua scripts.

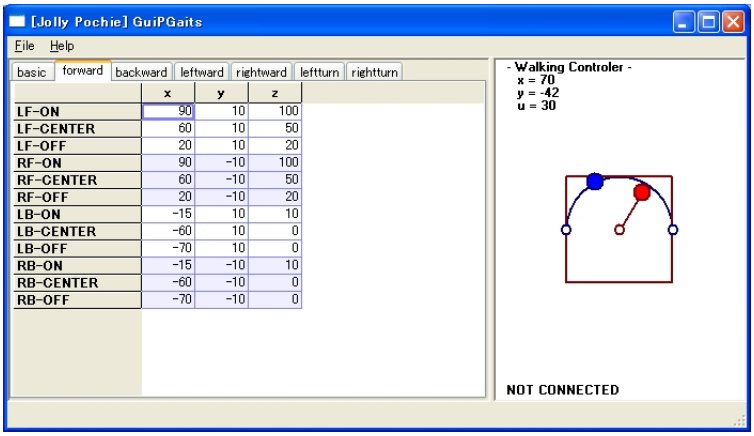

Fig. 2. Gaits Developing Tool

3 Motion

Designing a good motion, which is quite important in the game, is an apparently time consuming task. A lot of trials and errors are required. In order to ease these tasks, we have developed various tools. Fig 2 shows the gaits developing tool. However, fine tuning of various parameters by hand are very hard tasks. Thus we have also examined an automatic adjustments by applying genetic algorithms. The score function evaluated the speed of the gait, by measuring the distance of the movement for a fixed period, from a camera equipped on the ceiling. After several thousand trials, we established a fast gait, 429 mm/sec for ERS-7.

3.1 Measurement Environment

In order to capture the position of a robot from the ceiling camera, we use two colored balls equipped on the back of the robot, as makers. By these makers, the ceiling camera can determine the location and the direction of the robot easily and accurately. The robot can receive these information from the camera via wireless network. Fig 3 shows the robot with the markers.

3.2 Genetic Algorithm

Every gait in our locomotion system is specified by the following parameters.

number of frames
: The number of frames in one cycle of the gait.

landing point, leaving point, and top point
: The positions at which each leg reaches the ground, and leaves from it, and the highest position. In landing point and leaving points, the hight does not necessarily equal to be 0, which can changed arbitrarily.

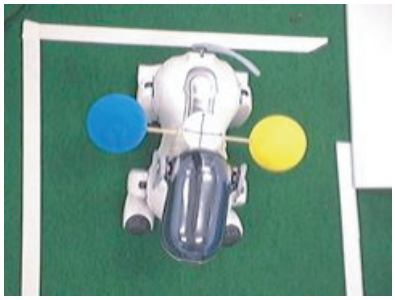

Fig. 3. Marker AIBO

front height of body, back height of body
They determines the position and posture of the body.
power rate
The time ratio of each paw touches the floor.
phase difference
It specify the phase difference betwen the front left leg and the rear right leg.

In our genetic algorithm, we set the size of populations to be 50. The mutation probability is 10%. In the initial population, each gene has random values as the parameters. For each gene, the robot moves actually and evaluates the fitness score. Among 50 genes, the best 20 genes will be alive in the next generation, and the rests are exchanged. The crossover operation is executed for these 20 genes, at any points. The fitness score is measured by the distance between the starting point and the end point for a fixed period. By this system, we could develop fast gaits automatically.

4 Localization

4.1 Self Location

Localization is an important task for automatic positioning, shooting and team coordination. We use Monte Carlo localization technique for self localization. The localization is based on the distances and directions of landmarks, e.g. poles and goals, but it is applicable even if robots are in the situation that the landmarks are invisible. Robots remembered the moving average of relative locations of landmarks in the fixed period, which is one cycle of swinging their head. The localization module has 1000 seeds.

4.2 Ball Location

Last year, our robots play soccer based on the location of the ball which is observed directly. However, in soccer game, the position of the ball is fluctuating.

Moreover, the robot has narrow view so that it frequently lost the ball. It is difficult for the robots to keep the correct position of the ball only based on the direct observation. Therefore we are trying to develop a system to infer the position of the ball even when it is invisible, based on Monte Carlo method.

The aim of this system is to estimate the position and velocity of the ball as accurate as possible. Especially, the velocity is critical to infer the trace of the ball which is out of view. It is impossible to estimate the velocity of the ball. But we consider the velocity of the ball. The hypotheses which have wrong velocity take low scores in the resampling phase. The procedure of the update is as follows.

```
b := array of sample[1, ..., m]; : set of hypothesis
procedure ballmcl(observation,motion)
 begin
  if (observation ≠ ϵ) then
    b := updatebyobservation(b, observation);
  if (motion ≠ ϵ) then
    b := updatebymotion(b, motion);
  b := resampling(b);
  bloc := decidelocation(a);
  output bloc;
 end;
```

5 Vision

In RoboCup, image processing is one of the most important tasks. In fact, our robots consume most time for image processing. Our vision system consists of three modules, CDTBox, Vision and DetectBall. CDTBox module detects 8 specific colors from the original 24-bit color image. Vision module recognizes the landmark objects and DetectBall module recognizes the orange ball, from the output of CDTBox.

Last year, we used 6 threshold values, y-max, y-min, u-max, u-min, v-max and v-min, to specify each color. In another word, each color is expressed by an axis-parallel rectangular region in YUV color space. However, it turned out that the rectangular does not fit well to separate the colors accurately. The HSV color space was more robust under the changing light conditions, but the conversion from YUV to HSV in realtime was not a trival task. Therefore in this year, we use 3 dimensional table in YUV space, where the colors are turned in HSV space.

