Tic-Tac-LEGO: An Investigation into Coordinated Robotic Control

Ruben Vuittonet and Jeff Gray
Department of Computer and Information Sciences
University of Alabama at Birmingham
Birmingham, AL 35294 USA
{rvito, gray}@cis.uab.edu

ABSTRACT

The Lego Mindstorms Robot Command eXplorer (RCX) is a popular robotics kit that provides an immediate "out-of-the-box" opportunity to explore software controlled robot interaction. The limitations of the RCX provide a direct challenge that is typical of real-world embedded system development. This paper describes the Java-based development of a set of robots that coordinate to play the game of tic-tac-toe. Three key challenges were investigated in the project: 1) recognition of the state of the game board, 2) computation of the next-move within a reasonable timeframe using robots working in parallel, and 3) navigating a robot to the proper board location to mark the desired move. Game board analysis takes the form of a robot that performs optical scanning. A min-max tree algorithm was implemented in the primary control robot to determine the next best move. Various robot components were implemented to affect the physical movement of the robots and to mark the appropriate tic-tac-toe cell. The inefficiency of the min-max algorithm provides an opportunity to explore the use of parallelism among the robots to compute the next best move at specific levels of game play. In addition to the research results, the project informs the appropriateness of using the RCX as the basis for introductory programming classes and to provide a platform to drive undergraduate research.

Categories and Subject Descriptors

C.3 [Special Purpose and Application Based Systems]: real-time and embedded systems, K.3.2 [Computers and Education]: Computer and Information Science Education

Keywords

LEGO Mindstorms RCX, Embedded Systems, leJOS

1. INTRODUCTION

Embedded systems are "special-purpose computer systems encapsulated by the devices they control" [6]. Although there is disagreement concerning the general nature of embedded systems, two characteristics are generally applicable [1]:

- An embedded system is typically deployed on a different platform from which it was designed and implemented.
- An embedded system is typically resource-constrained and forced to execute in an asynchronous environment while optimizing performance and predictability concerns.

The first characteristic is driven by the design of the hardware platforms on which embedded systems are implemented. Often, these hardware platforms provide little or no user interface through which a programmer can develop the embedded software. The second characteristic is driven by the uses to which embedded systems are commonly deployed (e.g., the measurement of real-time data through sensory input, or the performance of predefined tasks in a specific time-frame). Other characteristics exhibited by embedded systems are resource limitations (in terms of both computational power and memory capacity), safety issues inherent in embedded systems' lack of information hiding, the inevitable compromise between size and performance, and loop structures for continuous sensory input [10].

Opportunities to investigate embedded systems within an educational context can be explored by adopting the RCX as the primary hardware platform. Because of its limited features, the RCX naturally embodies the first of the stated general requirements for embedded systems (i.e., software is developed on one platform and deployed on the RCX). The second general requirement is a matter of convenience (i.e., robots naturally lend themselves to real-time applications). The RCX offers a flexible architecture for exploring embedded systems, including challenges regarding real-time CPU performance, memory constraints, looping mechanisms, implementation exposure, and the limitations introduced from the use of an interpreter in an embedded system. The embedded systems created for the RCX take the form of robots that function autonomously, responding to their environment based on their programmed instructions.

This paper describes the results of an undergraduate Honors research project that investigated the feasibility of using the RCX as the host environment for coordinated robots that play tic-tac-toe. The project contained all of the challenges encountered during typical embedded systems development [7]. A contribution of the paper is a technique for parallel computation among robots to determine the next move using a game-tree algorithm. The paper describes the design and implementation of the tic-tac-toe player, as well as a discussion of the specific challenges and limitations encountered.

The organization of the paper is as follows: Section 2 provides a brief introduction to the RCX and leJOS, and also motivates the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ACM SE'06, March, 10-12, 2006, Melbourne, Florida, USA
Copyright 2006 1-59593-315-8/06/0004…$5.00.

goals of the research. Section 3 outlines the major challenges of the project, followed by a discussion of the implementation details in Section 4. The final section provides concluding remarks and discusses the limitations of the work.

