ZJUNlict
Extended TDP for RoboCup 2014

Yue Zhao, Chuan Li, Yangsheng Ye,
Zeyu Ren, Lisen Jin 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 nine 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 new software systems. Also we will share our tips on some special problems.

1 Introduction

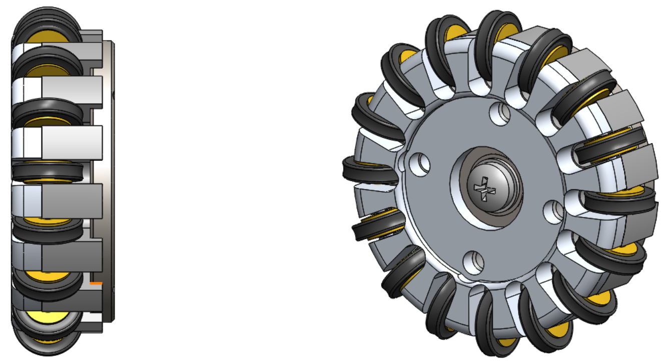
Our team is an open project supported by the National Lab. of Industrial Control Technology in Zhejiang University, China. We have started since 2003 and participated in RoboCup 2004-2012. The competition and communication in RoboCup games benefit us a lot. In 2007-2008 RoboCup, we were one of the top four teams in the league. We also won the first place in Robocup China Open in 2006-2008 and 2011. Last year, we won the first price in Netherland, 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

In order to prevent the ferry rubber band being bashed which happens in the game several times in the past years. We add the thickness of the big wheel bracket. At the same time we consider about reducing the weight. So we just add the thickness of the outer ring of the big wheel bracket. It is shown in Figure 1. We have also changed the length of the resistance arm in the chipping device. Exactly, we add the length of the resistance arm to make sure that the ball can be chipped higher.

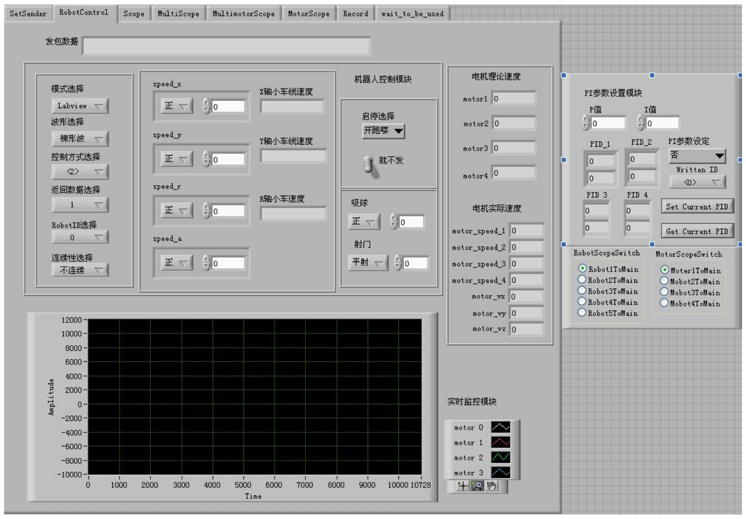


(a) (b)

Fig. 1. New wheel designed this year

2.2 Labview tools

We need several software tools to get the state of robot, which is inconvenient. So we use Labview to integrate all the functions, including speed-monitor, parameter-set, communication-test, etc. But the tool cannot run without Labview, so we packed the program and all the used plug-in in an installation file to make it available for any computer. Fig.2 is the GUI window of our Labview project.



模式选择: Labview
波形选择: 梯形波
控制方式选择: ⟨2⟩
返回数据选择: 1
RobotID选择: 0
连续性选择: 不连续
speed_x: 正 0
speed_y: 正 0
speed_r: 正 0
speed_a: 正 0
X轴小车线速度
Y轴小车线速度
R轴小车速度
启停选择: 开跑喽
就不发
吸球: 正 0
射门: 平射 0
motor1: 0
motor2: 0
motor3: 0
motor4: 0
motor_speed_1: 0
motor_speed_2: 0
motor_speed_3: 0
motor_speed_4: 0
motor_vx: 0
motor_vy: 0
motor_vz: 0
P值: 0
I值: 0
PID_1: 0 0
PID_2: 0 0
PID 3: 0 0
PID 4: 0 0
PI参数设定: 否
Written ID: ⟨0⟩
RobotScopeSwitch: [✓ Robot1ToMain] Robot2ToMain Robot3ToMain Robot4ToMain Robot5ToMain
MotorScopeSwitch: [✓ Moter1ToMain] Mobot2ToMain Mobot3ToMain Mobot4ToMain

