# ZJUNlict
# Team Description Paper for RoboCup 2014

Yue Zhao, Rong Xiong, Hangjun Tong, Chuan Li and Li Fang

National Laboratory of Industrial Control Technology
Zhejiang University, P.R.China
rxiong@iipc.zju.edu.cn
http://www.nlict.zju.edu.cn/ssl/WelcomePage.html

**Abstract.** The Small Size League is one of the most important events in RoboCup. The league is devoted to the advancing in mechanical design, artificial intelligence and multi-agent cooperation of mobile robots. ZJUNlict from Zhejiang University has participated in this league for ten years since 2004 and got the first place in RoboCup 2013. In this paper, we introduce the main ideas of the robots' hardware design of ZJUNlict, and emphasize the intelligent control system including the hierarchical architecture of strategy, the Lua script architecture, learning and selecting trajectories methods based on Dynamic Movement Primitives.

## 1 Introduction

Small Size League (SSL) is an important part of the RoboCup event. It is the fastest and most intense game in RoboCup's soccer competitions. The basic rules of SSL are based on the rules of a FIFA's soccer game, but each team consists of only six robots playing on field that is 6.05m long by 4.05m wide. There are two cameras mounted over the field to capture images, which are processed in real time by a shared vision system, SSL-Vision [1]. This vision system recognizes and locates the position and orientation of the robots and the position of the ball, then broadcasts the information package to each team via network. Of course, the objective of the game is to score more goals than the opponent. SSL emphasizes on the fast movement of robots in a complex, dynamic and competitive environment.

ZJUNlict from Zhejiang University has participated in this League for ten years since 2004. We won the regional competitions in RoboCup ChinaOpen 2006, 2007, 2008, 2011. In RoboCup world-wide competition, we run into the quarter finals in 2006, 2009, 2011, and got the semi finals in 2007 and 2008. Since 2012, we have made a great progress and received the second place in RoboCup2012, and won the championship in RoboCup2013.

The remainder of this paper is organized as follows. Section 2 briefly introduces the hardware design of ZJUNlict's robot. Section 3 introduces the intelligent control system including strategy selection and trajectory generation based on learning method. We present a play script writing in Lua and the results of the DMPs' execution in section 4, and section 5 concludes the paper.

## 2 Robot Design

### 2.1 Components of the Robot

The robot we developed is equipped with 4 omni-directional wheels driven by a 50 watt brushless Maxon motors. There are another three major mechanisms: a dribbling device, a shooting device and a chipping device. The robot and its mechanical components are shown in Fig.1.


**Fig. 1.** Mechanical design of the robot: (1)shooting device; (2)omni-directional wheel; (3)chipping device; (4)dribbling device

We employ the NiosII as the central processor module, which is a soft IP working in QuartusII and NiosII software programming environment. When powered on or reset, the robot initialize itself, then run into the main loop to execute the command sent from PC. The overview of our embedded software flowchart is show in Fig.2.

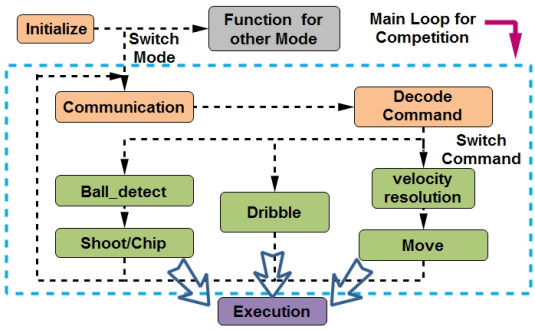



**Fig. 2.** Embedded software flowchart

## 3 Intelligent Control System

### 3.1 Hierarchical Architecture of Strategy

The software architecture of the intelligent control system is shown as Fig.3. 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 [2].

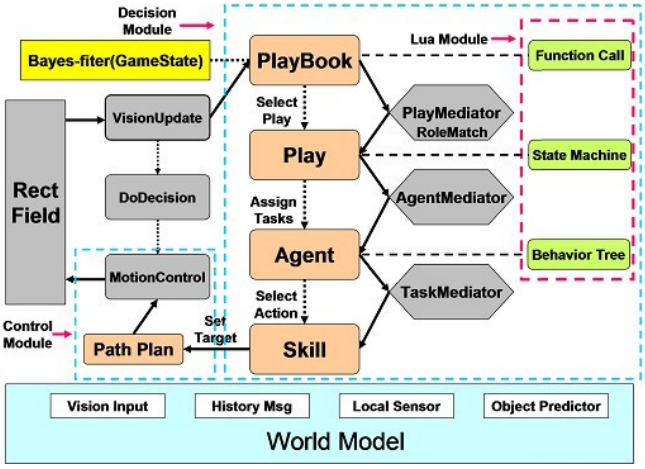



**Fig. 3.** Software architecture

The Decision Module is designed in a hierarchical structure, which is consist of the PlayBook module [3], 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 PlayBook, play-level and agent-level are all configured using the script programming language Lua, and will be detailed in Section 3.2.

