Int. J. Engng Ed. Vol. 19, No. 4, pp. 581–585, 2003
Printed in Great Britain.

0949-149X/91 $3.00+0.00

Teaching Embedded Programming Concepts to Mechanical Engineering Students*

J. EDWARD CARRYER
Design Division of Mechanical Engineering, Stanford University, Stanford, CA 94305, USA.
E-mail: carryer@cdr.stanford.edu

While the prerequisites to mechatronics courses often include a programming course, students are rarely prepared to deal with writing software for an embedded microcontroller. In typical introductory programming courses students write programs to run on relatively large computers with no mechanism for direct interaction with the hardware or real world. This paper describes the elements of Stanford's undergraduate mechatronics courses that are used to introduce students to programming on an embedded microcontroller directly connected to a simple autonomous mobile platform. The philosophy behind the approaches taken, the content of the lectures preceding the laboratory assignment, the assignment itself, the software framework provided as well as the physical platform are discussed.

INTRODUCTION

REVIEW OF SOFTWARE programs written by mechanical engineering students in the graduate mechatronics courses at Stanford indicated that these students were not well prepared to program embedded systems. While the prerequisite programming courses had been successful at teaching students the basic syntax and flow control structures of programming, they were not successful at teaching students to structure complex programs. This was especially true when the programs were, as most embedded programs are, required to respond to a number of asynchronous inputs relating to real-world events.

In 1994, the design of a new undergraduate Mechatronics course prompted the development of a sequence of lectures and a lab assignment coupled with a software framework to introduce event-driven programming constructs to undergraduate students. The design, evolution and implementation of these lectures and lab assignment along with the supporting infrastructure are the topic of this paper.

Background on the course structure

The undergraduate course, 'Introduction to Mechatronics' that is based on the approach described here was offered for the first time in the spring of 1994. It is a ten week long course that derives its hands-on laboratory orientation from the experiences of the graduate mechatronics course sequence that has been offered since 1978. The course consists of four hours of formal lectures a week, four laboratory assignments and a final four-week long open-ended team design project. The material presented here occupies a week's worth of lectures and one of the laboratory assignments.

The approach that has been taken to allow undergraduates to gain experience in systems integration is to give them tools that enable them to work at a relatively high level of abstraction. This has been facilitated by the development of both software and hardware function modules. By using these higher level tools, the students are able to concentrate on solving the larger problem rather than spending all of their time learning the details necessary to design every sub-system from scratch. This is consistent with the trends in the semiconductor and software industries, which have focused on supplying larger scale problem solutions and emphasizing re-usable software.

The primary goal for the lectures described in this paper is to introduce students to the concepts associated with structuring software for systems that must interact with the real world. The laboratory exercise provides them with an experience in which they can put these ideas into practice. A secondary goal is to motivate students for the balance of the course by providing them with an exercise that will not only be pedagogically sound but also feel like it is fun.

HOW THE MATERIAL IS INTRODUCED

The lecture content

The lecture material to introduce students to the embedded programming environment is delivered in two 110-minute lectures prior to the laboratory assignment. The material begins with a review of

* Accepted 5 June 2003.

the relationships between binary & hexadecimal notation. This is used as the introduction to the C bitwise operators and how to use them to isolate bits in a byte for testing and manipulation. This is a technique that is not typically taught in the introductory programming courses, but is essential in embedded applications where individual bits on input and output ports must be tested and manipulated. The lecture continues with an introduction to microcomputers, and the different types of input/output devices on typical microcontrollers. The following lecture focuses on software structures for event-driven programming. This includes the definition of an event, the concept of non-blocking code and an introduction to state machines as a design and implementation tool (Auslander *et al*., 1995). The state machine structure is introduced as a tool for structuring code and also as a tool for debugging at the design stage. This lecture is a somewhat atypical lecture in that it includes a group exercise.

