2005 American Control Conference
June 8-10, 2005. Portland, OR, USA

Industry Needs for Embedded Control Education

J. S. Freudenberg
Dept. of Electrical Engineering and Computer Science
University of Michigan
1301 Beal Avenue
Ann Arbor, MI 48109-2122, USA
jfr@umich.edu

B. H. Krogh
Dept. of Electrical and Computer Engineering
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-3890, USA
krogh@ece.cmu.edu

Abstract— **The rapidly growing embedded control industry has created a need for engineers whose training extends across the traditional pedagogical boundaries. To address this challenge, courses in embedded control systems have been created at the University of Michigan and at Carnegie Mellon University. These courses have in common a hardware laboratory based on the MPC555 microcontroller used to implement force feedback algorithms to enable a human to interact with a computer through a haptic interface. In this paper we describe the pedagogical challenges posed by the embedded control field, the hardware laboratory used at Michigan and Carnegie Mellon, and the individual courses that use this laboratory.**

I. Introduction

An embedded computing system, by way of contrast to traditional desktop, laptop, and workstation computers, is one in which the computer itself is but a component of another technological device. Embedded systems are faced with constraints on memory, power, cost, and the user interface. Examples of such embedded systems include cell phones and personal digital assistants. Embedded *control* systems are a special class of embedded systems wherein the microprocessor is embedded in another piece of technology with which it interacts by implementing a control algorithm. Examples include airplanes, automobiles, household appliances, copy machines, hospital beds, and machine tools.

An embedded control system shares the same constraints on cost, memory, and so forth as do other embedded system applications. Many other constraints are particular to embedded controllers, however. These include: the need to interface with and influence the external environment through sensors and actuators; the fact that real-time operation is critical for performance and safety; hybrid system behavior due to the interaction of continuous dynamics and state machines; and distributed control problems arising from embedded controllers interact with one another over a network.

Hence, to understand an embedded control system requires a skill set that goes well beyond that usually acquired by a computer engineering student, which would include detailed knowledge of microprocessor hardware, but not its interaction with the environment beyond the use of discrete switches and LEDs. A student with a traditional control background, on the other hand, would know about control algorithm design and analysis, but not understand important issues that constrain the computer implementation of these algorithms. These issues go well beyond the issues of sampling and quantization taught in a typical course on digital control systems.

Indeed, a design team for an embedded control system application will require expertise that extends across traditional disciplinary boundaries. Skills required include

- design of control and signal processing algorithms
- computer hardware (interfacing, memory, and timing)
- computer software (multitasking, real-time computation)
- interface and power electronics
- sensors and actuators (DC motors, encoders)
- mechanical design

Cost constraints in consumer applications require design tradeoffs to be made across disciplinary boundaries: a change in the mechanical hardware to save money may require changes in the computer hardware, software, or control algorithm. This poses a challenge in industry, where design teams must be assembled from individuals who have expertise in the separate disciplines, but may be unaware of the issues and problems facing members of the team from other backgrounds.

The situation in industry described above was explained to the first author in 1999 by Dr. Ken Butts, then working at Ford Motor Company. With advice from Dr. Butts, a new course in embedded control system was created at the University of Michigan to directly address the needs for embedded control expertise in the automotive industry. The hardware laboratory developed for the Michigan course has been introduced at Carnegie Mellon University, where it is being used to provide embedded control experience for students in a pre-existing digital control course.

The development of courses in embedded control systems impose pedagogical challenges similar to those faced in industry and for the same reason: the highly multidisciplinary nature of the field. The purpose of the present paper is to share with a general audience the ways in which these challenges have been addressed by the authors at their respective universities.

The development of the courses described in this paper was supported in part by NSF Grant EIA-0088064.

0-7803-9098-9/05/$25.00 ©2005 AACC

The courses in embedded control systems described below are not merely a response to a short term trend. On the contrary, the availability of low cost embedded microprocessors is enabling the introduction of embedded control systems into almost every conceivable technology. Moreover, courses in embedded control systems introduce students to issues at the forefront of control research, because such topics as hybrid dynamical systems and distributed control over a network arise very naturally.