The Control Module is responsible for the path planning and trajectory generation. It traditionally uses the Rapidly-exploring Random Trees (RRT's) algorithm [5] to find a feasible path and Bangbang-based algorithm [6] to solve two-boundary trajectory planning. We also adopt the idea of Dynamics Movement Primitives (DMPs) [9] to learn trajectories demonstrated by human, which will be shown in Section 3.3.

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 [10] 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 [7], 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. 4,

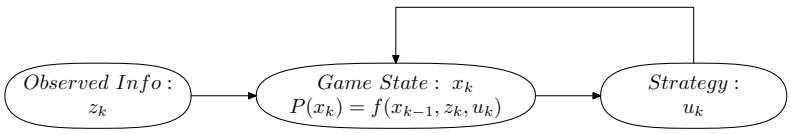



**Fig. 4.** Bayes-based evaluation method

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. 5 depicts the basic Bayesian filter algorithm in pseudo code.

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

**Fig. 5.** General algorithm for Bayesian filter

Fig. 6 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.

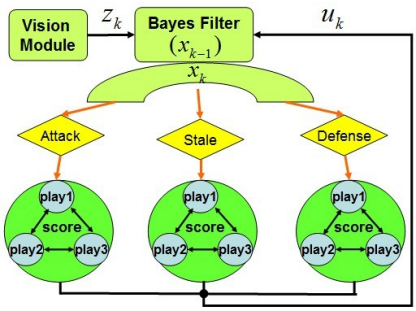



**Fig. 6.** Bayes Processing Flow Chart

The proposed method has two main features:

- **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 [3]. 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 [3]. We coupled a role match item in each state node to this basis. The new framework is shown as Fig.7, 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 [4], 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.

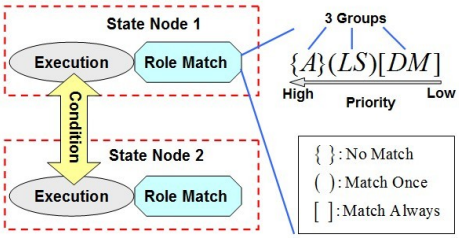
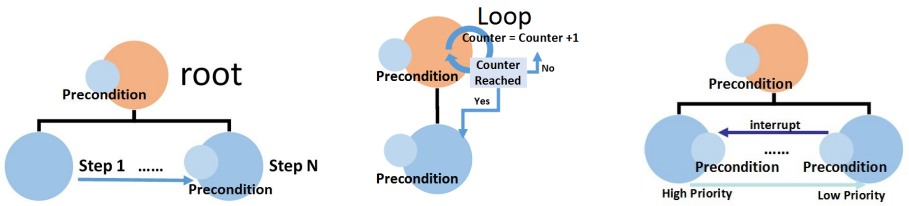



**Fig. 7.** FSM-based Role Match Framework. A role match item's syntax is described on the right: the letter in the string is short for the role's name in the state node(A-Assister, L-Leader, S-Special, D-Defender, M-Middle). We offer three modes for matching, *No Match* means to keep the role as same 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.8. 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.8(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.8(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.8(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. 8.** 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 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 [8]. 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.9.

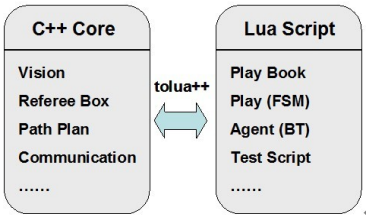



**Fig. 9.** Script architecture with Lua

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.
- **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.

### 3.3 Trajectory Generation with DMPs

We also introduce the Dynamic Movement Primitives (DMPs) framework [9] into our system to learn some special trajectories from human's demonstration. The DMPs presented below is just a basic model for learning a point-to-point trajectory without considering opponents and kinematic restrictions.