In the group exercise, the class is broken up into small groups to work on a state machine design problem. The problem is to design the controller for a hypothetical microwave oven. The students are given a drawing of the controls of the oven and a description the desired operation. They are asked to develop a state diagram to describe the details of the operation of the oven. The description that they are given is intentionally vague and incomplete. As the questions arise about how a particular feature should work, or what is meant by a portion of the description, the instructor uses the opportunity to not only answer the question but to emphasize that the process of creating the diagram is helping to bring out the parts of the specification that are either not clearly stated or omitted completely. The students take about 15 minutes in the groups to work up their state machine designs.

After the groups have worked up their designs, they are asked to capture them on a transparency. At this point several groups are asked to volunteer to share their designs for review by the class. The purposes of this part of the exercise are several. First, this is a way to rapidly assess if the students have grasped the mechanics of creating a state diagram and capturing the required information. By doing this immediately after the approach has been introduced, immediate feedback can be given before misconceptions have had a chance to get established. The second thing provided by presenting several groups' solutions is to emphasize that there is no one unique design that describes the operation of something even as simple as a microwave oven. The third purpose of the exercise is to graphically demonstrate the use of the state diagram as a debugging tool at design-time. The instructor and members of the class study the proposed design and assess whether or not the design captures the requirements laid out in the description. This generally leads to questions for the group about how the design would deal with particular aspects of the specifications. In every offering of the course at least one of the student-presented designs has had a fundamental flaw in the design. This offers the opportunity to demonstrate in a very convincing way how taking the time to design the structure of the software can help to find errors long before any code has been written.

The final portion of this lecture is a code review of a simple application written using the concepts discussed in the lecture. The code review is used as an opportunity to emphasize some of the significant issues that students need to deal with in designing an event-driven system. The first of these is the definition of an event and how it differs from a state.

When writing their first event-driven application, many students write event-checking routines that, in actuality, test the state of a sensor. The result is that the event checker reports a continuos stream of 'events'. A true event must defined as a transition in the state of an input. The examples of how to write an event checker demonstrate the use of static local variables to maintain the history necessary to detect a transition.

The other issue that is demonstrated is using hysterisis to deal with analog inputs. If an event is defined as an analog input value crossing some specified threshold value then multiple 'events' will be generated when the analog value switches across the threshold due to noise in the system. It is rarely desirable to treat each of these transitions as a separate event. To avoid this 'event chatter' the students are shown how to implement hysterisis around the threshold.

The students leave this lecture with a homework assignment that will be the first step in working the laboratory assignment.

The laboratory assignment

The concepts introduced in the lectures and homework assignment are solidified in a laboratory assignment. In this assignment the students use the programming concepts and course-supplied software tools to program an autonomous vehicle. The vehicle is referred to as a 'cockroach' for the behavior that is programmed. The vehicle can drive forward and backward, turn left and right, sense bumping into objects at each corner and sense the ambient light level. The students are asked, as a minimum, to program a behavior that mimics its namesake: drive around in the light, deal with obstacles and stop when darkness is found. The exercise is designed to not only give the students a chance to employ the concepts from lecture but also to give them their first exposure to programming that produces an output that is something more than dots on a video display. The exercise was designed to be motivational and is positioned at the beginning of the quarter to get students 'hooked' on the material.

The students are given ten days from the time

the assignment is handed out until the results will be evaluated. Each student will design and implement a solution to the assignment, though the students are required to work in groups to review their individual state machine designs before implementation. The culmination of this assignment is a mini-celebration where the whole class gets to see the behaviors implemented by their classmates.

THE INFRASTRUCTURE

Software tools and evolution

To allow students to quickly apply the structural concepts that have been presented in the lectures they are provided with a set of software libraries that ease implementation. These libraries provide for an event-driven programming framework, analog to digital (A/D) converter library, motor drive and timing functions.