The remainder of this paper is outlined as follows. In Section II we provide additional discussion of industry trends in embedded control systems, motivated largely by the automotive industry. The class at the University of Michigan is described in Section III, including a discussion of the laboratory setup, which includes the MPC555 microcontroller developed for automotive powertrain control applications, and a haptic interface, or force feedback system, designed by Professor Brent Gillespie at the University of Michigan. The course at Carnegie Mellon University, which also uses the MPC555 and the haptic interfaces developed by Professor Gillespie, is described in Section IV.

II. Industry Trends in Embedded Control Systems

Industry is being forced to change the way in which it develops embedded control software. Consider current developments in the automotive industry, which is faced with the problem of managing the rapidly increasing complexity of the embedded software used in powertrain and vehicle control. This growing complexity is caused to a large extent by the demands to meet environmental, fuel economy, and safety regulations under the cost constraints of the consumer market. In order to meet these regulations, many new developments in the technology for engine and vehicle control are being introduced. Novel actuators and sensors are being added to the powertrain, and many traditional mechanical control devices are being replaced by electronics. For example, an electronic throttle is replacing the mechanical throttle linkage between the driver and the engine. As another example, in the future the crankshaft and camshaft will no longer be directly linked, and the input and exhaust valves will be controlled individually. All these innovations require embedded microprocessor control.

Despite the potential improvements afforded by the use of embedded microprocessors, their successful integration into production vehicles presents a challenging software development problem. Current industry practice requires that too much software testing be done at a late stage in the design cycle, when the production code is tested on the vehicle. As a consequence, a heavy burden is placed on the engineers responsible for final software calibration. Significant software changes made late in the development process increase time-to-market, thus reducing profitability and rendering problematic the introduction of otherwise promising new technology.

As in many other areas of engineering, it is advantageous to conduct as much testing as possible in software, through simulation and verification, before hardware implementation. The *model-based embedded control software design paradigm* requires that the control system be modelled and simulated using tools such as MATLAB, Simulink, and Stateflow, before code is written. The latter tool is necessary because embedded control software contains a large amount of mode switching, diagnostics, and logic, in addition to more familiar control and estimation algorithms. There is great interest in using rapid prototyping tools, such as Real Time Workshop, to automatically generate executable code directly from a Simulink/Stateflow diagram. Rapid prototyping has proven invaluable in testing embedded control algorithms on hardware before they are coded into production software. Doing so enables several design iterations of the algorithms, even before production hardware is available. Discussion with industry contacts indicates that rapid prototyping has not been used extensively for producing actual production code, although there is considerable interest in moving in this direction, and many companies are actively engaged in doing so.

One implication of the preceding comments is that *C* remains the standard language for writing production embedded control software, and this will continue to be true even while the use of autocode generation tools increases. An instructor developing a course in embedded control systems must thus decide how much code students should write in *C*, reflecting current industry practice for writing production software, and how much is code should be autogenerated with rapid prototyping tools, reflecting the model-based control design paradigm toward which industry is moving. Indeed, there are many jobs available for students who want to work at making the code generation tools work better.

III. The Course at the University of Michigan

The course EECS 461, Embedded Control Systems, is now in its third year as a regularly offered class in the Department of Electrical Engineering and Computer Science (EECS) at the University of Michigan. Currently it is offered twice a year, with an enrollment of 48 students each semester, for a total of 96 students/year. The class is taught in Fall semester by Professor J. S. Freudenberg, and in Winter semester by Adjunct Professor J. A. Cook, also of Ford Motor Company.

The student body consists of seniors in both electrical engineering (EE) and computer engineering (CE), some graduate students in these disciplines, and an occasional graduate student in mechanical engineering (ME), by permission of the instructor. Due to the heavy demand for the class, enrollment is normally restricted to students in the EECS Department. The undergraduate students in EE and CE have in common a low-level discrete signals and systems course, an introductory course in circuits, and background in C^{++} programming. Many EE students and all CE students

take a course in digital logic. In addition, students must have taken at least one of the following courses: EECS 306, a traditional course in signals and systems that covers Laplace and Fourier transforms, and makes significant use of MATLAB, or EECS 373, a laboratory based course in microprocessor systems taught in assembly language. The latter prerequisite is normally taken by the CE students, and the former by EE students, although EECS 306 is increasingly popular with CE students as well.

