Physical Interaction Design for Music

Michael Gurevich, Bill Verplank, Scott Wilson
CCRMA, Department of Music
Stanford University
Stanford, California USA
{*gurevich,verplank,rswilson*}*@ccrma.stanford.edu*

Abstract

Teaching physical interaction design for music combines aspects of embedded systems, sensors, electronics, sound synthesis, design and HCI. CCRMA's courses in this area draw students with a variety of backgrounds in these fields, and expose them to aspects of each. Students learn about technology and design theory, from the instructors and from each other. Multidisciplinary team design projects to create physical interfaces for music have resulted in a broad range of successful devices that can be described by their situation along a set of continua.

1 Introduction

CCRMA has offered courses focusing on human-computer interaction every year since 1996. Presently, there are two ten-week academic courses during the fall and winter terms: Music 250A and Music 250B. In addition, CCRMA now offers a two-week summer workshop entitled *Physical Interaction Design for Music*. Several aspects of the courses, including the students' varying backgrounds and levels of experience, the range of topics covered, and the technology employed present unique educational challenges for the instructors.

An overview of the goals and implementation of the courses is provided along with case studies of several recent projects. We propose a set of three descriptive axes helpful in situating the projects discussed.

2 Courses

2.1 Music 250A

Music 250A is officially titled *Human Computer Interaction Theory and Practice: Designing New Devices*. It is supported as a course in both the Music and Computer Science Departments. Consequently, while focusing primarily on interfaces for music, the course tries to address broader issues of interaction design, and some student projects do not deal exclusively with music.

Goals. There are two main purposes of the course. First, the students are exposed to important issues in the design of physical interfaces. This is accomplished by examining and thinking about existing devices, and through exercises in building simple new devices. This first part of the class covers a technological and theoretical background to interaction design so that the students have at least a portion of both the skills and knowledge they will need to design successful interfaces.

In the second part of the class, normally four weeks, the students undertake a group project, in which they design and build a physical interface. Here, the more general term *physical interface* is deliberately used in place of *music controller* not just to include non-musical projects (of which there have been very few), but rather to broaden the scope of kinds, styles or modes of interactions that the projects employ. Where *music controller* can have a very specific meaning, physical interfaces for music covers a broad range of devices from toys to instruments to art installations. The project enforces experiential learning, wherein the students' project ideas provide motivation to acquire resources and skills that would not otherwise be taught to them in structured labs. The team nature of the project forces students to manage group dynamics in the context of a team design, as well as to draw on each group member's expertise. It is our hope that in addition to learning something about interfaces and design, each student makes at least one "discovery" by working with or observing someone who has skills outside of their own area of expertise. These discoveries can be as simple as a programmer seeing the mechanical workings of a tape measure or a musician wiring a microcontroller circuit from scratch. It is often the demystification of some seemingly esoteric, yet relatively easy process that is the most powerful accomplishment a student can make. Such a discovery frequently opens a student's mind to new possibilities and is something that can otherwise be quite difficult to teach.

Students. One of the course's greatest strengths is indeed the diversity of experiences of the students who make up the course. It not only allows students to learn from each other, but for each to contribute in his or

her own way. The student body typically consists of musicians, electrical engineers, product designers and computer scientists. Project groups are ideally, but not always, made up of one student from each field. This is both a blessing and a curse for the class, and presents one of the most frequent difficulties for the instructors. In an interdisciplinary team, it is natural for each member to focus on his or her own area of expertise to maximize productivity, but as a result it can be difficult for students to expand their skills outside of what they already know. Brainstorming and discussion sessions help promote sharing among group members as well as between groups, but a consistently successful method of encouraging the kinds of "discoveries" discussed above is still elusive. The hands-on interdisciplinary nature of this course appears to be unique at Stanford, where there are other largely theoretical interdisciplinary design courses, and very practical but discipline-specific courses in mechanical and electrical engineering.

Practical Instruction In the first part of Music 250A, we use lectures and lab assignments to get students started with the theory and tools. Lectures focus alternately on design theory and technology to support the labs.

