A New Set of Eyes and a New Pair of Legs: A Robust Learning Environment for Advanced High School Robotics

Jeremy Karnowski[1] and **David S. Touretzky**[2]

[1]Department of Cognitive Science, University of California, San Diego
9500 Gilman Drive, La Jolla, CA 92093-0515
jkarnows@cogsci.ucsd.edu
[2]Computer Science Department, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213-3891
dst@cs.cmu.edu

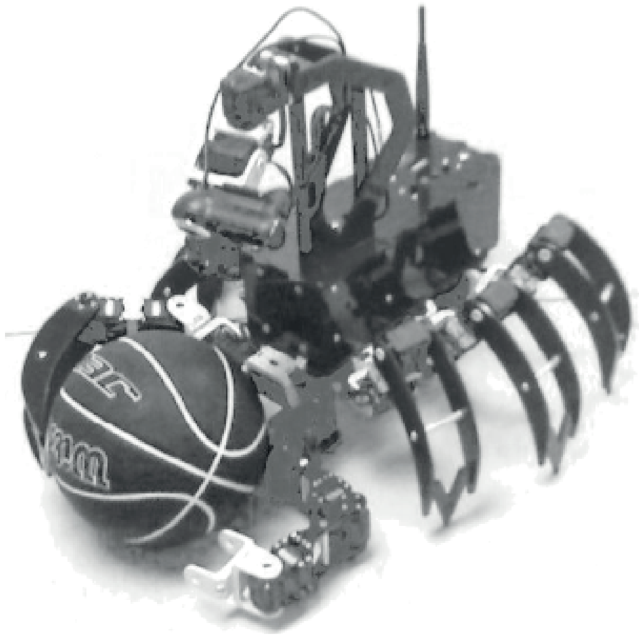
Abstract

Tekkotsu (see Tekkotsu.org) is an open source application development framework for intelligent mobile robots. Originally designed for undergraduate computer science majors, recent refinements to the framework have led us to explore its use with high school students. We developed a pilot course curriculum to introduce high level robotics to students with little or no programming experience in a way that provides improved feedback and error detection on multiple levels. The use of visualization tools and pair programming techniques scaffolds the learning process and provides a systematic way to introduce robotics as a fun and worthwhile endeavor to novices, and helps instructors efficiently address students' concerns in a real-time manner.

Platform

Tekkotsu was created as an open source software framework for developing intelligent mobile robot applications (Tira-Thompson and Touretzky, in press). It is available for free at Tekkotsu.org. The framework itself is based on C++, with GUI tools for teleoperation and monitoring written in Java for portability. Tekkotsu is primarily used for teaching computer science undergraduates, and for graduate level robotics research. But recent refinements to the framework suggest that it could be used to introduce high school students to a more sophisticated form of robot programming than is possible with the currently popular LEGO Mindstorms and VEX platforms, which suffer from impoverished sensors and limited processing power. Tekkotsu-based robots have color cameras and can see the world. Tekkotsu also provides primitives for locomotion, navigation, and speech generation. Using Tekkotsu, students can avoid low-level robot programming where individual motors must be turned on and off, and focus instead on problems of perception, navigation, and control.

Tekkotsu was originally implemented on the Sony AIBO robot dog, but has since been made platform-independent and supports a variety of robot designs, including the Chiara hexapod created at Carnegie Mellon (Figure 1; see also Chiara-Robot.org). The Chiara features a 1 GHz processor running Ubuntu Linux, 1 GB of RAM, an 80GB hard drive, WiFi and Ethernet connectivity, and a Logitech QuickCam Pro 9000 webcam. An updated version of the Chiara with a specialized gripper competed in the Small Scale Manipulation Challenge at AAAI-10 in Atlanta, playing chess against other robots. Six Chiaras were available for our course.


apod robot.

