

Using the AIBOs in a CS1 course*

John Chilton and Maria Gini

Department of Computer Science and Engineering, University of Minnesota
200 Union St SE, Minneapolis, MN 55455
{chilton,gini}@cs.umn.edu

Abstract

We describe an experiment that we have conducted in our CS1 course, where we used the robot dogs AIBO for a laboratory assignment. We briefly outline the course objectives, motivate the choice of using robotics and, in particular, of using the AIBOs in the course, and provide details on the software we developed for the lab assignment.

Introduction

Robotics is often used in undergraduate education to teach specific robotics and AI concepts (Dodds *et al.* 2006), but also as a way of engaging the students with hands-on experience, and of attracting students to study computer science (Gini, Pearce, & Sutherland 2006).

What we believe is unique about our experiment is that (1) we used as robots the Sony robot dogs AIBO, (2) we did the experiment in our CS1 course, (3) our CS1 course is based on Scheme, and (4) we used a combination of cooperation and competition to build teamwork while challenging the teams to compete. AIBOs are used by others for teaching, but they are used in advanced courses (for instance, (Veloso *et al.* 2006)) often designed for teams playing in RoboCup (RoboCupFederation), and typically not with Scheme. Scheme or Lisp are rarely used with robotics (but see, for instance, (Klassner 2006) for an exception).

In the rest of the paper we briefly outline the course objectives, motivate the choice of using robotics and, in particular, of using the AIBOs in the course, and provide details on the software we developed for the AIBO lab assignment.

Description of our CS1 course

Our CS1 course, the first course required for the Computer Science and Computer Engineering majors, is modeled after the MIT 6.001 entry-level computing course.

Our students are expected to take this course in their freshman year, but some take it in the sophomore year. The

course is offered with a single section per semester. The students are divided into smaller groups (30-34 students each) for a 2 hours/week lab, where they work on programming assignments with the help of the TAs. The course enrolls 100-150 students per term.

The course offers an introduction to the fundamental principles of programming and to different programming paradigms, with emphasis on the design of abstract data types and recursive algorithms. The course teaches how to think as a computer scientist, by teaching the process of decomposing problems into simpler problems, and of controlling the complexity by using abstractions that hide implementation details. Although students may not use in the course much of the material covered in calculus, they need solid reasoning skills and the willingness to use mathematics as a tool for analyzing and solving problems.

The course does not assume any programming knowledge. Students come to the class with very different backgrounds. Some have never programmed, some have significant programming experience, but they have rarely gone beyond the mechanical understanding of how to write a program to reach an understanding of how to organize their thinking process.

The language taught in the course, Scheme, is concise, has a clean and consistent semantics, and it is perfect to teach the students how to think. An additional reason for the choice of Scheme is that it is a language unknown to most of the students taking the course. This provides a more even starting point for all the students in the course. However, the students are often reluctant to put a significant effort into understanding the material, since they do not see how knowledge of Scheme will increase their short term marketability.

We want to maintain the rigor of the course and its contents, but find ways of engaging the students more in the learning process. Two years ago we were selected as one of the twelve teams at the University of Minnesota that participate in a campus-wide project on "Promoting Student Learning in Large Classes." As part of the project we have experimented a variety of active learning techniques, ranging from different small group activities in class, to placing a strong emphasis on problem-based learning,

In problem-based learning, relevant problems are introduced at the beginning of the instruction and are used to provide the context and motivation for learning. As described

*Work supported in part by a Bush grant on Innovative Teaching and Technology Strategies at the University of Minnesota and by the National Science Foundation grant DUE-0511304
Copyright © 2007, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

in (Prince 2004), “While no evidence proves that problem-based learning enhances academic achievement as measured by exams, there is evidence to suggest that PBL “works” for achieving other important learning outcomes. Studies suggest PBL develops more positive student attitudes, fosters a deeper approach to learning and helps students retain knowledge longer than traditional instruction.” (p. 229)

Why use the AIBO in a CS1 course?

Making the students excited about their first computing course by including creative contents and building bridges to other disciplines is recommended (Klawe Dec 1 2005) as a way to encourage students to consider majoring in CS.

The Sony AIBO comes with a rich set of sensors (camera, proximity, touch, microphones), a complex body (with 20 degrees of freedom), and preloaded software that allows it to interact with humans via voice and vision without the need for any programming. We believe they are specially suited for beginner courses because of their simple and intuitive use, but, at the same time, they will allow students to grow with them. As students become more knowledgeable, they will reprogram the AIBOs and make them do more interesting movements and interactions.