2. BACKGROUND AND MOTIVATION

This section provides the necessary background information about the RCX, as well as a generalization of RCX software development using Java on a small virtual machine. The section also motivates the goals of the research.

2.1 RCX and leJOS

The RCX was first released in 1998 with two additional versions following. The core of the robot consists of the Hitachi H8/3292 CPU, which contains 16x8 general registers, 57 base instructions, 8 addressing modes, 16K Flash ROM, 32K RAM, 16MHz @ 5V, 2x8 bit timers, 1x16 bit timer, 8x10 bit A/D converters, 43 I/O lines, 8 input lines, and a serial port [2]. The default firmware for the RCX supports 5 program slots, each of which can manage 10 threads that support 8 subroutines, 32 global variables, and 16 local variables per thread. The firmware also provides 4 timers at 100ms precision, and 4 other timers at 10ms. The RCX contains 3 input ports (configurable for each sensor type, either active or passive), 3 output ports, 1 sound output, and 1 LCD unit. Although 8x10 bit A/D converters are available within the processor core, the RCX only takes advantage of 3 sensor ports, 3 power ports, and an infrared (IR) port. RCX is distributed with the LEGO Robotics Invention System programming environment, which is a graphical programming environment for beginning programmers. Of the 32K of RAM in the RCX, 28K is available for user applications [2].

RCX development is possible with several programming environments. The leJOS framework is a Java-based derivative of the TinyVM for use in the RCX [3]. leJOS implements a variety of language features that are not typically found on other RCX programming systems (e.g., object-oriented programming, preemptive threads, synchronization, exceptions, arrays, and recursion). The combination of preemptive threads, synchronization, and exceptions enables leJOS-based applications to receive input from the external environment in a stable manner, while providing the design and implementation benefits of an object oriented language [4]. Among the features incorporated into leJOS that are not available in the standard TinyVM are floating point operations, string constants, dynamic casting of integers, multi-program downloading, trigonometric functions, and various APIs for reading sensors and controlling motors. Although the TinyVM's footstamp is 10K, the leJOS footstamp is 17K, which is a by product of leJOS' attempt to be comprehensive in its coverage of the Java features that are appropriate for the RCX.

Various development tools are also provided by leJOS. An Eclipse plug-in for leJOS provides an effective integrated development environment (IDE) for developing RCX embedded systems based on the leJOS runtime [5]. The plug-in enables developers to create leJOS-specific projects that may be uploaded to the RCX directly from the IDE. If required, the leJOS firmware may also be uploaded to the RCX directly from the IDE. leJOS also provides an RCX emulator called *emu-lejos*, which is a textual tool that can receive more detailed exception information. Such information can assist in debugging a leJOS application before it is deployed to an actual robot [3].

2.2 Motivation

Embedded systems present special challenges in regard to the development of system software. As a platform for the development of embedded systems, the RCX provides opportunities for exploring these challenges. Of special interest is a specific set of embedded system issues that relate directly to the capabilities and limitations of embedded systems in general. These are 1) the development of embedded systems on one platform for deployment on another, 2) execution of real-time scheduling constraints, 3) systems that deal with sensors and activators, 4) the trade-offs between machine language, compiled and interpreted systems, 5) the use of loop structures for repetitious input, 6) information hiding concerns, and 7) the compromise between size and performance. Each of these general challenges finds specific expression when embedded systems are developed on the RCX.

The development of embedded systems on one platform for deployment on another requires special consideration in regard to system design and implementation. Embedded systems may lack the external user interface features necessary to develop software on the host system itself. Therefore, general purpose desktop computers are generally employed as development platforms [1]. When necessary, the source code is compiled on the general purpose platform. Afterward, the compiled code is uploaded to the embedded system through a special interface designed for that purpose. In the case of the RCX, there are several options provided for the development platforms that run on a variety of operating systems, including Windows, Macintosh, and Linux.