**DMPs Framework** Generally speaking, a movement can be described by the following set of differential equations [11], which can be interpreted as a point mass attached to a spring and perturbed by an external force which is applied artificially in demonstration from an intuitive point of view:

$$\tau \dot{v} = K(g - x) - Dv - K(g - x_0)u + Kf(u) \tag{1}$$

$$\tau \dot{x} = v \tag{2}$$

$$\tau \dot{u} = -\alpha u \tag{3}$$

where

- $x$ is the position for one DoF of the system,

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

– $v$ is the velocity for one DoF of the system,
– $x_0$ is the start position,
– $g$ is the goal position,
– $K$ is the spring constant,
– $D$ is the damping term,
– $u$ is the phase corresponding to time which going from 1 towards zero,
– $\tau$ is temporal scaling factor,
– $\alpha$ is a pre-defined constant,
– $f$ is a non-liner function which simulates as a extern force.

The equations (1) and (2) are referred as *transformation system*. And the differential (3) is called *canonical system*. Usually, $K$ and $D$ are chosen such that the transformation system are critically damped. Moreover, $\tau$ and $\alpha$ are chosen such that $u$ is close to zero as time. These equations are time invariant and make the duration of a movement alter simply by changing $\tau$. In addition, once the non-linear function $f$ is defined, the shape of the resulting trajectory will be determined as well. Specifically, $f$ can be defined as

$$f(u) = \frac{\sum_i^N w_i \psi_i(u) u}{\sum_i^N \psi_i(u)}, \tag{4}$$

$$\psi_i(u) = exp(-h_i(u - c_i)^2) \tag{5}$$

where $\psi_i$ are Gaussian basis functions with center $c_i$ and width $h_i$, $N$ is the total number of Gaussian basis functions. And $w_i$ are the weights need to be learned. All the equations written above are used for a one dimensional system. Our robot can be decoupled kinematically into three DoFs: $X$ translational direction, $Y$ translational direction, $W$ rotational direction, which are shown as Fig.10.

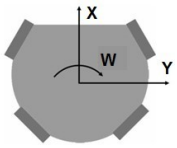



**Fig. 10.** Three DoFs of robot in SSL

**Learning and Executing Trajectory** An important advantage of DMPs is that a demonstration could be imitated by one-shot learning. In order to learn from the demonstrated trajectory, we should record the movement $x(t)$ first, then derivative $v(t)$ and $\dot{v}(t)$ from $x(t)$ in each time step, $t = 0, ..., T$. By combining the *transformation system* and *canonical system* together, $f_{target}(u)$ is obtained:

$$f_{target}(u) = \frac{\tau \dot{v} + Dv}{K} + (g - x_0)u - (g - x), \tag{6}$$

where $x_0 = x(0)$ and $g = x(T)$. We can find the weights $w_i$ in (4) by minimizing the error criterion $J = \sum(f_{target}(u) - f(u))^2$, which is a linear regression problem and can be solved efficiently [11]. Once $w_i$ is determined for a demonstrated movement, we can generate trajectory with a new goal position by resetting $g = g_{new}$, $x_0 = x_{current}$ and $t = 0$. After this setup procedure, we could adjust $\tau$ to determine the duration of the movement.