Lab exercises make up the practical portion of the first six weeks of the course. Conceptually, the labs are divided into two sets of three labs, the second set being an iteration on the first, going into more detail and with slightly different focus. This has a number of effects. We try to dispense with the "difficult", yet more mundane details of microcontroller architecture and programming and circuit building early on to allow deeper thought into the nature of human-computer interaction during the second iteration. This also implicitly exposes students to the idea of iterating on a design and the importance of revisiting the first attempt. By the third week, the students "complete the chain" as depicted in figure 1. By this, we mean understanding the model of the technology we use:

$$Gesture- > Signal- > Sound,$$

and implementing a basic system that incorporates the technology to complete this chain:

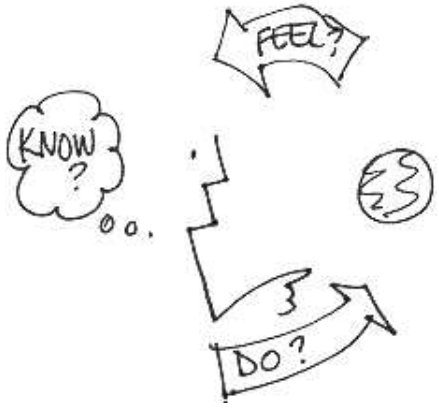
$$Human- > Sensor- > Microcontroller$$

$$- > PC- > Loudspeaker.$$

The second iteration on the lab exercises revisits the components of the system, focusing more on how each relates to human performance. For example, last year students programmed and performed reaction time tests using the microcontroller, and built signal conditioning circuits to deal with issues of human and sensor bandwidth. The final lab exercise combines all the elements, where students build a simple controller (often sensors taped onto an existing object) and demonstrate that it makes music for the rest of the class.

Theoretical Instruction The theoretical portion of the course deals with some "design" lessons, two of which we will address in detail. The first is to become aware of the difference between handles and buttons, the second is a framework or checklist of design considerations.

Handles vs. Buttons. There are three questions for interaction design: *How do you do? How do you feel? How do you know?*. The first question is the one about devices, not displays or conceptual models. Our first exercise is to sketch two real controls (one a handle, the other a button) and to comment on the difference.



Buttons give control over to the machine; handles allow continuous manual control. With a button, the user is normally forced into a sequence of presses; with a handle, a sequence becomes a gesture. Buttons are more likely *symbolic*; handles are *analogic*.

The realization, in the end, is always that we need both. Buttons determine *modes* while handles control whatever variable has been chosen. The button unlocks the door, the handle allows me to open it. A mouse is a two-dimensional handle with two buttons.

A Framework for Interaction Design. Successful interaction design involves balancing a variety of concerns using a variety of methods or representations. These are not suggested as stages in a design process but as a framework for checking, in the end, that all the necessary concerns have been addressed. The framework is illustrated in figure 1. Across the top are the overviews, across the bottom are the details. Left to right the columns are "motivation, meaning, modes and mappings" or "observation, invention, engineering and appearance".

On the right is some input device or control and an output display for showing the result (*mappings*). Next to that is the task analysis that details the step-by-step sequences that are organized by the mental model both for implementers and users (*modes*). Next, is a set of scenarios describing for whom you are designing and a metaphor or two to connect your idea to the solution

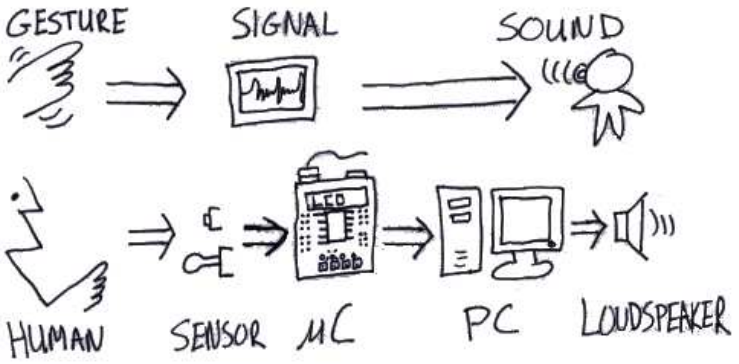


Figure 1: Theoretical and Practical Tool Chain.

