**Leveraging laptops for low-cost, full-fledged outdoor robotics**

Zachary Dodds

Harvey Mudd College Computer Science Department  
301 Platt Boulevard  
Claremont, CA 91711

909-607-1813

dodds@cs.hmc.edu

**Abstract**

The growing ubiquity and capabilities of off-the-shelf laptop computers give CS educators remarkable opportunities to include hands-on robotics within their curricula. At our college of 700 students we have developed several laptop-controlled robots based on the very inexpensive PowerWheels line of FisherPrice toys. Such a chassis offers a capable, low-cost base for large-scale outdoor navigation and planning tasks. It enables cost- and time-effective undergraduate engagement in the ongoing community of robot- and vision-themed venues, exhibitions, contests, and conferences. This work describes both successes and caveats from how laptops enable the use of these robots in AI electives and independent-study projects. We conclude that leveraging Moore's law helps make robotics not only an engineering challenge, but a truly computational endeavor at the undergraduate level.

**Section 1 - *Motivation*: self-driving cars for all?**

Progress in robotic autonomy has recently achieved deservedly well-publicized results through DARPA's grand challenges. In 2005 Stanley, an autonomous Volkswagen Touareg, successfully completed a six-hour desert course without human intervention (Thrun et al., 2006); In 2007 CMU's vehicle, Boss, wended its way through the dynamic and unpredicatble obstacles of urban-style driving (Urmson et al., 2007). Both systems attracted ample coverage from worldwide media; indeed, they have inspired many students to think more deeply about the possibilities inherent autonomy, robotics, and computer science and engineering.

Yet the opportunities available to many students have failed to keep pace with these cutting-edge advances. Even if not strictly true, the conventional wisdom -- that autonomous cars require even more money to build than the millions given away to the contest winners – accurately reflects how few institutions can participate in such venues. Over the past two years we have sought to provide our students a simpler and less expensive autonomous vehicle based on the PowerWheels line of children's toys. Figure 1 depicts three of our four vehicles. The result is a team of robust and capable robots that use students' or departments' laptop computers for computation and control. What is more, the total system cost of approximately $500 makes the platform accessible to student clubs, CS and engineering departments, or adventurous tinkerers.

**Figure 1:** Three PowerWheels platforms for student investigations of outdoor robotic autonomy. Because they rely on laptops for control, these platforms cost about $500 each to create. The vehicle on the right is the "Ford F150"-design: it alone offers a narrow enough chassis to fit through doorways without turning.

Penn State Abington emerged as the first college-level advocate of autonomous PowerWheels platforms (Avanzato, 2005). That school has held a competition for such robots, appropriately named the *Mini Grand Challenge*, since 2003 (Avanzato, 2007). Since then adoption has spread to other institutions and has continued to grow. This paper contributes the lessons learned by one such early adopter, including

* how we overcame hardware challenges in constructing our platforms
* accessible hardware and software interfaces to sensors and laptop computers
* the curriculum and algorithms that PowerWheels platforms have opened up for our students

Although progress in developing our own fleet of autonomous outdoor vehicles has not been entirely smooth or straightforward, we have emerged from our two-year experiment enthusiastic about the results thus far and the potential for future student work. This paper seeks to simplify that journey for other institutions and investigators.

**Section 2 - *Background*: putting the pieces together**

Especially for educators whose background is more software than hardware, creating a laptop-controlled vehicle can be a daunting task. Indeed, our experiences had all of the deadends, roundabout realizations, and sadly smoking components expected of exploratory hardware design. We hope that by sharing our travails - and triumphs - others will have a less tortuous path to an autonomous vehicle. This paper is not an appropriate medium for step-by-step instructions; those details are available at the URLs provided (URLs, 2008). Rather, this section seeks to give a broad overview of an autonomous PowerWheels's hardware and the subsystems it comprises.