Students

The Summer Academy for Mathematics and Science (SAMS) is a rigorous summer enrichment program for students from underrepresented groups, designed to "encourage good students to become excellent students" (Carnegie Mellon University, 2010) and thereby improve their chances for admission to selective colleges. Students take essay writing and mathematics refresher courses, participate in several mock standardized national exams, and work on two engineering or science projects for three weeks each. Students who selected the Robotics II activity participated in a three week course (2 hours a day, 5 days a week) introducing them to the Chiara and Tekkotsu programming. We taught two sections of Robotics II during the summer of 2010. The first had 2 boys and 4 girls; the second had 4 boys and 2 girls. Only 4 of these students reported any prior programming experience.

Course Syllabus

We designed a curriculum to give students an introduction to robot control structure, vision, and locomotion. The course had two weeks of hands-on instruction followed by one week of project work.

Since two weeks was not enough time to teach students C++ programming, we focused on Tekkotsu's extensible state machine language, which uses a concise notation for defining node classes and creating instances of nodes and transitions. The notation is automatically translated into C++ code by a state machine compiler.

Each instructional day followed the same general pattern: students were introduced to a new concept, given a few lines of novel code in the state machine language, and instructed to solve a task that required altering the code in meaningful ways. In their solution attempts, they made use of several debugging methods taught to them: diagramming out their ideas with their partners (Figure 2), observing robot behavior to see if it matched expectations, drawing graphical representations of their actual state machine code using a tool called the Storyboard, and using the storyboard to create a graphical execution trace as their robot ran their program.

Week 1 Instruction

Monday: *Introduction, Teleoperation, Creating postures*
The first day of the course introduced students to the robots and the laboratory setup, reviewed safety procedures for preventing damage to the robots, and concluded with a crash course in Linux shell commands. The majority of the students had never worked with robots, let alone robots of such complexity. Many of them were apprehensive or intimidated, and at least two students considered dropping the course after first observing the robots move. One student actually screamed and had to leave the room to adjust to seeing what she had considered unnatural. After repeating basic power up/power down processes a few times students were shown how to teleoperate the robots on their own. They used a tool called the ControllerGUI, which allowed them to "see the world through the robot's eyes" by displaying its camera image, and also explore the Chiara's walking behavior. Students were then shown how to record their own robot postures in posture files; these files would later be used in constructing complex behaviors. After experiencing the robot's behavior on their own and under their own control, those students who were originally concerned about the unnaturalness of the robots began to reinterpret them as machines that only followed their commands. They were willing to continue with the course, and in fact went on to be very successful in their exercises and projects.

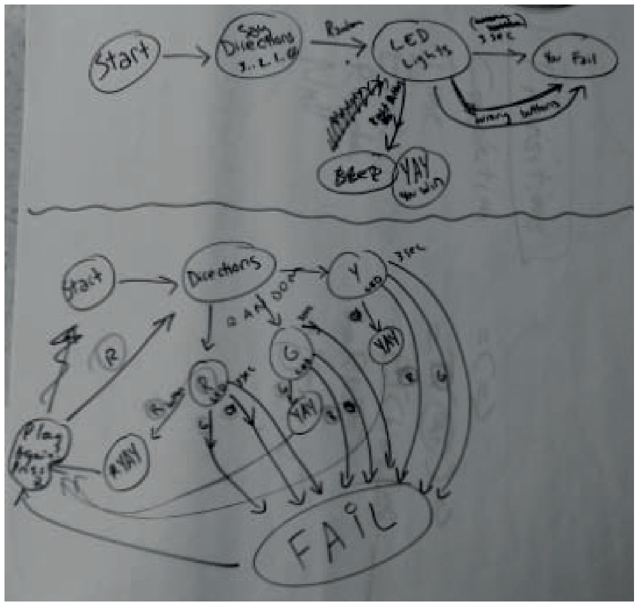


Figure 2 – Student diagram of the secret password game.

Tuesday: *State Machines, Basic Code, Run first program*
The second day began with an introduction to the concept of state machines. Students were asked to translate a daily task, such as brushing their teeth, into a logical flow that might be used in a computer program. They were then taught to express their ideas in the form of a state machine. Next they were introduced to some of the state node and transition classes Tekkotsu provides for the Chiara, and given the opportunity to run their first robot behavior.