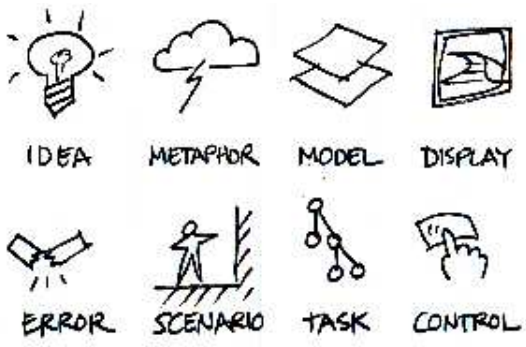


Figure 2: Interaction Design Framework

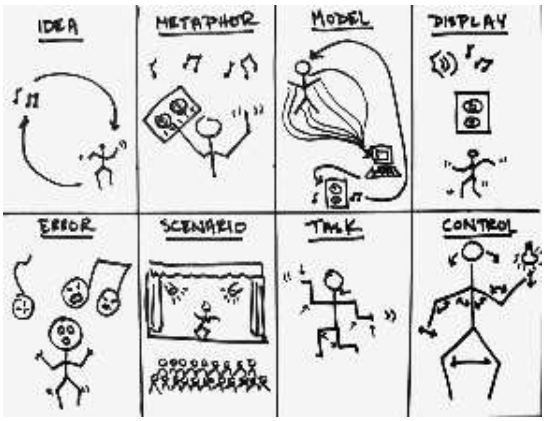


Figure 3: Framework Sketch Example

(*meaning*). At the left, is the original error or *breakdown* and the idea that must be there to notice something wrong (*motivation*).

Jeremy Faludi, Audrey Tsang, and Bradley Zimring, for their project "Dance-former-busta-movatron", have sketched (see figure 2) the eight aspects from uncertain dancers dreaming of creating the music as they dance to the details of control and auditory display. In between, are the dream of dancing on stage while making music like a DJ and conductor, and the task of moving while sending signals to a PC for making music.

They avoided the latency of camera sensing by attaching bend- and optical-sensors to the dancer.

2.2 Music 250B

Music 250B, "HCI Performance Systems: Music Controller Design and Development" is a ten-week continuation of 250A, that deals more with project development. This course has been subject to external forces, such as scheduling problems and changing degree requirements that have prevented it from having a clear, consistent focus from year to year. Music 250B is still trying to find its niche after existing in a number of different formats in the past few years. However, the course is not unsuccessful and there are many positive results that both the students and the instructors have taken away from the course.

Goals. The primary educational goal of Music 250B is for students to choose a project and excel at it. Though it may seem somewhat simplistic, to define, plan, and carry out an appropriate project in ten weeks can be challenging. There is a tendency for students to choose a more sophisticated engineering problem than they tackled in the first term, leading to varying degrees of success. An added objective this year was to expose students to more advanced topics, both technical and artistic, as a source of knowledge, motivation and inspiration, through a series of guest lectures.

Students. The diversity of students from Music 250A is often maintained in the second course, but in smaller numbers. This leads to more individual than group projects, which fosters more discoveries and exposure to fields outside the students' respective areas of expertise, but also leads to more isolation and an increased risk of "getting stuck". Frequently lacking is a critical mass of students such that those doing interesting, high-quality work provide an implicit motivation for the other students to do the same. With a smaller number of students, there is a tendency for all to get in a rut

of mediocre work.

Topics / Labs. The instructional content of this course does not add any new technologies to those learned in the first term, but students are free to explore other technologies on their own. A certain level of confidence is gained in the first term that allows them to do so. Lab exercises in the second course revisit some of the earlier topics in more detail. This year, the sequence of labs dealt first with *mappings* (different ways to assign gestures or signals to sounds); then with *modes* (ways of dynamically altering mappings; and finally with feedback and displays - how do you know what state your controller is in? An additional lab dealt with programming Pd external objects (externs) in C.

