

Creating a Hands-On Robot Environment for Teaching Assembly Language Programming

James Wolfer¹ Haroun R. A. Rababaah²

Abstract --- *Real-world computer applications and assembly language programming are often taught with abstract exercises. To provide hands-on, real-world experience early, we introduce robots into our assembly language class. This paper describes the creation of a bridge from assembly language to a publicly available simulator, programming exercises, and a reconfigurable maze environment compatible with the simulator. This, with other open-source software, provides a complete development environment at no cost to the student.*

Keywords: *robot, assembly language, pedagogy, laboratory.*

INTRODUCTION

Since most contemporary computer applications are programmed in high-level languages such as Java or C/C++, assembly language programming is often included as a component in a broader class such as computer organization or, sometimes, operating systems. Since it is taught in the context of general computer operation, the time allotted to assembly language is limited. Furthermore, assembly language is typically taught on a desktop computer, making it an abstract exercise in programming.

To help provide a more meaningful experience, still within the context of a computer organization course, we reviewed the course with the following objectives in mind:

- Expand the experience of our students in a manner that enhances the student's insight, provides a hands-on, visual, environment for them to learn, and forms an integrated component for future classes.
- Remove some of the abstraction inherent in the assembly language class. Specifically, to help enhance the error detection environment.
- Provide a kinesthetic aspect to our pedagogy.
- Build student expertise early in their program that could lead to research projects and advanced classroom activities later in their program. Specifically, in this case, to build expertise to support later coursework in intelligent systems and robotics.

As one component in meeting these objectives we, in cooperation with the Computer Science department, the

Intelligent Systems Laboratory, and the University Center for Excellence in teaching designed a robotics laboratory to support the assembly language portion of the computer organization class. The balance of this document describes the laboratory, the trade-offs, and the resulting student programming environment.

BACKGROUND

Robots have long been recognized for their potential educational utility, with examples ranging from abstract, simulated, robots, such as Karel[1] and Turtle[2] for teaching programming and geometry respectively, to competitive events such as robotic soccer tournaments[3]. As the cost of robotics hardware has decreased their migration into the classroom has accelerated [4, 5].

As robots become a more common component of pedagogy care must be exercised to ensure their effective use. Fagin and Merkle[5], for example, report that robots actually had a negative effect on student performance when introduced into the laboratory. They attribute a significant portion of this result to the lack of an appropriate development environment, specifically the lack of a robot simulator to allow development outside of the laboratory.

ROBOT SELECTION

Hardware

There are many approaches to incorporating robots into a classroom, ranging from off-the-shelf robots to custom designs. Since our goal in this class is to teach assembly language programming, not robotics per se, we had specific criteria as follows:

- The device must be mechanically robust since we would have a limited budget.
- The robot must have sensors capable of supporting behavior-based algorithms, since we expect the students to write programs that interact with their environment.
- The robot must be small since we have limited laboratory space.

¹ James Wolfer, Associate Professor of Computer Sciences, Indiana University South Bend, Department of Computer and Information Sciences, 1700 Mishawaka Ave., South Bend, IN 46615 USA, jwolfer@iusb.edu

² Haroun R. A. Rababaah, Adjunct Lecturer in Computer Science, Indiana University South Bend, Department of Computer and Information Sciences, 1700 Mishawaka Ave., South Bend, IN 46615 USA, hrababaa@iusb.edu

- The robot must be programmable either in assembly language or controllable from assembly language programs running on an external computer.
- For our longer-term objective of leveraging student expertise in advanced classes, the robot must also have a high-level language applications programming interface.

There are a variety of robots currently available that meet some or all of these requirements. Possibilities ranged from building-blocks, such as LEGO Mindstorms, to custom-build devices, as well as a variety of hobby-level components.

Software

Another factor that must be considered is the programming environment itself. Since assembly language is unique to each processor, and the code embedded in each robot controller is custom, we needed to consider the development requirements and available tools for each device. We were also budget limited so we found it necessary to maximize our use of publicly available development tools.

Selection

Our choice, driven by the combined goals for this class and the future research objectives, as well as the software availability was to use off-the-shelf, Khepera II, robots from K-Team[7].

The K-Team Khepera II is a small, two-motor robot which uses differential wheel speed for steering. Figure 1 shows a functional diagram of the robot. In addition to the two motors it includes a series of eight infrared sensors, six along the “front” and two in the “back” of the robot. This robot also comes with a rich embedded system-call library, a variety of development tools, and the availability of several simulators.