Fig. 2. GUI window of the Labview project

2.3 Frequency-set mode design

Organizing Committee of the RobotCup 2013 required specified frequency. But our frequency-set mode can only provide 15 specified channels. Thus, we change the way we set the frequency. Firstly, the new mode was programed to set the start-point and the step. Then we use the e2prom to store the parameters. So the frequency-set expression was transformed from (1) to (2). In this way, all the frequency channels are available.

$$frq\_pre = \begin{cases} 2400 + dial\_num \times 4 & dial\_num \le 7 \\ 2400 + dial\_num \times 4 + 58 & dial\_num > 7 \end{cases} \tag{1}$$

$$frq\_now = \begin{cases} 2400 + startpo\mathrm{int} + dial\_num \times steplength & dial\_num \le 7 \\ 2400 + startpo\mathrm{int} + dial\_num \times steplength + 58 & dial\_num > 7 \end{cases} \tag{2}$$

where

- frq_pre is the position,
- frq_now is the velocity,
- $dial\_num$ is the start position.

3 AI System

3.1 Lua Script Architecture

We introduce a script language into our system to improve the flexibility and robustness of the system. Here we choose the script programming language Lua [1]. We have transplanted some repeated logic code to Lua such as positioning tactic, FSM configuration, Behavior Tree's generation, while left the complicated algorithms such as path planning, vision handling in C++ workspace. So the code is divided into two parts, as illustrated in Fig.3.

Tolua++ is an extended version of tolua, which is a tool to integrate C/C++ code with Lua. At Lua side, we need to access some variables and functions written in C++. Tolua++ helps us deal with this using a package file. For more details about tolua++, please refer to its Reference Manual[1].

The design advantages of script architecture with Lua are:

- **Clear Logic:** Like other scripting language, Lua is easy to understand. We can pay more attention to the logic of the code rather than the syntax, and it's really easy for different people to express their tactics by Lua scripts even if the script's author has little knowledge on programming.

[1] Tolua++ Reference Manual: http://www.codenix.com/ tolua/tolua++.html

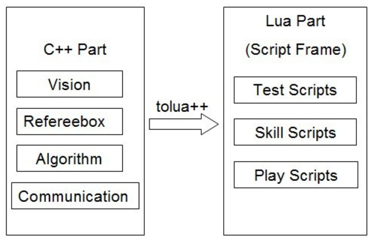


Fig. 3. Script architecture with Lua

- **No Compiling:** In RoboCup Small Size League competition, each team has only four chances for time out, the total time is 10 minutes. Therefore, it is very important to rebuild the code as quickly as possible in the limited time. Usually, a modification in C/C++ code takes about 10-20 seconds, but the compiling takes 1 minute or more. Lua helps us to solve the problem, we can just modify our strategy in about 10 seconds, and then do a syntax check by Lua's own debug tool, which takes almost no time. So we can spend more time on modifying logic code rather than compiling and debugging.
- **Online Debugging:** A play script will be loaded every cycle in our code. So tuning some parameters or functions such as a FSM's switch condition or Behavior Tree's node action do not need to stop the whole program, the effects will be shown as soon as the modifications in a script file are saved, which enables easier and faster strategies adjustment.

Fig.5 is an play script used for indirect kick in the frontcourt.

