Carlos: The Fire-Fighting Robot

Anne Neilsen, Justin Popek, Lyle Shearer
Computer Engineering Senior Design Project
University of Nebraska - Lincoln

5 May 2010

Abstract - *For a senior design project, we created a fire protection system made up of a fire-fighting robot and ground control station. The key requirements for this system were to create an effective and low cost system that can be used in hazardous situations. The developed solution is a prototype that might be built upon to create a production model and uses a combination of open-source and Microsoft technologies. The motivation, solution (including design, implementation, and testing), and analysis of the results and cost are discussed in this paper.*

Keywords: remote-control robot, fire-fighting, Arduino Microcontroller

1 Introduction

A difficulty plaguing developing nations is the lack of support infrastructure for handling common problems. Our senior design team recognized that a shortage of resources in these countries could create dangerous situations for public servants. We narrowed this broad problem down to target a specific customer need to:

create an effective, low cost, and safe fire protection robot and control system for areas of the world with limited resources and environments where it is too dangerous for humans.

Current solutions that address this need are either cost prohibitive or have a limited supply of available resources.

Driven by an Xbox controller that is connected to a ground control station, the robot will use a fire-extinguisher to put out a flame. A rapid response emergency device, such as this, reinforced with stable technologies, could immediately benefit low resource communities across the globe.

This paper describes the fire protection robot and control system solution developed by our team as part of our senior design project and is organized as follows: Section 2 discusses the high-level design and a high-level description of the system integration. Detailed descriptions of the required algorithms (3.1) and hardware components (3.2) are found in Seciont 3. The implementation is explained in two parts (4.1 Ground Control Station and 4.2 Remote Control Vehicle) of Section 4. The testing strategies are explained in Section 5. Section 6 provices a comprehensive discussion of the resulting solution and possible extensions. An analysis of the cost in Section 7 estimates the hours of work to be over 200 hours and the monetary cost to create Carlos at $555. Some of the key difficulties encountered were getting the router to communicate with the Arduino (8.1), the Arduino only supports one serial connection when using the standard serial library (8.2), Arduino pin problems (8.3), and integrating the Xbox controller to Silverlight (8.4).

2 Design Concept

Initial idea discussions resulted in some primitive sketches (see Figure 1) and a high-level understanding of the requirements for a fire-fighting robot and controls for such a robot. Some of the original ideas included giving the robot artificial intelligence using ultra-violent sensors to detect the fires and having various modes where at times the robot would search for fires itself and other times it would be manually controlled. While possible, these were not the best suited solution to meet our requirements. The primary requirements for this solution are:

- easy to use,
- minimal initial and maintenance hardware cost,
- minimal software maintenance cost,
- robot can run while being a significant distance from people,
- robot can put out hazardous fires

