Multi-player Soccer and Wireless Embedded Systems

Gaetano Borriello, Carl Hartung, Bruce Hemingway, Karl Koscher, Brian Mayton
Department of Computer Science & Engineering
Box 352350
University of Washington
Seattle, WA 98195-2350 [USA]
+1.206.543.1695

{gaetano, chartung, bruceh, supersat, bmayton}@cs.washington.edu

ABSTRACT

Embedded systems are increasingly becoming connected through wireless networking. These devices now form the basis of many of today's consumer products including cell phones and video game controllers. In the "Software for Embedded Systems" class in the Department of Computer Science & Engineering at the University of Washington, we used the design of a multi-player video game as motivation for the principal concepts in wireless embedded systems. Each student in the class designed an accelerometer-based game controller and then, the class as a whole, developed a multi-player video game that allowed 28 players (the number of students in the course) to play simultaneously. In this paper, we first describe the context of the course and its goals followed by the hardware/software platform we used to realize the game controller. We then detail the pedagogical approach we used to collectively design the video game (loosely based on soccer) and conclude with the lessons learned from this group design experience and how we would enhance the project and course in the future.

Categories and Subject Descriptors

C.3 SPECIAL-PURPOSE AND APPLICATION-BASED SYSTEMS, Real-time and embedded systems.

B.4 INPUT/OUTPUT AND DATA COMMUNICATIONS, B.4.1 Data Communications Devices, B.4.2 Input/Output Devices, B.4.5 Reliability, Testing, and Fault-Tolerance;

General Terms

Design, Reliability, Experimentation.

Keywords

Multi-player video games, sensor-based input devices, accelerometers, pedagogy, group design.

1. INTRODUCTION

Wirelessly connected embedded systems are increasingly common in today's consumer electronics ranging from wireless keyboards/mice to music players to running shoes. These devices often involve a simple point-to-point connection, often between a simple sensing device (e.g., mouse) and a base station that is attached to a more capable computing device (e.g., a PC). On the horizon, there are even more products on the way that will extend this model to multi-point connections. These include home automation systems built around the new Zigbee standards [1] as well as various applications of multi-hop sensor networks [2].

Computer Engineering curricula have traditionally included the interfacing of sensing and actuation devices to microcontrollers but have not emphasized wireless communication. However, wireless communication is quickly becoming the preferred method of interconnecting ubiquitous electronic devices. The elimination of cables and the ports at which to connect them helps greatly in improving the form factor of consumer products and permits automatic transfers of data that lower the cognitive burden on the user. For example, a cell phone that automatically connects to the speakers/microphone in a vehicle for hands-free operation is much more useful, convenient, and safe than one that the user must remember to connect manually.

Wireless communication between embedded devices has been part of the Computer Engineering curriculum at the University of Washington for several years [3, 4]. We began with simple "virtual wire" radio for point-to-point connection and then moved on to sensor networking platforms such as the UCB motes. In our courses, we always strive to provide projects that motivate students. The last few years, we have used the creation of a "flock of birds" as the large project in our "Software for Embedded Systems" course [5]. Students used a UCB mote platform [6], augmented with an additional board of our own design that gave each node sound synthesis capabilities, to develop an autonomous "bird" that could sing a variety of bird songs. The birds communicated with their neighbors to coordinate their song choice through a simple algorithm that gave the illusion of flock behavior including propagating a song from bird to bird and occasionally initiating new songs. The "flock", with a "bird" contributed by each student, was demonstrated in our building's atrium at the end of every quarter to entertain our community.

This past year, we undertook the task of updating the platform we use in this course to the Intel iMote2 which runs an embedded Linux operating system [7]. Along with the transition, we also undertook the creation of a new project. The remainder of this

paper describes the curricular context of the course, the platform we created, and the project we collectively designed in the class to demonstrate key concepts for wireless embedded systems. The paper concludes with the lessons we learned and how we are likely to extend and improve the project in the future.

2. COURSE CONTEXT AND GOALS

CSE466, Software for Embedded Systems, is a required course in our Computer Engineering curriculum. It has data structures and computer architecture as prerequisites. Students also complete courses in operating systems and networking but these are not in a prerequisite chain with CSE466. Students are mostly seniors with some juniors. This is the course where students are first introduced to microcontrollers, detailed timing of I/O operations, basics of device drivers, and interfacing techniques [3].