Real-time development demands that embedded systems assume specific characteristics. In terms of signal processing, real-time refers to the acquisition, transformation, and disposition of the incoming data in a manner efficient enough to ensure a sufficiently dense sampling of input data. What constitutes sufficient density differs among applications. For instance, a thermometer in a domestic setting for the control of a central air conditioning system has less stringent requirements than a heart monitor in a hospital's intensive care unit. In other scenarios, embedded systems must meet efficiency requirements in terms of sufficient output. For example, when a robotic manufacturing line is controlled by an embedded system, the embedded system must perform its assigned tasks with sufficient efficiency to ensure the manufacturing line does not become congested. For this research, the RCX is used to receive and analyze input from the external environment and react to the input received.

An embedded system is often deployed with accompanying sensors and activators, which are separate components (often embedded systems themselves) that provide information and perform actions for an embedded system [6]. Consider the thermostat for an air conditioning system: the embedded system is manifested in the form of a process that takes sensory input from a thermometer and controls (activates/deactivates) the air-conditioning unit. The RCX represents a similar architecture in the form of three sensory sources, three activator sources, and one IR port.

The computational and size limitations of embedded systems lead to special considerations in terms of programming language support. Although assembly language provides an opportunity for optimization and efficiency, it also introduces the developmental complexities associated with lower-level programming. Compiled languages provide an abstraction level between the developer and the hardware, facilitating development. However, an inefficiency shared with assembler language is the lack of debugging information. Interpreted languages provide more robust debugging capabilities, but at the cost of efficiency.

A common characteristic of embedded systems are loop structures that poll repetitious input [6]. Often, input is continuous and must be monitored (e.g., altimeters, heart monitors). For other scenarios, embedded systems must perform a task in a repetitious manner. The RCX can be used as the controller for robotic systems that continually await for sensory input and react to the input in ways defined by the specific application.

The resource constraints of embedded systems often limit the application of software engineering principles (e.g., information hiding) that can improve the stability and modularity of software systems [8]. Such limitations are encountered when an embedded system is developed in an object-oriented programming language that provides access to encapsulated data through interfaces. This problem is compounded when an interpreter is used (such as the leJOS TinyVM) because an interpreter requires additional resource overhead to perform a computational task.

Several of the issues outlined in this section deal with the most fundamental characteristic regarding embedded systems, which is the compromise between size and performance. In computational domains that do not require critical, real-time performance, it may be possible to increase the functional capabilities of the embedded system. However, many applications of embedded systems do not permit this flexibility because of the effect on the performance and schedule.

3. KEY CHALLENGES OF TIC-TAC-LEGO

This section outlines several of the challenges encountered when developing the tic-tac-toe player.

3.1 Cross Platform Development

One of the issues regarding embedded systems is the frequent necessity for developing the embedded systems software on one platform for deployment on another platform. In leJOS, both the interpreter and the user application program are uploaded from a USB Tower on a PC to an infrared port on the RCX (see Figure 1). On average, it takes 167 seconds to upload the leJOS firmware to the RCX. For our tic-tac-toe software, the bytecode (i.e., the .class file) for the Scanner robot was 13,572 bytes, which required 108 seconds (1.8 minutes at 125 bytes/second) to upload. The time to upload the bytecode to the RCX presents a challenge during development – each change that is made to the source code must be uploaded to the RCX to be tested, which impacts the time of the edit-compile-test cycle.

