MOBILE ROBOTIC PLATFORM

By

Chris MacKenzie

Senior Project

California Polytechnic State University

San Luis Obispo

2012

Table of Contents

I. Introduction

Robotics is a field that is becoming more and more widespread and it can cover a lot of different disciplines and even cover a wide array of topics within one discipline. Because of this, designing a robot can have a lot of challenges to overcome and new things to learn, making it an educational yet fun task. Since many people tend to have a specific concentration for what part of a robot they would want to work on, especially when senior project rolls around, it is nice to not have to deal with all areas of design.

The purpose of this project is to develop a working mobile robotic platform with various sensors and capabilities so that if someone wants to focus on one aspect of robotics, more in a software sense, they can have a platform to develop on. This platform has sensors to detect objects as well as a camera to see in front of the robot, leading the way to the ability to drive autonomously. To go with the camera, there is also a video transmitter/receiver pair to be able to offload the vision processing and just have to send data back. With the structure of the robot and the nature of the microcontroller board that is mounted on it, it also leads to being able to mount additional sensors for whatever might be needed with this new focus.

II. Requirements and Specifications

There are several requirements that we came up with in the broad sense to make this platform something that could be expanded upon in the future. These considerations included making a platform that had the necessary hardware to control the robot remotely as well as move autonomously, if coded for that purpose.

- Design/find a platform that can maneuver around or over various obstacles.
- Have the ability to add or remove sensors, actuators, etc. easily.
- Ability to detect close-by objects for object avoidance.
- Camera mounted on the platform.
- Send video wirelessly back to a computer for off-board processing.
- Ability to send commands to the robot and get sensor data back.
- Everything should be Mac compatible to run on Dr. Seng's computer.

