

PolyBot and PolyKinetic™ System: A Modular Robotic Platform for Education

Alex Golovinsky, Mark Yim, Ying Zhang, Craig Eldershaw, Dave Duff
Palo Alto Research Center
3333 Coyote Hill, Palo Alto, CA, USA
alexg@mit.edu and {yim,yzhang,celdersh,dduff}@parc.com

Abstract—Modular robotics has been an active area of research for the last decade. In this paper, we argue that it also provides an excellent platform for education. PolyBot is a type of modular reconfigurable robot developed at PARC. The PolyKinetic™ System provides a robotic scripting language and programming environment for controlling this type of robot. Based on experience with mentoring high school students, and running a tutorial for robotic researchers at IROS'03, we show that, PolyBot and the PolyKinetic™ System is effective for educational activities at multiple levels. These include playing, exploring, building, programming, competing, and most of all, learning while having fun.

Index Terms— Education, PolyBot, PolyKinetic™, Modular robot, Scripting language.

I. INTRODUCTION

In the past, robotics was considered an advanced research topic and robotics courses mostly consisted of theoretical foundations on electromechanics, kinematics, dynamics and control [2]. Robotics in education is receiving more attention [18]. During the last decade, hands-on robotics courses [1], [5] have been introduced to classrooms as an introduction to the field. The emphasis of these courses is on *construction*, i.e., *building complete and working systems* with sensors, actuators and control. There has been an increasing number of nation-wide robotics competitions at all levels from elementary through graduate school. Most existing robotic platforms for education are either based on the LEGO® Mindstorms™ Robotic Invention System™ [7] or mobile robot systems. In this paper, we make the novel proposal that *modular* robots provide an ideal platform for such educational activities.

Modular reconfigurable robotic systems [6], [9], [10], [12], [13] are systems containing a small number of different module types relative to the total number of modules. Such systems promise a simplified design and construction of components, while allowing a broad range of functionality due to the large number of combinations. The Palo Alto Research Center (PARC) has created three generations of PolyBot [16], a type of modular reconfigurable robot. Nearly 200 modules have been built so far showing a variety of capabilities, including moving like a snake, lizard or centipede as well as humanoid walking and rolling in a loop. Modular reconfigurable robotic systems are similar to the Mindstorms™ in that they are built up from parts. The primary difference between this system from PolyBot, is that the Mindstorms system uses primarily one controller and up to three motors and sensors, whereas the PolyBot system

supports an arbitrary number of active modules, each of which has its own controller sensors and motor.

The PolyKinetic™ programming system [19] consists of a robot scripting language and programming environment. It provides a means for creating interesting and complex behaviors for robot systems with high degrees of freedom, such as modular robots, legged robots, snakes and humanoids. The system enables developing and programming this type of robot for both the novice and experienced robot programmer. It supports poseable programming, gait table composition and editing, for users with little computer knowledge. It also provides advanced mechanisms for programming automata, closed-loop feedback control, and teleoperation.

For the past three years, PARC has been hosting the Institute for Educational Advancement (IEA) [4] apprentices. These are advanced high school students who work at PARC for three weeks. During this short span, the students have been able to create an interesting variety of configurations from PolyBot modules and develop control systems for real applications, such as search and rescue. PARC also conducted a one-day hands-on tutorial for robotics researchers at IROS'03 [8]. Within a couple of hours, participants were able to build and program PolyBot systems with complex behaviors.

The remainder of the paper is organized as follows. Section II gives an introduction to PolyBot hardware platform. Section III provides an overview of the PolyKinetic™ software environment. Section IV discusses different levels of programming in the PolyKinetic™ system. Section V reviews our experiences with IEA and the IROS'03 tutorial.