Wednesday: *Additional state nodes and transitions*
The third day expanded on the state machine coding students had learned the previous day, adding new node classes for speech generation, playing sounds, lighting LEDs on the robot, and making head movements. Random transitions as well as transitions from button press events added new ways to alter the program flow. With these concepts, students created state machines that mimicked unlocking a safe by entering a secret password in the form of a specific sequence of button presses (Figure 2).

Thursday: *Storyboard, Loops, Parallel Processing*
The fourth day introduced the Storyboard tool as a way for students to visualize the structure of their state machine on the computer and generate a trace of its execution to see their code running in real time. To fully utilize the tool's capabilities, students were taught how to use fork and join operations, which allowed for concurrent state node actions. The project for the day was to create a metronome. Students used parallelism to have the robot light an LED and say 'tic' when its arm moved to the right, and light another LED and say 'toc' when the arm moved to the left.

Friday: *Mini Project – Robot Dance Competition*
The final day of the first week was a robot dance competition that required students to make use of all the concepts learned up to then. Students viewed a YouTube video of an international hexapod robot dance competition for inspiration, and were asked to similarly choreograph their dances. Each group successfully combined robot postures with lights, sound, and voice to create an elaborate dance routine set to a piece of popular music.

Week 2 Instruction

Monday: *Image segmentation, Simple visual search*
The sixth day of instruction introduced robot vision by showing students how Tekkotsu uses color image segmentation to simplify the visual world. Students were given code that used a Tekkotu MapBuilder node that instructed the robot to locate a green ellipse in its visual field. Students were then asked to create new nodes that looked for other shapes (lines, blobs) or ellipses of other colors. They then experimented and created nodes that could locate many of these colored shapes at once.

Tuesday: *Egocentric and Allocentric frames of reference*
The seventh day delved into the differences between Tekkotsu's camera, local, and world maps. Using various images in each, students were correctly able to deduce that the camera map gave a first person view, while the local map created an overhead view of the environment centered on the robot's body. Where objects appeared in the local map was a function of the robot's position but independent of its head direction. Students were then challenged to fool the robot's vision system as they currently understood it. After uncovering certain weaknesses, such as the problems posed by occluding objects, methods were given to correct for these errors.

Wednesday: *Basic locomotion, Environment interaction*
The eighth day focused on locomotion and the Pilot node class. Using Pilot nodes, students learned how to instruct the Chiara to move forward, backward, or turn to any angle. Students were encouraged to combine this with their previous knowledge to have their robots locate an object in the world, approach it, and knock it over.

Thursday: *Experimenting with virtual environments*
The ninth day gave students a way to view robot behavior without using physical hardware. Mirage is a virtual environment simulator for Tekkotsu that provides 3D rendering of the robot and its surroundings, along with realistic physics. Using Mirage, students gained an understanding of how to teleoperate a robot in a virtual world. The second half of the day was spent learning how to use the WorldBuilder tool to create new virtual worlds for the robot to explore.

Friday: *Project Planning and Proposal*
The tenth day was devoted to brainstorming in groups to design course projects for the final week.

Course Projects

Each group was required to complete a one week project that made use of three key features of higher level robotics. The requirements for the project were:

Vision: The robot must use camera images to do calculations with shapes in the environment and use that data for meaningful behavior.
Locomotion: The robot must move within its environment, using the data from its visual system.
Interaction: The robot must interact with its environment, either with objects, other robots, or with the roboticist.

One example project had a Chiara robot navigating to positions in the world that matched button press inputs. After a group member pressed one of the three colored buttons on the back of the Chiara, the robot would scan its immediate environment for a similarly colored ellipse on the floor, walk to that spot, and then turn around for the next round (Figure 3). One of the group members explained their architecture in this way: "It does this by using MapBuilders, which help the robot to see what color and what shape it's looking for, and the Pilot nodes will get the robot to actually walk to that color and shape."