III. Design

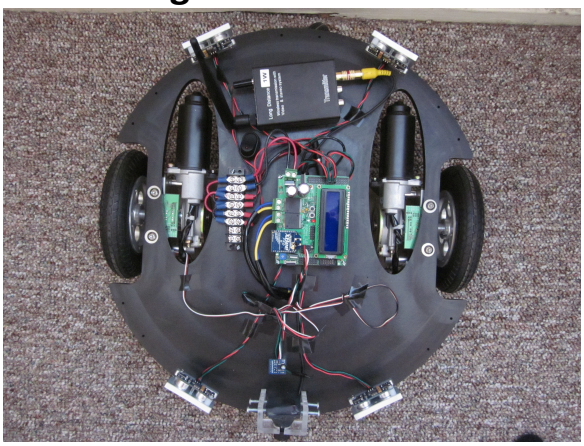

Figure 1: Top View of Completed Robot

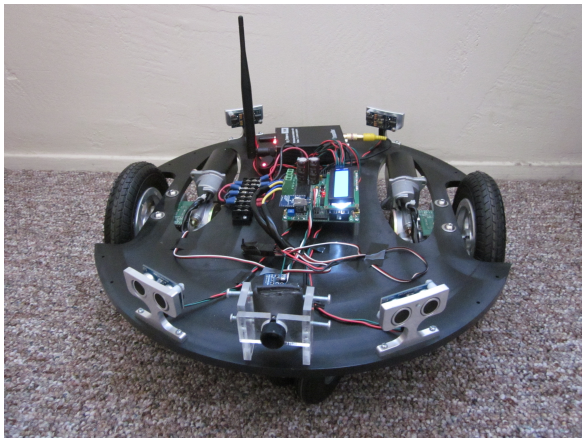

Figure 2: Angled View of Completed Robot

System Architecture/ Hardware Block Diagram

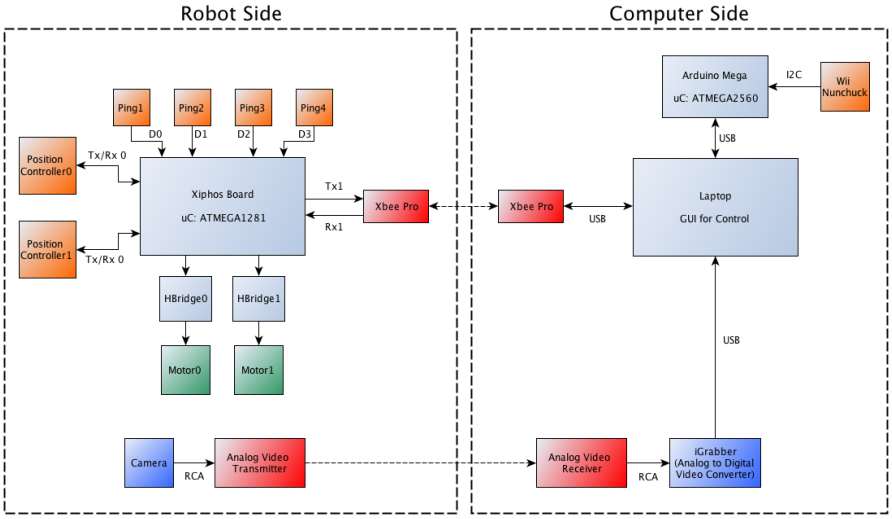

Figure 3: Hardware Block Diagram

The two main controllers for the robot are the Xiphos Board on the robot side and the computer on the computer side.

Xiphos Board

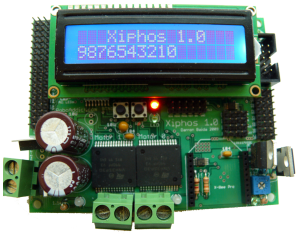

Figure 4: Photo of Xiphos Board

The Xiphos Board is a robot controller board featuring an ATMEGA1281 microcontroller, as well as hardware to program the board, run motors, and interface with sensors or actuators. The important features for this project are the XBee headers to connect an XBee for wireless serial communication, the two H Bridges with terminals to attach and run the two motors on the robot, the broken out digital header pins to interface the Ping sensors, broken out UART pins to interface the position controllers, and the broken out I2C header pins to interface modules such as the compass.

Computer

The computer is the one that does the overall processing and gives the Xiphos Board commands. To do this, the computer has an XBee module connected to USB for getting sensor values back and sending data to the robot, an analog to digital video converter to get a camera feed, and for another control option, has an Arduino Mega 2560 attached to USB to convert I2C values from the Wii Nunchuck to motor commands.

Connected to the two main processors are four Ping sensors, two motors, two position controllers, an XBee module for each side, the Arduino and nunchuck, as well as the camera sending video feed over the wireless transmitters which is run through the iGrabber to convert the video to digital.

Ping Ultrasonic Distance Sensor

Figure 5: Photo of Ping Ultrasonic Distance Sensor

The Ping sensors are ultrasonic object detectors, so they send out an ultrasonic pulse then see how long it takes to get back, then convert that to a distance. These are connected to the Xiphos Board and used to stop the robot if it is about to run into something.

Motors with Position Controllers

Figure 6: Photo of Motors with Position Controllers Attached

The two 12V motors have a position controller attached to them, as seen in Figure 6, with a flywheel that has 36 encoder positions per revolution. The position controllers then keep track of the encoder counts and using a single-wire UART, the Xiphos Board can get a position value or a speed value.

XBee Pro Modules

Figure 7: Photo of XBee Pro Module

There is an XBee module connected to the Xiphos Board and one connected to the computer through a USB breakout board. This opens a serial line of communication between the robot and the computer using 802.15.4, enabling them to talk to each other. The XBee's range according to the datasheet is 1-mile line of sight, 300ft in an indoor/urban environment.

Arduino with Nunchuck

Figure 8: Photo of Arduino Mega and Wii Nunchuck

The Wii Nunchuck is connected to the Arduino through an I2C communication, and the Arduino is then connected to computer to give motor commands to the GUI.