**2.1 Chassis**

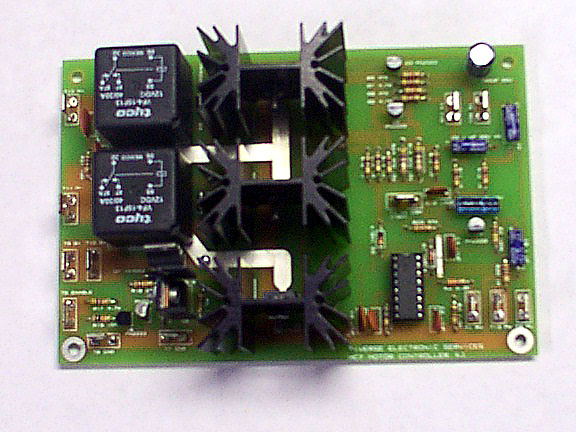
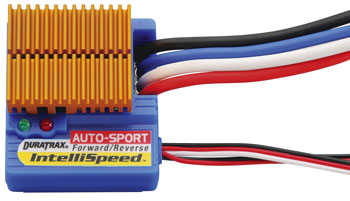
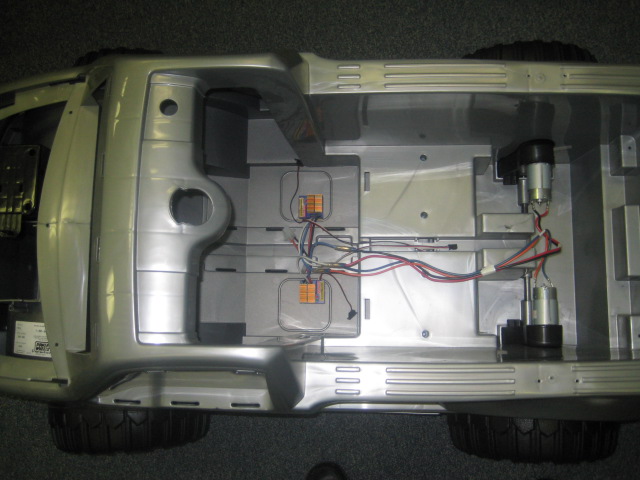
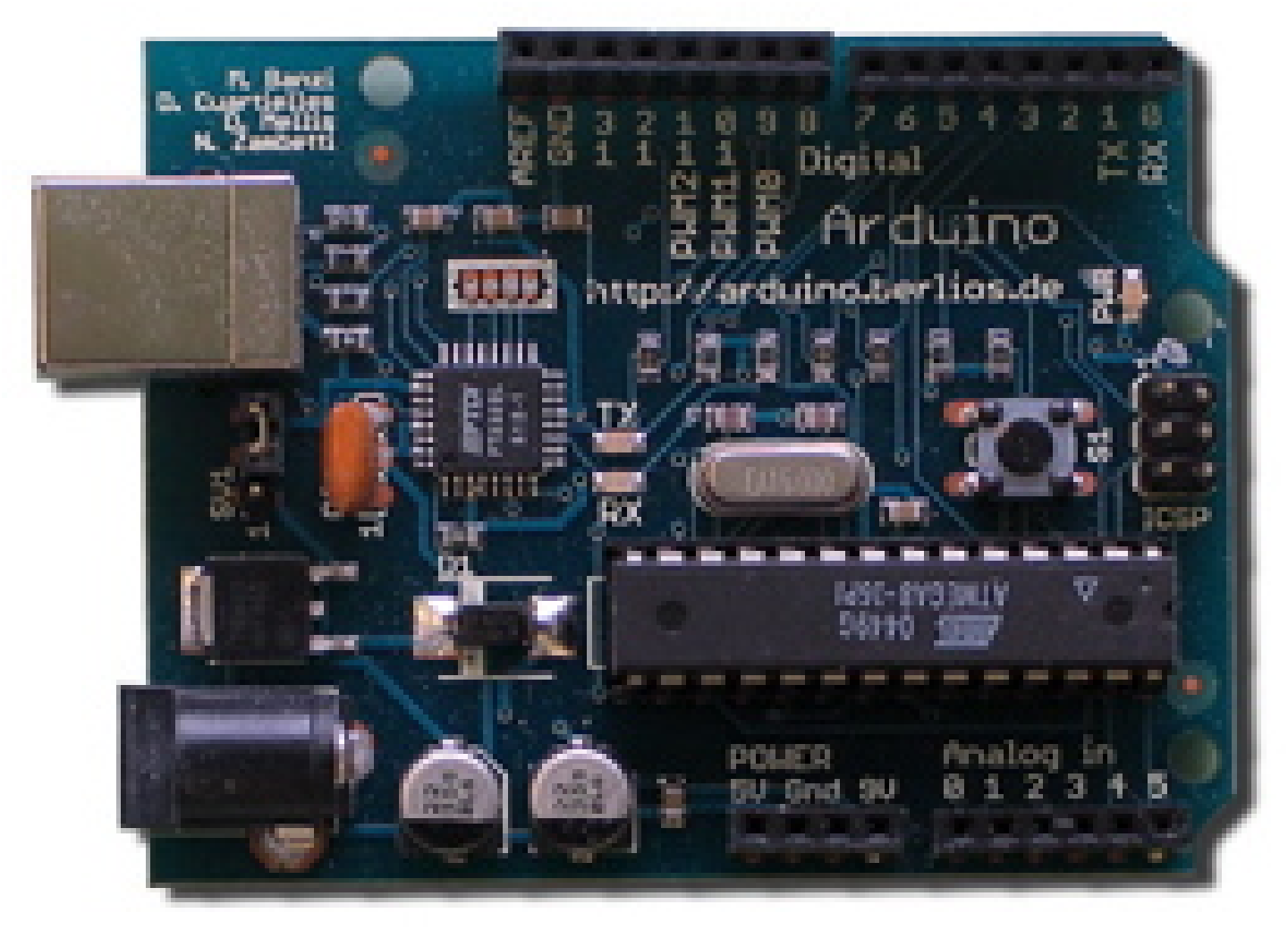
PowerWheels vehicles are easily obtained from large toy retailers. Because of the size of the cars, they are not often available on the shelf in stores, but on-line ordering has proven quick and efficient. What is more, some retailers waive the cost of shipping if you pick the box up at the nearest store. This reduces the price for a "Ford F150" toy to less than $150 from WalMart.