SIMULATION

Since these are relatively expensive robots, we elected to create a supervised laboratory for student use. Unfortunately, a closed lab also limits the time the students can spend programming the robots directly. This limitation, in addition to being pedagogically unsound as described in [5] would have an exaggerated effect on our students. As a non-residential university the majority of our students work in the community and live some distance from campus, putting constraints on their ability to spend many hours in a scheduled laboratory session.

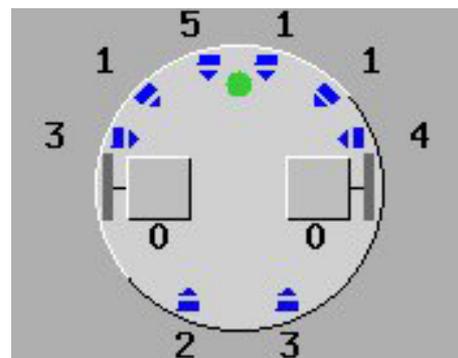


FIGURE 1
SIMULATED ROBOT DIAGRAM

Simulator Selection

To provide these students an infrastructure that would allow them to work from home we surveyed a variety of publicly available simulation tools: Khepera Simulator (SIM) [8], YAKS [9], Pyro [10], Breve [11], and others. Unfortunately, none of these tools work at the assembly language level. So, for example, while we could cross-assemble on a PC for the Motorola CPU on the robots, we could not directly run this code on the simulator. This would effectively force the student to have access to the physical robot throughout the development cycle.

Fortunately, the embedded code in the Khepera robots includes a relatively simple, but adequate, command level interface which communicates with the host via a standard serial interface. This allows students to write their programs using the host instruction set (Intel Pentium in this case), send commands, and receive responses such as sensor values, motor speed and relative wheel position from the robot.

Also, some of the software included the ability to simulate the serial interface protocol. Ultimately we elected to use the Khepera Simulator, SIM [8], as the basis for this class. The SIM Khepera simulator includes source code in C, and provides a workable subset of the native robot command language. It also has the ability to redirect input and output to the physical robot from the graphics display. Figure 2 shows the simulated Khepera robot in a maze environment.

Simulator Adaptation

To provide a seamless interface to the simulator and robots we modified the original simulator to more effectively communicate through a pair of Linux pipes, and we developed a small custom subroutine library callable from the student's assembly language programs.

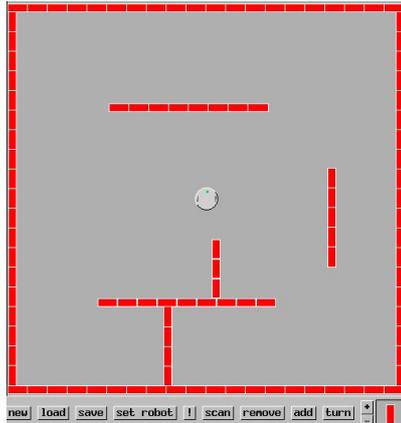


FIGURE 2
SIMULATED ROBOT IN MAZE

Specifically, we provide the following routines:

- `sndRcv_0` sends a command string composed by the student to the simulator/robot, receives a response string from the robot, parses the result, and places relevant values in successive locations of an integer array.
- `sndRcv_1` sends a command string composed by the student to the simulator/robot, receives a response string from the robot, partially parses the response into an array of strings, one string for each value.
- `sndRcv_2` sends a command string composed by the student to the simulator/robot, receives a response string from the robot, and makes the unparsed string available to the student program.

Note that each variation places more of the responsibility for decoding the response from the robot on the student giving them experience ranging from array traversal to number conversion in assembly language.

With this arrangement the student can send and/or receive information from the simulator, or the actual robot, using exactly the same program. This also allows the student to develop code on their own computer at home, test it with the simulator, freeing lab time for testing and refining their program on the actual robots.

ROBOT PHYSICAL ENVIRONMENT

Another component of the robotics laboratory is providing an effective area for the robots to run. In this case we developed a reconfigurable maze built from readily available materials. The base unit is an unmodified steel erasable marker board. The maze walls were fabricated from sheets of insulating foam painted white and cut to length and width. Attaching self-adhesive magnets to the bottoms of the insulating foam walls allows us to reconfigure the maze at will. Figure 3 shows the Khepera II robot in the resulting maze.