It is important to tailor the lectures and laboratory exercises to the background of the students. For example, the laboratory uses DC motors, a topic is not covered anywhere in the EECS curriculum. On the other hand, students can be expected to be familiar with *C*. Were the class taught in the Mechanical Engineering department, students would have studied DC motors, but might not have seen *C* programming or know that data is stored in registers.

A. Laboratory Hardware and Software

The laboratory consists of four major components: an embedded microprocessor,a software development environment, mechanical hardware, and the MATLAB based environment for autocode generation.

The microprocessor used in the EECS 461 laboratory is the MPC555, originally manufactured by Motorola, and now by Freescale, a division of Motorola that was recently launched as a separate company [1]. The MPC555 is specially designed for use in automotive applications. The many novel sensors and actuators used in advanced technology powertrains to control such things as ignition, fuel injection, and valve timing, imply that a microprocessor must be capable of processing a large number of inputs and outputs. Real-time performance is critical to safe and reliable engine operation. The microprocessor must operate in a harsh environment, over a wide range of temperatures and is subject to significant electromagnetic interference. The MPC555 microcontroller was developed to meet these constraints: it has a 32 bit PowerPC core with a floating point coprocessor, runs at 40MHz, and can function over a temperature range of -40° to $+125^\circ$ C. The MPC555 has programmable time processing units (TPUs) that are actually separate processors used to handle I/O duties that would otherwise require either a separate chip or CPU interrupt service. They are used, for example, to perform quadrature decoding. The MPC555 also has modules to interface with a Control Area Network (CAN), a networking protocol popular in automotive applications.

The EECS 461 laboratory uses the CME-0555 development board for the MPC555, available from Axiom Manufacturing [2]. The CME-0555 board is pictured in Figure 1, together with an additional interface board designed at the University of Michigan. The purpose of the latter board is to provide additional buffering, and some dipswitches and LEDs for simple I/O to the MPC555.

The software development environment currently used in EECS 461 consists of the Diab *C*-compiler and the SingleStep on-chip debugger available from WindRiver [3]. The latter tool allows students to step through the code one line at a time, view the contents of individual registers on the MPC555 as they change in response to external inputs, and study the assembly code produced by the compiler. These professional development tools greatly speed the process of writing and debugging embedded *C* code.

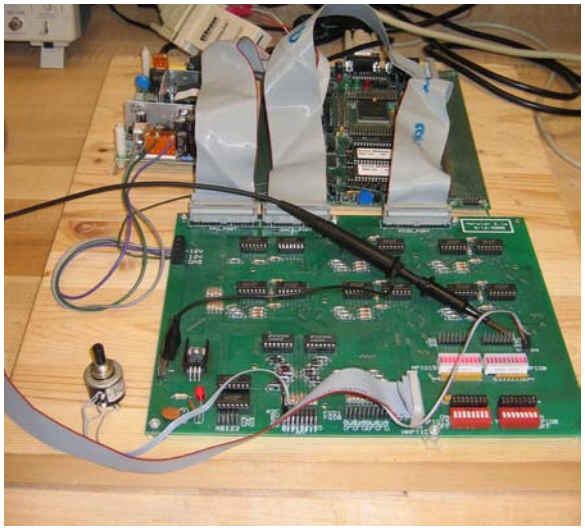

Fig. 1. MPC555 development and interfacing boards.

Although the MPC555 was developed for use in the automotive industry, it is not feasible to use it for engine control in the EECS 461 laboratory. Instead, we use another interesting and challenging application of embedded control, a haptic interface, or force feedback system, developed by Professor Brent Gillespie of the Mechanical Engineering Department at the University of Michigan. The specific haptic interface we use is the wheel depicted in Figure 2, and described in [4]. This wheel is equipped with an optical shaft encoder to measure angular position, and a DC motor driven by a PWM amplifier in current control mode.

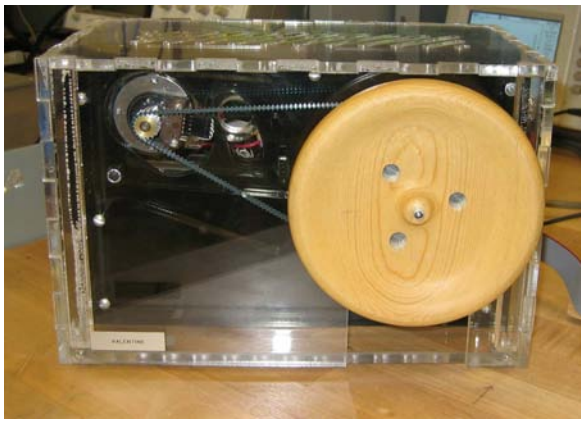

Fig. 2. Haptic wheel.