The 'events and services' programming library supports the event-driven paradigm by providing a round-robin scheduler for calling event-checking routines and an automated method for calling an associated response, or 'service' routine. To use the library, the programmer makes a single call to a master initialization routine followed by any number of calls to a function that associates event-checking functions and response functions. The typical main( ) takes a form like that shown in the listing below:

```
SES_Init(SES_ROUND_ROBIN, SES_NO_UPDATE);

SES_Register(getKey,putKey);
SES_Register(TestForLightOn,RespForLightOn);
SES_Register(TestForLightOff,RespForLightOff);
SES_Register(TimeOut, TimeOutResp);

while (1)
  SES_HandleEvents( );
```

Using this library, the students focus their programming efforts on identifying what the relevant events are, how they should be tested for and what to do in response to those events. The `SES_HandleEvents( )` function efficiently handles the process of calling the event checkers to test for events and then calling the associated service function if the event checker reports that an event occurred.

The A/D converter interface library is quite simple consisting of only two functions: `AD_Init( )` and `AD_Read( )`. It is worth noting that the greater flexibility of a larger A/D library provided later in the quarter is rarely used by the students.

The motor drive library provides control over the pulse width modulated (PWM) outputs that drive and steer the platform. The students were provided with a set of functions to separately control the left and right drive motors on the third generation platform. The underlying PWM functions are implemented using interrupt driven routines and the 68HC11 timer output compare system. As such, their operation is transparent to the students.

The last significant library provided is an elapsed timer library. The timer library implements eight virtual timers. Each of these timers can be started while specifying a time-out period and tested for expiration. This makes it easy for students to start a timer and use the ability to test for expiration as the event checker to create a new event when the timer expires. The result of adding this library has been a much greater use of time-based behavior in the code written for this lab.

Electronics platform

The students implement their software on an embedded controller board based on the Motorola MC68HC11. This is an 8-bit processor that has sufficient computing power to facilitate programming in the C language. Programming in a high level language is an essential feature for a one-quarter course and C is the de-facto standard implementation language for embedded systems. For this and all other programming in the course, the students work in an integrated development environment running on a PC platform (Man, 1999).

The latest generation of 'Cockroaches' is based on a commercial single board computer, the MicroCore-11 (Barnes, 1999). The version of the MicroCore-11 used provides 8K bytes of EEPROM storage and 32K bytes of RAM. The EEPROM is programmed with a customized version of the BUFFALO debugging monitor. Student programs are downloaded into the 32K of RAM, providing a large program space and quick downloading of code.

To provide the circuits necessary to drive the motors, as well as the interface to the light and mechanical sensors on the vehicle, a supplemental interface board was designed and built. The contents of this board are more fully discussed in the section on the mechanical platform.

Mechanical platform

The mechanical platform for the 'cockroach' is currently in the third generation. While the first two generations used commercial radio control (RC) model cars as the vehicle base, the platform was recently redesigned to make it more robust and base it on more easily available components. One of the goals for this third generation design (Fig. 1) was that it be reproducible at other institutions.

To the greatest degree possible it was based on easily available commercial parts. To this end, the RC car platform was abandoned for a simple acrylic platform driven by DC gear-motors and riding on roller-blade wheels. The platform is in two parts. The inner chassis carries the drive motors and a standard 8.4 V RC battery pack on the bottom side (Fig. 2), and the single board