FIGURE 3
KHEPERA ROBOT IN MAZE

ASSIGNMENTS

Assignments for the class range from initial C assignments to call the robot routines to assembly assignments culminating in the robot traversing the maze. To provide an incrementally increasing difficulty level, and to encourage leveraging the knowledge these students have from their C/C++ coursework, a sampling of the assignments include:

- A C program to interactively send commands to the robot and to poll the sensor and print the results.
- An assembly language program to poll the sensors and print the results in a nicely formatted string. This illustrates calling the same routines as the previous assignment plus interfacing with the C dynamic libraries from assembly language.
- An assembly language program to move forward and stop when in proximity to an obstacle. Perform both under simulation and on the real robot, and report sensor differences required.
- An assembly language program to move parallel to a wall, stopping when an obstacle is in front of the robot.
- An assembly language program to traverse a simple maze.

Classroom discussion also includes topics such as motor control, timing issues, and sensor calibration.

PROGRAMMING SUITE

As previously stated, as an integral part of our objectives, we wanted each student to have access to a consistent development environment. While we made the simulators, including source code, available in the Computer Science Linux-based laboratories, we also wanted our students to have a viable environment at home. Many, if not most, of our students do not run Linux at home or work. Effectively, we needed to provide either a system running under their

operating system of choice, or an alternative operating system. We chose to provide our students a Linux based alternative by adapting and remastering the Knoppix Linux distribution.

Knoppix Linux is a distribution of Linux, a freely available operating system, that runs "live" from a CDROM. Once placed in the CDROM drive and the computer rebooted, Knoppix builds a "ram disk", and runs from that. Because Knoppix does not install itself onto, or write to, the hard disk it is a benign platform for supplying a Linux environment. Knoppix contains a complete development suite, including C, C++, Assembly, Perl, and Python.

Our custom distribution supplemented Knoppix with the modified simulator, the interface library (including source code), manuals, and assembler documentation. It was also mastered in a manner that allowed access to the documentation whether Knoppix was running, or the CDROM was being read from another operating system. Collectively, this provides a complete development platform.

CONCLUSIONS

Observing the physical action of robots can generate valuable feedback to students and demonstrate the real-world consequences of their programs -- robots hitting walls make students instantly aware of program errors. It also provides insight into the realities of physical machines such as motor control, sensor calibration, and noise.

To effectively develop code in a time and access limited laboratory algorithm development should occur under simulation followed by testing on physical machines. This allows students to work from home and reduces wear on robots. Here we have described a robot laboratory that addresses these needs, its integration into the curriculum, and a complete development environment that can be made available at no cost to the student.

REFERENCES

- [1] R.E. Pattic, *Karel the Robot: a gentle introduction to the art of programming*, 2nd edition. Wiley, 1994
- [2] H. Abelson and A. diSessa, *Turtle geometry: the computer as a medium for exploring mathematics*. MIT Press, 1996
- [3] M. Amirijoo, A. Tesanovic, and S Nadjm-Tehrani, "Raising motivation in real-time laboratories: the soccer scenario" in *SIGCSE Technical Symposium on Computer Sciences Education*, pp. 265-269, 2004.

[4] E.C. Epp, "Robot control and embedded systems on inexpensive linux platforms workshop," in *SIGCSE Technical Symposium on Computer Science Education*, p. 505, 2004

[5] B. Fagin and L. Merkle, "Measuring the effectiveness of robots in teaching computer science," in *SIGCSE Technical Symposium on Computer Science Education*, PP. 307-311, 2003.

[6] F. Klassner, "Enhancing lisp instruction with rxclisp and robotics," in *SIGCSE Technical Symposium on Computer Science Education*. Pp. 214-218, 2004

[7] <http://www.k-team.com>

[8] O. Michel, "Khepera Simulator package version 2.0: Freeware mobile robot simulator written at the university of nice Sophia-Antipolis by Olivier Michel. Downloadable from the world wide web. <http://diwww.epfl.ch/lami/team/michel/khep-sim>

[9] <http://r2d2.ida.his.se>

[10] <http://emergent.brynmawr.edu/pyro/?page=Pyro>

[11] <http://www.spiderland.org/breve>

[12] <http://www.knoppix.org>

[13] <http://www.knoppix.net>