The HallWalker Robot: An Interdisciplinary Design Project

William P. Lovegrove, Timothy S. Owens, Matthew S. Bronkema
Bob Jones University

Abstract

The fall 2000 Bob Jones University capstone design project is presented as a model of a successful interdisciplinary design project. It directly addresses the hardware/software co-design that is an integral part of many modern electronic devices by employing a software team of Computer Science majors and a hardware team of Electrical Engineering majors. In order to facilitate hardware/software co-design, the software team implemented a full simulation of the robot and the target environment. This forced the development of complete and detailed specifications early in the project, and made the standard design process an integral part of the project rather than a mere academic exercise.

Introduction

The goal of this capstone design project was to have the students experience an interdisciplinary hardware/software co-design. The two design teams consisted of three students enrolled in *Cps 420 Software Development* and four students enrolled in *Ele 406 Advanced Microprocessors*. To enable the students to focus on microprocessor and software issues, we gave them a mechanically sound robot to begin with, with fully functioning drive and sensor systems. The robot was a 1980's vintage Heathkit Model ET-18, a limited-capability robot intended for the home hobby market (Figure 1).

In keeping with the *Ele 406* emphasis on embedded systems, we required the students to replace the outdated and under-powered Motorola 6800 microprocessor with a Motorola MC68HC11 microcontroller (Figure 2). The hex keypad and 6-digit display were replaced with a serial port for communication with a personal computer for software development. The Engineering team was assigned the task of installing and interfacing the 68HC11 board, and then of writing all of the device drivers from scratch. To do that, they had to learn the workings of the not-too-well documented drive and sensor systems. The students were given none of the original Heathkit-written code - they had to develop their own drivers from scratch.

The Computer Science team was assigned the task of writing the main control and navigation algorithm. Since the real hardware and drivers were not expected to be working until late in the project, this team was also assigned the task of writing a software simulation of the robot and its environment in order to develop and test navigation algorithms.

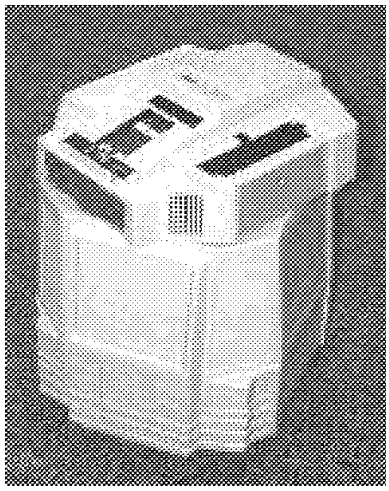

Figure 1. Heathkit Model ET-18 robot.

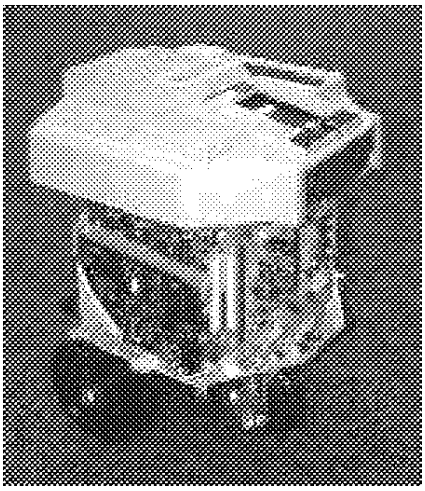

Figure 2. Robot with shell removed to reveal 68HC11 board added.