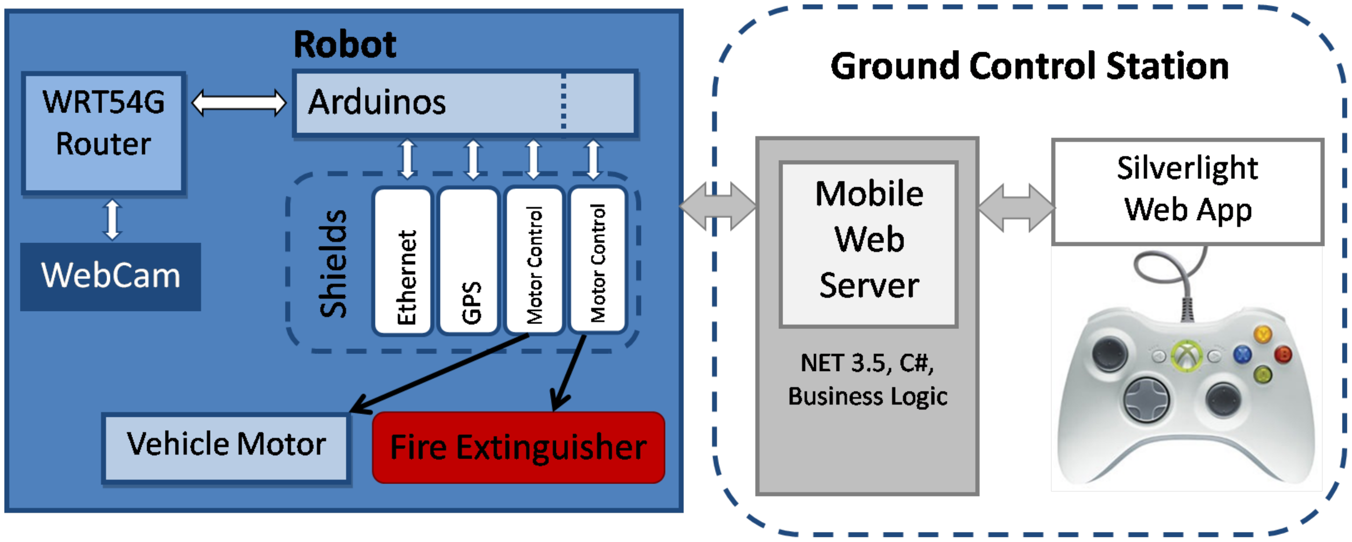

Figure 2: High-Level Design of Robot and GCS

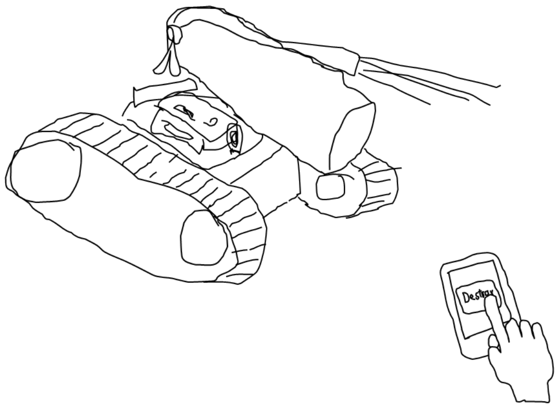

Figure 1: Sketch from an initial brainstorming session

Given our target market and requirements, keeping the fire-protection system as simple as possible is ideal, because it reduces the cost to make and maintain. Therefore, it was determined that the system would have a remote controlled robot and Ground Control Station (GCS) to manage the robot. Figure 2 shows the high-level design, which addresses the requirements for this system. The robot contains several components (discussed in greater detail in Section 3: Analysis of Design Choices and Architecture), moves according to directions from the GCS, and engages a fire-extinguisher to put out fires. The GCS includes a Mobile Web Server and Silverlight Web Application that is used to in conjunction with the Xbox controller for a simple user interface.

3 Analysis of Design Choices and Architecture

The high-level design contains many pieces and there are multiple ways to design and implement a fire-fighting robot. Various decisions were made as to what sort of algorithms and hardware to use. A further discussion of these decisions follows. In Section 3.1, we discuss the require algorithms and in Section 3.2, we discuss the Hardware Components (3.2.1 Arduino Microcontroller, 3.2.2 Arduino Shields, 3.2.3 Linksys WRT54G Router, 3.2.4 Panasonic BL-C1A Network WebCam, 3.2.5 Xbox controller).

3.1 Algorithms

With regards to the the required algorithms needed, all communication between the robot and GCS is straightforward. The primary interaction between the robot and GCS is handling and forwarding events and data through the component level APIs. As such, algorithm requirements are negligible.

3.2 Hardware

The key hardware components used are (add references, websites, etc.):

- Arduino Microcontroller (ATmega 328)
- Arduino Shields
- Linksys WRT54G Router

- Panasonic BL-C1A Network WebCam
- Xbox Controller

These components have been researched and found to be the most ideal to meet the requirements of being part of a low-cost fire-fighting robot. Further explanation for each component is included below.