II. POLYBOT HARDWARE PLATFORM

Three distinct generations and numerous variations of PolyBot have been designed. The first generation (G1) of PolyBot is the most prolific, and was used by the IEA students and in the IROS'03 tutorial. PolyBot consists of only one type of active module (a *segment*), with one degree of freedom and two connection plates. Figure 1 shows a segment and a connected chain of them. Keeping in theme with the cost requirements of the educational market, G1 is built from “hobby servos” and laser-cut plastic. Each module also has a PIC microprocessor and a variety of sensors (touch, accelerometers...). Computation takes place either on a host PC, with communication to the modules via a serial bus, or on the onboard PICs. With extra

passive pieces, PolyBot modules can be configured into various forms, such as centipedes or humanoids (Figure 2).

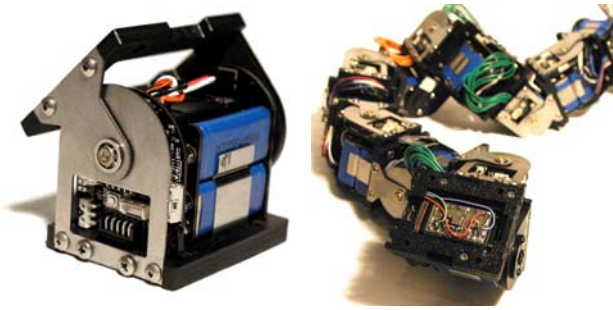


Fig. 1. A PolyBot segment and a connected chain

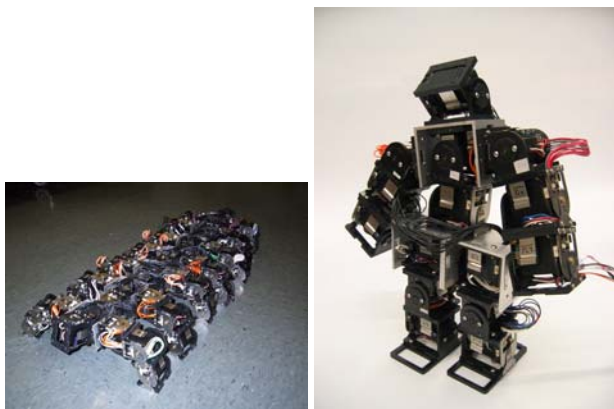


Fig. 2. A 55-module, 14-leg centipede and a 15-module humanoid

From a more abstract point of view, modular robotic systems follow the philosophy employed in many branches of computer science and engineering, in which complex and powerful systems are built from many simple components. When creating these robots, the students use their imagination and learn not only about mechanics of the modules, but also the electronics, communications systems, and programming approaches. We believe that it is important to have a simple yet expressive mechanism to construct and program such robots.

III. POLYKINETIC™ SOFTWARE ENVIRONMENT

The PolyKinetic™ software environment consists of an XML-based robot scripting language, a PolyBot simulator, and a programming environment. This environment enables the running and debugging of a PolyBot program from a host PC, which can simultaneously control both the hardware (via a serial bus) and the simulator. Programming high degree of freedom systems usually requires the simultaneous motion of many joint subgroups through a variety of trajectories.

A. Robot Scripting Language

Building upon prior work in the programming of scalable gaits [20], [21], PARC created the PolyKinetic™ Phase Au-

tomata Robot Scripting Language (PARSL) [19] — a markup scripting language defined in XML. PARSL allows the user to define robot configurations through module groups and their associated sensors and actuators. It also permits definition of gait control tables and automata, and applies these to the module groups. This shifts the focus away from low-level implementation to high-level gait specification. PARSL is powerful enough to be used by researchers, but friendly enough to serve as a teaching tool in classrooms.

Figure 3 is a simple PARSL program that shows the overall structure of a PARSL specification. A PARSL script consists of a robot definition $\langle robot \rangle$, where the structure of the robot may consist of one or more groups, each of which consists of a set of modules or groups. Named sensors and actuators, which are optional, can also be defined in this section. From a top down perspective, the section $\langle main \rangle$ consists of a sequential series of behaviors defined in the section $\langle behaviors \rangle$. Each behavior consists of a set of automata, each defined in the section $\langle automata \rangle$. The execution of a set of automata in a single behavior is “parallel”, i.e. they simultaneously run on different non-overlapping groups of modules. For example, the motion of one arm can run in parallel with a different motion on another arm or leg. In contrast, multiple behaviors are executed in succession. For example, a humanoid stands up and then walks, where “stand up” and “walk” are behaviors. Each automaton is defined by a sequence of states; the actions associated with a state is defined in the section $\langle states \rangle$.