The AIBOs are not intimidating and are intuitive to use, yet they will allow us to show students how to use the programming language they learned in the course for a sophisticated and exciting application.

Currently, many college-level robotics courses require group robot building projects using Lego robots with Handy-board controllers or the like. Many students are not drawn to soldering, shrink wrapping or low level programming in C. To the contrary, the AIBO is designed and built to be used by a broad population. Everything is enclosed so that the robot need not be checked for loose wires or incorrect connections each time it is run. This will allow the students to go further since they won’t have to spend a lot of time on finding sensors that work and tracking down faulty wires.

A unique feature of the AIBO is its autonomy. AIBO can be controlled using wireless commands, but they can also be programmed to explore their environment and to make autonomous decisions. We believe this is an important feature, which will enable the students to appreciate the power of computing by understanding what it takes to make autonomous decisions, in the face of a complex and unknown real-world that can only be partially observed via sensors.

Recent studies with adults (Friedman, Kahn, & Hagman 2003) and with children (Melson *et al.* 2005) show that the majority of them views the AIBO as having mental states, life-like essence, and social communication skills. To the best of our knowledge, this is the first robot for which people have shown the type of psychological, cognitive, and emotional reactions they tend to have with pets (Beck & Katcher 1996). With the AIBO no one will get scared by the complexity but, at the same time, no one will get bored.

Educational Objectives

Felder (Felder, Felder, & Dietz 2002) reports a study of the effect of personality types on engineering student per-

formance. Although it is known that in order to be successful in an engineering career, a student should experience learning styles other than only the one they prefer, Felder states “severe mismatches commonly occur between the teaching style of instructors and the learning style of their students.” The study shows a positive correlation between students who do not learn well in a lecture-based environment and both women and first generation college students. Many engineering courses are lecture-based, so it is important to adopt different teaching styles and to make the students more active participants.

The project we describe addresses specifically some of the major issues that affect recruiting and retention of students in Computer Science:

1. **Increase confidence by hands-on programming experience.** Beyer *et al.* (Beyer *et al.* 2003) state that one of the reasons for poor self-image around computing is that women are less playful than men with computers. Robotics programming, by nature, is playful. Through the class experience we hope to give students experiences that “boost their self-confidence” (Beyer *et al.* 2003) and that improve their self-perceptions about computer science and career goals.

Furthermore, it is common wisdom that students gain confidence by hands-on manipulation, and by seeing concrete effects of their work. It is our experience that robot programming teaches and reinforces a variety of skills, and that most students enjoy using robots. Our goal is not to create robotics experts but to give students an opportunity to use their newly acquired programming skills in an environment that gives quick and concrete feedback, and which is also fun.

2. **Solve real world problems.**

We believe that focusing on solving real world problems, rather than just learning how to write programs, will increase student interest in the course. It has been shown that female retention in CS improves when the focus is not simply on the computer itself, but on the connections between computer science and other areas (Margolis & Fisher 2002). Robotics forces the students to deal with the real world. Robots do not always act as expected, and making the AIBOs to dance while keeping their balance proved to be harder than anticipated by the students. However, seeing the robot performance provided immediate feedback, which in turn helped the students in the process of debugging and improving their program.

3. **Work in groups.**

Work by Carol Gilligan (Gilligan 1982) suggested that women score differently on Piaget’s scale of moral development because they do not think in the same terms as men. In particular, Gilligan claimed that women are more oriented toward cooperation than competition. Recent studies indicate that “pair programming” (Williams & Kessler 2000) is an effective tool for increasing the retention of women in computer science classes (Werner *et al.* 2005).

In our CS1 course the students work in pairs for all their

weekly lab assignments. We developed a lab assignment centered on the use of the AIBO. Because of logistic for the AIBO lab we used larger groups, as explained later, and we made the lab assignment more open ended than any other lab assignment. We measured success simply by considering the participation in the process.

4. Combine collaboration with competition.

We know that many students respond positively to competition. Competition tends to provide students with motivation and focus, while cooperation has been shown to improve student learning outcomes. To combine the two, in the AIBO lab students cooperate within their group and compete between groups. The students were divided in large groups (6-7 students per group). Each group had to write a program to make the AIBO to dance and had to compete with the other groups for the best dance. The best dance from each of the lab sections was then presented to the entire class, and competed for having the best dance of the course.