In the first part of the course, students design their own USB device from basic ICs wired together on a breadboard. They connect some simple sensors to a microcontroller and write the interfacing firmware. Since they do not have any run-time support, they must access the microcontroller configuration registers directly and use the timers to schedule their sensor readings and communications. They use a USB interface chip (connected via an SPI bus) to create a connection between their microcontroller and a PC and then write a simple USB device driver for the PC so that their application can communicate with the microcontroller and sensor [8].

In this edition of the course, we used a light sensor to construct a heart-rate monitor. Light readings from a person's finger vary as blood moves through his or her capillaries. Readings from the light sensor are sent to the PC where peaks in the signal are detected by a C program. From the spacing of the peaks, the PC application determines the heart rate and sends that value back to the microcontroller, which then blinks an LED at the same rate. The PC application also displays the numeric value of the heart rate in a window.

The second half of the course introduces students to wireless communication and moves them to a more capable pre-fabricated platform that includes runtime support. We do this to ensure that larger projects are built on reliable hardware and firmware primitives, but only after students have built their own device and have understood the underpinnings of microcontroller-based systems.

At this point, the course focuses on wireless networking protocols including common standards such as Bluetooth, Zigbee (802.15.4) and WiFi. Lectures describe the MAC layers and protocol stacks of each with their advantages and disadvantages. Application examples include consumer electronics (e.g., Bluetooth headsets for cell phones) and multi-hop sensor networks (e.g., UCB motes and their data collection and aggregation functions).

The course concludes with a large design project that connects sensors over a wireless link to a PC application. In this latest edition of CSE466, we decided to develop a multi-player video game using controllers modeled on the, then just introduced, Nintendo Wii video game controller. Each student designed their own Wii-like accelerometer-based 2-D game controller [9]. We added an LCD screen so that game state could be displayed on the user's controller during the game.

3. HARDWARE/SOFTWARE PLATFORM

The platform we used is based on Intel's iMote2 [7]. The iMote2 was designed as a highly capable wireless sensor network node. It consists of a PXA271 processor running at adjustable speeds from 13 to 416MHz. It uses embedded Linux for its run-time support and comes with 64MB of memory (half RAM, half Flash) within the same package as the processor. In addition, the base board includes an 802.15.4 radio based on the ChipCon2420 for a bandwidth of up to 250Kb/sec in the 2.4GHz unregulated band. It is commercially available from Crossbow Technology, Inc. [10].

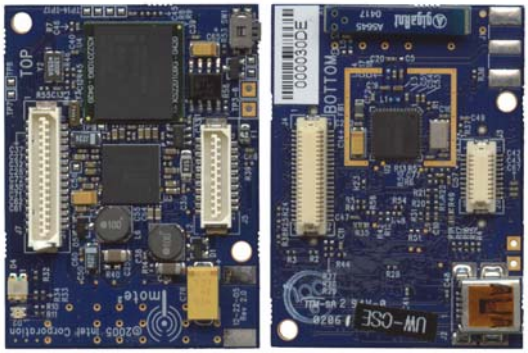

Figure 1. The top and bottom of the Intel iMote2. The top, on the left, has the PXA271 and memory in the large chip at the top center. The bottom, on the right, has the 802.15.4 ChipCon2420 radio (within the square in the center). The long connectors on both sides connect to the other two boards.

The iMote2 has expansion connectors for attaching sensing and actuation hardware. On one side, we attached Intel's basic sensor board (BSB) which includes: a 3-axis accelerometer, two temperature sensors, a humidity sensor, a light sensor, and a 4-channel A/D for further additions. On the other side, we designed our own board to provide some actuation. We included a cell phone-size color LCD screen as well as sound generation capabilities, a speaker, microphone, and audio jacks. This board also includes a cell phone camera, jog dial, USB host port, barometer, and a heart rate sensor and is the form factor of a large cell phone. While not all of the board's capabilities were used, we developed the board to be flexible enough to accommodate a variety of projects (such as a video phone, music player, or an enhanced version of the "Flock") in the future.

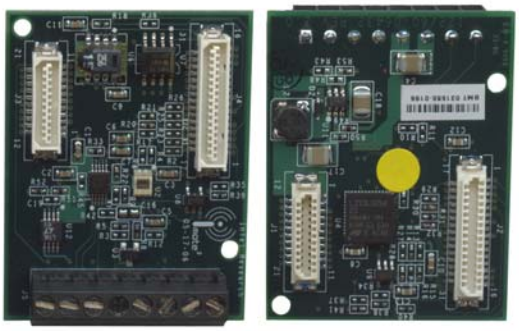

Figure 2. The top and bottom of the Intel iMote2 basic sensor board. The 3-D accelerometer sensor is just below the yellow dot on the right.