The program in Figure 3 describes a robot with eight modules divided into two groups named “group1” and “group2”. The code consists of one behavior “simpleWave”, which applies the automaton “wave” to the “group1” modules, with a phase delay of 0.25 between each successive module. The total cycle period of the automaton is 1 second. The automaton “wave” consists of two states, “up” and “down”, where the “up” state causes the module to change its angle in a linearly interpolated fashion from its current angle to 30° in joint space. Likewise, the “down” state causes a move to -30° . This program, when controlling a snake-like chain of modules, produces a simple travelling wave gait.

The purpose of developing an XML-based scripting language rather than implementing control in a more standard programming language like C or Java is two-fold. Firstly, to separate specification from implementation, so that the specification has a clear and simple syntax and semantics that are directly related to the core of control. These specifications can be easily shared by researchers who develop complex gaits for modular robots. Secondly, specifications have multiple uses and can be implemented on different platforms. One may develop a graphical representation for PARSL, or implement a code generator for an embedded controller.

The program environment that runs PARSL is called the PolyKinetic™ Phase Automata Robot Programming Environment, (PARPE). PARPE is implemented in Java-1.4 with an XML parser. PARPE is designed to run and debug PARSL scripts, connected to the PolyBot simulator via RMI and/or the

```

<?xml version="1.0" ?>
<program>
  <robot>
    <structure>
      <group1 elements="0,2,4,6" />
      <group2 elements="0,2,4,6" />
    </structure>
  </robot>
  <main>
    <simpleWave />
  </main>
  <behaviors>
    <simpleWave>
      <wave group="group1" phase="0.25" period="1" />
    </simpleWave>
  </behaviors>
  <automata>
    <wave states="up, down" />
  </automata>
  <states>
    <up angles="30" />
    <down angles="-30" />
  </states>
</program>

```

Fig. 3. An “hello world” example of PARSL: a snake gait

hardware via a serial bus. The reader is referred [8] for detailed information.

IV. PROGRAMMING MODULAR ROBOTS

With all computer systems it is important to cater to users with different skill levels. This is especially important in an educational environment. The PolyKinetic™ programming system provides a flexible model for programming modular robots. It supports a range of paradigms, from posable programming to behavioral coordination. The next five sub-sections outline progressively more complex (and, as might be expected, more powerful) levels that this system can be used.

Level I: Pposable Programming

Posable programming is a very simple and effective way of programming modular robots with high degrees of freedom. It is done by manually moving the robot (back-driving the joints) into a certain pose and recording the joint angles that constitute the pose. The program is simply a recorded sequence of such poses. This recording may be saved to a file, and subsequently “played back” at different rates with the motions interpolated in joint space between each pose. In the PolyKinetic™ programming system, it is possible to record the motion of just a subset of the modules. Later, the programmer can combine several such recorded fragments using different periods or phase shifts for each. For example, one can record the movement of one leg, and then apply that one recording to both legs with a phase delay between them to produce a walking gait. Pposable programming is an effective technique for those with no programming experience and has been tested with children as young as 5 years old.

Level II: Gait Tables

Time	Module ID											
	1	2	3	4	5	6	7	8	9	10	11	12
1	15	15	15	15	-15	-15	-15	-15	15	15	15	15
2	-15	15	15	15	15	-15	-15	-15	-15	15	15	15
3	-15	-15	15	15	15	15	-15	-15	-15	-15	15	15
↓ 4	-15	-15	-15	15	15	15	15	-15	-15	-15	-15	15
5	-15	-15	-15	-15	15	15	15	15	-15	-15	-15	-15
6	15	-15	-15	-15	-15	15	15	15	15	-15	-15	-15
7	15	15	-15	-15	-15	-15	15	15	15	15	-15	-15
8	15	15	15	-15	-15	-15	-15	15	15	15	15	-15

Fig. 4. A gait table for a travelling wave snake-like locomotion

Gait tables are another effective way of programming modular robots [14], [15], [17]. Gait tables are two-dimensional tables that describe the state of each module (in this case, joint angles) at each successive point in time. Figure 4 is an example of such a table which produces a travelling wave gait for a 12-module chain. In PARSL, a gait table can be directly specified in the section of *automata*. A gait table, just as an automaton, can be applied to a specified subset of the modules. Like posable programming, little computational background is required to use gait tables.