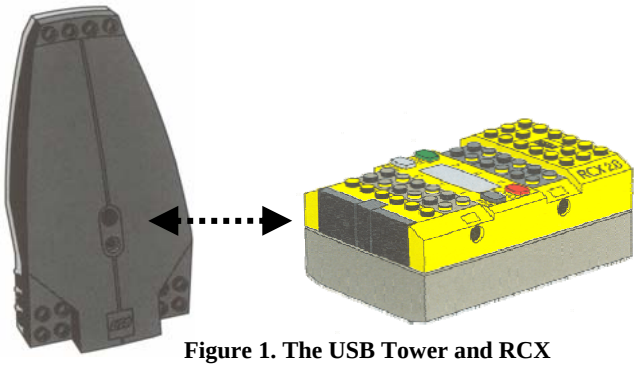

Figure 1. The USB Tower and RCX

3.2 Performance Considerations

For the Tic-Tac-LEGO project, two challenges related to performance issues emerged during development: 1) the computations for the next-best game move in a time-constrained, game-playing context, and 2) the physical movement of the robots to scan the game board and mark the result on the appropriate cell on the tic-tac-toe game board.

The first challenge manifested primarily as a result of selecting the min-max algorithm [9] as the basis for exploring parallelism among RCX units. This choice highlighted two of the identified concerns of embedded systems: 1) the compromise between size and performance and 2) issues regarding the choice of language implementation. With the leJOS interpreter installed, and running an implementation of the min-max algorithm, the tic-tac-toe player was able to evaluate only 109 tic-tac-toe boards per second. Given this average processing time, the run-times for the specified remaining moves are indicated in Table 1.

Table 1. Tic-Tac-LEGO Run Times

Moves Remaining	Run Time
4	0.5 seconds
5	2 seconds
6	11 seconds
7	58 seconds

As expected, the growth is exponential. At 7 remaining moves in a game, 58 seconds may not meet the real-time requirement for computing the next, best move in a time-constrained, game-playing context. The solution, as discussed in section 4.3, is to parallelize the next move computation among a set of collaborative robots.

The real-time concerns involved in scanning the board and placing the game pieces derived from 1) the disjoint nature of the sensors and activators and 2) the limited number of sensors and activators. For the Scanner robot, one "rotation sensor/motor activation" combination was required for rolling forward/backward and rotating the sensor arm (see Figure 2). This combination relied on the RCX's hardware interrupt system in conjunction with the virtual machine's messaging system to coordinate the starting and stopping of motors based on a count of axel rotations. Because the motors operated independently of the axel sensors, a slight discrepancy might be incurred after each movement. This issue was overcome through the use of gear reduction to suppress the discrepancy to sufficient levels.

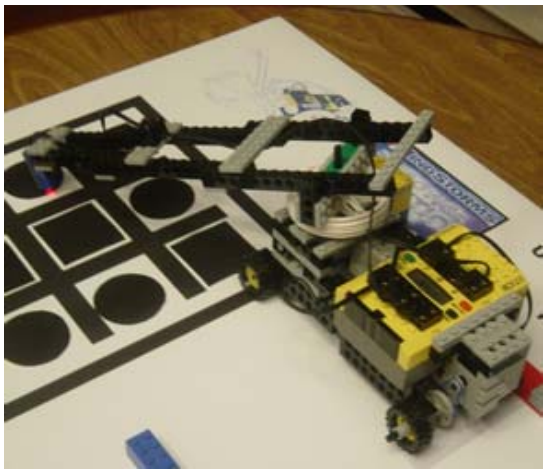

Figure 2. The Scanner Robot scanning the board

Another aspect through which the RCX reflects its nature as a platform for embedded systems is the limited number of sensors (three) and activators (three) available for use on the RCX unit. One sensor and one activator are required for each of the rolling and rotating behaviors. For the Scanner robot, this leaves one sensor for scanning. To actually affect the physical placement of pieces on a game board, another robot is required. For Tic-Tac-LEGO, a second robot performs the role of board-marker (see Figure 3). This robot is also used as a second processor when parallel computation is required.