We believe this approach has the potential to maximize the benefits of each kind of structure. In particular, we have seen how the competition invigorated the group assignment and prodded students who tended not to work well in groups to enjoy working collaboratively. This combined the best of cooperative learning and the motivational energy of academic competition.

Details on the lab assignment

The constraints on the kind of assignment utilizing the AIBOs that could be given were as follows. There were four lab sections, all on the same day, with roughly 30 students per lab section. This was many more students per lab than AIBOs available, so the assignment needed to be able to be completed by relatively large groups of students. Also, the labs were only two hours long and allowing 120 students to work on this project outside of the designated lab time was not practical. Hence, the assigned task needed to allow the students to experiment with the AIBO and the provided procedures for controlling it, write up a program, and test it in less than two hours.

The lab assignment that was devised to satisfy these criteria was to have the students program the AIBOs to dance. In each lab session, the students were divided into groups of roughly eight people and each group was given a little under two hours to make up a dance and program the AIBO to perform it. Each group was given one computer and one AIBO. The students were told the dance performances should be around 30 seconds long.

Given the time constraints the students had, it was decided their lab grade would be determined solely by attendance. To motivate the students, extra credit points were awarded to the top dance performance from each lab section. Shortly before each lab period was finished, the students were gathered together and each group demonstrated their dance for the other groups of section. As a group, each lab session voted for a lab section winner. During the large lecture session on the day after lab, an AIBO was brought to class, and

the best dances from each lab section were performed for the whole class.

Instead of the students programming their dances as a linear sequence of commands, they were instructed to make use of a specialized priority queue data structure, described in the course textbook (Abelson & Sussman 1996), called an agenda. An agenda is a table of times sorted in increasing order with associated events, where events are “thanks” stored in a queue and executed when their time comes. The textbook introduces agendas as an approach for discrete event simulation.

The students used a modified version of this data structure that would insert actual time delays between the discrete times. This allowed the students to control what events happened when easily. Making use of the agenda data structure this way, we were able to accomplish all of those things we hoped to with the AIBOs that is not accomplished by our more typical labs, while still reinforcing the course material like any other lab.

To create the dance movements the students were instructed to use the agenda either simply by inserting events with assigned time delays (the time delay is measured in milliseconds) or in a more complex way by adding actions that add other actions.

For instance, in this simple example the dog will move forward for a second, then stop and wait a second, and then look left and make a sniffing sound. This example also illustrates passing named and unnamed procedures to after-delay.

```
(after-delay 0
  (lambda () (walk .7)))
(after-delay 1000 stop)
(after-delay 2000
  (lambda () (look-left)
             (sniff)))
```

A better approach is to break the dance into different repeating patterns and to create actions that insert each other into the agenda. For instance, the AIBO can be made to nod its head up and down repeatedly by using two procedures. One procedure moves the dog’s head up and adds the other procedure to the agenda. The other procedure moves the dog’s head back down and adds the first procedure to the agenda again. Since this will create an infinite sequence, the agenda has to be reset at some point or a timeout to be added.

```
define (my-nod-down)
  (nod-down) ; nods dogs head down
  (after-delay 200 my-nod-up))

(define (my-nod-up)
  (nod-up) ; nods dogs head up
  (after-delay 200 my-nod-down))
```

```
; Put first call in the agenda, to get
; the process started
(after-delay 0 my-nod-up)
; Stop the loop after 20 seconds
(after-delay 20000 reset-the-agenda!)
```

Students were given a large variety of commands they could use to control the movements of the AIBOs, to control

the LEDs on the AIBO's face, to play music and produce different sounds, and to control individual joints.

Description of the software

During the lab, each AIBO ran the server-side implementation of a framework for remote communication with the AIBO we developed called A MODular Remote AIBO Interface (AMORAI). The computers that the students used for this lab assignment were Linux PCs equipped with 802.11b wireless cards to communicate with the AIBOs. The students did all of the programming for this lab in Scheme using the Guile Scheme interpreter. Issuing commands to the AIBO was done using a collection of procedures we wrote that implement the client-side interface of AMORAI.