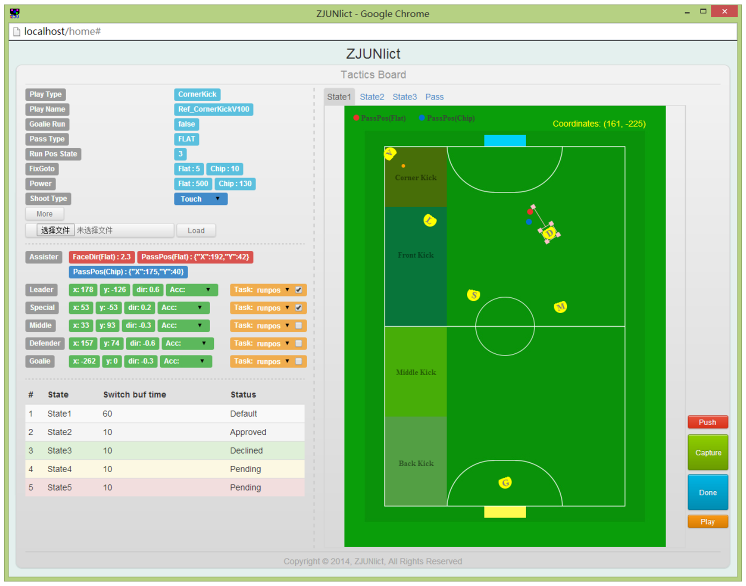
3.2 Tactics Board

Fig.4 is an application for planning the free kick and the indirect kick. With this application, we can just drag the robots on the board, and it will automatically generate code which can run the real robots instantly. So, we can generate any tactics in just few minutes.

This application is based on Web. It uses Node.js(Koajs) as its server, and uses Angular.js and HTML5(canvas) as its client. And it uses socket.io and udp packets to communicate with our C++ Core. In short, this application generate a Lua script, then send the script to the C++ Core to run the robots. Because of the efficient of graphical interface, we do not need to write much code for our strategy.

3.3 Style Adaptive Dynamic Movement Primitives

Trajectory learning and generation from demonstration has been widely discussed in recent years with promising progress been made. In some reaching goal tasks, different goals requires trajectories of different styles. How to reproduce a trajectory with a suitable style is an issue that must be resolved. In this year,


ZJUNlict - Google Chrome

localhost/home#

ZJUNlict

Tactics Board

Play Type	CornerKick
Play Name	Ref_CornerKickV100
Goalie Run	false
Pass Type	FLAT
Run Pos State	3
FixGoto	Flat : 5 Chip : 10
Power	Flat : 500 Chip : 130
Shoot Type	Touch

More

选择文件 未选择文件 Load

Assister: FaceDir(Flat) : 2.3 PassPos(Flat) : {"X":192,"Y":42} PassPos(Chip) : {"X":175,"Y":40}

Leader	x: 178	y: -126	dir: 0.6	Acc:	Task: runpos ✓
Special	x: 53	y: -53	dir: 0.2	Acc:	Task: runpos ✓
Middle	x: 33	y: 93	dir: -0.3	Acc:	Task: runpos ☐
Defender	x: 157	y: 74	dir: -0.6	Acc:	Task: runpos ☐
Goalie	x: -262	y: 0	dir: -0.3	Acc:	Task: runpos ☐

#	State	Switch buf time	Status
1	State1	60	Default
2	State2	10	Approved
3	State3	10	Declined
4	State4	10	Pending
5	State5	10	Pending

Coordinates: (161, -225)

Copyright © 2014, ZJUNlict, All Rights Reserved

Fig. 4. Script architecture with Lua

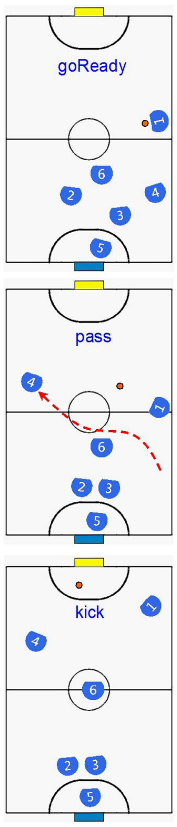
we propose a style-adaptive trajectory generation approach based on DMPs [2], by which the style of the reproduced trajectories can change smoothly as the new goal changes.

System Architecture The basic DMP equations are detailed in **??**Ijspeert2002, and now we discuss how to change the trajectory style according to different goals, here we propose a new method named Style Adaptive Dynamic Movement Primitives (SADMPs). Fig.6 shows the architecture of the SADMPs. We first collect and analyze different motions demonstrated by human. The captured motions are modeled and clustered using PDM [3] in order to get the principal trajectory of each cluster, which is the average of the trajectories in a cluster [4]. Then we train these principal trajectories using the DMPs trainer separately to get their weight parameters. Finally, we use the adapter to combine the training results with the new goal to reproduce new motions. Here a new desired goal g_{new} acts not only on the transformation system, which is similar to the original DMPs, but also helps to generate a fused style of the reproduced motion.