Camera, Wireless Transmitters, iGrabber

Figure 9: Left to Right. Camera Mounted to Robot, Transmitter/Receiver Pair, iGrabber

The camera has an analog video output through an RCA cable, which is connected to the 1W 2.4GHz RF video transmitter/receiver pair, which is in turn connected through an RCA cable to the iGrabber, which converts the video to digital and outputs it over USB to the computer. The given range for the transmitter/receiver pair is 1000 meters line of sight, and 80 meters with walls.

Mechanical Design

The brunt of the mechanical design was done for me by getting a premade platform from a company called Parallax. This included the platform base, the battery shelf, the motors with mounting blocks and air-filled tires, as well as the Ping sensors and their protector stands. The platform base is shown below in Figure 10. The other main mechanical design was a mount for the camera, which was contributed by Troy Weber, a mechanical engineering student at Cal Poly. The only additional mechanical considerations were bolting down the Xiphos Board, Ping sensors, and power terminal block, velcroing the transmitter down, and bungee cording the battery on, for easy removal.

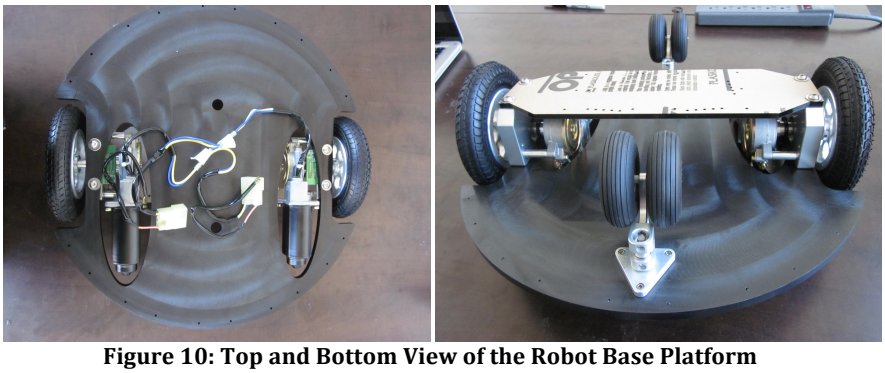

Figure 10: Top and Bottom View of the Robot Base Platform

Software

The software is mainly broken up into three entities: Xiphos board code, a C# GUI to control the robot from the computer, and Arduino code to interpret data from the nunchuck.

Xiphos

The Xiphos Board is done as normal microcontroller code, meaning that most of the code is in an infinite loop, so the program never ends. Shown in Figure 11 below is the flowchart for the Xiphos code.