## 4 Experiment Results

### 4.1 Play Script Writing in Lua

Fig.11 is an play script used for indirect kick in the frontcourt.

```
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. 11.** 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).

### 4.2 Learning from Demonstration by DMPs

In the DMPs' experiment, we evaluated how well the DMPs' formulations generalize a quick turning and breaking through movement, as shown in Fig.12(a). First, we collected the movements in the three DoFs using the cameras mounted over the field. In the second step, we used DMPs method to train the data and got $w_i$ for each DoF. Finally, we reset the DMPs formulas and adapted to a slightly nearer target, see Fig.12(b). In this experiment, the robot successfully moved to new positions which are at most $0.7m$ away from the original target, and the duration of the movement is about 1.5 second. Note that in the practical 3-DoFs motor system, every DoF sets the transformation system independently, but they share the canonical system as they are coordinated, the results of the motion reproduction in different DoF is shown as Fig.12(c) and Fig.12(d).

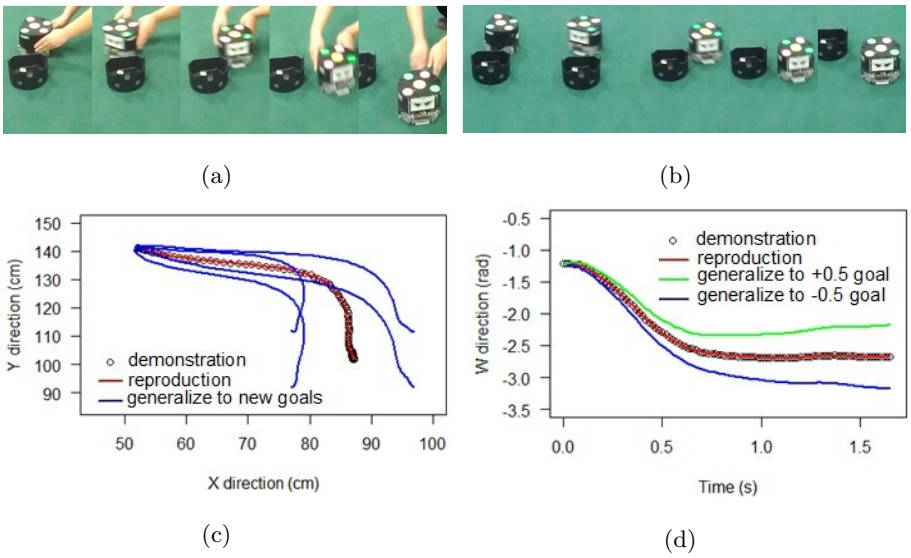



**Fig. 12.** Turning and breaking through example: (a)demonstration; (b)generalization to a slightly nearer target; (c)reproduction and generalization to new goals in X-Y panel; (d)reproduction and generalization to new goals in W rotational direction

In addition to the basic DMPs presented in section 3.3, we also try to add a dynamic potential field item to the DMPs' formula as in [12] for the purpose of obstacle avoidance. Moreover, we can change the target velocity of the movement while maintaining the overall duration and shape by developing several extensions and modifications to this approach as in [13].

## 5 Conclusion

In this year, we paid more attentions to the intelligent control system. First, Bayesian evaluation method and a role match framework based on STP [3] are proposed to make stable and flexible strategy decision. Second, we develop behavior tree for action connection. The FSM and BT both can be configured with Lua scripts. Finally, the DMPs are introduced to help generate human-like trajectory. These efforts make ZJUNlict score 43 goals and concede 2 goals in RoboCup2013. In next year, we will continue to focus on multi-agent cooperation and the trajectory generation model, also we hope to use 3D physics engine [14] to help simulate and predict common situations encountered in the game.

## References


1. S. Zickler, T. Laue, O. Birbach, M. Wongphati and M. Veloso: SSL-Vision: The Shared Vision System for the RoboCup Small Size League. RoboCup 2009: Robot Soccer World Cup XIII. pp. 425-436 (2010)
2. Yonghai Wu, Penghui Yin, Yue Zhao and Yichao Mao, Rong Xiong: ZJUNlict Team Description Paper for RoboCup2012. In: RoboCup 2012 (2012)
3. 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)
4. 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)
5. J. Bruce, M. Veloso: Real-time randomized path planning for robot navigation. In: Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on. IEEE, 2002, 3: 2383-2388 (2002)
6. L. Sonneborn, F. Van Vleck: The Bang-Bang Principle for Linear Control Systems. In: Journal of the Society for Industrial & Applied Mathematics, Series A: Control, 1964, 2(2): 151-159 (1964)
7. S. Thrun, W. Burgard, and D. Fox: Probabilistic robotics. Vol. 1. Cambridge: MIT press (2005)
8. R. Ierusalimschy. Programming in Lua. Lua.org, 2nd edition (2006)
9. A. J. Ijspeert, J. Nakanishi, and S. Schaal: Movement imitation with nonlinear dynamical systems in humanoid robots. In: International Conference on Robotics and Automation. Washington, DC: IEEE, pp. 1398-1403 (2002)
10. G. Welch, and G. Bishop: An introduction to the Kalman filter. (1995)
11. P. Pastor, H. Hoffmann, T. Asfour, and S. Schaal: Learning and generalization of motor skills by learning from demonstration. In: Proc. of the International Conference on Robotics and Automation (2009)
12. D. Park, H. Hoffmann, P. Pastor, and S. Schaal: Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields. In: IEEE-RAS International Conference on Humanoid Robotics (2008)
13. J. Kober, K. Mülling, O. Krömer, C. H. Lampert, B. Schölkopf, and J. Peters: Movement templates for learning of hitting and batting. In: IEEE International Conference on Robotics and Automation, 2010, pp. 1-6 (2010)
14. S. Zickler, M. Veloso: Playing Creative Soccer: Randomized Behavioral Kinodynamic Planning of Robot Tactics. RoboCup 2008: Robot Soccer World Cup XII. pp. 414-425 (2009)