2.3 Summer Workshop

The two-week Physical Interaction Design for Music summer workshop is essentially a condensed version of the Music 250A course, focused more on exposing the participants to the relevant technologies, and motivating thought about interface design. Given the short amount of time, student projects are not as substantial as those in the regular academic course, but some are quite impressive nonetheless.

Unlike Music250A, most of the summer workshop participants are musicians, but with varying degrees of technological expertise. Some struggled with microcontroller programming and architecture more than others, but almost all were able to master the tools to a reasonable extent. The overall level of musicality of the student projects is quite impressive, in contrast to Music 250A, where the biggest weakness of many of the music-oriented projects is a lack of musical sensitivity that most of the summer workshop participants intuitively possessed. It is arguable whether or not this sensitivity can only come from the training and exposure that musicians have, but it has certainly alerted us to the necessity of at least trying to concisely teach non-musicians what is musically appropriate and viable in the context of Music 250A. Due to the varying levels of technical skills and short amount of time, the workshop consisted of lectures for the whole group and smaller, parallel tutorial sessions presenting topics like microcontroller programming, circuits, sensors and synthesis / signal processing in Pd.

A recurring theme in our course has been "Why Microcontrollers?". In other words, why not give students a black box that does 8 channels of A/D conversion on 0-5V signals and generates MIDI messages? The short answer is pedagogical. Using a programmable microcontroller allows the students to learn about computer architecture, digital logic, programming, A/D conversion and serial and parallel communication protocols.

In learning to program and use a microcontroller, students develop these skills and intuitions in a practical, hands-on way that would be difficult with theory alone. Furthermore, it gives students exposure to the technology used in actual commercial products, demystifying the world of embedded systems. In this respect, our choice of hardware platform is significant. The Atmel AVR series of microcontrollers is a professional, commercial-grade technology that is used in a wide variety of existing commercial devices. Though the technology can be daunting at first, both the hardware and and software programming platforms supporting the microcontroller are designed in such a way that it can be used quite simply initially, without blocking the more advanced features for advanced students. In a sense, this is an extension of the open source philosophy into teaching HCI. The system behaves well for the novice user, and advanced students' innovations or modifications can be incorporated into future versions. Students have without prompting become quite enthusiastic about documenting and preserving their work for the benefit of future students. In addition to teaching more about technology, microcontrollers have enabled students to produce innovative and highly successful projects that would not otherwise be possible.

The choice of a platform programmable in C has provided several benefits. There is both a wealth of knowledge and existing free code to be found at CCRMA, online, and in other fields. This code can be ported, modified and used at will in the students' projects. Using *gcc* as the compiler is advantageous because it provides a coherent transition from programming C in Linux and Mac OS X, with which many students have facility with, to programming for the microcontroller.

3 Project Case Studies

Of the many projects recently produced in the course, we have identified several continua along which the projects are situated. Three of these have been particularly useful in identifying the different kinds of interactions the projects employ.

Group vs. Individual. Several of the projects can be played by more than one individual simultaneously. Such an approach to collaborative music making is seldom found in traditional instruments.

Toy vs. Instrument. This distinction addresses the difference between toy-like devices which cause music to be generated in an incidental way, in the context of some other interaction (such as juggling or riding a bicycle) and interfaces that have deliberate musical results. The latter play a similar role to traditional instruments.

Mode Change vs. Expressive Gesture The third continuum identifies to what extent the player's physical gesture is directly expressive on the instrument as opposed to invoking mode changes at some granularity