Haptic interfaces are intended to enable humans to interact with a computer through the sense of touch. Perhaps the

most familiar example is a force feedback joystick. Haptics are used in virtual reality flight and driving simulators, in teleoperation for manufacturing operations, surgery training, and will be required in future generations of *X*-by-wire cars, where mechanical linkages between the driver and the automobile are replaced by electronics. Haptics represent a challenging embedded control task, in part because the human haptic sensor, i.e., the sense of touch, has a bandwidth in the range 500-1000 Hz, as opposed to the human visual sensor, which has a bandwidth of 30 Hz. This implies that the embedded software used to update the virtual reality simulation with which the human interacts must satisfy rather fast timing constraints. A description of the specific force feedback algorithms implemented in EECS 461 will be described in the discussion of the individual lab exercises.

The final component of the laboratory is the software used to support model-based control software development. The each workstation is equipped with MATLAB, Simulink, Stateflow, and Real Time Workshop, all products of the MathWorks [5]. We also use the MotoHawk rapid prototyping tool available from Mototron [6]. The latter requires the use of a real-time operating system (RTOS) which, in our case OSEKWorks, an OSEK compliant RTOS from WindRiver (OSEK is a European software standard).

B. Lectures

The embedded controls course is a bit unlike other 400-level courses at the University of Michigan, which tend to delve deeply into a single technical area. Instead, EECS 461 covers a number of topics from different disciplines, so that students see how they must fit together to enable the development of an embedded control system, and to begin to appreciate design tradeoffs that must be made across disciplinary boundaries.

Initial lecture topics include sampling and quantization, use of quadrature decoding to obtain position and velocity measurements, frequency response, pulse width modulation (PWM), DC motors and motor control. These topics are generally found in a mechatronics course, and are necessary to enable students to interface with the haptic wheel. The special features of the MPC555, including the Time Processing Units, are discussed, since these are used in interfacing.

Haptic interfaces are discussed, together with some simple virtual reality environments that students implement in the lab. These include the *virtual wall*, in which the idea is to implement a virtual mechanical stop to prevent the wheel from rotating past a certain point, and the *virtual spring mass* system, in which the wheel is made to behave as though it were attached to a virtual wheel through a torsional spring. It is necessary to discuss artifacts of the digital implementation that students will experience. One of these is "wall chatter", a limit cycle that arises due to quantization and sampling, that can make the virtual wall vibrate noticeably. Another is stability of the algorithms used to numerically integrate the differential equations describing the virtual spring-mass systems. Ease of implementation suggests using forward Euler integration, which is known to be numerically unstable. By adding a virtual damper to the system it is possible to precisely compensate for this instability. Students are required to construct Simulink models of the virtual worlds they create, and to make sure that their algorithms work in simulation before they are allowed to implement them on the processor.

A number of computing issues are described, including interrupts, which are used in the lab exercises to provide a time step for numerical integration. Shared data problems and multitasking are described, as are elementary concepts from real-time operating systems, such as task states and priorities. Issues such as deadlock and priority inversion are discussed, and how these are addressed (or not) by semaphores, priority inheritance, and priority ceiling protocols. The course discusses real-time computation, including rate monotonic and earliest deadline first scheduling.

Students will implement force feedback algorithms over a Control Area Network (CAN). This networking protocol is used in automotive applications to transmit relatively small message and data samples. Each message is assigned a priority; if two nodes attempt to write to the network at the same time, the highest priority message is transmitted. Timing aspects of CAN networking are discussed, and students learn how to estimate bus utilization.

Students are required to use Simulink and Stateflow to simulate DC motors, quadrature decoding algorithms, and the virtual reality environments they implement on the MPC555. This provides an introduction to the model-based control software development paradigm, in which extensive use is made of simulation and rapid prototyping to reduce development time and cost. A key tool in this paradigm is the use of autocode generation to produce executable *C* code directly from a Simulink model. This process and the potential advantages it affords are explained in lecture near the end of the semester, after students have enough experience with writing *C* code and with simulation to understand the procedure.