The AMORAI server framework is implemented as a collection of Tekkotsu (Tira-Thompson 2004) behaviors, each which listens on or writes to various ports, similar to Tekkotsu's own Tekkotsu Mon behaviors. AMORAI, unlike Tekkotsu Mon, is not tied to the Tekkotsu remote Java GUI and so remote applications can cleanly be written against it in any desired programming environment and no extra software or libraries are required by the client. Also, AMORAI was designed from the ground up to maximize code reuse and hide issues such as interfacing the wireless card and socket programming from programmers of new server-side behavior interfaces.

Each Tekkotsu behavior that comprises the AMORAI server framework implements an interface for some specific functionality. Some of the behaviors implemented at this time include, among others, an interface for controlling the 3D motion of the AIBO, an interface that allows the programmer to control individual LEDs and joint positions, and an interface allowing the programmer to play any of AIBO's on board sound files. Communication in the other direction is also implemented. An example of this is a behavior which continuously sends the current joint positions and sensor readings to the remote PC. Communication of this kind was not required for this lab assignment, but it would be necessary if more complicated tasks were assigned.

Any interface behavior the programmer sets up to use is started when the AIBO boots up, and each of these creates a socket and sets it to listen on a given and distinct port. Programming a client side application to interface with AMORAI is then very simple. All the application needs to do is use the AIBO's IP address and the known port number of the desired interface to connect another socket to the one that is listening on the AIBO. We have implemented such a client interface for Guile scheme, but little effort would be required to develop analogous interfaces in Python, C, Ruby, or any other programming language that allows for reading and writing binary data to sockets.

Setting up and using AMORAI

To install the server-side AMORAI framework on a Sony ERS-7 AIBO, one can simply download a memory stick image from the AMORAI wiki (<http://www.cs.umn.edu/~chilton/amorai/>), copy this to an AIBO memory stick along with the appropriate wireless configuration file, and boot the AIBO.

Statement	Seldom	Sometimes	Often
1	5	30	62
2	3	31	63
3	1	17	79
4	2	15	80
5	6	44	47
6	5	41	51
7	4	37	56

Figure 1: The values indicate the number of students that selected that response to that statement. The statements were: (1) Contributed ideas, (2) Positively encouraged others in my group, (3) Compromised and cooperated, (4) Was flexible and willing to follow others, (5) Took initiative when needed, (6) Helped to solve problems, (7) Did my share of the workload/tasks.

After the AIBO produces a roar sound, it will be ready to communicate with. Extending the AMORAI server functionality requires a working Tekkotsu environment. More information about extending AMORAI can be found at the AMORAI wiki. AMORAI will not work as is with AIBO models older than the ERS-7.

The file containing the Guile Scheme procedures that implement the client side of the AMORAI interface and documentation can also be found at the AMORAI wiki. This file contains a variable that indicates the IP address of the AIBO to communicate with. This variable should be changed accordingly. If a wireless network is setup correctly, after changing this IP address, all that one should need to do to begin sending commands to an AIBO running the AMORAI server framework is to start the Guile interpreter and load the file.

Preliminary evaluation of the experiment

The experiment was a success. A questionnaire was given to the students directly after the activity. In response to the question, "Do you have any thoughts you would be willing to share about the lab? Was it fun?", a great majority (83 out of 97) of the students indicated they enjoyed the lab, while 12 students did not answer the question and only two students wrote that they disliked the lab.

These questionnaires were not anonymous. However, shortly after the activity the students were asked to fill out an anonymous course evaluation. In response to a similar question, 52 out of 56 students indicated they enjoyed the lab, two students did not respond, and again only two students disliked the lab.

The questionnaire given directly after the lab also included a series of multiple choice questions designed to encourage the students to reflect on how they functioned in their groups. Figure 1 summarizes these results.

The following question was also included in the questionnaire to assess how the lab changed the students' view of Scheme, "Does it change your view of the scheme programming language to know the client code used to communicate with the AIBOs was written 100% in Scheme?". Twenty-two respondents indicated they had more positive

view Scheme, 20 additional students had other positive comments such as “The lab showed off Scheme well.”, and only six students indicated that their views of Scheme were unchanged.

The large size of the groups that worked on this lab did not appear to be a problem. One reason for this could have been that students with different strengths could all contribute. As one would expect, the strongest programmers tended to be the ones in front of the computer most of the time. However, many students whom were not very engaged by the rest of the labs were able and excited to contribute their ideas. The noise and energy level of the lab was considerably higher than any other lab that semester.