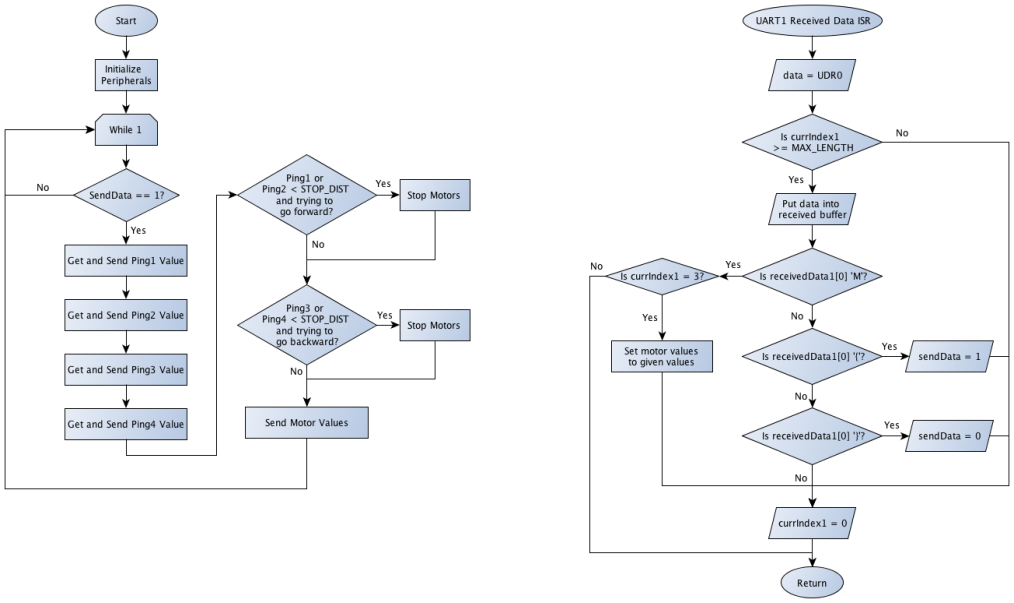

Figure 11: Flowchart for Xiphos Software

Main Function

Before the loop, all the peripherals are initialized, including the timers set in PWM mode for the motors, setting up the UARTs, initializing I2C, and setting up the timer used for measuring Ping pulse lengths. In the infinite loop, it first checks if sendData is equal to 1, if it is it gathers the sensor and data and transmits it back to the computer, checking to see if something is too close to a Ping sensor, and if something is too close it stops the motors. If sendData equals 0, then it just goes back to the beginning of the loop and circles until the computer tells it to start sending data.

Serial Library

The serial library contains functions for communicating over UART0 and UART1 on the Xiphos board. The library is more general, but the main functions used are the initialize functions, transmit functions and the ISR for receiving data. The initialize sets the baud rate into the register as well as enabling the Tx and Rx and the interrupts that go along with them. The transmit functions used take the given data and puts it into a transmit buffer than enables the Tx complete interrupt. The ISR for Tx fires when the device is done transmitting its previous

value. It then puts the new value from the buffer into the UDR register to be sent, provided there is data in the buffer. Otherwise it turns the interrupt off.

The data received ISR executes whenever the UART1 gets a packet of data from the XBee. As shown on the right in Figure 11, the ISR first stores the value in the received data register into a variable. If the current index in the receive buffer is at the max limit, set it back to 0 ignoring the data. Otherwise, put the data into the received buffer at the current index, and check the first value in the buffer for a command. If the first is M, see if the index is three meaning the full motor command of M## has arrived, if the index is three, set the motor value, otherwise just return. If the first value is an open curly brace, {, then set sendData equal to 1 to tell the main loop to start sending data. If the first value is a closed curly brace, }, set sendData to 0 so the main loop will stop sending data. If the value at index 0 in the receive buffer is not one of these commands, set the index back to 0 to get new commands.

Ping Library

This library contains the function to initialize the Ping sensor by setting up the timer that is used to measure the pulse given, and the function to get the value from the Ping sensor. To get the value, the function pulls the signal line low for 2 microseconds, then sends a high pulse for 5 microseconds, then pulls the line low and waits for the sensor to pull the line high. It then keeps track of how long the line is high for, and then converts that microseconds value to inches to return.

Compass Library

This library contains functions to change all the different configuration parameters, as well as the functions that were mainly used; such as initialize, get raw values, compute angle, and one to get the value converted into degrees already. The initialize function sets the initial parameters given by sending the given parameters to specific registers on the compass.

The function to get the raw values from the compass just goes through the six registers on the compass getting the values, then combines the three pairs into the three 16 bit data for x, y and z. This function takes in pointers to x, y and z variables to assign data to.

The compute angle function takes two coordinates from the raw values and using the arctangent to find the angle the compass is pointing at, shifting it into a 0-360 degrees range.

The last main function used is getting the compass angle in degrees, which calls get raw values and stores the data, then calls compute angle and returns back the end value.

Position Controller

The position controllers run on a single wire UART, so this library has an initialize function to change the baud rate and configuration of UART0. The request speed and position functions turn the Rx off to send data, so it doesn't get read by the same microcontroller sending it, then send a request for the desired value, then once that value is transmitted it turns the Tx line off and Rx back on. Once the data comes back it fires the ISR for UART0 receive and that parses the two 8 bit numbers into a signed 16-bit number and stores the value into an array. The clear position function does the same, except it doesn't get any value back from the position controller.

Computer C# GUI

The GUI starts out by calling a constructor that initializes the layout that is seen in Figure 12 below. This layout includes two tabs, one for controlling the robot and getting feedback, the other for setting up the serial ports (and if in Windows the camera to be displayed). Everything else is run off of events such as when a button is pressed, a combo box is dropped down, a key on the keyboard is pressed, serial data is received, or a timer fires.