C. Laboratory Exercises

A semester at the University of Michigan is fourteen weeks along, with labs starting the second week of the semester. During the next nine weeks, students perform eight laboratory exercises (one takes two weeks). These exercises are all intended to be performed by teams of students working in pairs, and completed within a three hour time slot with extra time required for prelab preparation and postlab analysis. After completion of these laboratory exercises, students spend the remainder of the semester working on a project.

Lab 1, Familiarization and Digital I/O: Students learn to write *C* for embedded applications, using the 'union" command in *C* to perform low level bit manipulations needed to access data registers. Doing so is required for

sensor interfacing, and to be able to examine a header file to determine the types of different variables. Students learn to program the Modular Input/Output System on the MPC555 to read numbers set on the dipswitches and write the result to LEDs.

Lab 2, Fast Quadrature Decoding Using the TPU: Students learn about the fast quadrature decoding function that is burned into ROM on the time processing units of the MPC555, and use this function to interface to the encoder on the haptic wheel. Given the gear ratio between the haptic wheel and the motor drive shaft, where the encoder is mounted, the length of the register holding the counter that keeps track of changes in wheel position, and the rate at which this counter is read by the CPU, they compute the maximum rate at which the haptic wheel can turn before the MPC555 loses track of position.

Lab 3, Queued Analog to Digital Conversion: A/D converters are relatively expensive, and many microcontrollers allow them to be multiplexed. Students learn to program the QADC module on the MPC555 for this purpose, and performing timing analyses to determine how long the conversions take.

Lab 4, Pulse Width Modulation and Introduction to Simple Virtual Worlds: The DC motors used to provide force feedback are driven by a PWM amplifier that, in turn, is driven by the MIOSPWM module on the MPC555. In this lab students learn to program this module and use it to construct their first example of a virtual reality, the virtual wall described above. To do so, the processor implements a force feedback law stating that if the wheel "penetrates" the virtual wall (i.e., turns past the position of the virtual stop), then a strong restoring force proportional to the penetration distance is applied, effectively a virtual spring. Students are required to work with the conversion factors between wheel angle and encoder counts, and between duty cycle and torque, so that 1° of wall penetration corresponds to 400 N-mm of torque. The simulation of changing contact conditions may cause the chattering effect described above, and students are able to experiment with this hybrid system phenomenon.

Lab 5, The Periodic Interrupt Timer and Frequency Analysis of PWM Signals: Simple virtual worlds such as the spring used for the virtual wall can be implemented in a software loop, and do not require a concept of time, as would be needed for numerical differentiation or integration. More complex worlds do require a time step, and students learn to supply this using the periodic interrupt timer on the MPC555. In this lab they use the timer to create an interrupt routine that modulates the duty cycle by sampling a sine wave. When the result signal is passed through an appropriately designed hardware filter and displayed on an oscilloscope, the sine wave is recovered. Students write three different interrupt routines to obtain the sine wave samples: by sampling the output of a signal generator, by using the *C* sine function, and by using a lookup table. The resulting interrupt routines have different execution times, which students measure by toggling a bit after entering and before leaving the routine.

Lab 6, Virtual Worlds with Haptic Feedback: With a concept of time available, virtual worlds that require numerical integration or differentiation can be implemented. In this lab, students program the MPC555 to simulate the effect of a virtual wheel attached to the physical wheel with a virtual torsional spring. The virtual inertia and spring constant must be chosen so that, in response to a 45° step in the physical wheel, the virtual inertia will oscillate at 1 Hz, with the maximum torque produced being at most 800 N-mm. Before the students begin writing software to implement the virtual inertia, they are required to develop a Simulink model to verify that their force feedback algorithm, if implemented correctly, will have the desired behavior. Another issue that arises is the choice of numerical integration routine. Although the forward Euler method is numerically unstable, it is simpler to implement than certain stable routines, such as trapezoidal, which have direct feedthrough terms that can introduce algebraic loops. Students are asked to compute the amount of virtual damping needed to offset the destabilizing effect of the Euler integration, and verify the result in simulation, before testing their code on the haptic wheel.