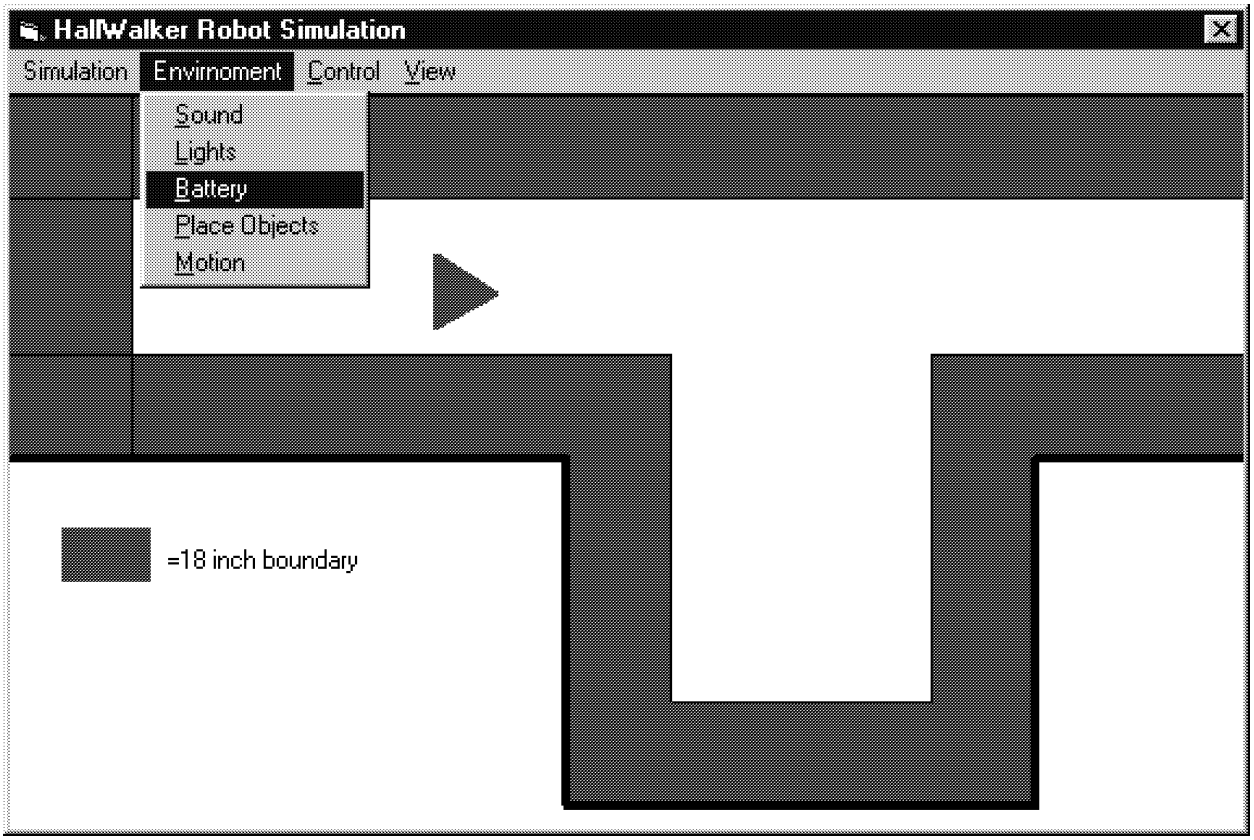

Figure 3. A screen shot of the simulator.

The task assigned to this robot was to wander the halls of the engineering building, avoiding obstacles and navigating until it detected low batteries or darkness. It would then navigate back to its garage and dock for recharging. We intend eventually to give this robot speech recognition and synthesis capability and use it to greet visitors to our building.

Specifications

A central feature of modern design methods is the development of specifications early in the design process. Students on the other hand tend to want to design using an ad-hoc "develop the specifications as we go" approach. One of the primary goals of any capstone design project should be to force the students to go through a full specify/design/build cycle. We found that the nature of this particular project particularly lent itself to this approach. It was absolutely essential that specifications be developed early to enable work on the simulator to begin. Furthermore, the specifications had to be grounded in the realities of the already-existing robot.

In addition, the interdisciplinary nature of this project increased the importance of the specifications. The specifications became the primary vehicle of communicating design information between the two teams. The Computer Science team, with limited knowledge of hardware and interfacing issues, was heavily dependent on the Engineering team for early, detailed, and accurate specifications of how the hardware worked. The Engineering team on the other hand was pressured by the nature of the project to deliver accurate specifications. When the specifications were vague, incomplete, or inaccurate, the resulting discrepancies between actual and simulated robot behavior were obvious and unavoidable.