3.2.1 Arduino Microcontroller

The Arduino is an open-source electronics microcontroller (see Figure 3) [1]. This controller has embedded I/O support and a standard programming language based on C/C++. Our team chose this microcontroller because we were already familiar with it and found several example projects online. Compatibility between the Arduino, sensors, router was verified early in the project through unit testing. The Arduino connects and controls the motor controller and GPS sensor. These Arduino specific components are known as shields.

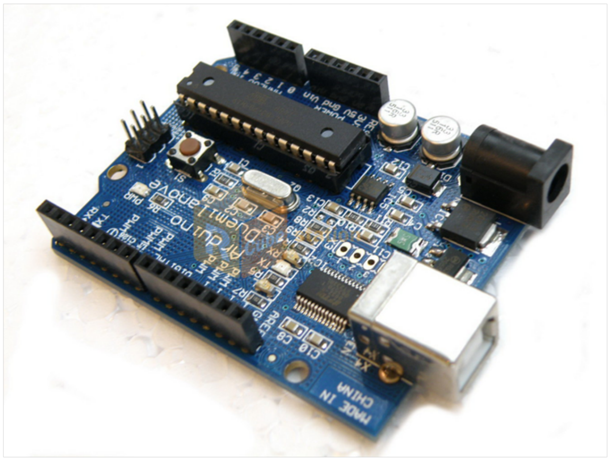

Figure 3: Arduino Microcontroller

3.2.2 Arduino Shields

There are four shields used in conjunction with the two Arduinos and a shield extender, which makes it possible to use more than one shield with the same Arduino at the same time. There are two motor controller shields, GPS shield, and ethernet shield. One motor controller shield is responsible for moving the vehicle [6] and the other is for engaging/disengaging the fire-extinguisher [7]. As expected the GPS shield determines the location of the robot and forwards this information to the GCS [8]. The ethernet shield connects the Arduino to the Linksys WRT54G Router [2].

3.2.3 Linksys WRT54G Router

The WRT54G router [5] is one of the most versatile routers in production (see Figure 4). Because of its popularity and efficiency, a large open-source community has supported embedded linux distributions as alternatives to the proprietary Linksys operating system. Installing OpenWRT or DD-WRT in conjunction with an Arduino Ethernet shield allows our robot to wirelessly communicate with the GCS [4] [3].

Figure 4: Linksys WRT54G Router

3.2.4 Panasonic BL-C1A Network WebCam

The Panasonic BL-C1A Network WebCam [11] (see Figure 5) was selected as the means for video feed because of the available documentation and positive reviews. This camera's popularity stems from its quality, low cost, and being "hacker-friendly," which are all key metrics for selecting a suitable WebCam for this project.

Figure 5: Panasonic BL-C1A Network WebCam

3.2.5 Xbox Controller

The Silverlight [9] front end empowers the user to control the modes and motion of the robot. A feature of using the Xbox controller [10] is the familiar controls and button mapping for the user. The features of both Silverlight and the Xbox 360 controller create a user experience that is both efficient and easy to learn. Because both of these user interface technologies are Microsoft, there is a seamless integration and extensive documentation available.

4 Implementation

There are several components and technologies that have been used to implement this solution. For clarity, the implementation will be discussed in two sections relating to the two parts parts of the system (Ground Control Station and robot).

