Centralized Server Environment for Educational Robotics

Randolph Voorhies, Christian Siagian, Lior Elazary, and Laurent Itti
Computer Science Department, University of Southern California

***Abstract*— One of the main challenges when creating an undergraduate introduction to robotics course is connecting the theory taught in the lectures with the current practices of research. The primary cause of this difficulty is an inability to find a hardware solution that is powerful enough to run complex cutting-edge algorithms yet inexpensive enough to be purchased by an undergraduate class budget. An ideal system needs to have a gentle learning curve to allow students with minimal background in the field to get a robot up and running. Lastly, a fleet of classroom robots needs to be easy to administrate and maintain given the limited time of a Teaching Assistant. Our approach is to implement a centralized server system. In this system individual robots are inexpensive yet capable of establishing a WiFi link to a main server so that all the compilation and system administration, as well as much of the computationally intensive processing, are done on that server. We find that this solution saves both time and money and provides an effective teaching tool. This paper describes the hardware and software architecture of the system, and example applications implemented by undergraduate students.**

I. INTRODUCTION

For many students, an undergraduate Introduction to Robotics course represents the first taste of what it would be like to work in the field. Invariably, one of the main purposes of such a class is not just to provide a basic knowledge or content of the history and current state of robotics, but at the same time to motivate students and instill in them an excitement for the subject.

A primary means of achieving this important thrust is through a hands-on laboratory experience that engages students to think more deeply about different approaches to solving a variety of problems faced by roboticists. We find that laboratory experiences which lead to "moments of discovery" after a series of trial-and-errors, off-the wall ingenuity, and a bit of luck make all the difference in engaging students and making them want to learn even more. This, we believe, cannot be reproduced by lectures alone, or even a simulator-based lab.

However, like many universities which have offered these type of courses [1], [2], [3], [4], in the past ten years the University of Southern California (USC) [5] has offered a curriculum which has focused more on the mechatronics rather than the computer science aspects of robotics. For example, previous iterations of the class — using a microcontroller-based system — could only implement robotic movement at the level of PWM or PID control and had no facilities to explore more modern probabilistic approaches to motion. Our class was structured in this way only because the university did not possess a suitable hardware platform to run complex algorithms. Since the course's creation nearly ten years ago, the lab curricula has remained fairly static, while robotics research had progressed considerably, thus widening the gap between what is taught in the lab and the state-of-the-art covered in the lectures. Thus, while lectures could easily be updated to follow the latest advances in machine vision, probabilistic learning, simultaneous localization and mapping, group robotics, sensor networks, and neuromorphic algorithms, none of these more advanced and computationally-intensive techniques could be tested in the lab. In essence, the lectures were keeping up with the forefront of research but the lab was stuck at the hobbyist level.

Prior to our proposed solution, we used LEGO as hardware building blocks and the relatively low-cost of $250 ($ denotes US dollars hereafter) although computationally limited Handyboard featuring a Motorola HC11 8-bit microprocessor with 32KB RAM [6] as a computing platform in the class. Despite the computational limitations of the Handyboard, new students can learn most of the nuances of programming them within a three-hour laboratory session through the use of easily understandable Interactive-C software. Furthermore, these boards are fairly lightweight, low-power and small in size, which allows for a smaller, and thus, cheaper robot locomotion system.

A very different approach which is taken by many universities [7], [8], [9], [10], [11] is to utilize much more capable, though expensive (in the range of $3000 to well above $10,000 per robot) alternatives such as MobileRobots Pioneer [12], AIBO [13], and Chiara [14] robots for their undergraduate robotics courses. These robots, which tend to be larger in size, can carry a full-size laptop as the designated main computing module. However, these complex robot systems may not be as easily used by first-year students or easily maintained by the Teaching Assistant. The ER1 by Evolution Robotics [15] is the most affordable in this category of solution ($499) as it uses bare-bones 8020 aluminum frame to hold together the locomotion system and the laptop. Note that the price of the laptop also has to be accounted for as we cannot assume that students will be able to provide their own. We decided to forego this type of solution because we believe that building robots from the ground up provides a valuable learning experience and a taste of the cross-disciplinary requirements of the field.