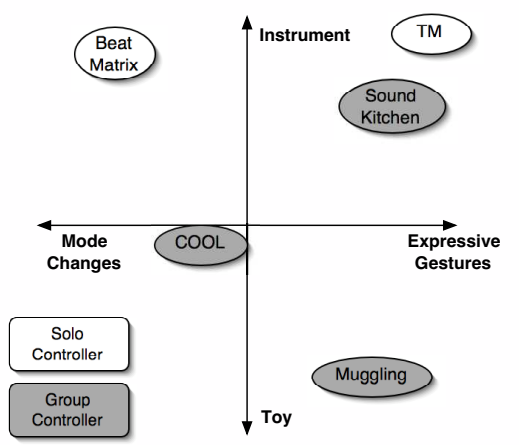


Figure 4: Interaction type space

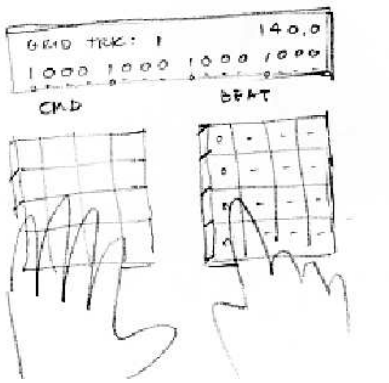


Figure 5: Beat Matrix

through contact with the device.

Figure 3 depicts these axes populated with the projects we discuss below. Given these continua, it is significant that we choose to use the term *physical interface* and not *music controller*, as described above. Music controllers would most likely be situated at one extreme point of this space: devices that are individual instruments whose interactions consist of expressive gestures. We have tried to open up this space for student work, and the resulting projects of been quite diverse and innovative.

Beat Matrix. The Beat Matrix, by David Lowenfels and Gregor Hanuschak, is a MIDI drum sequencer using two 4x4 keypads in conjunction with our development board. The goal was to create a 4 beat sequencer, using one 4x4 grid to represent the sixteenth note subdivisions of the beats. This project's success lies in its self-contained nature, which was afforded by the microcontroller. The controls and display are completely integrated into the device, relying on a computer or synthesizer for sound generation only. Our platform's LCD is integral to the device, giving the user immediate visual feedback for the controls, allowing the user to navigate through drum tracks, and displaying the state of the sequencer. David and Gregor learned to manage the microcontroller's internal memory to create and store note sequences. They used interrupts to effectively program the controls, the LCD display, and MIDI communication to ensure the steady timing essential for a sequencer. The students intend to add features including external memory to store user presets, which should be relatively easy with the current platform and the AVRlib's existing I2C support.

Figure 6: Muggling

The Beat Matrix is similar in some respects to a traditional controller in that it is played by an individual, and and is more like an instrument than a toy. But it is different in that and it quite far toward the *mode change* end of the third axis. This distinction is made because interaction with the controller consists of making persistent updates to the state of the system. These updates are fine grained mode changes, altering the current and future behavior of the system, rather than expressive gestures.

Muggling. "Muggling", developed by Pascal Stang, Jeff Bernstein and John McCarty, comes from "musical juggling", describing their intention to instrument 3 juggling balls to send signals that could be used to control musical parameters. The group successfully implemented one ball as a remarkable proof of concept. The ball contains 4 two-axis accelerometers, from which linear and angular acceleration in the x, y and z planes are calculated. Analog acceleration values are sampled, low-pass filtered and interpolated to 14-bits on an AVR323, then transmitted via a Linx wireless radio transmitter to a Linx base station receiver. The portability of the AVRlib code used in class allowed the group to easily switch to a more capable and substantially smaller microcontroller in the same family without any major code changes. The technology's low cost and small package size options allowed the accelerometers, 3-color LEDs, microcontroller and radio transmitter/receiver to be mounted inside a 3" diameter ball.

In contrast to the Beat Matrix, Muggling takes its place at the opposite end of all three continua proposed earlier. It is group friendly, permits direct gestural expression, and is much more a toy than an instrument.

SoundKitchen. Hiroko Shiraiwa, Vivian Woo, and Rodrigo Segnini created the SoundKitchen, a collection of sensors designed to sense various parameters of chemical reactions to create sound. Combinations of reagents including orange juice, red wine, and baking soda were mixed in vessels and the resulting voltages were measured. The result was a set of continuous sensors that were essentially wet cell batteries whose voltages changed predictably as reagents were added to the mix. The SoundKitchen was presented in a dramatic, performative manner with a unique aesthetic that combined mad-scientist wizardry and contemporary music performance.