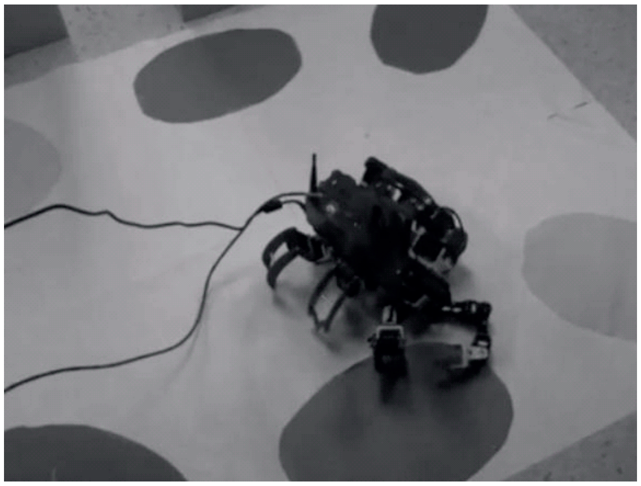


Figure 3 – Chiara robot navigating to a colored ellipse.

Another successful project featured two Chiara robots interacting, one of which wandered the environment and another that went looking for the first (Figure 4). The first robot wandered around by scanning its environment and walking to the nearest blue object. Upon reaching it, it would repeat the process and head towards a new blue object, as the previous one had become hidden underneath its body. The seeking robot would also scan the environment, but responded to a different set of colors. If the seeker saw a yellow patch in its environment, which was the color of the paper marker taped to the wandering robot, it would state "I found you," and head towards the hiding robot's position. If yellow was not seen, it would walk toward the nearest green object and begin its search again. While the locations of the green and blue objects were predetermined, the dynamics between the robots and the environment were never known prior to running the programs, and this led to rich, complex behaviors.

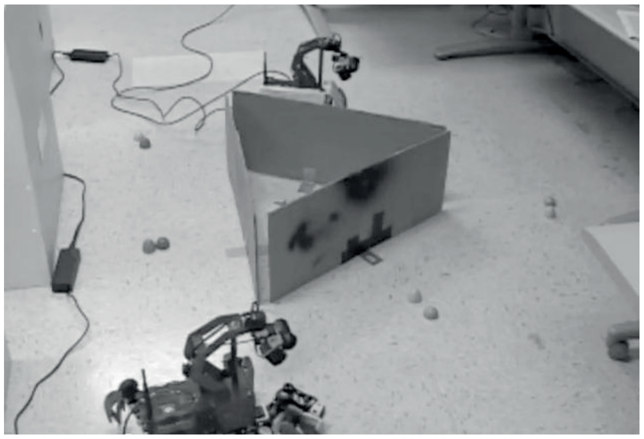


Figure 4 – The Chiara at the bottom of the photo is looking for the wandering Chiara, above.

Novices working with complex systems are less likely to comprehend how complex behaviors can result from simple rules (Jacobson 2000). While most students in our program immediately understood linear programs, these group projects clearly indicated that students, after only two weeks of instruction, could develop nonlinear projects that employed loops and parallel processing.

Forms of Feedback

Robot Behavior

Using robotics as an introduction to computer science has been a popular way to attract new students to the field. A major reason for its success is that it provides an integrated approach that exposes students to problems that involve action in the physical world, and that require critical thinking skills and oftentimes teamwork (Beer, Chiel, and Drushel 1999). The use of robotics allows students to see the implementation of their code in a way that is not usually apparent in a traditional programming assignment. By observing the robot's behavior, students can compare their predictions against actual outcomes to gain feedback about possible avenues of error correction. Students understood that incorrect behavior indicated poorly designed or improperly implemented code, and consequently returned to the drawing board. In establishing this loop between coding, observation, and debugging, students were able to gain more insight into the programming process.

State Machine Visualization

Observing a robot's actions is sometimes not enough to understand what went wrong. In visualizing the robot's state machine structure, students could often gain insights that were previously hidden. Students were instructed in the use of the Storyboard tool, which generated a graphical view of their state machine and allowed them to inspect the structural layout. Additionally, the Storyboard tool could be used to trace the execution of the program to ascertain if the state machine was sequencing correctly. This tool often provided students a way to discover inconsistencies in structure and function of their state machines that were not diagnosable using the robot behavior alone.