We introduce a robotics system architecture (figure 1), which is composed of a mobile robot platform that is capable of communicating with a remote but powerful server. In this way, the mobile robots are not bound by their own on-board computational power. By adding seamless communication

Fig. 1. "Gumbot" Robot Controllers And Server

between server and robot, our system allows the implementation of much more advanced capabilities than is possible with isolated systems. Additionally, the centralized server environment lends itself to easy administration of the whole system which is an important feature in the classroom. While the concepts used to implement our system are not new, the following paper is a case study in designing a state-of-the-art classroom system and should provide enough details to allow interested educators to follow suit.

II. DESIGN AND IMPLEMENTATIONS

In this section, we first describe the mobile robot's on-board hardware (subsection II-A), and then the software components which run on that hardware and the centralized server in the following subsection II-B. The key to the system is the seamless integration between the two. Thus, we will touch on communication issues such as establishing reliable connection with enough throughput for high bandwidth data such as images from the robot's camera. In addition, we also have to ensure that the on-board platform has enough ability to perform most of the common tasks required for a mobile system, while relying on the server to execute specialized computationally intensive algorithms.

A. Mobile Computing Platform

The two tasks of the mobile robot's on-board computer are to control the robot and to be able to communicate with the server. We decided early on to only research platforms capable of running Linux. This would allow us to use our in-house expertise to facilitate easy set up of a programming and administration toolchain, as well as to reduce costs by utilizing the many open source tools available for this operating system. Furthermore, because of our familiarity with the Handyboard controller, we decided to make our feature wishlist resemble the low-level characteristics of the Handyboard, but with much more computational power. Our final feature requirements were as follows:

- Ability to run Linux with little hassle
- Minimum 500Mhz CPU
- USB Connectivity
- 802.11 Connectivity
- 10 Digital I/O
- 10 Analog I/O
- 6 Servo Outputs
- 4 DC Motor Drivers
- 8 - 13V input voltage for use with Ni-Cd batteries
- Cost below $500 per unit

In selecting a mobile platform, we were faced with many options which were roughly broken into two categories: microcontroller-based designs, and full-featured CPU-based designs. Traditional microcontrollers [16], [17] were ruled out as too limited and cumbersome to handle the types of algorithms planned for the course. Additionally, there were no plug-and-play solutions for performing the kind of wireless communication we envisioned for the architecture.

We then researched many existing CPU-based systems, but found all to be unsuitable for our exact needs due to factors such as current consumption [18], limited computational power [19], or unavailability [20]. The clear winner for our needs was the Gumstix Verdex [21] (a miniature 600MHz XScale-powered system), which could be coupled to both a Robostix board (an I/O board designed around an Atmel microcontroller), and a WifiStix board (an 802.11g board). We passed on this design, however, because it lacked both DC motor drivers and USB connectivity, despite meeting our other criteria.

To remedy this, we decided to use the Gumstix Verdex computer board and the WiFiStix, but replace the RobotStix with a custom interface board to provide the remaining basic robotics interfaces. A Console-Vx board adds both USB connectivity as well as easily-accessible header pins to functions such as I2C and UARTs on the XScale processor. These header pins were used to provide easy electrical and mechanical interfaces to the underlying baseboard. In order to fulfill our remaining requirements the baseboard needed to be able to drive up to 6 servos, provide up to 10 analog inputs, and 10 digital I/Os, drive 4 DC motors, as well as distribute power to the Gumstix and all other peripheral hardware. Taken together, these components form a system that we call the Gumbot (see Figures 2 and 3).