It also became clear that given the task decomposed itself in to subtasks that smaller groups could work on easily. Without any instruction to do so, many groups formed subgroups that would work on different subtasks such as programming the LEDs, sounds, or the movement pattern of the AIBOs. These groups would divide their time on the AIBOs, and while one team was testing their code or experimenting, the other team would brainstorm, read through the provided documentation, and write out small pieces of code on paper or on their own laptops. Combining these separate pieces of code was easily done because of the nature of the agenda data structure that was used for program flow.

In addition to not requiring as much time to program or test as say traversing a maze, programming the AIBOs to dance was an inherently incremental task. Each group was encouraged to program a simple dance first and if time allowed add more to it. Doing this, every group was able to come up with at least a simple dance in the two hours given, and many groups came up with dances that were more sophisticated, creative, and entertaining than we could have expected.

Conclusions and Future Work

We have presented preliminary results on an experiment done in our CS1 course, where we used the AIBOs for one of the lab assignments. The experiment showed that the students were engaged by the robots and the novelty of the assignment, and that they enjoyed the opportunity to see a practical and fun use of the programming language Scheme they were learning in the course. We plan on repeating the experiment this year and to do a more systematic analysis of the results once we have the additional data.

The fact that the AIBOs are no longer commercially available, will make continuing this activity and replicating it elsewhere challenging. We are currently exploring options for using different robots and we are considering using a simulator instead of the real robots.

Acknowledgements

We would like to thank Paul Baepler and J. D. Walker from the Bush program for providing support and ideas and for helping us in experimenting with ways to increase student participation.

References

- Abelson, H., and Sussman, G. 1996. *Structure and Interpretation of Computer Programs - 2nd Edition*. The MIT Press.
- Beck, A., and Katcher, A. 1996. *Between pets and people*. Purdue University Press.
- Beyer, S.; Rynes, K.; Perrault, J.; Hay, K.; and Haller, S. 2003. Gender differences in computer science students. In *Proc. of the 36th SIGCSE Technical Symposium on Computer Science Education*, 49–53.
- Dodds, Z.; Greenwald, L.; Howard, A.; Tejada, S.; and Weinberg, J. 2006. Components, curriculum, and community: Robots and robotics in undergraduate ai education. *AI Magazine* 27(1):11–22.
- Felder, R.; Felder, G.; and Dietz, E. 2002. The Effects of personality type on engineering student performance and attitudes. *Journal of Engineering Education* 91(1):3–17.
- Friedman, B.; Kahn, P. H. J.; and Hagman, J. 2003. Hardware companions? what online AIBO discussion forums reveal about the human-robotic relationship. In *SIGCHI Conference on Human Factors in Computing Systems*, 273–280.
- Gilligan, C. 1982. *In a different voice: Psychological theory and women’s development*. Harvard University Press.
- Gini, M.; Pearce, J.; and Sutherland, K. 2006. Using the Sony AIBOs to increase diversity in undergraduate CS programs. In *Conf. on Intelligent Autonomous Systems (IAS)*, 1033–1040.
- Klassner, F. 2006. Launching into AI’s october sky with robotics and lisp. *AI Magazine* 27(1):51–66.
- Klawe, M. Dec 1, 2005. Blue skies ahead for it jobs. *CIO Magazine*.
- Margolis, J., and Fisher, A. 2002. *Unlocking the Clubhouse: Women in Computing*. MIT Press.
- Melson, G. F.; Kahn, P. H.; Beck, A. M.; Friedman, B.; Roberts, T.; and Garrett, E. 2005. Robots as dogs?: children’s interactions with the robotic dog AIBO and a live Australian shepherd. In *SIGCHI Conference on Human Factors in Computing Systems*, 1649–1652.
- Prince, M. 2004. Does active learning work? a review of the research. *Journal of Engineering Education* 223–231.
- RoboCupFederation. <http://www.robotcup.org/>.
- Tira-Thompson, E. 2004. Tekkotsu: A rapid development framework for robotics. Master’s thesis, Carnegie Mellon University.
- Veloso, M. M.; Rybski, P. E.; Lenser, S.; Chernova, S.; and Vail, D. 2006. CMRoboBits: Creating an intelligent AIBO robot. *AI Magazine* 27(1):67–82.
- Werner, L. L.; Hanks, B.; McDowell, C.; Bullock, H.; and Fernald, J. 2005. Want to increase retention of your female students? *Computing Research News* 17(2).
- Williams, L., and Kessler, R. R. 2000. All I really need to know about pair programming I learned in kindergarten. *Communications of the ACM*.