Gait tables are an intuitive approach that works well for a small number of modules. However manual input can become difficult for complex motions involving larger numbers of modules. A logical substructure in a programming sense is not apparent in these gait tables and would ease the scaling of programming many modules with complex gaits. Phase Automata, described in the next level of programming complexity, addresses this problem.

Level III: Phase Automata

Phase Automata introduce a scalable way of implementing a variety of gaits. In certain motions, such as the travelling wave forward locomotion of a snake robot, each module undergoes the same periodic motion with only an offset between consecutive modules. Zhang et. al [20] abstracted this notion to create *Phase Automata*, which can be used to create gaits as varied as the walk of a centipede, or the forward motion of a loop. Figure 5 illustrates an automaton for snake locomotion, where two states are defined: one for moving to 15° and another for moving to -15° . The transition between states is time-driven — half of the total cycle time. The phase shift between neighboring modules is 0.25. Unlike gait tables, where the number of modules is fixed, this automaton can be applied unchanged to a group of modules of any size.

In PARSL, a phase automaton is specified in the section *automata* in the following format:

```

<$name states="$s_1, $s_2, ..., $s_n" times="$T_1[E_1], T_2[E_2], ..., T_n[E_n]" />

```

where $\$name$ is the name of the automaton and $\$s$'s are names of the states defined in the *states* section. The nominal period for a state s_i is T_i unless its optional exit condition E_i becomes

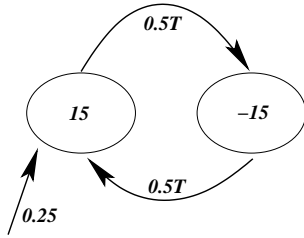


Fig. 5. A phase automaton for snake locomotion

true before T_i expires. The total nominal period for such an automaton is $\sum T_i$. The T_i 's are mathematical expressions evaluating to numeric values. E_i is a logical expression resulting in a Boolean. In the current implementation, the JEP [11] parser is used for parsing these expressions. When the optional `times` attribute is absent, the default duration for each state is 1 second. In the absence of an exit condition, the default expression is *false*, i.e., reducing to a purely time-driven transition. Every state referenced in `<automata>` has to be defined in the `<states>` section. For details can be found in [8].

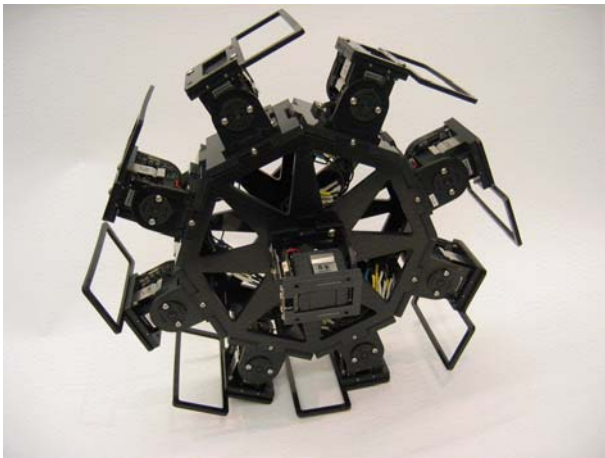


Fig. 6. A PolyBot cart wheel

Using phase automata, it is very easy to construct locomotion gaits. Figure 6 is an example of a PolyBot, consisting of a passive piece in the center and eight PolyBot modules, each with an attached passive “foot” for balance. Figure 7 shows the code that drives this wheel.