4.1 Ground Control Station

The Ground Control Station (GCS) interfaces between the user and robot and contains a laptop, router, and Xbox controller. As indicated by the project requirements, the system must be easy to use, which is primarily done through the GCS (Figure 6 is a screen shot of the Silverlight user interface). The technologies selected for the user interface and communication with the robot were Windows 7, .NET Framework 3.5, C# web services, and XML messaging. The GCS user interface gets user commands from an Xbox 360 controller and provides feedback to the user through a Silverlight application. The Xbox controller generates commands based on user input through the joysticks and a button. The joysticks of the controller are used to drive the robot and by pushing/releasing a button the user engages/disengages the fire-extinguisher. The Silverlight web application shows the streamed video that is overlaid with the most current telemetry data. The signal strength and remaining battery life are indicated by the icons in the upper-right corner. In the upper-left corner, a globe icon can be pressed using the laptop to get GPS information. The arrow icon in the bottom-left corner high-lights the direction the robot is being driven. In the Figure 6 screenshot, all directions are gray, because the robot is sitting still. However, if the robot was to drive forward, the arrow pointing up would be highlighted green? to make it clear to the user which direction the robot is being told to travel. This sort of user feedback makes the system intuitive to use and minimizes the amount of training. The user interface is designed using a C#

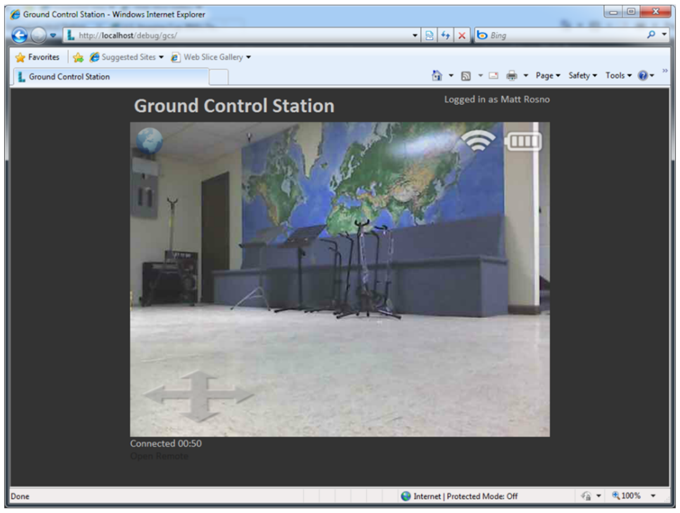

Figure 6: User interface caption

web service and a modular design. As such, the interface could be abstracted without changing the XML service or GCS. This would be applicable if an application was written to use a mobile devices to either control the robot instead of the Xbox controller or to provide the video feed or other feedback information to other fire-fighters.

4.2 Robot

The robot is implemented using simple open-source technologies. For communication, a modified Linksys WRT54G wireless router is used. This router receives the instruction from the GCS and forwards the commands to the microcontroller (see Figure 7 for a state machine representation of how robot recieves and processes requests). Once a command is processed, the router sends XML responses and telemetry data back to the GCS. The Linksys router also enables the use of a network web-enabled camera attached to the robot. This gives the user a live video feed and enables intelligent remote control.

The heart of the robot is a programmable Arduino microcontroller. Arduino is an open-source embedded system that provides a low cost interface between high-level software and the robots hardware [1]. The microcontroller provides a means of engaging the fire extinguisher, controlling the movement of the robot and providing communication between the various sensors and communication devices. The robot is constructed from scratch using a plywood framework and metal exterior (see Figure 8 to see the robot without metal exterior). The chassis provides a means of movement and a central location for attaching all