Satisfying all of these requirements with the Gumstix alone would be impossible due to the low number of I/O pins available. We thus decided to use a microcontroller to implement the remaining functionalities, which would then communicate with the Gumstix via a serial link through the header pins of the Console-Vx board. The Parallax Propeller [22] was chosen as the microcontroller due to its impressive computational power (eight 80Mhz cores) and 32 I/O pins. These parallel cores enabled us to quickly implement the many different functions required without needing to implement a complex scheduler, and the high pin count eliminated the need for any kind of I/O multiplexing.

The Propeller is connected to two Texas Instruments L293N ([23]) 1-Amp dual H-Bridges capable of simultaneously driving four of the LEGO motors that are used in our

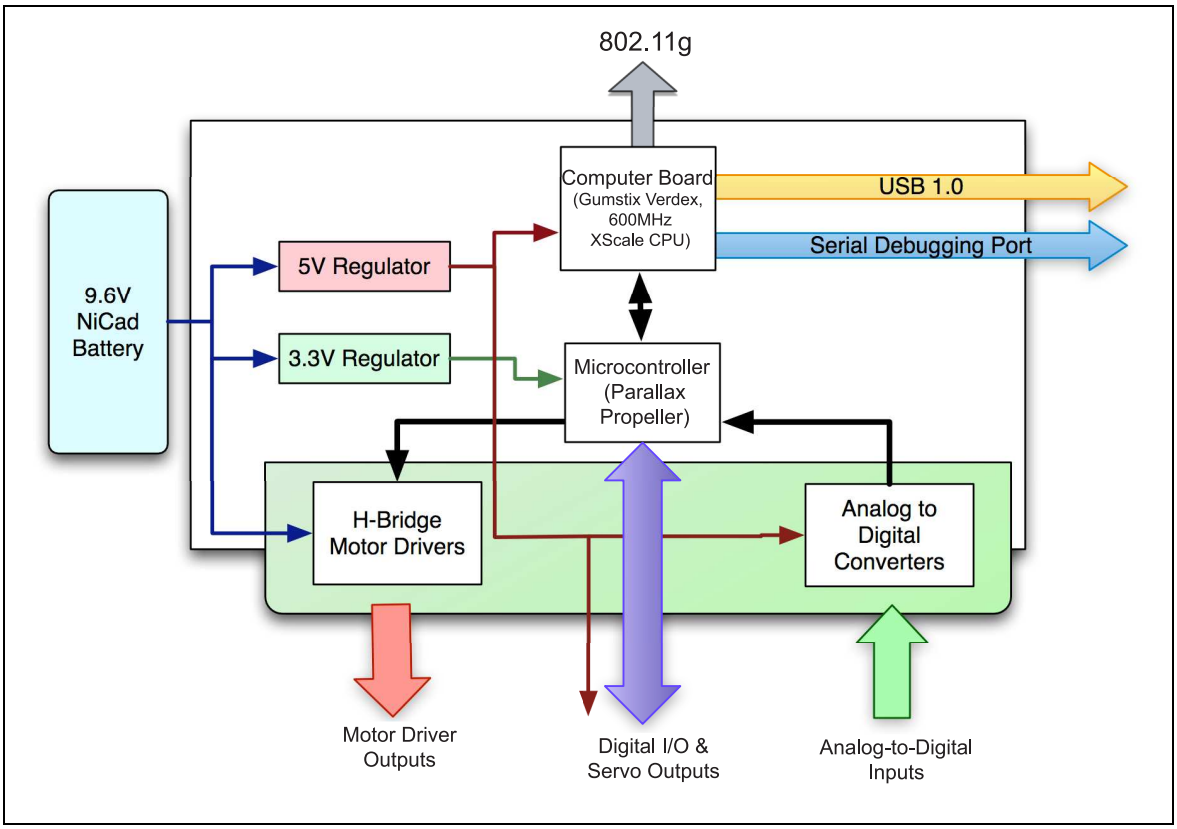

Fig. 2. Gumbot Data and Power Connections.

classroom. The Propeller also interfaces with three National Semiconductor ADC0834 ([24]) 8-bit analog-to-digital converters (ADC) with four channels each. These ADCs are set up to sample voltages between 0 and 5V with optional pullups to 5V. Digital I/O and servo control is shared between the remaining twelve pins of the Propeller, and operate with 3.3V output and 5V tolerant input. An onboard 20A DC-DC switching power supply provides power regulation from any 8V-15V input down to the 5V required by the board and any attached servos, and a single 3.3V linear regulator provides power to the Propeller.