Level IV: Behavioral Composition

A behavior of the modular system consists of a set of automata, each of which is applied to one of the groups defined in the robot specification. For example consider the case of a robot that moves forward while executing some other action (such as grabbing an object or making a scan of the environment). The two tasks are likely to be independent. Defining this in one gait table would link the timing of the two tasks. An alternative would be to define two automata and run them in separate threads. A programmer would most likely rather not deal with

```
<?xml version="1.0" ?>
<program>
  <robot>
    <structure>
      <wheel elements="0,1,2,3,4,5,6,7" />
    </structure>
  </robot>
  <main>
    <rotate time="150" />
  </main>
  <behaviors>
    <rotate>
      <wave group="wheel" phase="0.25" period="2" />
    </rotate>
  </behaviors>
  <automata>
    <wave states="down, up" times=".25, 1" />
  </automata>
  <states>
    <down angles="-10" />
    <up angles="45" />
  </states>
</program>
```

Fig. 7. PARSL script for wheel locomotion

thread timing and synchronization, instead focus on constructing the gaits. In PARSL, running two tasks simultaneously is no more difficult than defining them independently. For example,

```
<walk>
  <wave group="!left" offset="0" phase="0.75" period="3" />
  <wave group="right" offset="0.5" phase="0.75" period="3" />
</walk>
```

defines a “walk” behavior for a centipede in Figure 2. It consists of two automata: one for the group of left legs and one for the group of right legs, where “!” indicates the mirror symmetry between the left and the right legs. The phase shift between left and right leg pairs is 0.5, and the phase shift between the legs of the same group is 0.75.

Multiple behaviors can be composed sequentially in the `<main>` section. For example,

```
<main>
  <standup time="10" />
  <walk />
</main>
```

defines that the centipede first stands up interpolated over 10 seconds and then walks forward.

Level V: Sensors and Teleoperation

To encourage good coding practice, a PARSL specification may optionally be written to use parameters given at run time. These parameters are specified in the `<parameters>` section and may be referenced simply by name in any expression in the XML code. When the program runs, the user can enter the desired values in a parameter GUI. These values remain fixed for the duration of the program. Such parameters are important when writing scalable code. PARSL also allows the user to specify a series of custom controls (e.g. speed or direction)

which the user can vary during execution. Similar to parameters, controls are specified in the `<controls>` section. While executing, the value currently specified by the user through the interface may be referred in an expression within the behavior code by using `$name`, where `name` is the name of the control input. The exact form and appearance of these parameters and controls vary with implementation. The current PC implementation uses standard Windows slider bars, radio buttons and text boxes. The controls and parameters are automatically generated when the program is loaded.

Control inputs may also be used in conditional behaviors. For example, to control the direction of a snake by teleoperation, one can modify `simpleWave` in Figure 3 to `dirWave`.

```

<dirWave>
  <if cond="$direction==forward">
    <wave group="group1" phase="0.25"/>
  </if>
  <if cond="$direction==backward">
    <wave group="group1" phase="0.75"/>
  </if>
</dirWave>

```

where `direction` is a control variable.

The use of sensors is similar to the use of control inputs. The current reading of a sensor that is specified in the robot's description can be referenced. Sensors, like control inputs and parameters, can be used anywhere within an expression.

A complete tutorial and reference manual for PARSL is available electronically [8]. The PolyKinetic software and documentation can be downloaded from <http://www.parc.com/downloads>.

V. EDUCATIONAL EXPERIENCES

We believe that the PolyBot/PolyKinetic™ system is a useful educational tool. It does not require much background; in particular, it does not require knowledge of any computer language. PARSL avoids the technical setup needed for a language that students may find daunting. PARSL only requires students to understand the concept of state machines which is useful in a variety of fields. Getting started in PARPE is simple, with simulation and hardware debugging facilities.

The PolyBot/PolyKinetic™ system has been taught in two environments so far: in an apprenticeship program run by the Institute for Educational Advancement (IEA) [4], a program for high school students interested in robotics, and also in a one day tutorial for researchers at IROS'03.