Algorithm for Learning The learning of $\mathbf{w_{new}}$ in SADMPs can be accomplished within two steps: (I) Obtaining the weight parameters $\mathbf{w}_k = [w_k^1 \cdots w_k^N]^T$ for the kth principal trajectory. (II) Employing an adaptive goal-to-style mechanism to merge different $\mathbf{w}_k$.

In the first step, different with the training method used by [2], LMS method is used to train the parameters $\mathbf{w_k}$ of every principal trajectory in this paper.

```
 1: firstState = “goReady”,
 2: name = “Ref_FrontKickV8”,
 3: attribute = “attack”,
 4: timeout = 200,
 5: [“goReady”] = {
 6:       match = “{A}{L}[SMD]”,
 7:       switch = function()
 8:             if reachTarget(“L”) then return “pass” end
 9:       end,
10:       A = getBall(dir1), L = goMultiPos(posList),
11:       S = rush(pos1), M = rush(pos2),
12:       D = rush(pos3), G = goalie() },
13: [“pass”] = {
14:       match = “{AL}[SMD]”,
15:       switch = function()
16:             if kickBall(“A”) then return “kick” end
17:       end,
18:       A = chip(shootPos), L = receive(shootPos),
19:       S = middle(), M = leftBack(),
20:       D = rightBack(), G = goalie() },
21: [“kick”] = {
22:       match = “{ALSMD}”,
23:       switch = function()
24:             if kickBall(“L”) then return “finish” end
25:       end,
26:       A = goAssist(), L = shoot(),
27:       S = middle(), M = leftBack(),
28:       D = rightBack(), G = goalie() }
```
(a)



(b)

Fig. 5. An example of indirect free kick in frontcourt: (a)Lua script; (b)positions of robots in every state. In the script (a), the basic settings are from line 1 to line 4. Then we define the specific states in this play. For example, the state *goReady* comprises a role match item (line 6), switch condition (line 7-9) and execution (line 10-12).

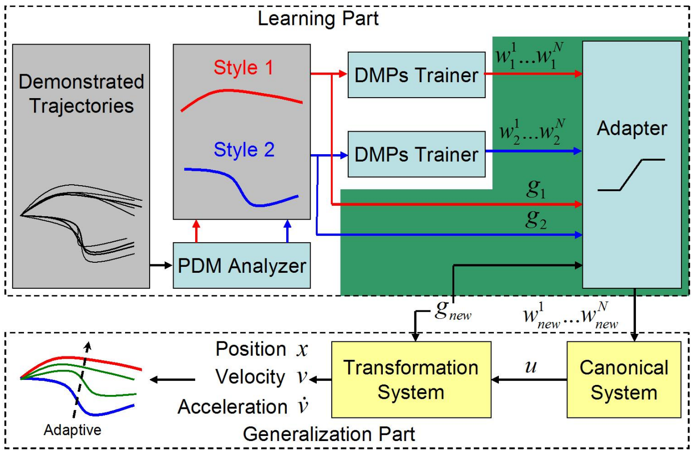


Fig. 6. The architecture of the proposed SADMPs model. This method comprises two parts, one is for learning, and the other is for generalization. The darkgreen-shaded components constitute an adaptive goal-to-style mechanism. Two points should be noted for the extension: 1) the architecture is for one-DoF, J copies of this architecture could be employed for an J-DoF application. 2) Although there are only two clusters obtained in this figure, the SADMPs can work for the situation existing multiple clusters.

The main reason for this design is that LMS method can obtain the weight parameters by using the same kernel functions for different principal trajectories in the same dimension, which is the basis of the goal-to-style mechanism in the next step. Here is the update rule:

$$w_k^i \leftarrow w_k^i + \alpha_r \sum_{t=0}^{T} \frac{(f_{target}(u(t)) - f(u(t)))\psi_i(u(t))u(t)}{\sum_{i=0}^{N} \psi_i(u(t))}, \tag{1}$$

In our study, we have more than one dimension for a principal trajectory, and the weight parameters of each dimension should be learned independently and in parallel. The different principal in one dimension trajectories share the common differential equations and kernel functions in one dimension. But in different dimensions, they could use different kernel functions.

In the next step, an Adapter is responsible for the adaptive goal-to-style mechanism. In the original DMPs formulations, the goal position g and the temporal scaling factor τ determines the style of the trajectory. In SADMPs, we further coupled g to the weight parameters, thus the style of a new reproduced movement changes smoothly between movement primitives in different styles.