The software side of the classroom system consists of three components:

1) Server side software which allows students to log into the centralized server, compile code, and run it on their robots;
2) Software on the Gumbot which is mostly in charge of interfacing with the physical world (reading sensors, activating actuators), possibly via the Propeller;
3) A set of software libraries (both server-side and Gumbot-side), which provide functionality for basic access to the controller board's hardware interfaces, as well as image processing and other advanced functionalities.

1) Server Side Software: The machine designated as the server is a quad-core 3.6 Ghz Xeon server running Mandriva 2008 with a GNU C++ cross-compilation toolchain set up for the XScale processor on the Gumstix. Additionally, the machine runs an SSH server and has user accounts for every student team in the class. SSH clients are installed on all of the existing Windows terminals in the lab so that students can use any of the available machines, or their own laptops, to log into the server. A source code management system [25] enables seamless backups of students' code to an off-site storage machine and allows for easy updating of shared code libraries. GNU Make was used to create scripts which allow the compilation and transfer of student code to their robots via SCP.

As mentioned above, one of the motivations for using a centralized server was to be able to provide extended computational power in order to remotely run algorithms which would otherwise be impossible on an embedded platform. In our undergraduate class, we like to give the students a broad overview of many of the available state of the art vision algorithms. By implementing these algorithms as server-side applications, we not only offload the computation but also abstract away the details of the algorithm to a level that is easily manageable by an undergraduate student. The details of the actual image transfer from the Gumbot to these server-side applications are made trivial by an RPC library from ZeroC called the "Internet Communication Engine", or "Ice" [26]. The use of Ice allows us to concentrate solely on the implementation of the needed algorithms rather than being burdened by the details of the transport.

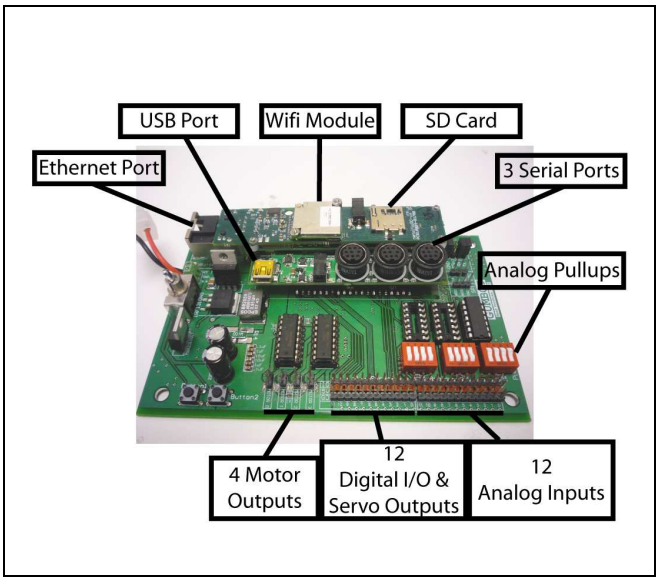

Fig. 3. Features of the Gumbot.

B. Software System

1) Robot Side Software: In order to keep Gumbot administration as simple as possible, the Gumstix run a light distribution of Linux 2.6 with an SSH server. Cross compiled binaries are transferred to the boards via SCP from the server, and are then run remotely through an SSH connection. All of this communication takes place over an 802.11g WiFi connection so that robots may be left on the lab floor while students program them and monitor their programs' output from their desks. Additionally, the Gumstix is loaded with a USB camera driver so that students may acquire images from commonly available webcams.