A. IEA High School Students

PARC has served as a host site for IEA for the last three years. In 2003, eight high school students from IEA aged 14–17, (see Figure 8). were working on the modular robotics project for three weeks. These students learned about robotics in general, and in the final week used PolyBot with the PolyKinetic™ system in a variety of exercises.

This early version of PARSL did not include any of the advanced features described in the Level V subsection, but it did

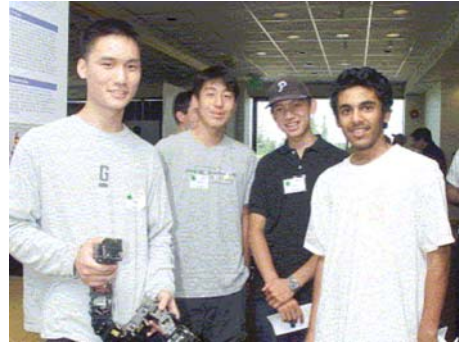


Fig. 8. IEA students in front of their poster at PARC

contain enough to allow the students to assemble and program the system in a variety of configurations. Their final task was to construct a robot for use in search and rescue. This involved teleoperated mobility through a highly constrained environment. The students developed a snake-like robot (Figure 9) that carried a small CCD camera and lights. Their programming included five different gaits which could be activated at arbitrary times depending on the demands of the environment.



Fig. 9. IEA student assembles a PolyBot snake

Overall, the robots performance through a simulated rubble pile was mediocre, however, the students expressed great satisfaction with the program in general and with PARSL in particular. Many of the students specifically expressed the preference of PARSL over using the pure Java interface they had also been taught.

B. IROS'03 Hands-on Tutorial

PARC gave a full-day, hands-on tutorial at IROS 2003 [3]. There were 12 participants, from Japan, Europe and the US, whose jobs ranged from managers to graduate students, from academia, government and industry. Most had no previous experience with any kind of modular robot.

Participants were initially split into six pairs. The first task for these groups was to create and program a robot to compete in a race, the winner of which was to the fastest forward

moving robot. Each group created a unique design, including fast waveform snakes, frog-like gaits, sidewinding snakes and a very original flipping/rolling gait. Even though there was only one winner, each was very creative and interesting — many of gaits were new even to the inventors of PolyBot. In creating these gaits, the students used phase automata, posable programming and a variety of passive pieces.

The final competition was conducted between two groups of six. Since the venue for IROS 2003 was in a casino in Las Vegas, the task was to create a robot that could gamble! This involved a very challenging set of tasks:

- walking up to a toy slot machine,
- placing a poker chip into it,
- depressing a lever,
- catching the winnings (ejected poker chips) in a cup.

The winner was the group whose robot collected the most chips. Participants were able to use the advanced features of PARSL, such as control user interfaces for teleoperation and parameters. One group completed a very complicated design (Figure 10) within only two hours. In the final demonstration they were able to do almost all the tasks, only barely missing the chips as they were ejected from the toy slot machine.

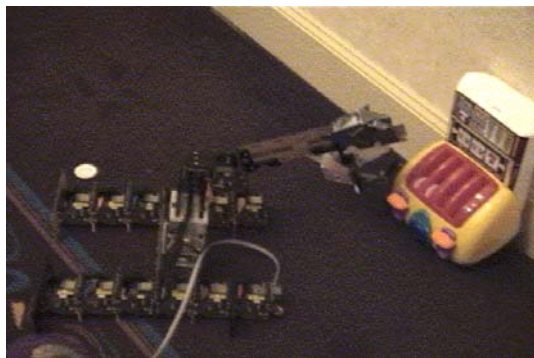


Fig. 10. Final competition: a gambling machine

VI. CONCLUSION AND FUTURE WORK

The PolyBot/Polykinetic™ System is an effective platform for robotics education. PolyBot modules are simple, robust and easy to assemble. The PolyKinetic™ Programming System allows users of diverse skill levels to develop control programs for modular robots.