The key component of the specifications became the driver "API" (Application Programmer Interface) specification. A section of the API from the project web site is included in the Appendix.

Hardware/Software Co-design.

A key feature of this project was the use of hardware/software co-design. With many embedded systems projects, it is desirable to start working on the software before functioning hardware is available. This particular project was chosen because it allowed for a natural co-design project in a student environment.

To facilitate early software design, the Computer Science team was required to build a software simulation of the robot and its environment. One risk of co-design projects on a student level is the difficulty in assessing project difficulty ahead of time. Particularly on the hardware side, a project is needed which is doable within the time limits of the class, yet no so easy that the hardware is done early enough to eliminate the need for the simulator.

The embedded software was written in a combination of ANSI C and 6811 assembly language. The drivers were all written in assembly language for two reasons: 1) The hardware team was

already experienced in assembler but was weak in C, and 2) assembler allowed direct low-level hardware control, the reason many drivers are written in assembler in a wide variety of situations.

An Archimedes C cross-compiler was chosen for development. This compiler runs on an IBM-PC compatible host computer and generates 6811 machine language.

The Archimedes C compiler was clearly intended only for 6811 embedded system development and was not at all suitable for developing the simulator. Thus the software team chose a different programming environment, the Borland *C++ Builder* ™ development system, to write their simulator in. A screen shot representative of their simulator is shown in Figure 3. Note the menu items to implement environmental conditions such as darkness. The display is a crude two-dimensional map of the hallway.

The two teams thus had to deal with a host of portability issues. We required that the main control program run in both environments without modification. Figure 4 illustrates the dual software environments used by the design teams.

Communication Issues

As expected, communication between the two design teams proved to be one of the major challenges for the students. This is in fact a primary purpose of an interdisciplinary design project: to allow the students to experience firsthand the communication problems inherent in interdisciplinary work. The students early in the project complained, “we don’t even speak the same language!” Vocabulary familiar to a Computer Science student (use cases, methods, API) was completely unknown to some of the engineering students in spite of the fact that they had all taken some programming classes. The assumptions made by the programmers (motors instantly turn on and off, sensors have infinite accuracy and precision) were astonishing to the more hardware-experienced engineers.

To address these problems, we took the following steps. First, project-wide meetings were held. We experimented with representatives from each team meeting, with limited success. There is no substitute for a meeting of the entire project teams from both departments.

Second, a project web site was established (Figure 5). Specifications and other project-related documents were posted here. It took some pressure to keep the students from circumventing the web site occasionally and dealing directly with their counterparts in the other department. Nevertheless, we found it essential to insist that everything on the web site be kept current.