To perform physical actions, the Gumstix sends control words over a serial connection to the attached Propeller microcontroller. The Propeller is running very simple software which waits for control words from the Gumstix, and on receiving a valid command executes the proper action and returns a value if one was requested. Example serial commands include setting a motor speed, setting a digital output, or requesting a voltage reading from one of the analog to digital converters. More complex functionalities were also implemented allowing easy communication with some of the common sensors that we use in the classroom such as compasses and sonars.

2) Software Libraries: In order to concentrate students' lab time on the development of robotics algorithms, a set of C++ software libraries was written to abstract away many of the details of the hardware. A single C++ class encapsulates simple methods for accessing all necessary functionality of the robot. Additionally, a sophisticated Image class allows easy access and manipulation of image data, including many built-in primitive image processing operations (see http://iLab.usc.edu/toolkit/).

For example, the following code demonstrates the simplicity of writing a program for the Gumbot. In this program, we turn the robot, access the commonly used sonar and compass sensors, and grab an image from an attached USB webcam. Then, we run a visual saliency algorithm, which analyzes the image along several multiscale feature dimensions thought to exist in the primare brain, to find the most "interesting" point in the image. The algorithm used for this computation is an implementation of Itti et al.'s model [27]. Because this algorithm is computationally intensive (12 multiscale image pyramids are created for each input image), we offload the processing to the central server, yet the details of this are hidden from the user. This function call shows both the simplicity and power afforded by the centralized server architecture.

```
#include "gumbot/Gumbot.H"
int main()
{
  Gumbot g;

  //Set motor 0 to 75% power forwards,
  //and motor 1 to 100% power backwards
  g.setMotor(0,75);
  g.setMotor(1,-100);

  //Get a heading from a CMPS03 compass
  //connected to pin 1
  int heading = g.getCompass(1);

  //Get a distance from an SRF04 sonar
  //connected to pins 2 and 3
  int distance = g.getSonar(2,3)

  //Grab a new image from the
  //USB webcam, draw a red pixel
  //in the middle, and send it to a
  //server side application for display
  Image<PixRGB<byte> > img;
  img = g.getImage();
  img.setVal(img.getWidth()/2,
             img.getHeight()/2,
             PixRGB<byte>(255,0,0)
            );
  g.displayImage(img);

  //Ask the server to compute saliency
  //(Itti & Koch, 2001) on the image,
  //and return the location of the
  //most salient point.
  Point2D<int> p =
    g.getSalientPoint(img);

  return 0;
}
```

In order to compile and run an application like the one written above, a student needs only to:

1) SSH into the server

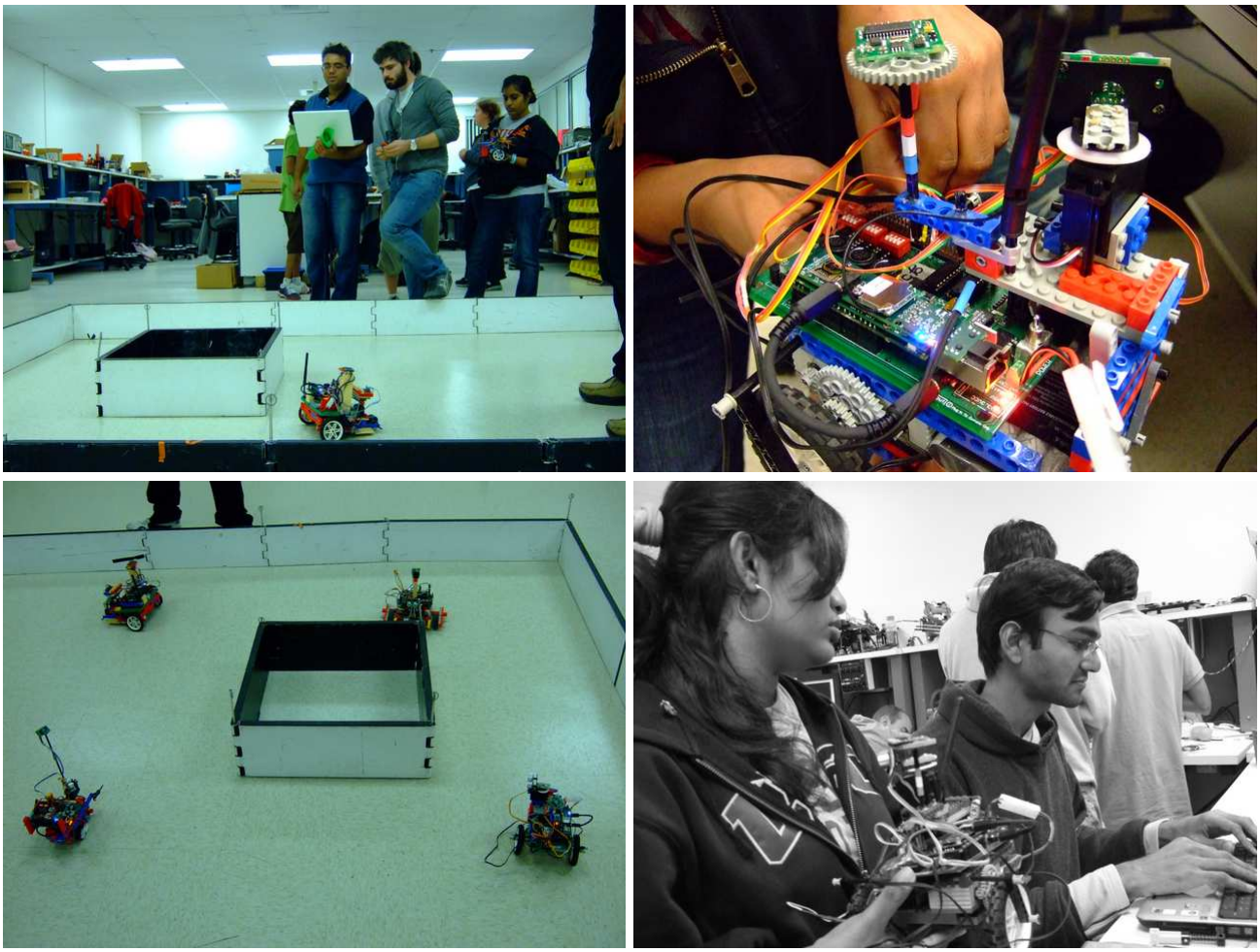

Fig. 4. Students completing coursework with the Gumbot and Server in the lab.

2) Write the above code in their favorite text editor or IDE
3) Compile the code using a provided makefile by typing `make`
4) Transfer the compiled binary to robot by typing `make install`
5) Run the program by executing a shell script generated automatically by the makefile by typing `./run_MyProgramName`

The shell script generated by the makefile opens an SSH connection to the Gumbot and executes the binary that was created in step 4.

III. EVALUATIONS

In the first semester of using this system, we found both the hardware and the software components to have exceeded our expectations in easing the job of teaching robotics algorithms. The Gumbot boards were very stable and hassle free. For example, the high efficiency DC-DC converters allow for a very good runtime. During normal operation, the board draws only $\backsim$1 Amp, giving students up to 2 hours of runtime using 2000mAh NiCad batteries. The on-board WiFi connection proved to be indispensable for both programming the robots wirelessly and for printing out debugging information during testing.

Architecturally the system worked very well, as the centralized server allowed for an extremely fast initial setup. The process of installing Linux and the GNU cross-compiling toolchain, as well as setting up the necessary user accounts took only a day. Administration of the class was equally easy, as all user accounts were accessible from a privileged teacher account. A few shell scripts were all that were required to update the software on all user accounts, and such updates could be performed in minutes. Having only one set of shared libraries eliminated any potential versioning problems that could arise from trying to create build environments on multiple machines, and any problems that were found could be fixed on all user accounts simultaneously.