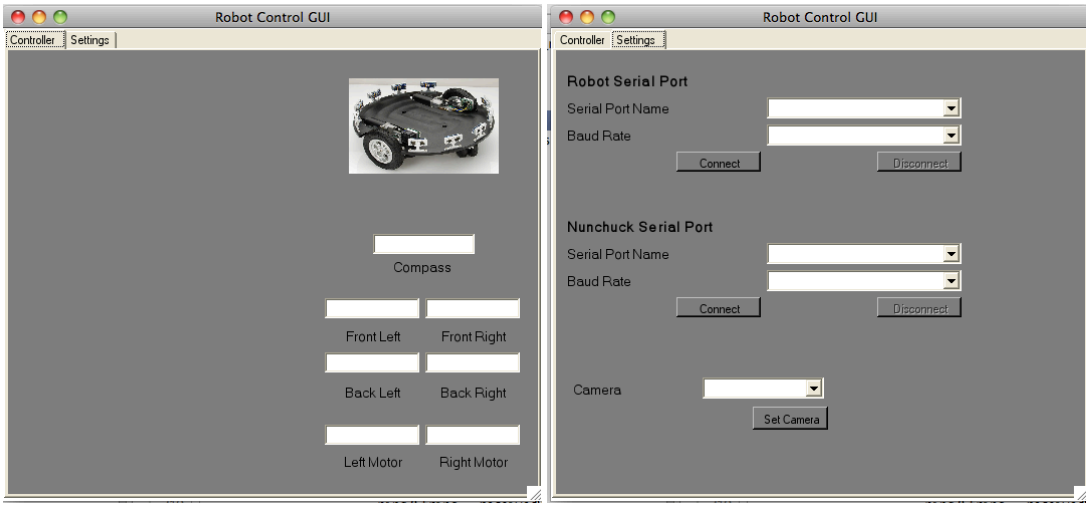

Figure 12: Controller Tab and Settings Tab in C# GUI

Controller Tab

The controller tab, as seen on the left in Figure 12 above, is where you enter commands to control the robot by pressing keys and get feedback from the robot. There are seven text boxes that update when the GUI gets data back from the robot. In Windows, the GUI uses a data received handler for the serial port that gets called every time a byte of data gets read by the serial port. In Mono in Mac that isn't implemented, so there is a timer that interrupts approximately every microsecond to check if there is data to be read. For either, when there are three bytes of data to be read, it figures out which value it is being sent by the first character, then takes the values given in the second two to print the data into the text box. For the Ping sensors, if the value is less than the STOP_DIST the box turns red to show which sensor it too close to something.

For robot control, you can either setup the serial port to read values from the nunchuck through the Arduino and use the joystick on that, or you can use the keyboard. For the keyboard, there is a listener on the controller tab to get keys that are pressed. For the keys, "w" corresponds to forward, "a" to left, "s" to backward, and "d" to right. When pressed it sends a motor command to run the motors at the current speed set, which can be change to one of 4 values by pressing 1, 2, 3, or 4, 1 being the slowest, 4 the fastest. Any other key pressed on the controller screen will send a stop motors command to the robot.

Settings Tab

The settings tab, as seen in the right part of Figure 12, is where the serial ports are set up, and if in Windows, the camera is set up too. The first group of items on the settings tab is for setting up the serial port that talks to the robot. There is a port name drop down box that when clicked will refresh itself with the available serial ports, COM* for windows, /dev/tty.usbserial* for Mac, when an item is clicked it sets that as the text in the box. Below that is a baud rate box so you can choose a specific baud rate for the device. The baud rates are a predetermined list of the common values, and when one is clicked it sets that in the box. Once those have been set, if Connect is clicked it uses the name from the port name box and the baud rate from the baud rate box to open the serial port. If opening it fails for any reason, a box will pop up saying there was an error. Once the Connect button has been clicked, provided no error, that button will disable and the Disconnect button will enable, which when clicked closes the serial port. The next group is the serial port for the nunchuck, which is set up the same as the robot serial port but for the second serial port object instead.