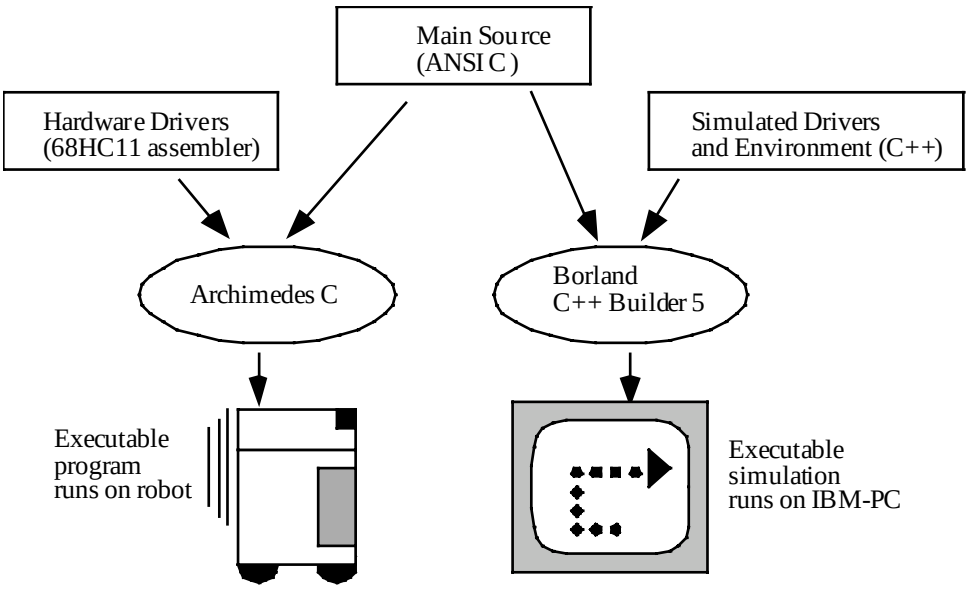

Figure 4. Dual software environments

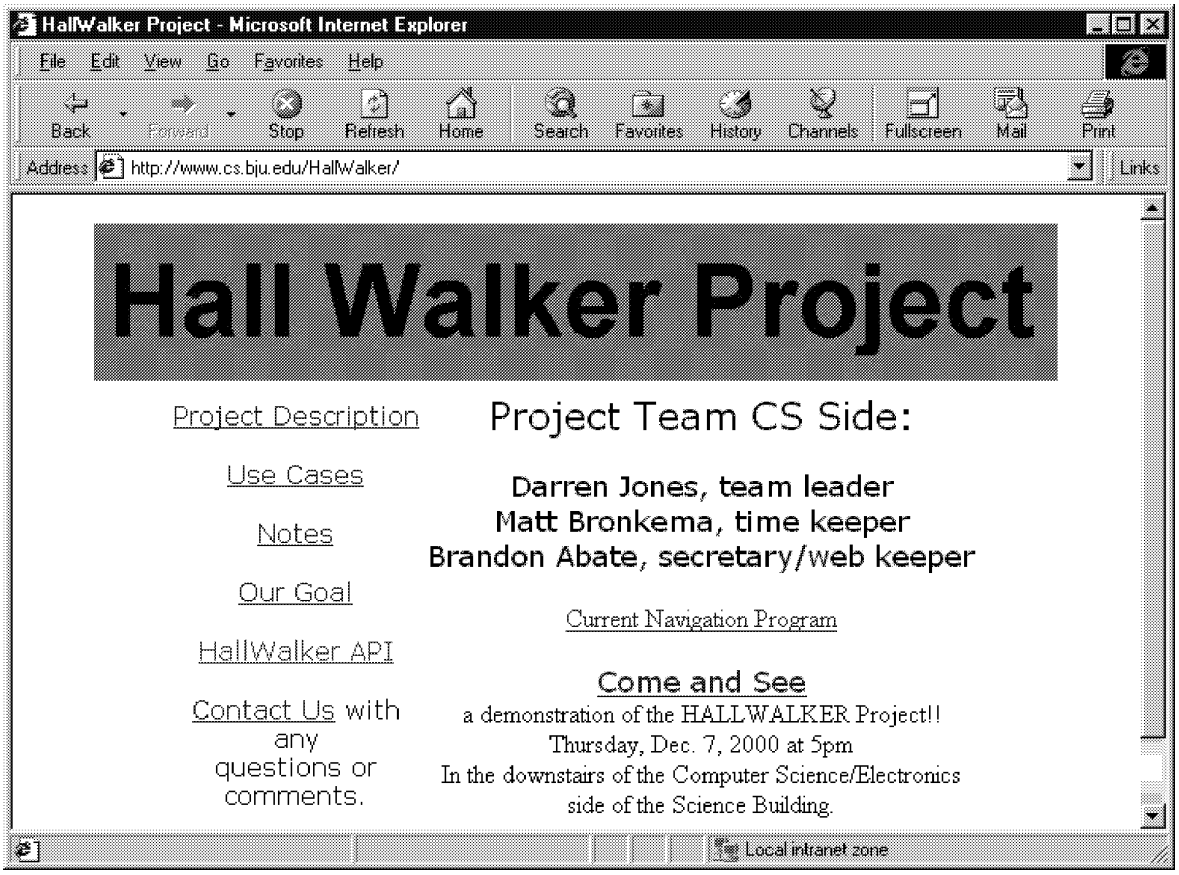

Figure 5. Project web site.

Third, we encouraged all between-meetings discussion to take place via group email, as a means of keeping all team members (and the faculty) informed.