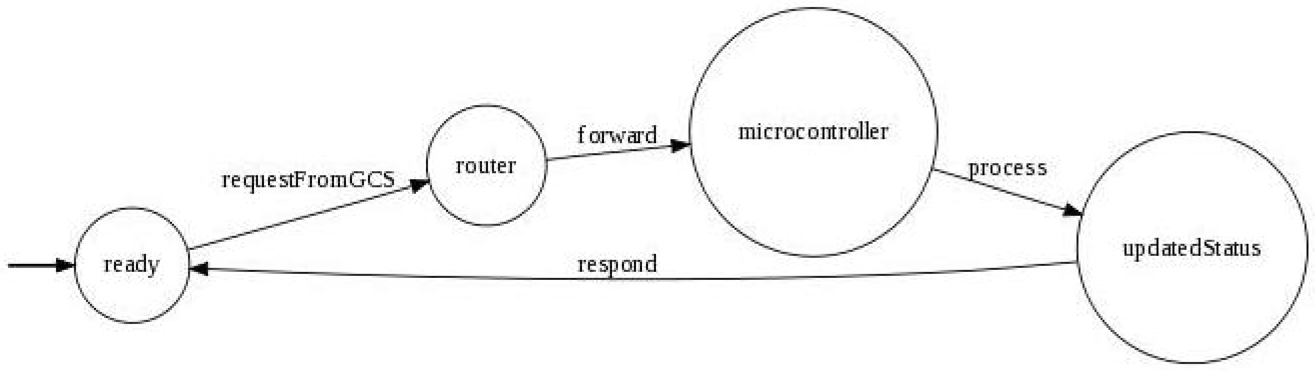

Figure 7: State Machine Representation of Robot Recieving and Processing Requests

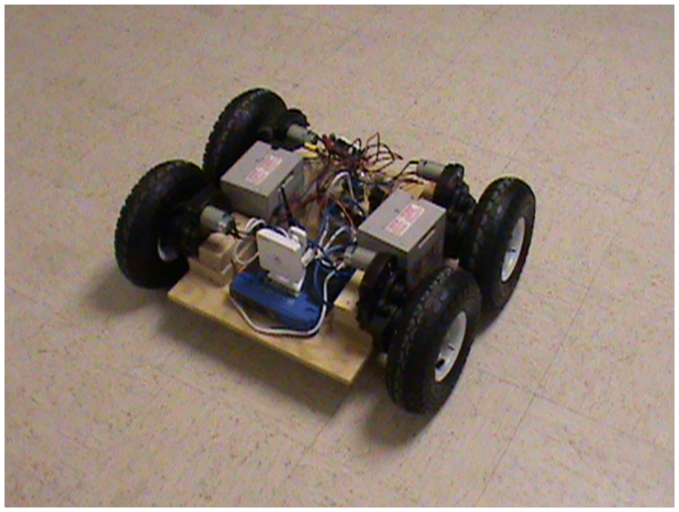

Figure 8: Robot Without Metal Exterior

of the sensors and electronics. Using plywood for the framework is desirable for its stability and strength. The weight and size of the chassis can be leveraged to help counteract the horizontal force of the extinguishing agent leaving the fire-extinguisher. The chassis uses four metal and rubber wheels for movement and steering. The robot turns by using skid steering, which is rated as one of the easiest way to implement steering. This works particularly well with the robots square-like shape. Skid steering is implemented by making the tires on one side of the robot move forward and the other side move backwards. The motors are controlled by a motor controller, which is attached to the Arduino. The robot is provided with location data via GPS signals interpreted by the GPS module that communicates with the microprocessor. The location data provides the robot with the ability to navigate to desired locations and report its location to the base station which is particularly applicable in the patrolling mode.
Two 12-volt DC batteries along with circuitry are used to power all the parts of our robot. This allows proper power distribution to different components. These batteries are the same used on Power Wheels cars. To protect the robots internals from conductive and radiant heat, we outfitted the robot with a metal covering. We bent and cut aluminum sheet metal to conceal the components and then riveted the pieces together to make a removable shell for facilitating maintenance.