Narrow vehicles, such as the Ford F150, have all the stability and weight capacity for a modest-speed autonomous robot. They also have the advantage of much easier storage and retrieval, as they fit through standard doorways without problem. The GatorJeep and BarbieJeep deisgns offer plenty of customizable surface area, as shown in Figure 1, but their wider profile is a recurring disadvantage.

**2.2 Drive wheels**

An interface allowing computer control of the rear wheels is a natural first addition to the system. We tried two commercially available motor interfaces that can handle the 30 amperes (surge) and more normal 5-10 amps the back wheels demand. The MC-7, available from **divelec.tripod.com/html/mc7.html**, lists at $95 and the DuraTrax DTXM1059 (**www.duratrax.com/caraccys/dtxm1055.html**) electronic speed controller costs $38. We have used the former to drive both back wheels in sync, in the way the vehicle ordinarily runs. An alternative is to use two speed controllers in order to drive the back wheels differentially. Figure 2 shows such a set up. In this case the vehicle does not even need front-wheel steering in order to turn. However, we have found that the rear wheels do not generate enough friction on all surfaces to turn the vehicle while the front wheels slip. Additional traction can be gained by adding small strips of adhesive rubber around the circumference of the back wheels.

Both of these interfaces can take pulse-width modulated input of the type that the USB Arduino board produces. The Arduino has a well-deserved reputation for its ease of setup and use: Linux, Mac OS X, and Windows are equally well-supported, and sending the board signals from a program involves no more than writing ASCII strings to the resulting serial port. We have used this chain of connections without incident from both Macs and PCs -- both with and without USB-to-serial converters. Figure 2 illustrates these components.