Recommendations

We were extremely pleased with the outcome of this project and hope to implement similar projects in the future. As always, however, there are a number of things we hope to improve on in future iterations. We will list several of them here for the benefit of instructors considering similar projects of their own.

Our milestones for the simulator development were not aggressive enough. When it fell only slightly behind schedule and the hardware advanced ahead of schedule, there was a period of time when algorithm development was being done on the real robot instead of on the simulator. Pragmatically this was effective in getting the project done, but pedagogically it weakened the co-design experience somewhat. Next time we intend to structure the milestones and grading criteria to put greater pressure on early simulator development.

One of the great difficulties in all capstone design projects is matching the difficulty of the project to the ability of the students. As mentioned above, the simulator development fell behind schedule. This happened in part because the programming skills of the software team were weaker than we anticipated. We will give more attention to structuring the simulator requirements to better match the abilities of the students.

Communication between the two departments was hampered by schedule conflicts. The Engineering and Computer Science students were enrolled in separate classes meeting at separate times. Finding time for joint meetings was challenging. We do not feel that the two classes should be combined and meet together. Outside of the project, they have different goals and course content. We do intend, however, to make some scheduling adjustments to facilitate meeting together.

We made ad-hoc and informal use of email in this project, and we feel like better use of email will facilitate future projects. In particular, we intend to mandate that all emails be sent to the entire group – no private emails between individuals. Second, we intend to log and post all emails on the project web site, to maintain a history and provide additional documentation.

A valuable third discipline to add to a project like this would be mechanical engineering. For this project we gave the students a completely built and functioning mechanism to begin with, but adding a mechanical team and designing the robot hardware simultaneously with the electronics and software would be an excellent experience for the students.

Late in the project we acquired a copy of the IGRIP virtual robot simulation software from Delmia Corp (www.delmia.com). This powerful software is capable of highly accurate modeling, both kinematic and dynamic, and 3D-rendering of complex robots and environments. It arrived too late in our project to replace the student-written simulator. In fact, we feel there is great value for the Computer Science students in the experience of writing a simulation program themselves. Nevertheless, IGRIP is a powerful and useful tool that in theory could replace or supplement the student simulator. Among other features it has a C-language interface, allowing us to drive the IGRIP simulated robot from a C control program just as the students did in their simulation. We plan in the next iteration of this project to make use of IGRIP, possibly requiring the Engineering students to model the robot in IGRIP. Among other things, it would allow the Computer Science students to visualize the robot and its behavior long before it is physically operating. Figure 6 shows the robot and the hall in which it operated modeled in IGRIP.