Lab 7, Controller Area Networking: The six workstations in the EECS 461 lab are connected over a CAN network. In this lab students learn to transmit and receive messages over the CAN bus. They first implement a virtual wall over the network, and experience degradation in the quality of the wall they can implement due to networking delay. They then implement a virtual chain of wheels – when one is turned all the wheels in the room also turn using a simple position tracking algorithm. If the position of one wheel is held constant, then the person turning the master wheel experiences an opposing force.

Lab 8, Introduction to Rapid-prototyping: To this point, all the embedded software has been implemented in hand written *C* code, although Simulink modelling has been used to verify that the force feedback algorithms have the correct functionality. In this lab, students generate embedded software directly from the Simulink diagrams. To do so, they use Real Time Workshop from the MathWorks, a set of device driver blocks for the MPC555 obtained from Mototron, and an OSEK compliant RTOS used to specify task priorities and to address shared data issues. Students construct a Simulink model of two virtual wheels with significantly different natural frequencies, thus motivating an implementation in task states executed with different periods and thus different priorities. Students see both the advantages and disadvantages of the rapid prototyping process. One the one hand, the generated code is more difficult to understand and less efficient than handwritten *C* code. On the other hand, it is very simple to modify the virtual environment simply by changing or adding new Simulink blocks.

Project At the end of the semester, students spend about

three weeks on a project that uses a graphical display for a driving simulator. For example, it can be arranged that each of the six haptic wheels is used to steer a car around a virtual track with force feedback used to put "feel" into the steering wheel. All six cars appear on the monitor, and interact through an "adaptive cruise control" algorithm that maintains a minimum distance between the cars. Rapid prototyping is used extensively for the project, which would essentially be impossible without it given the time allowed. Further discussion of the driving simulator is in [7].

IV. The Course at Carnegie Mellon University

The Department of Electrical and Computer Engineering (ECE) at Carnegie Mellon University has offered embedded systems courses of various sorts for three decades. Grason and Siewiorek describe an early embedded controls project course [8] that included student projects on motion control, flight simulation, train control, and ultrasonic obstacle detection. Over time, various courses have evolved to cover a variety of topics and areas of specialization, including, mechatronics and microcontroller applications, distributed embedded control, system on chip, networking, embedded PCs, safety critical systems, robotics, computer peripherals, wireless data systems, signal processing, and command and control systems. Additional courses cover cross-cutting skills that are important to embedded system designers include: security, dependability, energy-aware computing, software/systems engineering, real-time computing, and human-computer interaction. A recent paper reviews the full scope of the undergraduate embedded systems education at Carnegie Mellon [9].

In addition to the rich set of courses related to all aspects of embedded systems and embedded computing, two features of the ECE curriculum at Carnegie Mellon strongly influence the type of course that can be offered in embedded control systems. First, many courses have a significant project component, so the students are quite familiar with computing technology and tools for software and system development. Second, the ECE curriculum offers the students considerable flexibility in choosing courses [10]. This makes it imperative that the prerequisite sequences for individual courses are not extremely deep.

In this context, we have offered a course focusing on computer control systems with an emphasis on implementation of experiments in the laboratory for several years. This course is offered one semester each year, with enrollments of 12 to 24 students. Most of the students are juniors or seniors in ECE, but graduate students in ECE, mechanical engineering, robotics, and even architecture (for building control systems) have taken the course. The explicit prerequisite is a basic course in signals and systems, but students typically have some experience programming microprocessors. They also have a basic understanding of embedded computing.

The computer control systems course has developed over more than a decade to address the need for students to understand the issues that impact the design and implementation of real control systems. We found that students who simply take control courses along with embedded systems courses do not make the connection between the theory of control system design and the actual implementation of controllers. By focusing exclusively on the mathematics of feedback control system design, traditional control courses fail to introduce students to the realities of embedded control system implementation. Control-oriented models neglect important details such as sampling jitter, finite precision arithmetic (which is typically much more restrictive on an embedded processor than in the computers used for design), data conversion, timing constraints, limits imposed by the device interfaces, and physical saturation in sensors and actuators. These issues are addressed and understood adequately only when the feedback control design is carried through to a full implementation. On the other hand, an embedded systems course that does not consider feedback control applications fails to introduce students to the types of constraints and limitations that arise when the "environment" for the embedded software is a complex physical dynamic system.