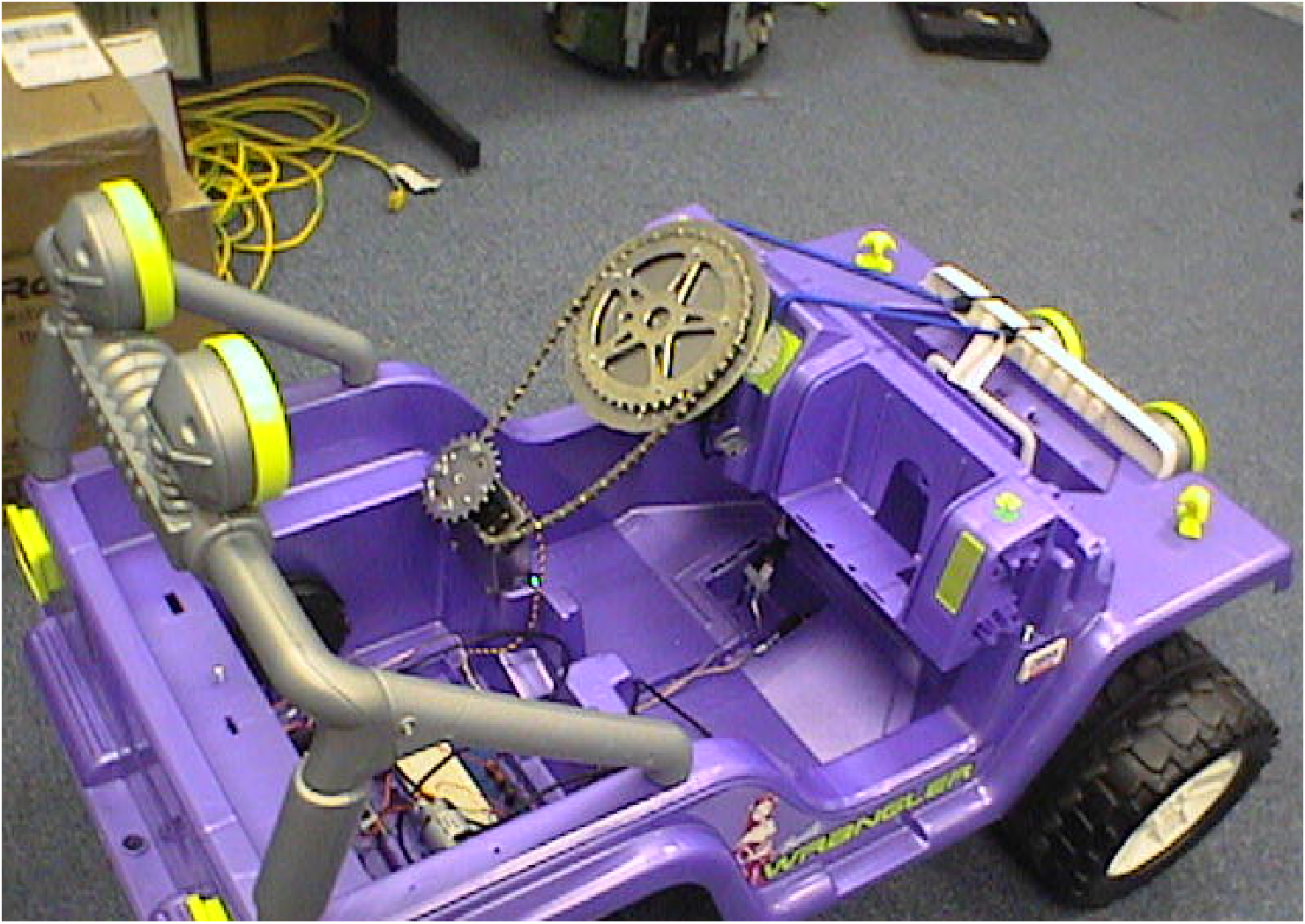
**Figure 2:** Shown at left are the MC-7 and DuraTrax controllers; in the differential-drive system (middle) each rear wheel is controlled independently through its own DuraTrax controller. At right is the general-purpose Arduino I/O interface board. It comes with excellent, free software and documentation and it works well under any major operating system.

**2.3 Front-wheel steering**

The most difficult part of creating an autonomous PowerWheels vehicle is motorizing the front wheels. A child driving the toy uses the steering wheel to turn the front wheels up to thirty degrees left or right. A natural design, then, is to build a mechanism that turns the steering wheel itself. Inspired by Penn State's bicycle-chain-and-servo-motor solution, we tried this but two difficulties prevented us from succeeding: (i) the plastic chassis was not sufficiently stiff to maintain the needed tension in the bicycle chain and (ii) the servo motor we used did not have enough torque to consistently turn the steering wheel.

Instead, we opted to directly push the front axle with a linear actuator, shown in figure 3. The force generated more than suffices to turn the wheels on any surface. The control is also straightforward: a single-wire voltage signal that can be generated by the same Arduino board that drives the rear wheels. The biggest disadvantage of a linear actuator is its speed. It requires 3-4 seconds to cover its full range of motion – this leads to stop-and-start navigation strategies in which the vehicle stops, turns, and resumes driving afterward. This is less compelling for observers than adjusting the steering angle on the fly, but only large turning radii can be followed when the vehicle is moving.