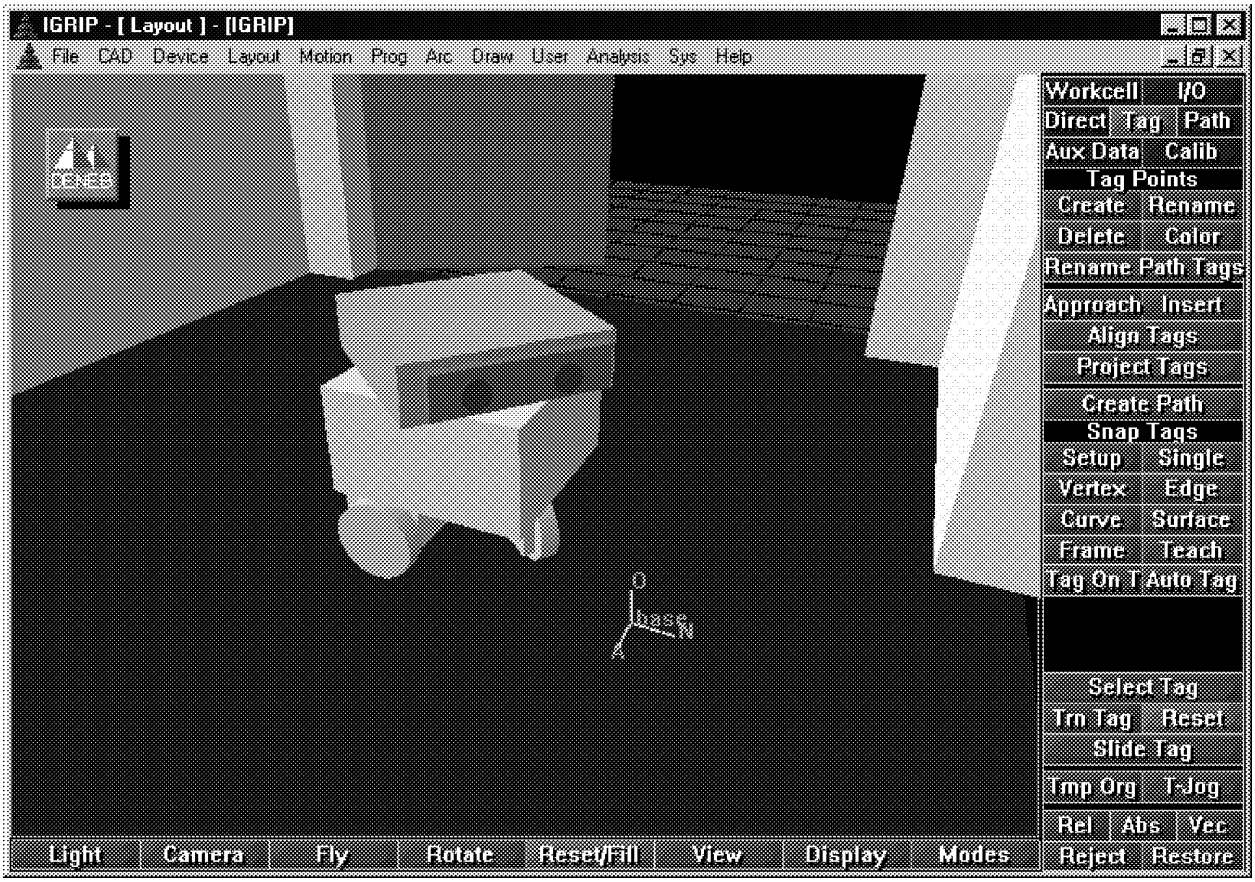

Figure 6. The robot modeled in IGRIP.

Appendix

Below is a sample taken from the actual robot API. Notice the failure to define a "click," an omission that would come back to haunt the design team later. An approximate value was verbally communicated between the two teams, and the exact value was never specified precisely. Note that although the API is defined in C, the actual drivers were written in assembly language.

Distance Traveled (position)

Definition:

unsigned int position(unsigned char ucReset)

Parameters:

In:

unsigned char ucReset = an 8 bit flag to determine whether or not to reset the distance counter.

If ucReset is nonzero (true), then reset the distance counter and return a zero indicating the motor is at rest.
If ucReset is zero (false), then return the current distance traveled by the motor in clicks.

Out:

unsigned int = a 16 bit int indicating the current number of clicks traveled by the drive motor since the last reset.

If the return value is nonzero, then set value of variable is equal to the distance traveled
If the return value is 0 (zero), assume the steering motor was reset to straight forward.

Example:

```
unsigned char ucSpeedDir = 0x10;
unsigned int uiDistanceToTravel = 352;

drive(ucSpeedDir);
while(distance > 0 && distance <= uiDistanceToTravel){
      distance = position(0x00);
}
```

WILLIAM P. LOVEGROVE
William Lovegrove is a professor of Electrical Engineering at Bob Jones University in Greenville, South Carolina, where he directs the senior capstone design projects. He received a B.S. degree in Physics from Bob Jones University in 1984, an M.S. degree in Electrical Engineering from Clemson University in 1986, and a Ph.D. in Computer Engineering from Clemson University in 1990.

TIMOTHY S. OWENS
Timothy S. Owens is a senior Industrial Electronics major at Bob Jones University. He did the majority of the programming of the device drivers for this project.

MATTHEW S. BRONKEMA
Matthew S. Bronkema is a junior Computer Science major at Bob Jones University. He did the majority of the programming of the simulator for this project.