Instructor Course Correction: A key feature of Tekkotsu not found in LEGO Mindstorms is the ability to create new classes of nodes and transitions using the full power of C++. Mindstorms users can create macro-like combinations of system-supplied block types (the MyBlocks feature), but they cannot add new functionality beyond what those blocks provide. Because Tekkotsu programs are semantically much richer, they may require complex logic, e.g., determine whether there is a pink blob that is closer than the nearest blue blob. Tekkotsu instructors can implement these bits of logic as new node classes, thus allowing students who are not yet ready to tackle C++ to still solve the bulk of a problem using just the state machine notation. In some cases the need for a new primitive is suggested by the students themselves.

The distinction between node classes and instances is a subtle point that unexpectedly caused problems for students. Several times during the first three week course, we noticed students referring to a node class as if it were an instance. This would fragment the state machine; the compiler would create a new instance each time the class was referenced, so there would be several streams of program flow that did not connect together (Figure 5).

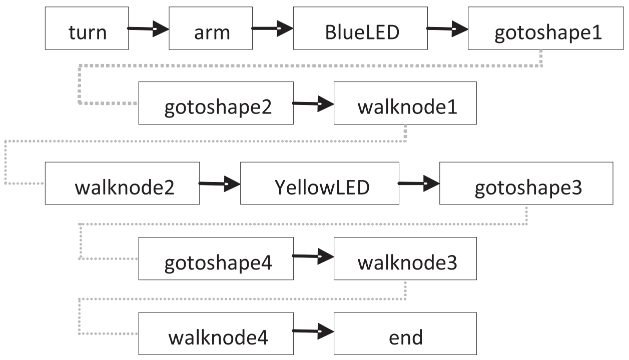


Figure 5 – A straight line program became fragmented due to students' failure to reference previously-created instances of the GoToShape and WalkNode classes. For example, gotoshape1 and gotoshape2 should be the same instance. Instead, separate instances were created.

A second common error that occurred frequently in the first session was that students who had instantiated a node of a particular class assumed that they could use this same instance in multiple places. For instance, if they instantiated a Pilot node that turned the robot 90 degrees to the right and named it ‘right’, they might refer to this node in multiple places in their code:

```
right: RightTurn =PILOT=> node1
node1 =T(5000)=> right =PILOT=> node2
```

The code above, written in the state machine shorthand notation, indicates that the student wished to turn the robot twice. Every occurrence of a class name (RightTurn) creates an instance of that class, in this case the sole instance is assigned the label “right”. The PILOT transition fires when the Pilot has completed the right turn, and the T(5000) denotes a five second timeout transition between node1 and what should be the second right turn. But the code was written in a way that creates a loop between the *right* node and *node1*, and therefore the program never reaches the final node, *node2*. Unfortunately, students also had difficulties understanding that they could solve this problem by creating two different instances of the RightTurn class inside their state machine:

```
right1: RightTurn =PILOT=> node1
node1 =T(5000)=> right2: RightTurn
                        =PILOT=> node2
```

Since we had the rare opportunity of participating in two sessions of SAMS, we took the lessons learned in the first session to inform the second. We switched to an approach that encouraged students to instantiate each new node on its own line before any references to that node:

```
right1: RightTurn
right2: RightTurn
right1 =PILOT=> node1
node1 =T(5000)=> right2 =PILOT=> node2
```

While this process might seem unnecessarily verbose to an expert, it eliminated two issues for novices: confusing a node class with an instance of that class, and using an instance of a class as some kind of universal symbol that could appear throughout the state machine. Understanding these common errors in the first SAMS session proved to be informative for the next, as students using the new coding convention were less prone to errors and better able to comprehend the distinction.

Pair Programming

While graphical displays and simulation aid learning, inexperienced students can benefit greatly from the addition of verbal explanations (Mayer and Sims 1994). Pair programming allows for multiple individuals to communicate and work in collaboration, which provides additional levels of intellectual support. It has been observed that “students who pair produce higher quality programs, are more confident in their work, and enjoy it more” (McDowell et al. 2003).