To gain the advantages both of directly steering the front axle and of fast steering response, we have begun experimenting with a strong servo motor mounted to the underside of the vehicle’s front battery compartment, with a rack gear transferring force to the wheels. Although still in its development, we will report on this design, once complete, through our website.

**Figure 3:** Though tempting, the chain-driven steering wheel did not work for us (left). Instead, a linear actuator (middle left and middle right) provides more than enough torque to turn the front wheels. Though not yet complete, a directly attached servo motor (far right) may work as well.

**Section 3 - It's built – what now?**

With the front and rear wheels under computer control, the PowerWheels vehicle becomes, in essence, a very large remote-control vehicle. This, alone, offers software architecture challenges suitable for an early-semester project, for example: writing a set of routines and a user interface that enables teleoperation of the vehicle. We use Python as the basis of all of our control software, though any language capable of using a serial connection would suffice. One advantage of having a laptop mediate this computation is that our campus’s wireless network is integrated immediately and with no additional effort into the car’s resources.

Autonomous driving, however, requires a suite of on-board sensors to provide information about the vehicle’s environment. Indeed, the first deficiency students notice is the lack of position feedback: the vehicle does not know where it is.

**3.1 Position and motion sensing: GPS**

We have used GPS in order to estimate our vehicles’ positions. Units such as the US GlobalSat’s BU-353 receiver cost about $80 and provide a stream of ASCII serial data. As with the Arduino interface, we use Python in order to pare and interpret that data. Our full source code is available online at (URLs, 2008). The one concern about GPS is that it only provides accuracy within about 10 feet. Thus, it is useful for providing a first approximation to the robot’s global position, but finer adjustments require sensors that measure the chassis’s immediate surroundings.

**3.2 Range sensing: Sonar**

Each of our completed vehicles relies on both a front-right and front-left ultrasonic sensor in order to determine whether an obstacle has moved into the current trajectory. These Devantech SRF-04 sonars cost $30 a piece and provide range readings between 3 inches and about 8 feet. The electronic protocol for triggering an ultrasonic ping, reading its time-of-flight, and converting that time into a distance is somewhat involved. However, again the Arduino proves its value: that board not only can generate and monitor up to three sonars without additional external power, but its libraries have pretested software for a large number of low-cost sonars, including Devantech’s. Figure 3 shows our headlight-mounted sonars.

