The Robot is the Program: Interacting with roBlocks

Eric Schweikardt
Computational Design Lab
Carnegie Mellon University
tza@cmu.edu

Mark D Gross
Computational Design Lab
Carnegie Mellon University
mdgross@cmu.edu

ABSTRACT

The roBlocks construction kit is a tangible concurrent programming environment that encapsulates sensory, kinetic, and computational behavior in modular building block units that snap together to construct robots. The choice of a protocol for propagating values through the constructed robot affects its behavior.

Author Keywords

construction, distributed, programming, robot, toy

ACM Classification Keywords

H.5.2 User Interfaces; K.3.1 Computer Uses in Education

INTRODUCTION

Two revolutions in computing are upon us: the advent of parallel and concurrent computing everywhere, and the physical (tangible) embodiment of computing. Together they have the potential to change the way we think about things, and in particular our ability to think about things computationally. Our roBlocks construction kit [1] is a physically embodied programming environment for building toy robots that employ distributed concurrency as a programming model. roBlocks has three important characteristics:

(1) It is a *tangible programming language* – users build programs by arranging physical blocks.

(2) It is *distributed and local*—computation is carried by a set of local processes rather than a single sequential process.

(3) It is a *language for programming robot constructions*, that is, how inputs from sensors control the behavior of a set of actuators. Figure 1 shows several actuator roBlocks.

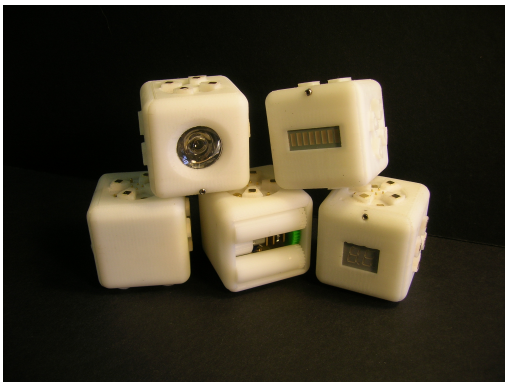

Figure 1. Several actuator roBlocks. Clockwise from top left: Flashlight, Bar Graph, Numeral, Tread, Tread.

It is easy to understand the basic idea of roBlocks by considering a simple light seeking robot made of two roBlocks: a light sensor block placed atop a tread block. The sensor measures the ambient light level and produces a number. The tread block gets that number from the light sensor block that sits on it, and runs its motor with a speed that corresponds to the magnitude of that number. To make the robot avoid light, take the two blocks apart and insert a red *Inverse* block between them. This operator block takes the number produced by the light sensor block, inverts it and transmits it to the tread block at the bottom. The new three-block robot moves away from a light source just as the previous robot moved toward it. This sort of modularity is possible because each of the blocks operates independently without knowing its place within the construction.

ROBLOCKS AS A PROGRAMMING LANGUAGE

While roBlocks do not immediately match traditional concepts of language (as textual or visual), we argue that in an important sense roBlocks are a vehicle for programming — they compute over sensor inputs and produce values (as actuator outputs). They can be configured in different ways to compute and produce different values from sensor inputs. And roBlocks is deterministic: any configuration of roBlocks will perform in predictable fashion.

Dataflow Models

We can describe a robot abstractly as a graph, with the nodes representing blocks and arcs representing adjacent communicating faces. Graph abstractions, like those shown in Figure 2, remove the physics of the robot, so we cannot tell from the graph how the robot will actually behave in the physical world. For example, if a robot has a right and a left motor block, the graph won't tell us which way the robot will actually move. But the graph helps us think about how information is transmitted from sensors, transformed by operators, and consumed by actuator blocks.

A difficulty with the directed-graph model of roBlocks, though, is the problem of cycles. It is possible to build a robot in which information flows from a sensor, into one or more configurations of blocks that are connected cyclically in the graph. So we could see feedback loops in which a small signal from a sensor is indefinitely amplified through a cycle.

An solution to this difficulty considers a robot as using a "diffusion model" to propagate values throughout the construction). We have considered various diffusion models and their effects on the robot behaviors.

Consider the simple two-sensor, two-actuator construction in figure 2. Sensor 0 on the left is producing a zero value, and sensor 1 on the right is producing 100. The question is, what value should each actuator compute?

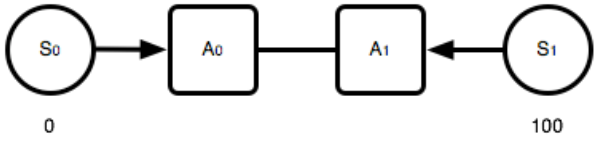

Figure 2. Simple two actuator, two sensor robot construction. What values for A0 and A1?

In a very simple diffusion model each block would just average the sensor values reaching it. Both A0 and A1 would compute a value of 50. We call this the *continuous average* method of computing the value.

In a more complex diffusion model, each sensor block broadcasts its data in a packet that contains the sensor scalar and a "hop count" that increases each time one block passes the packet to a neighbor. The further from a sensor, the larger the hop count on the data packet. We can use this to implement a diffusion effect in which the strength of a signal diminishes with the distance from its source.

One diffusion model for propagating values throughout the system has a fixed falloff. That is, we set an arbitrary threshold distance within which sensor blocks have influence. A light sensor in the middle of a construction, for example, might only be able to alter values in the set of blocks within four hops away, with blocks further away "ignoring" or not receiving the data. A variation is to use a diffusion model with a variable falloff. This is the model that we currently use in roBlocks, and we call it the *continuous gradient* algorithm. Continuous Gradient can be thought of as mixing. With only one sensor source, all blocks in the construction reflect the same value - the value in the data packet is not reduced as it propagates throughout the system.

In a construction with more than one sensor input, however, the values are mixed with blocks physically close to sensors showing values that are numerically close to the sensor's value and blocks between sensors reflecting an average. (Those familiar with analog circuits can think of this as a *voltage divider* model; it is similar to how voltage drops across resistors in series.) This global behavior is implemented locally by weighting each sensor input reaching a block by the inverse of its hop count. Unlike the continuous average model, data values don't degrade on their own, so the continuous gradient model lets users build robots of any size.

If the blocks in Figure 2 were programmed to calculate a *continuous gradient*, A0 would compute 33 and A1 would compute 67. The blocks divide each sensor scalar they receive by its hop count. A signal from a far away sensor has a weak effect on the actuator, whereas a signal from an adjacent block has a hop count of one, which strongly influences the actuator.

In earlier versions of the roBlocks system we have considered different propagation protocols and corresponding variations of the local block behaviors. Each of these, of course, requires reprogramming the code in the blocks. We plan to build a programming environment for more sophisticated programmers who wish to modify the behaviors that are built into the blocks. This would make our development cycle easier, but more importantly it would enable roBlocks users to take the next step toward programming distributed robotics algorithms.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant ITR-0326054.

REFERENCES

1. Schweikardt, E. and Gross, M.D. roBlocks: A Robotic Construction Kit for Mathematics and Science Education *International Conference on Multimodal Interaction*, Banff, Alberta, Canada, 2006.