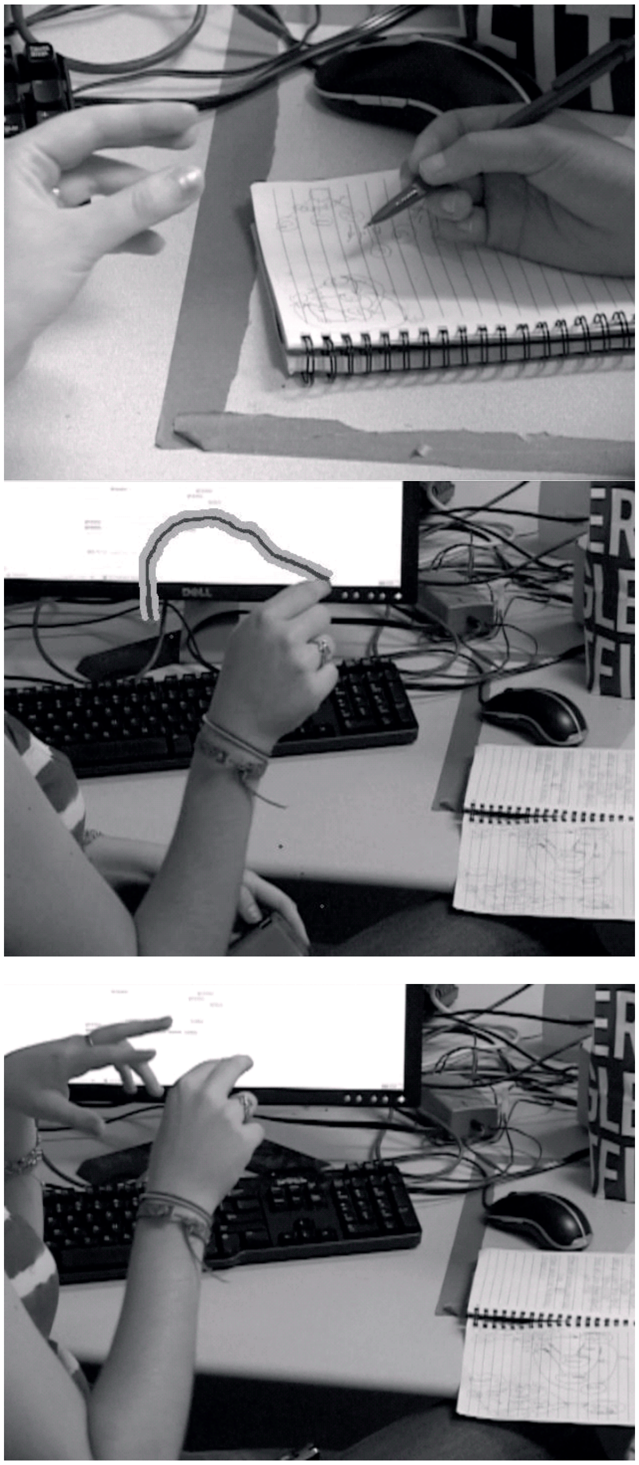


Figure 6 – Analysis of video data. Top: Two students discussing their state machine using gestures (left hand) and written notes (right hand). The student on the left curls her fingers to equate a fixed local 3D space with an imagined state node. Middle: Student’s finger trajectory introduces imagined motion through the air as a transition. Bottom: Student combines gestures to explain to a partner a more complicated setup. The three fingers on the left hand are associated with three distinct state nodes and the movements of the right hand with transitions.

A less commonly considered but important feature of pair programming which we considered in our course was the ability of group members to provide their partners with an embodied understanding of difficult and hard to grasp concepts through gestures. We did this by asking students to diagram out their solutions and discuss them with their partners. For example, when one student was not comprehending a concept, either on paper or the computer screen, another could explain the concept by equating states with imagined physical structure in space and transitions as embodied feelings of motion throughout that space. Adding these motions together, students could communicate more complicated systems, such as random transitions coming away from three state nodes, as shown in the bottom frame of Figure 6.