6 Script Language and Simulator

The biggest change of our system in this year is innovation of scripting language. Last year, we use C++ to describe our strategy. It causes high cost of debugging and we had to compile to verify changes. This year, we embedded Lua [3, 5] in our system to resolve these problems.

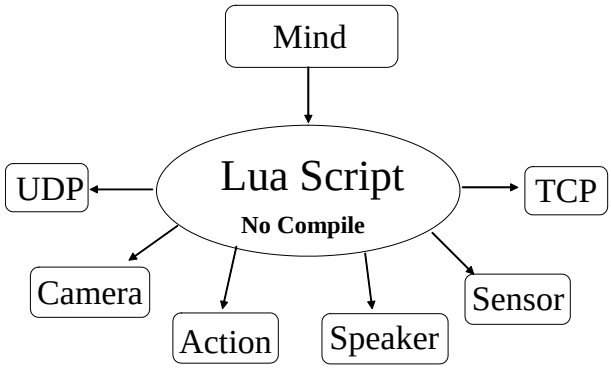

Fig. 4. Embedding Lua Script

6.1 Lua

Lua is a scripting language designed to support embedding. A basic Lua engine, including parser/compiler/interpreter but excluding standard libraries, weighs in at under 100kb. Moreover, it is highly portable. Therefore, we could easily embed Lua in the Jolly Pochie Framework.

The program which embedded Lua called *host*, and this host program can invoke functions to execute a piece of Lua code, can write and read Lua variables, and can register C/C++ functions to be called by Lua code.

Generally, we have to operate a *virtual stack* using API for connecting between script and C. However, in the Jolly Pochie Framework, we have used Luabind library [4] to do it automatically.

6.2 Lua Simulator

Testing scripts needs robots, it requires a lot of trials and prevents speedy development. Therefore, we developed a simulator which runs Lua scripts without any change. The simulator enables us to develop strategies without robots. It improves our development process drastically. Fig 5 shows that the difference between real machine and simulator: pseudo environment simulating environment in the real world, pseudo active modules emulating avtive modules, pseudo passive modules emulating passive modules. But lua scripts is the same in both environments.

Pseudo Environment A pseudo environment simulates the real environment, including a soccer field, a ball and AIBOs, in a real world. In the simulator, we can verify that the robot can watch the ball or not by represented these objects as 3D models. It is implemented in VPython [6] which is a 3D graphics module for python.

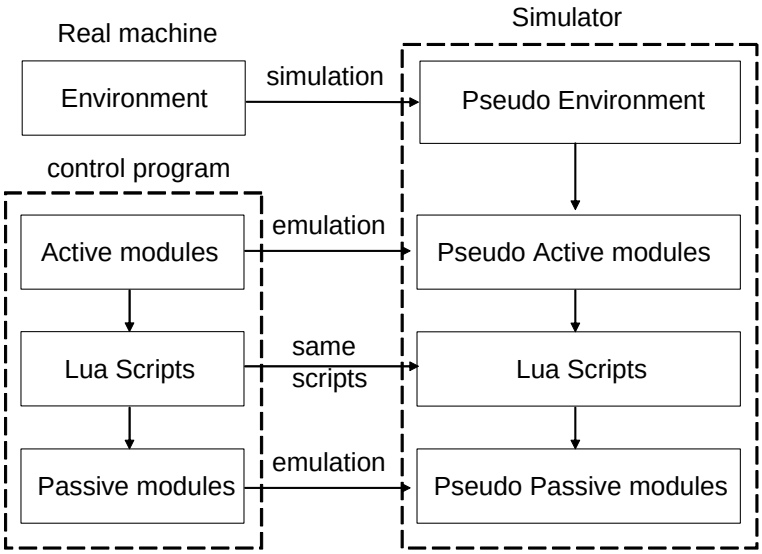

Fig. 5. real machine and simulator

Pseudo Active Modules Pseudo active modules emulate the modules in the Jolly Pochie Framework. These modules call functions in scripts at either regular intervals or an event happens. Specifically, the pseudo active modules consist of a mind module, action module, UDP communication module, and so on.

Pseudo Passive Modules Pseudo passive modules emulate passive modules outside of the Jolly Pochie Framework. Functions in these modules are called by functions in scripts. For example, pseudo passive modules are modules which make the robot walk or shoot, and calculate a position of a ball for the image prcessing. In the simulator, we only define dummy functions on Lua side, because it doesn't need a self localization and an object recognition.

7 Concluding Remarks

We have explained the current status of our development. This year, we improved many things in our system. Especially, embedded scripting language has a large increase in speed of our development, and genetic algorithm found good parameters for various gaits. As other team always do, we are continuing the development and improving day by day. Therefore it is quite natural that our programs will be changed drastically.

Fig. 6. Lua simulator

References

1. JollyPochie —team for robocup soccer 4-legged robot league—. `<http:://www.i.kyushu-u.ac.jp/JollyPochie/>`.
2. AIBO SDE Homepage. `http://openr.aibo.com/openr/jpn/index.php4`.
3. Roberto Lerusalimschy, Luiz Henrique de Figueiredo, and Waldemar Celes Filho. Lua—an extensible extension language. *Software—Practice & Experience*, 26(6):635–652, June 1996. Johon Wiley & Sons, Inc.
4. Luabind. `http://luabind.sourceforge.net/`.
5. The Programming Language Lua. `http://www.lua.org/`.
6. VPython. `http://vpython.org/`.