5 Testing

We implemented several methods to test the mechanical design of the robot and the correctness and performance of the software. The mechanical workings of the robot are tested by manually controlling and observing the resulting activity. We were specifically interested in the robot driving in the expected direction based on the provided user inputs. For purposes of testing, we created a Java program that converted keyboard inputs to commands that were sent to the robot. This allows for mechanical testing without having to use the Xbox controller. Additional testing was done adding the Xbox controller to the mix. Sample test cases for the manual testing include cases such as when a user pushes both joysticks up (or pushing the up arrow key), it is expected that the robot will drive forward. Similarly, when the user pushes one joystick up and one down (or a left or right arrow key), it would be expected that the robot will drive in a circle. For these and other cases, our robot reacted as expected, thus passing all test cases. As previously mentioned, Arudinos use a language based of of C/C++. To unit test the software running on the Arduino, we created small test projects. A test project acted as a test harness and could test a specific piece of functionality such as the software controlling one of the Arduino shields. In addition to testing the correctness of the software running on the Arduino, we also tested its performance, looking specifically for memory leaks. We wanted to address efficiency concerns with all the processing done by the Arduino. To test for a memory leak in any given part of the system, we simply allowed it to run long enough (at least 128,000 times) that any leak would be detected. In our case, this translated to allowing the software on the Arduino to run for around half of an hour. Through this process, we found one memory leak with a library (wstring) that we were using. By using an alternative library, we were able to provide the same functionality, but greatly improve the performance. Testing the major components separately facilitated the debugging of the system. This included testing the GCS's Silverlight user interface and C# backend code without needing to be connected to the Arduino. We accomplished this by creating a web service to mock the interaction of the GCS and the Arduino. Using this web service, we conducted manual unit and system tests. This also allowed us to concurrently develop the GCS and Arduino code.

6 Results

Our project resulted in successfully creating a robot that could operate a fire extinguisher and be controlled remotely, via Wi-Fi communication. The robot has ample steering control and maneuverability, torque, and weight to navigate through non-ideal terrain and carry a payload large enough to allow for fire-fighting ability. The metal shielding makes the robot resistant to conductive and radiant heat and helps protect the components inside. This implementation is, however, not incredibly heat-proof. The rubber wheels and Plexiglas camera window are prone to melting and the components inside will also melt once the metal shielding is heated enough.

The robot has experienced a battery life of around 3-6 hours of activity, as seen in Figure 9. A maximum of about 13-14 hours can be achieved in an immobile state. These values were obtained by measuring the amperage drawn from the battery at each speed and using the watt-hours of the battery to determine the life of one charge. The watt-hours of the battery were calculated from the batterys voltage and the amp-hours, which were marked on the battery.

The plan for the Ground Control Station was to use an Xbox 360 controller for controlling the robot. However, after integrating the controller, we found it to be slightly buggy and unreliable. Input from the controller travelled through the ActiveX control, Internet Explorers JavaScript engine, the Silverlight control, the GCS web server and finally to the router and Arduino on the robot itself. Through this chain of handlers, the input sometimes was delayed and miscommunicated. The Arduinos Ethernet shield could also become overloaded by too many control requests.

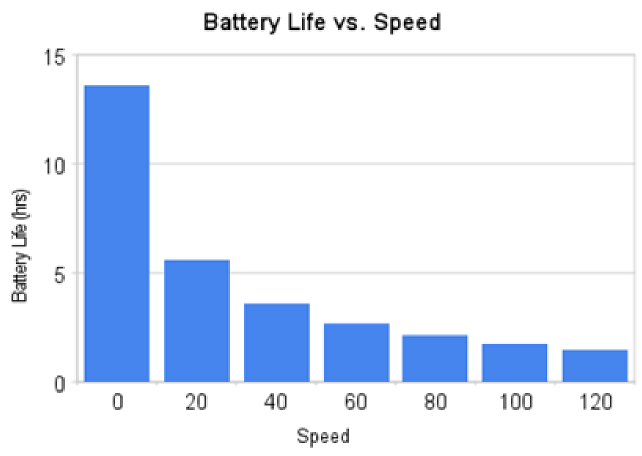

Figure 9: Average Battery Life Based on Speed Robot is Run

One element of the original design that we did not implement was the maps integration with the GPS shield. We encountered difficulty in providing the robots location on a satellite map. This was mainly because the maps services we intended to use require our ground control station or client machine to have an external internet connection while connected to the robots network.