**3.3 Vision via web cameras**

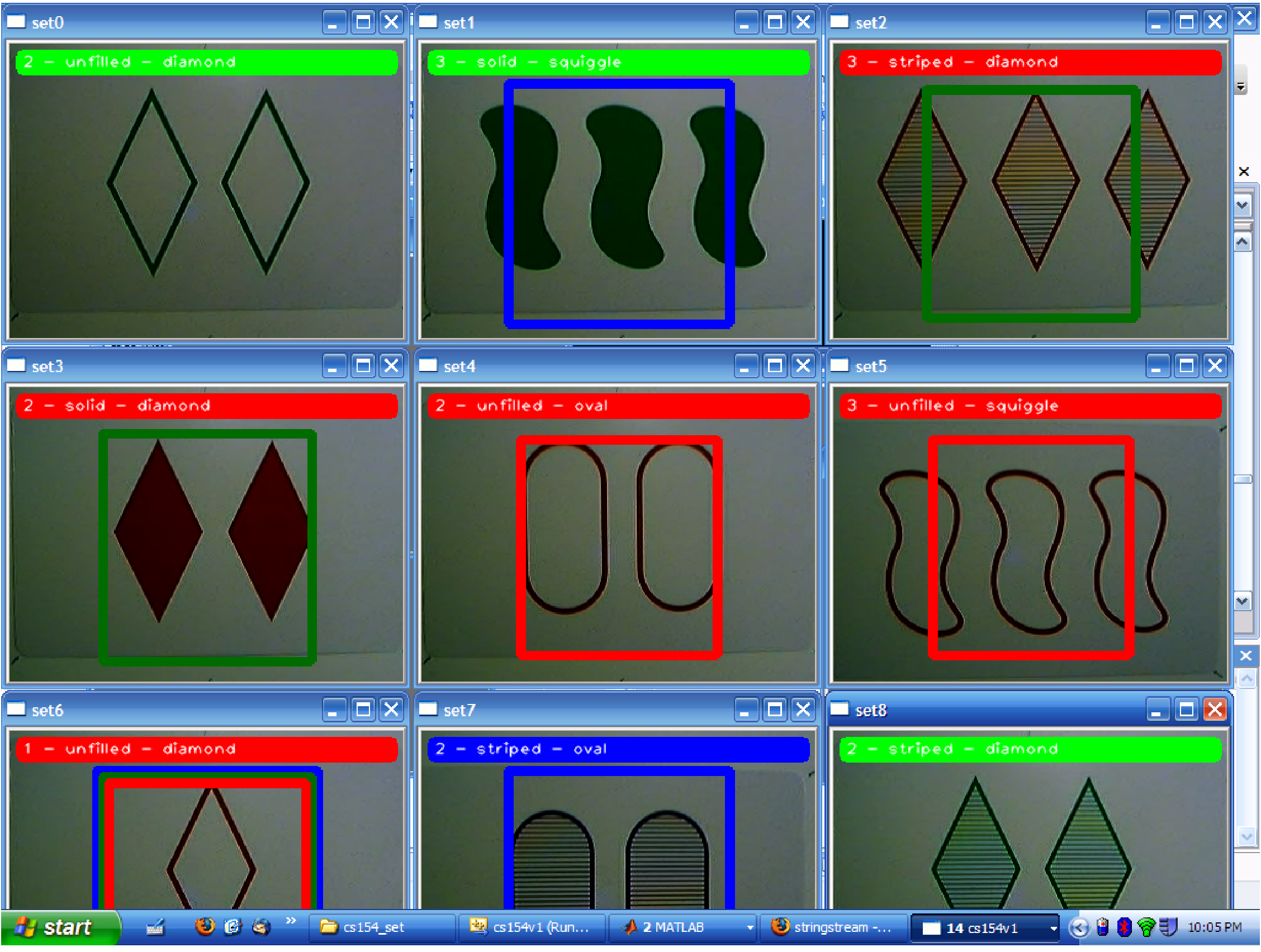
The most important advantage, perhaps, of having laptop computers on board a PowerWheels vehicle is the ease of access to visual data. Smaller robots, such as the Scribbler or Lego RCX, avoid cameras altogether because of the computational demands of the pixel acquisition and processing. Others, such as Botball’s XBC with a CMU cam, or the more recent Lego NXT, offer pre-segmented images based on a set of preprogrammed color definitions. With a full PC, however, students have the opportunity to push beyond blobs of color to investigate algorithms that rely far less on contrived lighting conditions or landmarks.

What is more, the PowerWheels’ cameras are not tied to the robot at all: they are ordinary webcams, available for $40 or less at any computer supply outlet. The software we use to access the cameras’ pixels is OpenCV, a powerful open-source library that runs under Windows, Mac OS X, and Linux. We have used both the Windows and Mac OS X versions and have developed a short, compelling homework project that familiarizes students with the libraries while simultaneously building a code foundation for their vehicles (URLs, 2008). The winner of the 2005 DARPA Grand Challenge, Stanley, used OpenCV as its starting point for its vision processing (Thrun, 2006). It is this full-fledged vision system, more than any other component, that distinguishes laptop-based robots from alternative educational platforms.

**3.4 Software flexibility**

A related advantage of using laptops – and often students’ own laptops – is the familiarity and power of the development environment and user interface. In contrast to many undergraduate robotics courses, our offering does not mandate the choice of a programming language or development environment. After all, by the time the students are juniors or seniors, they have started to develop efficient habits for prototyping, testing, and deploying code. We want to leverage that experience. Thus, we guide students to create software components – in any language – that do the work of controlling the motors, reading the sensors, or combining the two within a control loop. When different team members prefer different languages, information is passed through a loopback socket connection. When the sensing requires more than one computer, for example, if more than one camera is used, the loopback socket connections simply become true network connections between the two machines. To date, different teams and subteams have opted for C++, Java, and Python. We have not encountered difficulty in integrating those systems.