Not all novices come to computer science thinking in terms of symbols, states, and variables. By providing physical bodily simulations in addition to Storyboard visualization, students gain two routes to understanding the processes underlying state machine programming (Goldin-Meadow, Kim, and Singer 1999). Furthermore, students who produce a mismatch in their speech and gesture are often in a better position to learn a concept, and instructors can gain insights into a students' readiness by observing them explain their problems (Goldin-Meadow and Singer 2003). By requiring students to work together and communicate their understanding through diagrams, words, and gestures, we can hope to improve the quality of understanding in novices attempting to learn challenging concepts in robot control, such as parallelism.

Subjective Measures of the Course

Pre- and post- surveys were given to students to measure the effects of the course. Statistically significant changes in attitudes demonstrated that students held positive views about robotics after the completion of the course, as indicated with their agreement with the following statements: "I am already very familiar with the key ideas in robotics" (paired samples t-test, $t(10)=2.52$, $p<0.05$) and "I feel prepared to work independently on a robotics project" (paired samples t-test, $t(10)=3.03$, $p<0.05$).

Conclusions

Our novel course curriculum differs from current approaches in high school robotics and was designed with an emphasis on sophisticated robotics with primitives for vision and navigation. Robots like the Chiara, which include color cameras and on-board computational power equivalent to a laptop, are likely to supplant simple robot kits at the high school level once they can be efficiently mass produced (Touretzky, 2010). Tekkotsu's extensible state machine formalism allows non-programmers to take advantage of the Chiara's advanced capabilities and create interesting robot behaviors that include vision and navigation primitives.

By placing students in a rich environment designed to provide multiple levels of feedback and error correction, novices can more quickly and correctly learn aspects of computer science and robotics. Additionally, by observing the behavior of the students' robots, the graphical layouts of the state machines they design, and the gestures they use to describe their understanding, instructors can more efficiently address concerns and resolve issues with their courses in real time. Students in both sections of SAMS finished the course with high praise for the experience, and their performances indicated an increased understanding of higher level robotics concepts. By further developing our current structure and methods, we can hope to instill enthusiasm and confidence in a new generation of roboticists, and as one student put it, give them the "belief that anything is possible."

Acknowledgments

This work was supported in part by National Science Foundation award DUE-0717705.

References

Beer, R., Chiel, H., & Drushel, R. 1999. Using autonomous robotics to teach science and engineering. *Communications of the ACM* vol. 42 no. 6, 85-92.

Carnegie Mellon University. 2010. SAMS: The Summer Academy for Mathematics + Science. [online]. Retrieved November 21, 2010 from http://www.cmu.edu/enrollment/summerprogramsfordiversity/sams.html

Goldin-Meadow, S., & Singer, M. 2003. From Children's Hands to Adults' Ears: Gesture's Role in the Learning Process. *Developmental Psychology* vol. 39 no. 3, 509-520.

Goldin-Meadow, S., Kim, S., & Singer, M. 1999. What the Teacher's Hands Tell the Student's Mind About Math. *Journal of Educational Psychology* vol. 91 no. 4, 720-730.

Jacobson, M.J. 2000. Problem Solving About Complex Systems: Differences Between Experts and Novices. In B. Fishman & S. O'Connor-Divelbiss (Eds.), *Fourth International Conference of the Learning Sciences*, Mahwah, NJ: Erlbaum, pp. 14-21.

Mayer, R., Sims, V. 1994. For Whom is a Picture Worth a Thousand Words? Extensions of a Dual-Coding Theory of Multimedia Learning. *Journal of Educational Psychology* vol. 83 no. 3, 389-401.

McDowell, C., Werner, L., Bullock, H., & Fernald, J. 2003. The impact of pair programming on student performance, perception and persistence, *Proceedings of the 25th International Conference on Software Engineering*

Tira-Thompson, E. J., & Touretzky, D. S. (in press). The Tekkotsu robotics development environment. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-2011)*, Shanghai, China

Touretzky, D. S. 2010. Preparing computer science students for the robotics revolution. *Communications of the ACM* vol. 53 no. 8, 27-29.