We have designed a course that introduces students to several of these topics through lectures, homework assignments, and most importantly through a sequence of laboratory exercises. The current version of this course is 18-474 Embedded Control Systems. Because there are other courses in our curriculum in which students can learn the fundamentals of embedded computing systems and *C* programming, the emphasis in this course is model-based design using MATLAB Simulink/Stateflow and Real Time Workshop (RTW). Experiments involving motor control systems increase in complexity, culminating in experiments using the haptic device developed at the University of Michigan. To make sure students understand the real-time implementation, they are required write some *C* code without using autocode generation and also modify some of the automatically generated code on selected projects. The overall emphasis is on the practical interpretation and implementation of many topics typically studied in a digital control course.

A. Laboratory Hardware and Software

For several years, students implemented control experiments using PCs equipped with boards for digital and analog IO. Experiments included temperature control using hairdryers and temperature sensors, water-level control, and DC motor control using optical encoders for position and velocity feedback. The DC motor setups included linear amplifiers implemented with power op amps and optical encoder counters using the HCTL 2000 quadrature decoder/counter interface IC [11]. For these projects, students often used microcontrollers, such as the PIC microcontroller [12].

For PC-based control, time-based sampled-data control was accomplished in a number of ways over several versions

of the course. Initially, a custom routine was implemented to program the PC timer to generate interrupts at a specified frequency and the student's control programs were installed as the timer interrupt service routine. The real-time operating system RT Mach [13] was used for a couple of years, allowing the students to write real-time C programs that ran as real-time threads in the OS. When RTW was introduced to the course in 2001, the PC-based control experiments were run using MathWorks Real-Time Windows Target (RTWT) [14].

Although interesting control experiments can be implemented using PCs, these experiments do not give students a true embedded-systems experience in the laboratory. Consequently, in 2003 we introduced into the Carnegie Mellon control systems laboratory the MPC555 platform and haptic interface device developed at University of Michigan. This made it possible to give the students a better exposure to state-of-the-practice embedded control systems. We use the Metrowerks CodeWarrior C Development environment. Students initially write C code to become familiar with the basic functionality of the MPC555, and then use the MPC555 blockset in Simulink and generate code using RTW for the MPC555 embedded target [16].

B. Lectures

The emphasis of the Carnegie Mellon course is on the tools for designing and implementing embedded controllers for dynamic systems, particularly using DC motors to control mechanical systems. The lectures fall into four broad categories: control-oriented modeling, controller design, real-time implementation, and applications.

ECE students typically do not have a strong background in modeling mechanical systems, and although they have studied frequency domain concepts extensively in the context of circuits and signal processing, they often have not developed an intuition for how these concepts transfer to the understanding of mechanical dynamics. Consequently, about 20% of the lectures focus on the development of control-oriented models of mechanical systems, including the concepts of linearization and system identification. These lectures are very practical with an emphasis on using MATLAB and Simulink to build models and develop intuition about the dynamics of a system. One lecture is devoted to numerical simulation so that students understand the reasons for various tolerance parameters in Simulink and how to be confident that simulation results indeed reflect the real world.

Another 20% of the lectures deal with the details of implementing control systems, complementing the laboratory exercises with a more general perspective on the issues that need to be addressed when realizing a control algorithm using a real-time embedded processor. Since other courses in the curriculum cover microcomputers, programming, and real-time systems in depth, these lectures can focus on the issues that are particularly important in embedded control systems, such as the selection of sampling rates.

Approximately 45% of the lectures are devoted to controller design, covering topics such as root locus analysis, PID design, state feedback, linear-quadratic optimal control, reference trajectory generation, and discrete control logic. As with the lectures on modeling, the emphasis is on the intuition behind the various design methods and how to use MATLAB, Simulink and Stateflow to design and evaluate feedback controllers. Homework assignments reinforce the material in lectures by having students use these tools to explore control system behaviors as parameters vary and as various complexities are introduced, such as saturation effects. Several exercises help students understand the power and limitations of linear models. Advanced control techniques, such as gain scheduling and adaptive control, are surveyed briefly near the end of the course.

The remaining lectures are devoted to surveys of applications. Early lectures introduce applications directly related to the laboratory exercises, such as motion control systems and haptic interfaces for steer-by-wire systems. Other applications are described throughout the semester as they arise in the discussion of various control design methodologies and implementation issues.