The iMote2 software is all based on Linux. We provided the students with device drivers for most of the peripherals they used. The one exception is that they had to complete the accelerometer driver (we provided some skeleton code) and write the utilities to turn readings into 2-D movements. We also provided a very simple packet sending and receiving API for the 802.15.4 radio using a MAC layer and packet format borrowed from the UCB MicaZ motes [6]. The details of the packet payload were derived as part of the collective design experience that developed the precise specifications for the project.

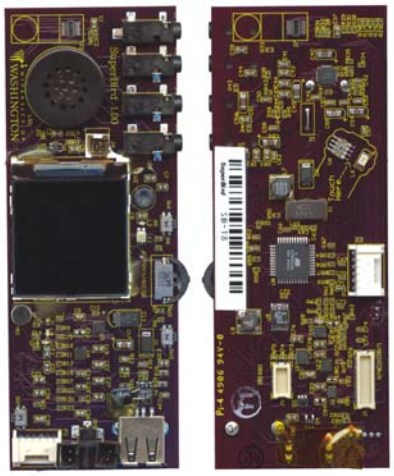

Figure 3. The top and bottom of the Intel iMote2 "SuperBird" board containing LCD, microphone, speaker, jog dial (near center), USB connector (left, on bottom right), and heart rate sensor (diagonally arranged area on right).

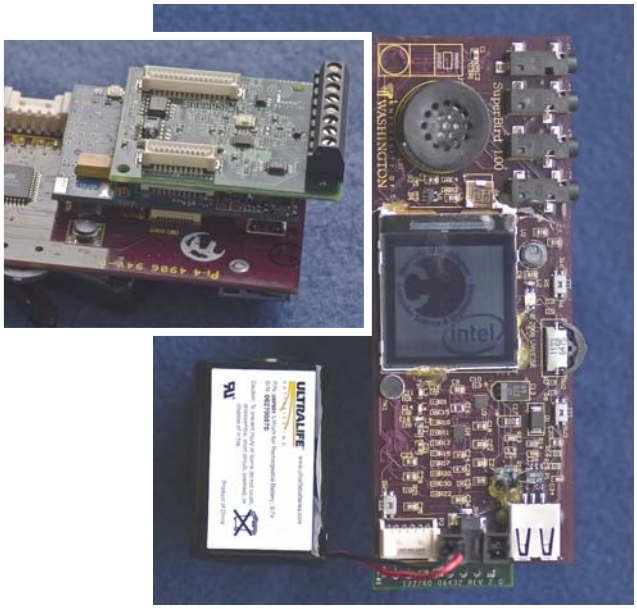

Figure 4. The three-board set stacked together with their rechargeable battery. The side view in the inset on the left shows the other two boards that lie below the large board in the larger picture.

4. DESIGNING THE GAME

Although we decided on the idea of a multi-player video game before the course began, the precise details of the game and its data packets were developed as the course progressed. We decided on utilizing controllers that could move a player in two dimensions. Soccer quickly emerged as a game that could be varied to accommodate our requirements for the project, namely, that it should involve every student simultaneously and require only two-dimensional control of each player. In addition, we wanted to have some collaborative elements between players that would spur real team play.

The variation on soccer that we developed does not use a ball! This greatly simplifies the dynamics that have to be modeled to render the game. Players score by moving through the goal rather than kicking the ball through it. When players collide, the game controller randomly moves them to their end of the playing field. The collaborative element is introduced when two players of the same team make contact. In that case, they are combined into a single larger player that occupies twice the area. In general, any number of players on the same team can combine into a single larger player whose area is proportional to the numbers of players combined to form it. When "merged" players from opposing teams collide, they are disaggregated and the individual players assigned to random locations. When a smaller player (n) collides with a larger player (m), the number of players in the smaller player is subtracted from the larger player to yield a single smaller player (of size n-m) and 2n individual players that are randomly repositioned on the field. In this manner, a defense can "nibble" away at a larger player heading for the goal.