The characteristics of the chemical reactions used were carefully worked out in advance. This determinism allowed the group to score their actions and as such, the SoundKitchen can be considered on the instrument end of the toy vs. instrument continuum. The controller is by nature a group interface and while mode changes do occur behind the sense, the interaction between the performers and the devices are quite expressive.

Circular Optical Object Locator (COOL). Tim Hankins, David Merrill, and Jocelyn Robert describe the COOL as follows:

> The Circular Optical Object Locator is a collaborative and cooperative music-making device. It uses an inexpensive digital video camera to observe a large rotating platter. Opaque objects placed on the platter are detected by the camera during rotation. The locations of the objects passing under the camera are used to generate music. (Hankins, Merrill, and Robert 2002)

The COOL is unique among the controllers presented here in that the object; the controller itself and the act of making music with it are just as important, if not more so than the musical result. Like the Beat Matrix, interactions with the COOL can be considered fine grained mode changes because the effects of the interaction are persistent, remaining in effect until another interaction is made to explicitly cancel or change them. This controller falls between the end-points of the toy vs. instrument continuum, and is most effective when used collaboratively by more than one person.

Musical TM. The Musical TM (TapeMeasure / TroMbone / ThereMin) is an instrumented tape measure designed by Becky Currano, Peter Solderitsch, and Unnur Gretarsdottir. The TM group successfully used a found object tape measure as the basis for their project by fixing a ten turn potentiometer to the rotating cylinder of the tape measure and digitizing the voltage across the pot. The tape was mounted inside a piece of cardboard poster tube sized to a comfortable length for one hand. The tube was outfitted with a series of force-sensitive resistors positioned to be played by the supporting hand, while the free hand "played" the tape by sliding it in and out of the tube.

Drawing the tape out of the body of the TM gave a very accurate and tactile continuous control that was effective when used to control the pitch of synthesis patches. In contrast to the previous examples, the TM is a project that most closely follows the paradigm of a traditional music controller. It is a solo instrument, with a repertoire of expressive gestures.

4 Teaching Platform

4.1 Hardware

Our current hardware platform is based on the Atmel AVR ATmega16 8-bit RISC microcontroller. We use the processor on the AVRmini custom development board (Stang 2003b) and program it in C from Linux, Windows and Macintosh OS X operating systems. The AVR series has a large active user base ranging from professional embedded-systems designers to hobbyists. One of the most important products of that community has been the AVR's inclusion as a standard build target in the open source Gnu *gcc* compiler.

The Processor. The AVR ATmega16 microcontroller has a maximum clock speed of 16MHz and the majority of its instructions complete in a single cycle thus providing assembly code performance near 16 MIPS[1]. The ATmega16 has 16KB of flash program memory, 1KB of data SRAM, and 512 bytes of EEPROM memory[2]. The processor has several interrupt sources enabling a program to respond to a number of internally and externally-generated events such as timer overflows and the completion of ADC or serial communication operations. The AVRlib (Stang 2003a) function library allows us to package the handling of these interrupts in a straightforward manner which the students quickly learn to use.

Standard microcontroller features found on most of the ATmega series of AVR microcontrollers include an interrupt-controlled UART for serial communication and three independent hardware timers, one can be synchronized to an external Real Time Clock (RTC) oscillator. The processor also supports up to three channels of pulse width modulation (PWM) output. The ATmega processors support the I2C and SPI communication protocols facilitating the addition of external peripheral ICs such as EEPROMs, programmable logic devices (PLDs) and digital to analog converters (DACs).

[1] Million instructions per second

[2] The flash memory stores the compiled program code, the SRAM holds run-time data, and the EEPROM is intended for storing calibration information or other data that must persist over power interruptions.