If the GUI is run in Windows, the last group is a drop down box that will update with the available cameras that it recognizes so the user can choose

one then click Set Camera to change what video is feeding into the controller tab.

Arduino

The Arduino, same as the Xiphos board, runs in an infinite loop since it is a microcontroller. As seen in Figure 13 below, it first initializes the serial port that it will send data over, the I2C communication that it will use to talk to the nunchuck, and sends the initialization sequence to the nunchuck. Once it's in the while loop, if readNunchuck is 0 it will just cycle back and keep checking. If readNunchuck is equal to 1, it gets the raw data from the nunchuck. It then checks if the joystick is pressed forward or backward, given some buffer for noise in coming back to the home position. If so, it sends a motor command to go forward or backward using the eight-bit analog y-axis value given from the joystick. If that isn't true, it checks if the joystick is going left or right using the same buffer. If it's going left or right, it sends a motor command sending the x-axis value to one motor and 255 minus the x-axis value to the other motor. Otherwise it sends a stop motor command. After any of the options, it will send a request for a new set of data from the nunchuck, then delay 50 milliseconds before repeating the loop.

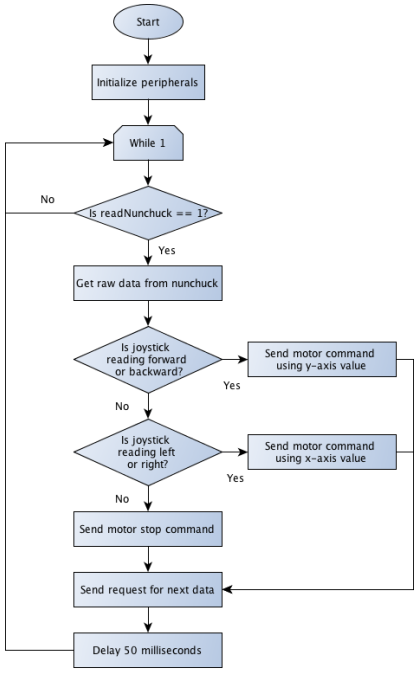

Figure 13: Arduino Code Flowchart

IV. Conclusions

Overall, this project was a fun project and a challenging task, even in ways that I didn't think would be challenging to overcome. The project was originally going to be to quickly interface the sensors and get it driving remotely to test it's capabilities, then feed the video into OpenCV and do image processing to have it drive itself. First problem with that was some of the sensors didn't want to behave as needed, and a lot of time was spent trying to get them to function. The main culprit of this was the compass module. The code was written fairly quickly for it, and I started getting values from it, but the values weren't what they should be. The 0 degrees was consistently pointing north, and it would go all the way to 359 degrees before rolling over, but turning the compass 90 degrees read about 60 degrees, turning 180 degrees read about 195, and turning it 270 read about 310. At first I hoped that it was off by a linear amount to be a simple scaling fix, but it wasn't a consistent error. After trying to read the compass in different ways, doing the self-test for correction and even having other members of the robotics club play with similar compasses, we could not get correct data, so the compass was abandoned. I had expected that to take a short time to interface then move on, but I ended up spending more time than I should have trying to get it to work

The next time consuming pitfall I had, was I made the GUI quickly, in about a week, and got it to function well in Windows, at least for the base needs of it, and went to transfer it into Mac and hit many issues. The most major issue was that the camera libraries were written specifically for Windows, so they would not work with Mac. So I tried to find a video library that would work with a C# GUI in Mac, and tried some that said they should work, but couldn't get any to work. That was when I decided that I had spent enough time on that task, and the video on Mac would just be run in another program. The other main issue that I had, which ended up with an easy, though not the most ideal, fix, was that for the SerialPort class, the data received handler wasn't implemented in Mono, so I had to create a timer object and have that interrupt every millisecond to check for data on the serial port line.