C. Laboratory Exercises

In the early versions of the Carnegie Mellon course, students learned basic switching control methods, including pulse-width modulation (PWM), using the temperature control apparatus. They also performed system identification to determine the dynamics of the hair-dryer and sensor. With the motor control setup, PID and state feedback control were implemented to control the position of a weight hanging from a pulley on the motor. Students also learned the concept of reference-trajectory generation using this simple setup. In the second half of the course, students designed and implemented their own projects, with the only criterion being that a computer must close a feedback loop to control some physical dynamic system.

In the current version of the embedded control systems course, students first learn about the MPC555 by writing C code to implement basic experiments using the QADC, digital outputs, the LCD display, keypad entry, and serial communication with the PC. They then move to using Simulink and RTW. The DC motor control experiments have been retained to introduce the use of the MPC555 fast quadrature decoder and PWM functions. The final set of experiments use the haptic device from University of Michigan, implementing "virtual worlds" similar to the examples in Labs 5 and 6 in the University of Michigan course.

V. Conclusions

There is a growing demand in industry for engineers who understand how to design and implement embedded control systems. The skill set for developing embedded control systems draws on domains that have traditionally been taught in different courses. The courses described in

this paper attempt to address the need for engineers that have a unified understanding of these domains.

The biggest challenge in an embedded control systems course is to strike a balance between teaching control systems principles and embedded systems principles. The structure of the curriculum influences what topics are emphasized in a particular course. At the University of Michigan, the embedded control course includes more material on microprocessor programming and real-time systems concepts than the course a Carnegie Mellon. Since these topics are covered thoroughly in other courses at Carnegie Mellon, more emphasis is put on modeling and control systems design, but the focus remains on embedded implementation issues when introducing control material.

At both universities, we have found that there is an enormous benefit gained from the model-based approach. Having the students design a system completely in simulation gives them an intuition about how the system behaves before dealing with all the details of implementing the system. It is much easier for students to understand the system from a "theoretical" perspective in simulation. Implementing the same system as a working embedded control system then creates a strong a connection between theory and practice; a connection students usually wouldn't make if they jumped immediately into the implementation. Of course, implementing systems in the laboratory gives students an appreciation of "real-world" issues that they never gain from just running simulations. This connection between theory and practice is an invaluable experience facilitated by the technology of model-based autocode generation.

REFERENCES

[1] www.freescale.com
[2] www.axman.com
[3] www.windriver.com
[4] R. B. Gillespie, M. B. Hoffman, and J. S. Freudenberg. Haptic interface for hands-on instruction in system dynamics and embedded control. In *Proc. of the 11th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pp. 410–415. IEEE Computer Society, March 2003.
[5] www.mathworks.com
[6] www.mototron.com
[7] P. G. Griffiths and R. B. Gillespie. A Driving Simulator forTeaching Embedded Automotive Control Applications. In *Proc. of the 2005 American Control Conference*, Portland OR, June 2005.
[8] J. Grason and D. Siewiorek. .Teaching with a hierarchically structured digital systems laboratory. *IEEE Computer*, Dec. 1975, pp. 73-81.
[9] P. Koopman, H. Choset, R. Gandhi, B. H. Krogh, D. Marculescu, P. Narasimhan, J. M. Paul, R. Rajkumar, D. Siewiorek, A. Smailagic, P. Steenkiste, D. E. Thomas, C. Wang. Undergraduate Embedded System Education at Carnegie Mellon. *ACM Transactions on Embedded Computer Systems: Special Issue on Embedded Systems Education*, 2005, to appear.
[10] S. W. Director, P. K. Khosla, R. A. Rohrer, R. A. Rutenbar. Reengineering the curriculum: design and analysis of a new undergraduate Electrical and Computer Engineering degree at Carnegie Mellon University. *Proceedings of the IEEE*, Vol. 83, No. 9, Sept.1995, pp. 1246-1269.
[11] www.agilent.com
[12] www.microchip.com
[13] info.isl.ntt.co.jp/rtmach
[14] www.mathworks.com/products/rtwt/
[15] www.metrowerks.com
[16] www.mathworks.com/products/target_mpc555/