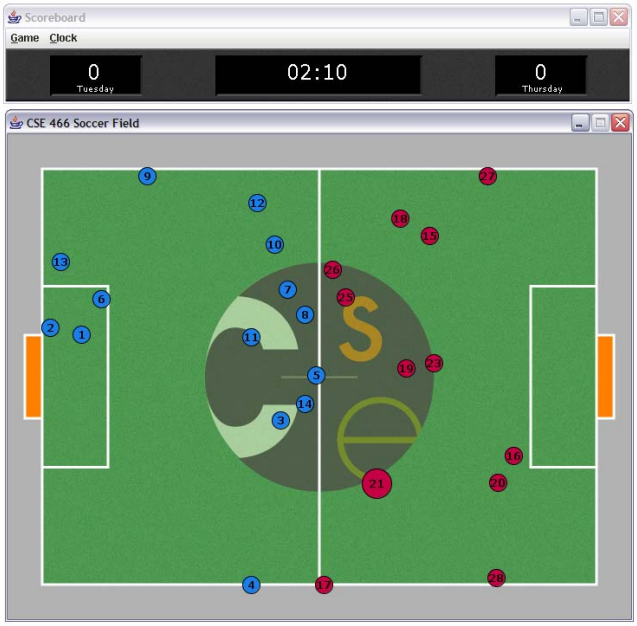

Figure 5. The game display showing 28 players with three red players merged into a larger player (#21 near the center).

There are many parameter values that need to be determined for making such a game visually pleasing and exciting for the participants. These include: the size of the players and the field, the speed at which the players can move, how much faster merged players can move, what defines a collision, etc. In addition, we

had to develop a game protocol to coordinate the information that needs to flow from each player's controller to the game coordinator (which updates the playing field), and back to each player so that a user can have appropriate feedback about what just happened to their player. We used the LCD screen and speaker to provide this feedback.

The game was developed in steps. We began with the basic player controller, that is, the mapping of values from the accelerometer to an X-Y velocity vector. One of the authors, Brian Mayton, who was also one of the students in the course, developed the game controller and display so that students were able to quickly observe the motions on the playing field. The experiences in the laboratory were reported back in the classroom where we collectively made decisions about what some of the parameter values should be.

Figure 6. Examples of different messages from the game controller to individual players indicating when a player merged with another player, when they scored a goal, or were teleported to a random position on the field.

Some of the most complex decisions involved merged players. We quickly decided that they should be able to move faster than singleton players in order to provide an incentive to merging. However, we also decided that the motions of the individual players that made up the merged player needed to be coordinated in order to have this advantage. Thus, the motions provided by the individual players were first averaged (so that motions of opposite sign would work against each other) and then scaled by a factor proportional to the square root of the cardinality of the merged player.

Finally, we had to devise a communication protocol – on top of the basic MAC of the 802.15.4 radio – to handle the communication between players and the game coordinator. To ensure that we included some inter-player communication, we decided to let the "captain" of the merged players (the player with the highest number) collect all the moves of the constituent players and report that result to the game coordinator. The scheme is basically round-robin. The game coordinator polls the first player for its move and waits for a response before proceeding to the next player. If a player is too slow in responding, its movement is ignored for that round. This provided some timing constraints on our students' implementations to quickly handle packets from the game coordinator. This forced a degree of parallelism in the implementation of each controller that would not have otherwise been required. When the game coordinator polls a merged player, its response is broadcast back to both the game controller and the merged player captain (we make the highest numbered player in each merged player the captain for this reason). The game coordinator ignores a response from a merged player and waits for the aggregated response from the merged player captain. At the end of a round, the game controller updates the field. With 28 players we were able to do a complete round at the rate of about 3 to 5Hz. With about 500 bits/packet, 28 players, and 2 packets/player (one to and one from each player), we used about 20% of the bandwidth available on our radio. The remainder of the time was used to allow adequate timeout intervals when awaiting a return packet from a player's controller in response to the polling packet issued by the game coordinator. The graphics in the game coordinator used these 3-5Hz samples to smoothly animate the motion.

An important exercise in determining the contents of each packet was to ensure that the game could recover from inevitable packet losses. This was a very educational exercise for the students as they quickly began to comprehend how networking protocols grow in complexity as design requirements add more constraints. To compensate for packet loss, we decided to send game status information redundantly in each packet to ensure that each player's display reflects an up-to-date picture of what is happening in the game.