To try to get it set up to move onto the next stage of autonomy, I also was trying to get the video feed coming into the computer to be read by OpenCV. This task proved a lot more difficult than originally planned. OpenCV wouldn't recognize the iGrabber, or the Dazzle that we tried, as a video device, and therefore wouldn't get the stream of images from it. I tried to find a work around or tell it exactly where to look for the video, but was ultimately unsuccessful.

These more major issues, with other minor issues, definitely showed that some of the "easy" things in the project could always turn out to be more work and take more time than originally planned. One of the reasons that is true, is because hardware doesn't always work like you think it will all the time. For instance, I got the position controllers interfaced and was getting sensible data back from them,

and I was going to use them to help correct one motor overpowering the other, but by the time both were integrated and started testing with values from them, one of them was resetting my Xiphos board. So one thing to keep in mind is that extra hardware is a nice thing to have to see if there is something wrong with that specific device, or if there is inherently something wrong with the system.

Overall, the basic goals of the project, to create a platform that will move when commanded and give video feedback, were met. So, someone coming along in following years can use this platform to develop different algorithms and test them.

Appendices

A. Senior Project Analysis

Student: Chris MacKenzie.

Advisor: Dr. John Seng

Summary

This project was a robot that would drive around via remote control from a computer. The robot is controlled through a GUI on the computer that sends commands to the robot wirelessly as well as getting feedback from the sensors and displaying it on the screen. Using the sensor values, the robot can also stop itself if it is going to hit an object. In addition to sensors, there is a camera feed getting sent back to the computer to watch where the robot is going. The basis for this project was to create a robotic platform that can be expanded upon in the future for other senior projects.

Primary Constraints

The project was somewhat limited by the hardware that I received through a sponsorship and the ability to find other hardware that would do what I wanted. I got a generous donation of parts from Parallax, but that limited the devices I could use for things such as the compass, which I never got functioning correctly, because I didn't want to spend too much on other sensors when I had a good amount donated. One of the other constraints that took more time than originally thought was to make the GUI and everything Mac compatible to run on the computers that will run the robot. So, the GUI had some issues transferring to Mac and took time to get at least mostly functional. Lastly, the hardware we could find to convert analog video to a digital USB connection was not recognized by OpenCV, so I spent a lot of time trying to get it to recognize it.

Economic

This project was mostly sponsored by donations from Parallax Inc., and the parts that weren't given by them were bought through the CSC department. The final cost was as expected from the beginning. The only things not accounted for in the beginning were connectors, such as nuts and bolts, which are cheap. At the bottom is a bill of materials for the cost of each part. The project does not earn any money, so therefore nobody profits of it, other than new knowledge. After this initial platform is built, there shouldn't be much in the way of cost. The main cost from this point on would be whatever sensors people want to add for a new project, or if one of the current sensors breaks and needs to be replaced. Most of the components on this project should last until they are long outdated, provided

proper use. The only component that will probably fail before the others would be the battery, and the only other imminent replacement might be replacing the microcontroller to something with more capabilities.

Manufactured on Commercial Basis

The base platform of this project is already manufactured on a commercial basis by Parallax, who I got it donated from. The main difference is that this platform has the sensors mounted, and has a microcontroller, battery, camera and video transmitter/receiver pair attached. Depending on the need for a platform this size, this could easily sell over 1000 units per year if the product was advertised enough. The manufacturing cost is unclear, because I don't know the cost that it takes Parallax to manufacture the parts I got from them, but I would guess overall this project would take about $500 to manufacture. This product would probably sell for about $900, making the profit for each year about $40000.

Environmental

This project would mainly affect the environment in production, not in use. The main components used are high-density polyethylene for the base, acrylic for the battery shelf and camera mount, and the printed circuit boards (most likely FR4), solder, the electrical components, and the metal in the nuts and bolts. The platform base is made using a CNC machine, which cuts away parts of the polyethylene and might be scrapped and thrown out, where as the acrylic was cut using a laser cutter. So it takes quite a bit of electricity to cut out the pieces used, and to run the reflow machine used to solder the components onto the various PCBs. Most companies sell internationally, so the solder used for most manufactured products are leadless solder, but the solder used to solder components onto the microcontroller board, done by hand, contains lead, but there is such a small amount used.