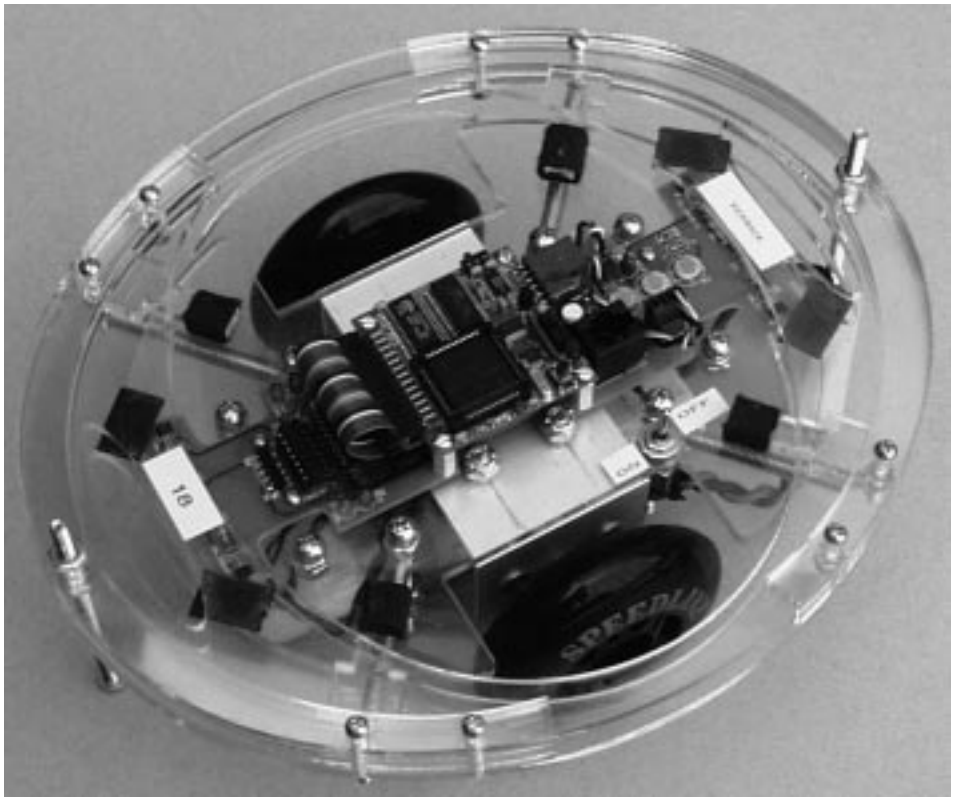

Fig. 1.