5. LESSONS LEARNED AND NEXT STEPS

This was an ambitious exercise as we developed an entirely new project along with updating the hardware/software platforms used in the course. Due to time constraints we were forced to leave the detailed design decisions for the project unbound when the course started. This turned out be an excellent approach as it led to much deeper student learning. We were able to collectively design the game through initial brainstorming, experimentation and data collection through lab assignments, and then further refinements as a group leading to a highly engaging final demonstration: a 30-minute soccer match between the two sections of the class (Tuesday and Thursday) on the final day of the course. The match was held in our building's atrium on a large projection screen during the lunch hour to guarantee a large audience and increase awareness of our efforts within our department. A video of the match can be found at the following web address: http://www.cs.washington.edu/homes/gaetano/CSE466.wmv [11].

As is usually the case, testing turned out be an important issue that we need to resolve for future editions of the course. Although we created a test environment that allowed students to check out the basic functionality of their player controllers, we did not pay enough attention to timing issues. For example, it would have been much more useful to provide round-trip timing for packet exchanges with the game controller. Several of the students ended up with implementations that often missed their timing deadlines during the round-robin polling (we know this because we instrumented the game coordinator to keep statistics during the final match).

Wireless congestion also proved to a bit more of an issue than we had anticipated. In retrospect, we should have had the communication between merged players happen on separate channels from the one on which the round-robin polling was occurring. This would require captains to keep track of which players have responded (rather than the game controller), and when to switch channels. The radio API allows us to switch channels on a per packet basis, so this is a feasible approach.

Finally, although there was no cheating during the final match (which we would have discovered through our monitoring of the wireless traffic), the collective design process helped to highlight

the many security issues associated with our protocols. Students were quite imaginative in figuring out ways they could theoretically cheat the system and make players that could disrupt others as well as enhance their own performance. Future editions of the course could consider adding security measures to the protocol. This will be especially important if we consider using multiple channels and thereby make it more difficult to simply monitor the packet transmissions at the game coordinator.

6. ACKNOWLEDGMENTS

Our thanks to the students of the Winter 2007 edition of CSE466 "Software for Embedded Systems" for the patience, perseverance, and enthusiasm in helping us develop a new approach to teaching the course and for a fantastically fun final soccer match. Special thanks to Intel Corporation and its University Relations Program for generously providing the wireless platforms (iMote2s) and sensor boards we used for this course as well as their consistent support of our educational mission. We would also like to thank Roy Want and his team at Intel Research for providing an excellent starting point for the 802.15.4 protocols we used.

7. REFERENCES

[1] IEEE 802.15 WPAN Task Group 4 (TG4): http://www.ieee802.org/15/pub/TG4.html.

[2] Estrin, D., G. Borriello, et. al. *Embedded, Everywhere: A Research Agenda for Networked Systems of Embedded Computers*, Committee on Networked Systems of Embedded Computers, Computer Science and Telecommunications Board , National Research Council, Washington, DC, 2001.

[3] University of Washington, Department of Computer Science & Engineering, CSE466: Software for Embedded Systems: http://www.cs.washington.edu/education/courses/cse466/.

[4] University of Washington, Department of Computer Science & Engineering, CSE477: Computer Engineering Capstone: http://www.cs.washington.edu/education/courses/cse477/.

[5] Hemingway, B., Brunette, W., Anderl,T., and Borriello, G. The Flock: Using Wireless Mote Networks in an Undergraduate Curriculum. IEEE Computer (special issue on Sensor Networks), Vol. 37, No. 8, pp. 72-78, August 2004.

[6] TinyOS Exchange: http://www.tinyos.net/.

[7] Intel iMote2: http://embedded.seattle.intel-research.net/wiki/index.php?title=Intel_Mote_2.

[8] University of Washington, Department of Computer Science & Engineering, CSE466: Software for Embedded Systems: (Winter 2007 edition): http://www.cs.washington.edu/education/courses/cse466/07wi/.

[9] Nintendo Wii: http://wii.com/.

[10] Crossbow Technology, Inc.: http://www.xbow.com/.

[11] University of Washington, Department of Computer Science & Engineering, CSE466: Software for Embedded Systems: (Winter 2007 edition), video of final project: http://www.cs.washington.edu/homes/gaetano/CSE466.wmv.

Figure 7. The big match on the last day of the course. The top three pictures show students playing the game with each holding the player controller they developed. The bottom picture shows the final result when time ran out after 2 15-minute halves: Thursday beat Tuesday by a score of 134-122.