4. TIC-TAC-LEGO IMPLEMENTATION

4.1 Class Decomposition

The use of Java and the leJOS TinyVM naturally led to a project implementation that decomposed into class hierarchies. The first hierarchy provided the core game playing and board scanning capability for one robot. The second hierarchy contained the parallel processing support and game piece placement for another robot. The design of each robot class was abstracted into a generalized class hierarchy. For example, both robots were required to roll forward/backward and rotate an arm, which suggested a generalized superclass called `RollRotateRobot`. One of the robots also required the ability to bend its arm to place game pieces on the board during game play. The class implemented to control the arm (`ArmRobot`) is a direct descendant of the `RollRotateRobot` class. Further, the processes of rolling, rotating, and bending are individually sensed and activated by an encapsulated class named `RotationAxel`. The `RollRotateRobot` is an aggregate class containing two (2) `RotationAxel` instances, one for rolling and another for arm rotation. These also exist in the `ArmRobot`, along with another for sensing and activating the bending of that robot arm. Other classes were also created for communication and location placement.

During development, the extensive decomposition of the project into an abstract class hierarchy, coupled with the implementation of interfaces to access class members, resulted in an implementation that was too resource intensive. Although an iterative and incremental approach to implementation was pursued, eventually a resource error occurred when the addition of a new class resulted in the exhaustion of resources.

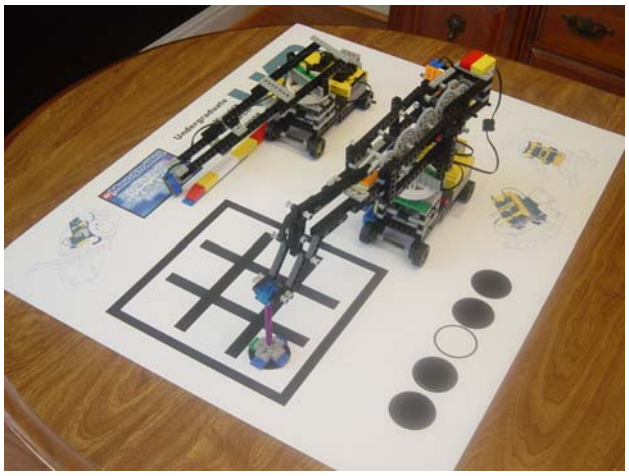

Figure 3. The Marker Robot placing a game piece

The solution to resource limitations was achieved by refactoring the source code. Specifically, a weakness of embedded systems programming known as data exposure was resolved. Embedded systems often require that data be pooled in a common area, or (due to space considerations) be publicly accessible to all parts of the embedded system. For the Tic-Tac-LEGO project, the conservation of memory was achieved through redefinition of all class members as public, and with the removal of all accessor/mutator interfaces for class members. Thereafter, the robots functioned within acceptable limits. Of course, this is a violation of traditional software engineering principles (e.g., information hiding), but was required to resolve the problems caused by resource limitations.

4.2 Event Polling

Polling occurs in both the Scanner robot and the Marker robot. The Scanner robot awaits input from the external environment and the Marker. The Marker robot only accepts sensor input from the Scanner robot. An important design constraint for the primary Scanner/Game-play robot requires that the robot be able to play consecutive games of tic-tac-toe; that is, on the completion of one game, the robot should be in a state to start another game immediately. To start a game, both robots are activated through their on-off button. At startup, each robot enters their respective polling loop. The Scanner robot is programmed to scan the board and determine a move when its *Prgm* (external input) button is pushed. Once it scans the board and determines the next move, it transmits the location of the move to be made to the Marker robot through the infrared port. After the Scanner robot informs the marker robot, it reenters its loop, awaiting another press of its *Prgm* button to begin scanning another move. The Marker robot, after receiving notification of a move, places the next piece at the correct location on the game board, and then reenters its polling loop.