An additional benefit of using a full-fledged computer for sensing, reasoning, and controlling the vehicles is that *any* software that runs on a PC or Mac can be brought to bear. We have used Matlab, speech-recognition libraries, speech-synthesis libraries, and other off-the-shelf algorithms simply unavailable to the special-purpose, dedicated processors on other robot platforms. The next section highlights some of the algorithms our students have been able to investigate via the computers on board these robots.

**Figure 4:** The BU-353 GPS unit (left) from US Global Sat provides a USB connector and a standard serial interface. Such off-the-shelf GPS receivers provide only coarse positioning: the error is illustrated in the image in the middle, in which the vehicle traveled along the blue path, but GPS reported the vertices of the red path. The OpenCV library eases pixel processing: the image to the right shows the results of a student-developed *Set* player that provides a gentle, motivating, and thorough introduction to OpenCV.

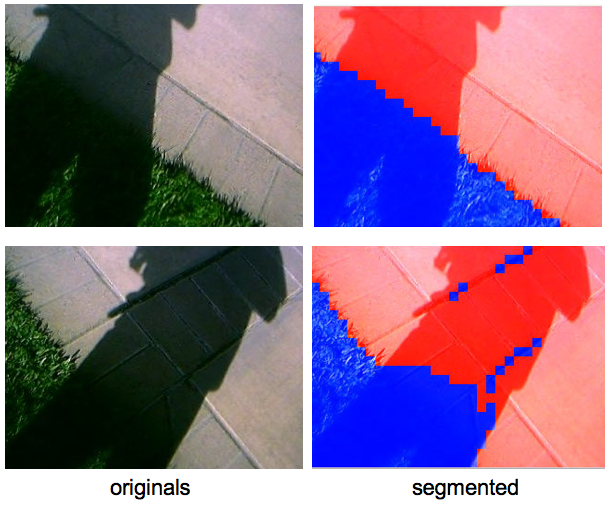
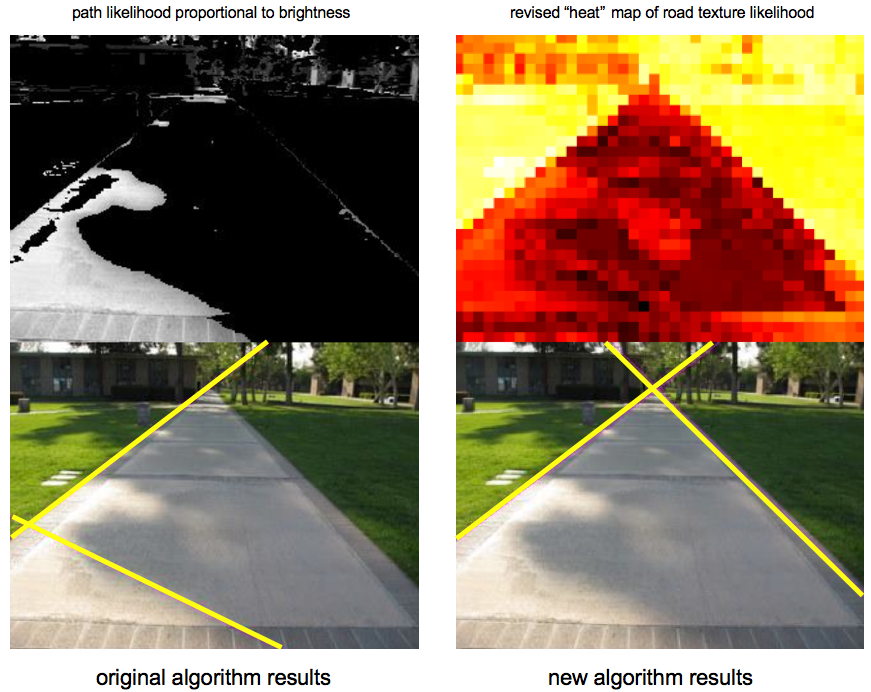
**Section 4 - *Results*: uncontrived autonomy**

As a result of the familiar, uncontrived computational interface, the PowerWheels vehicles are well-prepared to handle tasks in uncontrived environments. For example, the Mini Grand Challenge asks students to program a vehicle to autonomously navigate along campus paths without special landmarks except for ordinary traffic cones that block off incorrect branches at path intersections.