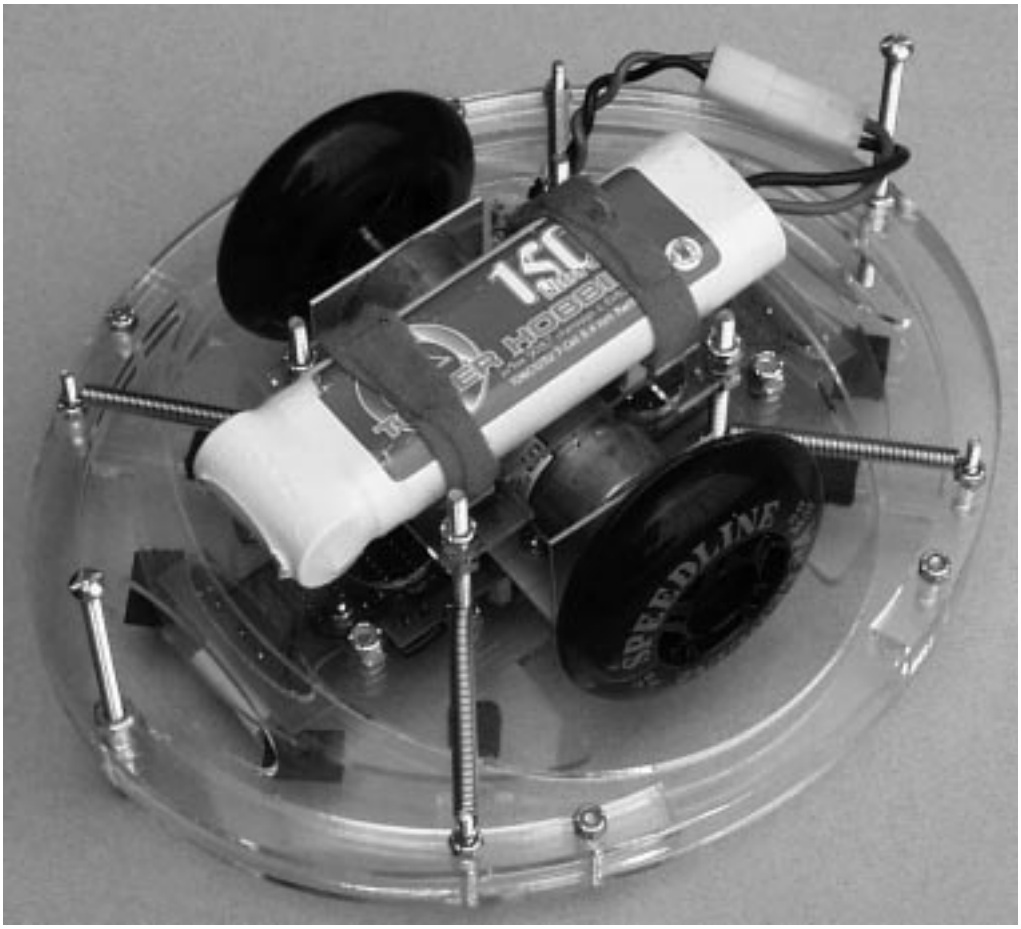

Fig. 2.

computer (SBC) and a custom circuit board that provides the communications connection, motor driver and sensor electronics on the top. The light sensor uses the same three-photo-resistor configuration that was introduced in the second generation vehicles. The bump sensors in the new vehicle are based on Hall-effect switches and magnets rather than the optical system used in the second-generation vehicle.

The bump sensor system used in this vehicle is a strong departure from the prior generations. To avoid problems with jamming, the bumpers are minimally constrained. The outer frame that is the bumper is free to move in-plane around the chassis. It is held in place by four springs that determine its static position. Therefore, the bumper structure is free to move in response to contact at any point around the perimeter. This allows it to respond to contact with fixed objects that are more or less in the direction of travel as well as contact with other vehicles that may come from any direction. The arrangement has proved to be largely immune to jamming. To detect motion of the bumper, four pieces of flexible magnet strip are glued in place near the four Hall-effect sensors at the corners of the top circuit board. A complete documentation package for the vehicle, its electronics and the software libraries is available at http://spdl.stanford.edu/Cockroach/

SUMMARY OF EXPERIENCES

The laboratory assignment presented here has been a part of ME118 since its initial offering in the spring of 1994. Since that time, the course has been offered more than a dozen more times, each including a version of the assignment. The same basic approach has also recently been applied to a new mechatronics course targeted to EE and CS majors.

From a pedagogical standpoint, the assignment has been very successful. The students report finding the experience of programming the mobile platforms to be very motivational and just plain fun. This is despite the fact that they generally spend a great deal of time getting their programs running. As the lectures and supporting libraries have grown to better support the event-driven paradigm, the use of this paradigm for the students' later projects has grown.

Acknowledgments—Tom Kenny has co-taught the ME course with me since its first offering and contributed many good ideas to its development. More recently, Matt Ohline has come on-board as a lecturer with the course and coordinated the design and build of the 'roaches'. Jose Aguilar did the conceptual and detailed mechanical design for the third generation vehicles. Matt Ohline designed the top circuit board. Finally, the initial development of this course at Stanford was funded through the Synthesis Coalition of the National Science Foundation.

REFERENCES

1. D. M. Auslander, A. Huang, M. Lemkin, A design and implementation methodology for real time control of mechanical systems, *Mechatronics*, **5**(7), 1995, pp. 811–832.
2. C. Barnes, *Using Your MicroCore-11*, Technological Arts, Toronto, Ontario, Canada M5R 1E9, (1999)
3. R. F. Mann, *Imagecraft Compiler Users Manual*, Imagecraft, Palo Alto, CA 94303 (1999).

J. Edward Carryer began his academic career by graduating from the Illinois Institute of Technology as a member of the first class of the Education and Experience in Engineering program (E cubed). After a short stint designing water treatment systems for power plants, he returned to school at the University of Wisconsin, Madison to earn a masters degree in bio-medical engineering. That was followed by 8 years in the automotive industry, working on turbo-charged engines. In 1992, he earned his Ph.D. in Mechanical Engineering from Stanford University. Since that time he has been teaching and developing mechatronics curriculum for both graduate and undergraduate students.