Because of the increased power and flexibility of the system over our old Handyboard based lab, we were able to vastly increase the scope of material that was covered. The first lab session taught the basics of DC motor control, and had the students implement PWM bit-banging directly on the Propeller microcontroller. By the second lab session, the students had learned the details of the server compilation system and were able to write code for the Gumstix to drive

their robots around the classroom remotely. Over subsequent lab sessions, the students learned and implemented concepts in both control theory and sensor filtering. We then moved on to cover basic image processing, in which the students developed their own blob tracking algorithms which they used to follow various objects around the classroom. The final assignment in the lab was to implement a grid-based Bayesian Markov localization algorithm to localize and navigate their robots in an arena on the floor of the classroom.

The cost of the system consists of the price of the server, a WiFi router and the Gumbots. The Additional costs of terminals to log into the server, sensors, and LEGO parts to build physical robots are left out as they are outside the scope of this paper, and independent from the system we have described. The cost of an individual Gumbot is:

- Gumbot: $377 each
  - Carrier Board PCB: $11
  - Carrier Board Components: $93
  - Gumstix Verdex Pro XL6P: $169
  - Gumstix Console VX: $25
  - Gumstix WiFiStix: $79

The exact specifications of the server are determined by the unique needs of a class. In particular, the number of students in the class will determine how powerful a server is necessary. In our case we had 5 groups (robots) per lab session, and a quad-core Xeon 3.6 GHz machine with 4GB of RAM and a 200GB harddrive was more than sufficient. Such a machine typically runs in the $1500 range.

IV. CONCLUSIONS AND FUTURE WORK

While the system is currently fully usable, there are many additions and modifications that are planned for the future. The current Gumbot boards are only a first pass, and there are a few minor kinks that will be worked out in the next revision. For example, a few of the mechanical interfaces on the boards ended up being a source of trouble as the connectors chosen proved quite fragile for every day classroom use. The main structure of the boards was quite robust, however, and was shown to be more than capable of the demands we required. If time and funding allow, we plan to build a new version of the boards based on Gumstix's new Overo platform [21]. The design of the board will remain similar, but the new Overo modules will allow us to decrease size and costs as well as provide a direct image sensor interface and an upgrade to USB 2.0. The Overo modules can also have a much sturdier mechanical connection to the Gumbot main board due to their connector layout.

On the software side, we plan to make only additions to the system in the form of more image processing modules to be run on the server. Currently in the works are a SIFT [28], gist [29], face detector, and shape recognizer modules, but more will be developed as the lab curriculum matures. The main benefit of these modules is that it allows us to brush on advanced topics in vision and allow the students to get a feel for the tools without necessitating a level of detail inappropriate for an undergraduate class.

Importantly, we found that this new platform has also allowed us to significantly enhance the lecture portion of our class. This was not only achieved by allowing the students to gain hands-on experience with algorithms covered in the lectures, but also by introducing students to modern computer science tools used in today's robotics research. Indeed, an ancilliary benefit was to present a more modern view of robotics, which is closer to the research state-of-the-art and farther removed from hobbyist-grade microcontroller hacking. Namely, students spent little time worrying about microcontroller or assembly-code details, except during an introductory lecture to the Propeller. In further lectures, students were exposed to a robotics platform that employs distributed Linux-based computing, cross-compiling, remote-login into your robot, wireless communications, the Ice transport library, using a central source code versioning system, makefiles, the C++ Standard Template Library, etc. We believe that using these essential computer science tools on a robot provides a drastically different perspective, whereby classroom robots become powerful computing machines with sufficient brainpower to support high-level algorithm design using modern tools, rather than being limited to spinal-cord-grade algorithms designed using low-level tools on limited-computation microcontrollers.

V. ACKNOWLEDGMENTS

The authors gratefully acknowledge the contribution of The University of Southern California and reviewers' comments. We thank Josue Martinez for classroom pictures.