7 Analysis of the Cost

Table 1 shows the breakdown of costs for the project.With a total of $555, this robot is much more cost-efficient than similar models that we found. A similar Raytheon product costs upwards of $8,000. This analysis does not include cost of labor; we put an estimated 200+ man hours into the robot and control station (development, construction, debugging).

8 Difficulties Encountered

In the development of this project, we have had to overcome several difficulties, including:

- Building a chassis
- Getting the router to communicate with the Arduino

Component	Price
Core	
Arduino Microcontroller	$30
Arduino Ethernet Shield	$45
Arduino Motor Shield	$20
Arduino GPS Shield	$30
Pololu Trex Motor Controller	$100
Subtotal	**$225**
Chassis	
4 x 12V Motors and Gear Assemblies	$90
4 x Tires	$20
Misc (bolts, washers, ...)	$10
Subtotal	**$120**
Fire Fighting	
Fire Extinguisher	$20
12V Gear Head Motor	$14
2 x 6mm Shaft Clamp	$12
Misc	$7
Subtotal	$53
Other	
Linksys WRT54G Router	$30
Linksys 160N Router	$35
Panasonic BL-C1A Network	$92
Subtotoal	$157
Total	**$555**

Table 1: Monetary Cost Analysis

- Arduino only supports one serial connection when using the standard serial library
- Pin problems
- Integrating Xbox controller to Silverlight

8.1 Router Communication

When trying to get our router to work with the Arduino, we encountered several hurdles. The serial port that we connected to the router did not work; we could not compile the OpenWRT SDK or run scripts to set the correct baud-rate. To remedy this issue, we opted to use an Ethernet shield to create the connection between the Arduino and router.

8.2 Serial Connection

Another issue regarding the serial connections was that the Arduino standard serial library only supports one serial connection. This means that a separate Arduino would be required to run each shield. After some research, an alternate open-source library was found (NewSoftSerial) (ref). This library allowed us to create the necessary multiple serial connections.

8.3 Pin Problems

Our decision to use the Ethernet shield resulted in multiple shields trying to use the same pins. The Ethernet shield and one of the motor controller shields required the use of the same pins. One option was to remove the overlapping pins from one of the shields and wire them to different output pins on the Arduino. Our team decided that this would be difficult and would limit future extendability. Therefore, we decided to use a second Arduino to run one of the motor controllers. This implementation was easier and allows for future extendability.

8.4 Xbox Integration

Integrating the Xbox controller with the Silverlight was more difficult than expected because we were unable to find a standard ActiveX control on the Internet that had this functionality. Our team was able to gain access to Agile Sport's existing code (an ActiveX control and installer package), and we were given permission to use and alter the code so we did not have to start from scratch.

References

[1] Arduino. http://www.arduino.cc/en/Main, 2010.

[2] Arduino ethernet shield. http://www.arduino.cc/en/Main/AruinoEthernetShield, 2010.

[3] Dd-wrt. http://www.dd-wrt.com/site/index, 2010.

[4] Openwrt: Wireless freedom. http://openwrt.org/, 2010.

[5] Cisco. Linksys wrt54g: Wireless-g router. http://homestore.cisco.com/en-us/Routers/Linksys-wrt54g2-wirelessg-broadband-router_stcVVproductId53779504VVcatId543809VVviewprod.htm, 2010.

[6] Pololu Corporation. Motor controllers: Pololu trex dual motor controller dmc01. http://www.pololu.com/catalog/product/777, 2010.

[7] Limor. Motor shield: Servos, steppers and dc motors!

[8] Limor. Gps shield: Diy location and data-logging. http://www.ladyada.net/make/gpsshield/, April 2010.

[9] Microsoft. Microsoft silverlight. http://www.silverlight.net/, 2010.

[10] Microsoft. Xbox 360 controller for windows. http://www.microsoft.com/hardware/gaming/ProductDetails.aspx?pid=091, 2010.

[11] Network Webcams. Panasonic bl-c1a - overview. http://www.networkwebcams.com/product_info.php?-products_id=363, 2009.