4.3 Parallelism in Computing the Next Move

The min-max algorithm was specifically adopted to explore the possibilities of parallelism among coordinated RCX units. As explained in Section 3.2, time constraints emerge when seven or more moves remain in a tic-tac-toe game. A trivial analysis identified the hardcoded moves that should be made at the beginning of a game when nine or eight moves remained. Parallelism was implemented for the specific case when seven moves remain. First, the Marker robot implementation was extended to wait for one of two messages while in its polling loop. The first message provided information regarding the placement of a game piece on the board. The second message contained the board layout when a parallel computation was required. When appropriate, the Scanner robot transmits the board configuration to the Marker robot. At that time, both robots compute the next, best move in parallel. Because seven moves remain, one robot receives three board locations, and the other robot receives four board locations; therefore, the computation requires more time for one robot than the other. However, the RCX's hardware interrupt system (in conjunction with the TinyVM's messaging system) ensures that the message received from the Marker robot after it completes its computation does not interrupt the processing of the Scanner robot. After completing the move computation, the Marker robot reenters its polling loop, waiting for the message indicating which move is to be made. After the Scanner robot receives the computed move from the Marker robot, it selects one of the computed moves and informs the Marker robot of the move to be made. The implementation of parallelism for the computation of the next, best move when seven moves remains, results in a 43% reduction in the processing time required for that move, as compared to a single RCX unit.

5. CONCLUSION

The RCX provides an interesting and entertaining platform to explore the challenges of embedded system design and implementation. Through the resources provided by its three sensors, three activators, IR port, limited memory, and limited processor capabilities, the RCX provides the opportunity to explore the core issues involved with embedded system design (i.e., dual platforms for design and deployment; real-time processing; language implementation features and limitations; loop control structures for embedded systems; information hiding concerns; size versus performance issues; and the use of sensors and activators). The research described in this paper encountered the traditional challenges of embedded system design and investigated the use of parallelism among multiple RCX units.

The use of a general computer for the development and the LEGO IR USB tower for deployment is a constraining factor when developing embedded systems for the RCX. When used in educational settings, especially with less experienced programmers, the slow transmission rate during uploads dampens the potential of the learning experience. We encountered this during our mentorship of summer high school interns – the upload time and unreliable nature of the IR tower presented several frustrations among the students (for information on this internship opportunity, please see http://www.cis.uab.edu/heritage).

The requirement for real-time processing in conjunction with other implementation choices resulted in situations common to embedded systems development. The extensive class decomposition, coupled to the memory requirements of the interpreter resulted in the over-use of the RCX's resources. The subsequently refactored code resulted in a functional product. However, the resulting code was less adaptable and maintainable due to the lack of information hiding.

The inefficiency of the min-max algorithm provided the opportunity to explore the use of parallelism to compute the next best move. To split the task into parallel activities required the robots to communicate with each other during the computation of the next move. Such communication is possible through the infra-red ports.

The RCX has definite shortcomings (the speed of uploads and the lack of an informative and useful emulator). However, the RCX provides interesting opportunities to explore embedded systems and other aspects of computer science. Class decompositions for the robotics aspects of RCX development provide clear and understandable delineations whose straightforwardness might be useful in an educational environment.

REFERENCES

[1] Bessin, Geoff. "Embedded Systems: A Primer." Staff, IBM Rational. November, 2003. Available at http://www-128.ibm.com/developerworks/rational/library/806.html

[2] "RCX Brick." Robotics Outreach Group. Available at http://robofesta.open.ac.uk/techspec.html

[3] Ferrari, Giulio, et al. *Programming LEGO Mindstorms with Java*. Syngress Publishing, Incorporated, Rockland, MA, 2002.

[4] "Tiny VM." Available at http://tinyvm.sourceforge.net/

[5] "What is leJOS or TinyVM?" Available at http://rcxtools.sourceforge.net/e_lejos.html

[6] "Embedded Systems." Available at http://en.wikipedia.org/wiki/Embedded_system

[7] Lee, E., "What's Ahead for Embedded Software?" *IEEE Computer*, September 2000, pp. 18-26.

[8] Parnas, D., "On the Criteria To Be Used in Decomposing Systems into Modules," *Communications of the ACM*, Vol. 15, No. 12, December 1972, pp. 1053-1058.

[9] Shannon, C.E., "Programming a Computer for Playing Chess," *Philosophical Magazine* 41, 1950, pp. 256-275.

[10] Simon, D., *An Embedded Software Primer*, Addison-Wesley, 1999.