Manufacturability

This product should be fairly straightforward to manufacture. It would take time and effort with all the components used having different PCBs and mounting methods that need to be fabricated or cut out and drilled, but overall should be straightforward to manufacture once someone gathers all the parts.

Sustainability

This device is fairly easy to maintain. The electrical components should last a while, even with heavy use, and the battery will last a while. I think the battery would be the first thing to go, and that will have to be recycled and dealt with carefully because it is a sealed lead acid battery. So, the amount of batteries needed over the years would be the main concern. The platform itself will last a really long time, and most of the other components are small, so if they do need to be replaced it isn't using much in the way of resources.

Ethical

The main ethical concern would be in how the robot is used. It is a remotely controllable robot with a camera on it, so the main thing is that people don't use this to spy on others.

Health and Safety

Regarding health and safety, the sealed lead acid battery could be a cause for a safety issue if something connects the two leads together and leaves it like that or punctures it in any way. Batteries in general need to be treated carefully or they could be dangerous. Another safety issue could be if the SLA battery is replaced with LIPO, because LIPOs can explode if impacted or puncture. The only other safety issue would be watching where it is driven, making sure not to run into people.

Social and Political

The only social or political issues that I can see this project bringing is if someone expands upon it in the future to make it autonomous. With something this size it shouldn't cause a stir or make a difference, but autonomy in general when it comes to object driving around could need to be regulated or monitored, especially if done poorly.

Development

Taking on Cal Poly's "Learn by Doing" philosophy, I learned to make a C# GUI by making one. I have coded a Java GUI for one class, and coded in Java, C and C++, but I had never coded anything in C#. Luckily this is really similar to Java in a lot of ways and Visual Studio is a good tool to use when starting to learn C# GUI design because you can drag and drop components and it updates the code, then you can go back and modify code and functionality. The main reference for learning C# was the MSDN library for C# at **http://msdn.microsoft.com/en-us/library/ms123401**.

The other main thing I learned by doing this project was how to edit the bootloader that is on the ATMEGA1281 chip on the Xiphos board to make it able to program over the XBee plugged into UART1 as well as over USB. Previously I thought the bootloader was magic that took my program and put it on the board, but now after looking through it and modifying it I realize that it's not.

B. Acknowledgements

I would like to acknowledge the great sponsorship of Parallax Inc. for donating most of the parts used for this project. Their total donation of parts amounted to a value over $700.

C. Bill Of Materials

Table I: Bill of Materials

Part	Quantity	Cost	Donated?	Total Cost
Robot Base Kit – Black	1	$49.99	Yes	$0.00
Motor Mount and Wheel Kit with Position Controller	1	$299.99	Yes	$0.00
Caster wheels	2	$39.99	Yes	$0.00
Ping Sensors with Mounts	4	$39.99	Yes	$0.00
XBee Pro 60mW Wire Antenna	2	$32.00	Yes	$0.00
XBee USB Adaptor Board	1	$21.99	Yes	$0.00
Compass Module	1	$29.99	Yes	$0.00
Gyroscope Module	1	$29.99	Yes	$0.00
Webcam	1	?	Yes	$0.00
iGrabber RCA to USB Converter	1	$24.99	No	$24.99
RF Video Transmitter/Receiver	1	$49.99	No	$49.99
RCA Coupler	1	$3.79	No	$3.79
8 Position Screw Terminal	1	$3.19	No	$3.19
Bag of Spade Plugs	2	$2.19	No	$4.38
#4-40 ¾” Nuts and Bolts (6 pack)	2	$0.98	No	$1.96
#4-40 3/8” Nuts and Bolts (6 pack)	1	$1.18	No	$1.18
12V 3.4AH SLA Battery	1	?	Yes	$0.00
			Sum Total Cost	$89.48