For this task, in contrast to many other college- or high-school-level tasks, color segmentation does not suffice. Even in stereotypically uniform climates, the lighting conditions along tree-lined paths vary dramatically, as do the path texture, the visual characteristics of non-path areas, and the variety of distracting image regions that may occur. Figure 5 illustrates how even carefully-tuned color definitions can fail; these failures, in turn, inspired our students to investigate alternative approaches that would improve performance. For example, OpenCV contains an adaptive segmentation approach known as the *watershed* algorithm; another group used a freely available backpropagation library to train a neural net to distinguish between pat and nonpath texture patches. Figure 5 shows these results, as well.

The growing community of mini-grand-challenge participants offers students an excellent opportunity to contribute to a shared knowledge base. For instance, the Penn State Abington grand challenge team of 2005 published a path-segmentation algorithm based on k-means segmentation (Patel, 2005). This year, a team of students opted to implement their algorithm from the source code provided (in Matlab) as their starting point. They noted that it performed well in many cases, but failed in the presence of strong shadows or paths that curved too sharply. Continuing in Matlab, they proposed, tested, and verified improvements to the original algorithm. Their report is now openly available (URLs 2008). Thus, the experience provided an opportunity to engage in the practice and communication of the broader research community – without demanding effort beyond the confines of an undergraduate elective.

Such an open-ended task as path-following need not – and perhaps should not – be the starting point for student work with these vehicles. Our teams have used indoor hallways or created visually distinct paper trails to follow as simpler scaffolding with which to develop and debug their systems. To date, our most successful run has been a 12-minute autonomous jaunt in the center of our campus. Other schools have pushed the platform much further, e.g., 2008’s Mini Grand Challenge team from Southern Illinois University Edwardsville.

**  **

**Figure 5:** At left is the original image and its color-based segmentation of path (shown in green) from non-path (in black) – the problem is that the color definitions were tuned under lower light conditions, so that the saturated part of the path is now lost. The middle figure demonstrates the results of a backpropagation-trained neural network, which classifies path in red and non-path in blue. Even heavily shadowed areas are handled correctly. Students have built atop prior work at Penn State Abington (right) in order to handle uncontrived image processing tasks such as this one.

**Section 5 - *Perspective*: self-driving cars for all!**

In the end, a robotic platform is valuable to the extent that it provides students motivation and opportunity to stretch their skills. Many platforms emphasize engineering skills over computational ones – this is by no means a bad thing! Yet the PowerWheels platforms described in this paper provide an accessible means to focus students’ efforts on the compelling CS and algorithmic challenges of robotics. What is more, these vehicles are simple enough and inexpensive enough to reach many more students than DARPA’s competitors possibly can. For example, a high-school team from the Pomfrt School built an impressive entry for the 2008 Mini Grand Challenge. In this spirit, we look forward to working with the robotics and computer science education communities in order to realize the full *computational* potential for robots in the undergraduate CS curriculum.

**Bibliography**

Avanzato, R. 2005. *Low-cost Outdoor Robot Platform for the Penn State Abington Mini Grand Challenge* Conference proceedings of the Association for the Advancement of Artificial Intelligence, 2005 Pittsburgh, PA, pp. 1716-1717, AAAI Press.

Avanzato, R. 2007. *Mini Grand Challenge Contest for Robot Education* AAAI Technical Report on the 2007 spring symposium "Robots and robot venues: resources for AI education." 3/26-28/2007, Palo Alto, CA, AAAI Press.

Patel, S. 2005. Monocular color based road following. Tech report of the Penn State Robotics Club. Available at **www.clubs.psu.edu/up/robot/files/MiniGC05VisionPaper.pdf**

Suehle, R. 2007. *Arduino: One board, many projects* RedHat Magazine Aug. 30, 2007.

Thrun, S., Montemerlo, M. et al. 2006. *Stanley, the robot that won the DARPA Grand Challenge* Journal of Field Robotics 23(9), pp. 661-692, Wiley Blackwell, publishers.

Urmson, C., Anhalt, J. et al. 2007. *Tartan Racing: A Multi-Modal Approach to the DARPA Urban*

*Challenge* DARPA Technical Report, April 2007. (www.darpa.mil/grandchallenge/resources.asp)

URLs 2008. All of the URLs referenced in this paper are at www.cs.hmc.edu/~dodds/PowerWheels.