The goal of kth principal trajectory in one-DoF is g_k. We sort the one-DoF goals of all the principal trajectories in an ascending order, $g_1 < g_2 < \cdots < g_M$, M is number of the principal trajectories. Note that the goals g may be ranked in a different location for different DoF. If $g_k \leq g_{new} \leq g_{k+1}$, then w_{new} can be represented as

$$w_{new}^i = \begin{cases} \dfrac{d(g_k)w_{k+1}^i + d(g_{k+1})w_k^i}{d(g_k) + d(g_{k+1})} & g_k \le g_{new} \le g_{k+1}, \\ w_1^i & g_{new} < g_1, \\ w_M^i & g_{new} > g_M. \end{cases} \tag{2}$$

where $d(g_k) = |g_{new} - g_k|$. It is clear that the $\mathbf{w}_{new}$ is determined by the distance between the new goal and the goals of principal trajectories. In another word, the nearer goal constitutes a high proportion of $\mathbf{w}_{new}$.

Fig.7 is the result for a shooting ball task.

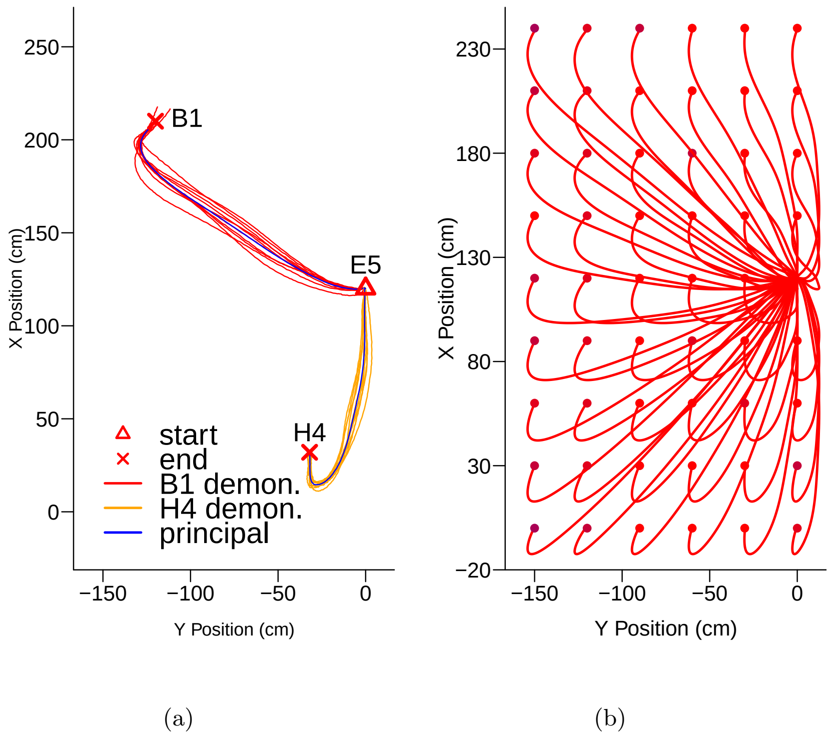


Fig. 7. Result on shooting ball task. (a) Two goals B1 and H4 is demonstrated for a shooting ball task; (b) The generated trajectories for other goals.

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. R. Ierusalimschy. Programming in Lua. Lua.org, 2nd edition (2006)
2. Ijspeert AJ, Nakanishi J, and Schaal S (2002) Learning attractor landscapes for learning motor primitives. In: Advances in Neural Information Processing Systems, vol. 15, 2002, pp. 1523-1530.
3. Roduit P, Martinoli A, and Jacot J (2007) A quantitative method for comparing trajectories of mobile robots using point distribution models. In: Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., 2007, pp. 2441-2448.
4. Stulp F, Oztop E, Pastor P, Beetz M, and Schaal S (2009) Compact models of motor primitive variations for predictable reaching and obstacle avoidance. In: 9th IEEE-RAS Interational Conference on Humanoid Robots, 2009, pp. 589-595.
5. Pastor P, Hoffmann H, Asfour T, and Schaal S (2009) Learning and generalization of motor skills by learning from demonstration. In: Proc. IEEE Int. Conf. Robot. Autom., Kobe, Japan. 2009, pp. 763-769.