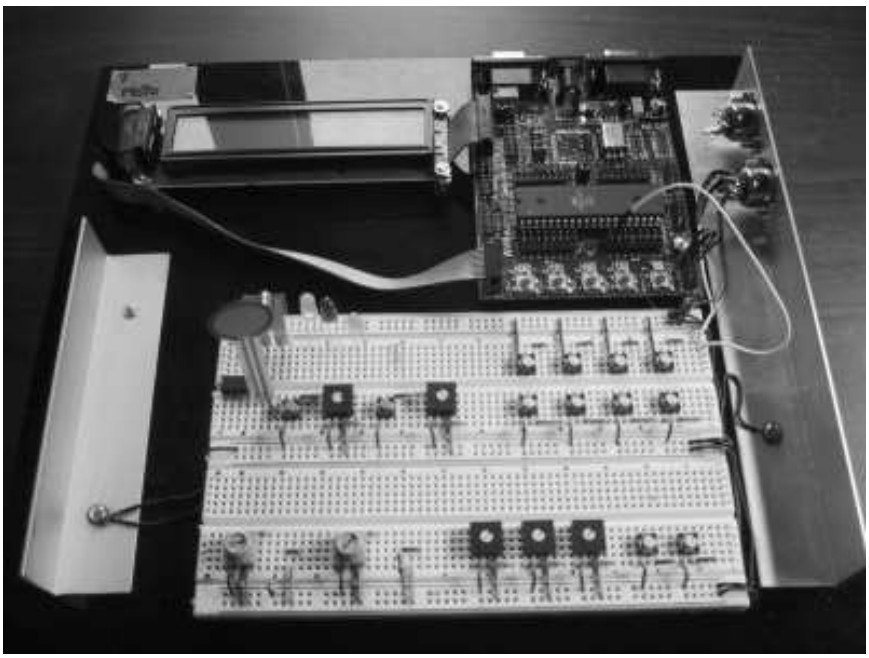


Figure 7: Prototyping Kit

We chose the AVR series of processors for the cost, memory size, and speed. Specifically we selected the ATmega16 for its eight channels of integrated 10-bit analog to digital conversion (ADC). The ADC support is the most heavily used feature in many student projects. Having integrated ADCs simplifies the task of reading continuous sensor circuits. The processor provides several choices of analog voltage reference for the ADC including an external reference, simplifying the task of scaling the sensor signals.

The Development Board - AVRmini. The AVRmini development board (Stang 2003b) provides convenient access to the I/O ports of the processor via blocks of headers. The headers enable individual pins or entire I/O ports to be exposed on a wireless prototyping board using jumper cables. Sensors are also easily connected directly to the pins of the AVRmini using jumpers. The board protects all I/O pins with a series resistor in place by default which can be bypassed as needed.

The AVRmini provides a header connection for an industry-standard character LCD module using four or eight I/O lines. The AVRlib (Stang 2003a) library provides several convenient options for output to the LCD. A built-in set of four LEDs and four push-buttons on the AVRmini may be connected via jumper cables to any of the I/O ports. This functionality has proven particularly useful in the early stages of learning about the processor, the development board and basic push-button and light circuits. The board provides an RTC clock crystal for clocking the third timer which can be enabled via jumpers.

Compiled code is downloaded to the microprocessor from a computer via an RS-232 serial connection. The AVRmini provides two RS-232 connectors. Either one may be patched to the UART so that the programming interface and a serial communications link may be kept connected simultaneously.

The AVRmini supports all 40-pin DIP and 64-pin QFP processors in the AVR series providing flexibility in choosing a processor with features suited to the specific application. Provision is also made for a bank of external SRAM of up to 512KB. An efficient switching voltage regulator prolongs battery life for autonomous applications and a separate analog voltage reference regulator may also be installed on the board.

4.2 Software.

We have used Linux almost exclusively for the programming and performance of the instruments created in these courses because CCRMA's systems are nearly all Linux-based. An IDE for the AVR exists for Windows, and thanks to the open source tool chain built around *gcc*, our development environment is usable from all the common operating systems in use today.

Compiler and Loader. Programs for the microprocessor are written in C with the addition of special purpose macros for the AVR. A makefile automates the process of compiling the AVR C code, linking it with the standard C library and the AVRlib library, generating the memory map and hex code files, and uploading them into the processor. The compiled code is uploaded to the processor using the command line utility *uisp*.