In general, describing robot behavior using state machines has been found to be intuitive, even to novice users. The extensible nature of the XML syntax means that PARSL can be readily augmented to incorporate additional functionality.

Possible avenues of extension are to add a graphical programming interface, and to extend PARSL to self-reconfigurable systems. Another interesting avenue is to automatically generate code from a PARSL script, to be executed on the embedded controllers in the modules. Overall, PARSL requires little programming background, and hides

low-level implementation details, allowing students to focus on creating novel configurations and constructing gaits.

ACKNOWLEDGMENTS

The first author would like to thank PARC's UIP (Undergraduate Internship Program) for sponsoring him on this project. The authors would also like to thank Richard Burton for suggesting this topic.

LEGO is a registered trademark of the LEGO Company. MINDSTORMS and the Robotics Invention System are trademarks of the LEGO Company. PolyKinetic is a trademark of the Palo Alto Research Center.

REFERENCES

- [1] T. Braunl. *Embedded Robotics*. Springer, 2003.
- [2] J. Craig. *Introduction to Robotics: Mechanics and Control*. Addison Wesley, 1989.
- [3] PARC Modular Robots Group. Exploring and programming modular robots. http://www.parc.com/cms/get_article.php?id=200/.
- [4] IEA. Institute for educational advancement. <http://www.educationaladvancement.org/>.
- [5] J. Jones, A. Flynn, and B. Seiger. *Mobile Robots: Inspiration to Implementation*. A. K. Peters, 1998.
- [6] H. Kurokawa. Modular transformer. <http://staff.aist.go.jp/e.yoshida/test/top-e.htm>.
- [7] Mindstorm. Lego mindstorm home. <http://mindstorms.lego.com/>.
- [8] Palo Alto Research Center Modular Robotics Group. Phase automata robot scripting language. <http://www.parc.com/modrobots/polybot/parc/doc/tutorial/>.
- [9] D. Rus. Self-reconfiguring robots. <http://www.cs.dartmouth.edu/~rus/self-reconfig.html>.
- [10] W.M. Shen. Conro project. <http://www.isi.edu/conro/>.
- [11] Singular Systems. Jep - java mathematical expression parser. <http://www.singularsys.com/jep/>.
- [12] C. Unsal. I-cubes. <http://www-2.cs.cmu.edu/~unsal/research/ices/cubes/>.
- [13] M. Yim. Modular robotics. <http://www.parc.com/modrobots>.
- [14] M. Yim. New locomotion gaits. In *International Conference on Robotics and Automation*, San Diego, California, USA, May 1994. IEEE.
- [15] M. Yim. *Locomotion with a unit-modular reconfigurable robot*. PhD thesis, Stanford University, California, USA, 1995.
- [16] M. Yim, D. G. Duff, and K. D. Roufas. PolyBot: a modular reconfigurable robot. In *International Conference on Robotics and Automation*, San Francisco, California, USA, April 2000. IEEE.
- [17] M. Yim, Y. Zhang, and D. Duff. Modular robots. *IEEE Spectrum*, February 2002. Cover Story.
- [18] X. Yu and J. Weinberg. Robotics in education: New platforms and environments. *IEEE Robotics and Education Magazine*, 10(3), September 2003.
- [19] Y. Zhang, A. Golovinsky, M. Yim, and C. Eldershaw. An xml-based scripting language for chain-type modular robotic systems. In *IEEE Eighth Conference on Intelligent Autonomous Systems*, March 2004.
- [20] Y. Zhang, M. Yim, C. Eldershaw, D. Duff, and K. Roufas. Phase automata: A programming model of locomotion gaits for scalable chain-type modular robots. In *International Conference on Intelligent Robots and Systems*, Las Vegas, NV, USA, October 2003. IEEE/RSJ.
- [21] Y. Zhang, M. Yim, C. Eldershaw, D. Duff, and K. Roufas. Scalable and reconfigurable configuration and locomotion gaits for chain-type modular reconfigurable robots. In *International Symposium on Computational Intelligence in Robotics and Automation*, Kobe, Japan, July 2003. IEEE.