REFERENCES

[1] University of Georgia, "CSCI-4530/6530: Introduction to Robotics," February 2009, http://www.cs.uga.edu/students/syllabi/CSCI4530_6530.html.
[2] University of Nevada Reno, "CpE 470/670 Autonomous Mobile Robots," February 2009, http://www.cse.unr.edu/~monica/Courses/CPE470-670/index.html.
[3] University of Louisville, "ECE500 Fundamentals of Autonomous Robots," February 2009, http://www.ece.louisville.edu/~t0inan01/teaching/ECE564/.
[4] Rensselaer Polytechnic Institute (RPI), "CPSC 452: Robotics and Spatial Intelligence," February 2009, http://www.cs.rpi.edu/~trink/Courses/Intro_to_Robotics/home.html.
[5] University of Southern California, "Introduction to Robotics," February 2009, http://www-scf.usc.edu/~csci445/.
[6] F. Martin, "Circuits to Control: Learning Engineering by Designing LEGO Robots," Ph.D. dissertation, MIT, Cambridge, MA, 1994.
[7] Carnegie Mellon University, "16-311 Introduction To Robotics," February 2009, http://generalrobotics.org/.
[8] University of Minnesota, "CSCI 5551: Introduction to Intelligent Robotic Systems," February 2009, http://www-users.itlabs.umn.edu/classes/Fall-2008/csci5551/index.php?page=assign.
[9] Northern Michigan University, "CS 295 Special Topics in Computer Science: Introduction to Mobile Robots," February 2009, http://euclid.nmu.edu/~jeffhorn/Classes/CS295/Winter2002/.
[10] Utrecht University, "AIBO Programming," February 2009, http://www.cs.uu.nl/education/vak.php?vak=INFOAIBOP\&jaar=2005.
[11] Harvey Mudd College, "Computer Science 154 Robotics," February 2009, http://www.cs.hmc.edu/courses/2009/spring/cs154/index.html.
[12] Mobile Robots, Inc., "Intelligent Robots for Commercial & Research Applications," January 2009, http://www.mobilerobots.com/.
[13] Sony Entertainment Robot Europe, "Aibo," January 2009, http://support.sony-europe.com/aibo/.
[14] D. Touretzky, "Chiara," January 2009, http://chiara-robot.org/.
[15] Evolution Robotics, Inc., "ER1 Personal Robot System. User Guide. Version 1.1.1." January 2009, http://www.evolution.com.

[16] Arduino, “Arduino,” February 2009, http://www.arduino.cc.
[17] Pololu, “Orangutan SV-168 Robot Controller,” February 2009, http://www.pololu.com/catalog/product/1225.
[18] Via, “Pico-ITX Mainboard Form Factor,” February 2009, http://www.via.com.tw/en/initiatives/spearhead/pico-itx.
[19] Charmed Labs, “Qwerk,” February 2009, http://www.charmedlabs.com.
[20] F. Martin and A. Chanler, “Introducing the Blackfin Handy Board,” in *Proceedings of the AAAI Spring Symposium on Robots and Robot Venues: Resources for AI Education*, Stanford, CA, March 2007.
[21] Gumstix, “Gumstix,” February 2009, http://www.gumstix.com.
[22] Parallax, “Propeller General Information,” February 2009, http://www.parallax.com/tabid/407/Default.aspx.
[23] Texas Instruments, “L293N Quadruple Half-H Drivers,” February 2009, http://focus.ti.com/docs/prod/folders/print/l293.html.
[24] National Semiconductor, “ADC0834 8 bit Serial I/O A/D Converter with Multiplexer Option,” February 2009, http://www.national.com/mpf/DC/ADC0834.html.
[25] Subversion, “Subversion,” February 2009, http://subversion.tigris.org/.
[26] ZeroC, “The Internet Communications Engine (Ice),” February 2009, http://zeroc.com/ice.html.
[27] L. Itti, C. Koch, and E. Niebur, “A model of saliency-based visual attention for rapid scene analysis,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 11, pp. 1254–1259, Nov 1998.
[28] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.
[29] C. Siagian and L. Itti, “Rapid biologically-inspired scene classification using features shared with visual attention,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 2, pp. 300–312, Feb 2007.