Support Library - AVRlib. The AVRlib library of C support routines (Stang 2003a) is extensive and invaluable for work on this platform. AVRlib includes functions wrapping many of the standard features of the AVR processors. These include support for managing the timers, using the A/D converters, the UART, SPI and I2C interfaces, and the PWM outputs. General purpose code in the library provides bit and byte oriented data buffers, an implementation of a convenient C-style printf function, and terminal emulation facility. There is also support for specific peripherals including character and graphical LCD modules, external SRAM, GPS, USB, 400Mhz RF wireless transceivers, ATA hard drives and an MP3 player! This monumental library continues to grow and improve as the author uses it in his own teaching and personal projects.

Control Output. In an effort to provide the students a flexible set of options for the output of their controllers we support both MIDI and Open Sound Control (OSC). For MIDI, the standard MIDI output circuit is attached to the UART pin of the processor. A series of MIDI-specific functions are provided which encapsulate the standard UART library routines. OSC support is provided by including a simplified implementation of the OSC message construction code and a similar set of wrappers around the UART library functions. The OSC messages were received on the Linux computers via RS-232 serial. The OSC UDP receiving object for Pd(Puckette 2003) was modified to create an object called OSCSerial that receives messages from the serial port rather than the network (Wilson 2002). Students can send MIDI or OSC messages with a single C function call, but have access to the libraries to see exactly what is going on behind the scenes and customize or extend the functions if necessary.

Pd is our software of choice for designing the musical side of the projects because of its availability on several platforms and its popularity at CCRMA. Students have also used the MIDI output functionality to communicate with Max, Pd, and commercial music gear.

5 Conclusions

CCRMA's three human-computer interaction courses expose an interdisciplinary group of students to technical and theoretical aspects of physical interaction design. The main pedagogical goals are that the students achieve technical skills necessary to design and build functioning physical interfaces, develop a vocabulary and critical thinking skills with which to evaluate existing interfaces and build a conceptual framework in which to design new ones, and to gain experience by carrying out a team-based interdisciplinary design project.

The interdisciplinary nature of the students, and of the subject of the courses themselves provides a set of unique challenges. Our approach is to have all students, regardless of their background individually tackle all of the technical issues before breaking into more multidisciplinary project teams. This allows students to first glean knowledge from fields other than their own, and then provides them with an opportunity to shine in doing what they do best. This approach also tries to encourage students to learn from one another. The empowerment of students who make simple discoveries in new fields is quite striking. Student projects demonstrate the technological and design mastery that can be achieved in a short amount of time drawing on these interdisciplinary groups, and provide us with useful continua for thinking about methods of interaction.

To support the courses, a powerful microcontroller-based hardware platform is used that provides relatively easy operation of basic tools for beginning students, while advanced students still have access to powerful, professional-level tools that promote portability, scalability, and ample support for refinement and true innovation in their projects.

6 Acknowledgments

Thanks to Max Mathews, Pascal Stang, Wendy Ju, Stefania Serafin, Chris Chafe, Gary Scavone, Fernando Lopez-Lezcano and the CCRMA and music department administrators. Thanks to all of the participants in the 2002 summer workshop who were good sports as our guinea pigs. Thanks to David Lowenfels, Vivian Woo, Rodrigo Segnini, Hiroko Shiraiwa, Pascal Stang, John McCarty, Jeffrey Bernstein, Gregor Hanuschak, David Merrill, Tim Hankins, Jocelyn Robert, Becky Currano, Peter Solderitsch, Unnur Gretarsdottir, Jeremy Faludi, Audrey Tsang, and Bradley Zimring.

References

Hankins, T., D. Merrill, and J. Robert (2002). Circular optical object locator. In *NIME 2002*.

Puckette, M. (2003, January). Pure data. `http://crca.ucsd.edu/~msp/software.html`.

Stang, P. (2003a, January). Avrlib: C function library code for atmel avr processors. `http://www.procyonengineering.com/avr/avrlib/`.

Stang, P. (2003b, January). Avrmini: Homepage of the diminutive atmel avr application/development board. `http://www.procyonengineering.com/avr/avrmini/`.

Wilson, S. (2002, August). Osc serial object. `http://www-ccrma.stanford.edu/~rswilson/OSCSerial`.