

## Project Final Report

# Arduino-based Object Detection System

April 27, 2010

Frank Ervin

### ***Project Abstract***

For some time, I have been interested in making some sort of robot based on the Arduino platform. This project is the first phase of this longer-term hobby project. The goal is to develop an object detection system which could be retrofitted to a semi-autonomous robot up the road.

The system leverages an ultrasonic range finder to detect obstacles located in proximity to the apparatus. The ultrasonic sensor is mounted on a sub micro servo in order to have a sweeping perspective. A Passive Infrared (PIR) sensor is also used to activate the SONAR when motion is detected.

The apparatus sends sensor readings and distance measurements over 802.15.4 to a computer for graphical display. Code running on the microcontroller generates audible alerts using a serial-driven voice synthesizer chip attached to a small speaker.

### ***Status***

This project has been completed successfully, and my goal of integrating all of the underlying technologies has been met. The apparatus is able to take SONAR readings at commanded servo angles and transmit them wirelessly to a Processing sketch, where they are visualized for simple analysis. The device is also able to talk and announce basic status messages and object detection events. In the sections that follow, I will describe various outstanding areas that could stand to see some further refinement. Two specific areas include XBee wireless programming and finding a more robust power supply.

### ***Specification***

#### **Hardware**

I built this apparatus around the Arduino Duemilanove microcontroller, namely because I had already purchased one and had some experience interfacing with it and wanted to learn more. The Arduino is an open microcontroller platform that uses an AVR-C derivative known as Wiring. The availability of shields for prototyping, SpeakJet, and XBee made for a fairly tidy package for this project.

## Project Final Report

For the SONAR device, I used a Devantech SRF10 ultrasonic sensor mounted on a Grand Wing Servo (GWS) sub micro servo. The SRF10 is mounted using two screws and the SRF10 mounting kit. I ordered two servos, which was handy because they each only shipped with one screw of the proper size to mount the servo horn. The circular horn was a near-perfect match to the mounting kit, but mating the two took a bit of patience as the parts and screws are really small.

The SRF10 uses an I2C interface, which means that communications with the device leverage analog pins 4 and 5 in alternate function I2C mode. When connected in this way, the `Wire` library can be used to communicate with the sensor on the Arduino's I2C bus.

The project also uses the Arduino's digital output capabilities for moving the servo on which the ultrasonic sensor is mounted. The `Servo` library does a good job of abstracting the Pulse Width Modulation (PWM) which happens behind the scenes to command the servo.

I also used a digital input for reading the state on a Parallax Passive Infrared Sensor (PIR). This simple device takes the pin high when it senses motion within its range.

To enable speech, my Arduino code communicates with a Magnevation SpeakJet via a software serial connection. I used the `NewSoftwareSerial` (<http://arduiniana.org/libraries/NewSoftSerial>) library instead of the built-in `SoftwareSerial` library. It is leaner and capable of higher speeds.

The XBee interface to the computer is simply a serial port replacement for taking sensor data off of the microcontroller. I used two Digi XBee Series 1 Pro chips for this purpose.

### Wiring Code

In order to make all this work hardware work together, I wrote four libraries. All Wiring code can be found in the Attachment for this paper. Wherever prudent, each library has `#define` statements for various configuration elements. This makes the libraries easily reusable. The library names and relevant configuration elements are articulated below:

The `pir_controller` is used to initialize the PIR and wait for a motion detection event:

```
#define PIR_PIN sets the digital pin number for PIR.  
#define PIR_CALIBRATION_TIME sets the time to wait for the PIR to calibrate.
```

The `servo_controller` is used to initialize the servo and move it as needed.

```
#define SERVO_PIN sets the digital pin number for the servo.
```

## Project Final Report

```
#define MIN_DEGREE sets the minimum degree which the servo can be moved.  
#define MAX_DEGREE sets the maximum degree which the servo can be moved.  
#define DEGREE_STEP sets the degrees which the servo moves when panning.  
#define MID_DEGREE sets the degree in which the servo is in the middle of a sweep.  
#define SWEEP_RESOLUTION sets the total number of steps in a complete sweep.
```

The `sonar_controller` is used to initialize the SRF10, get range values, and coordinate sweep rotation by calling the `servo_controller`.

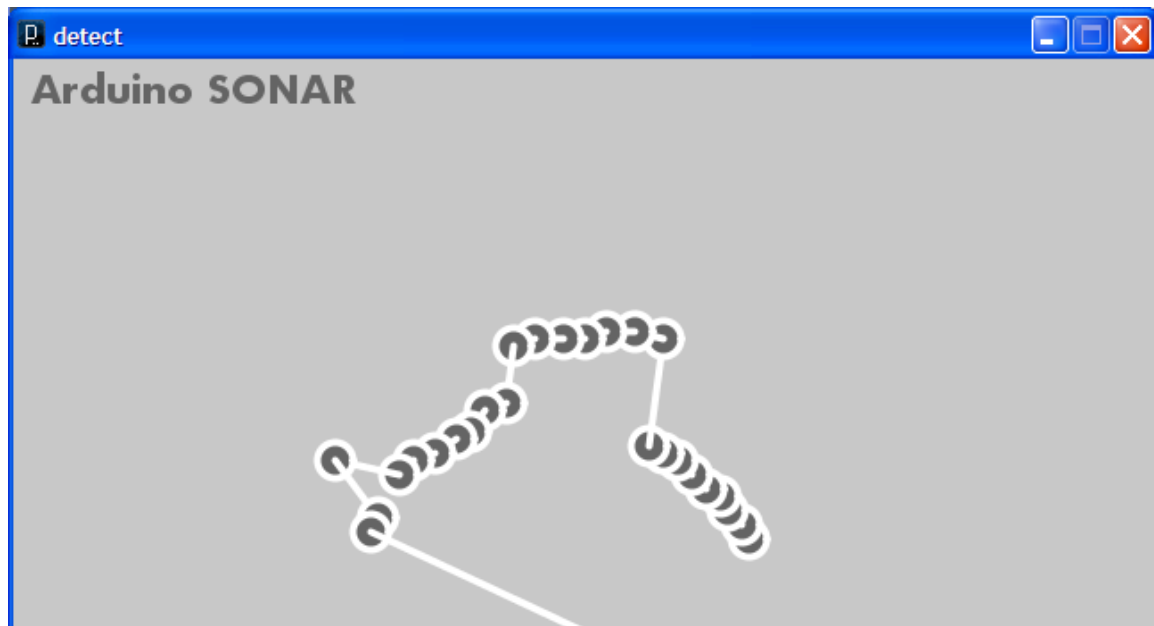
```
#define SRF_ADDRESS contains the address of the SRF10.
```

The `spekjet_controller` is used to initialize the SpeakJet and say various messages which are stored in the library.

## Processing Code

I wrote a very simple Processing sketch which simply plots the Cartesian coordinates of the sensor information and connects these data points with lines. The code was thrown together by mashing up an example I found

(<http://www.uchobby.com/index.php/2009/03/08/visualizing-sensor-with-arduino-and-processing>) with some J2ME code taken from a LIDAR plotting routine I wrote last semester. The Processing code can be found in the Attachment for this paper.



*Figure 1: Processing Sketch for Visualization*

Supply List	Price	SKU:	Supplier
-------------	-------	------	----------

Project Final Report

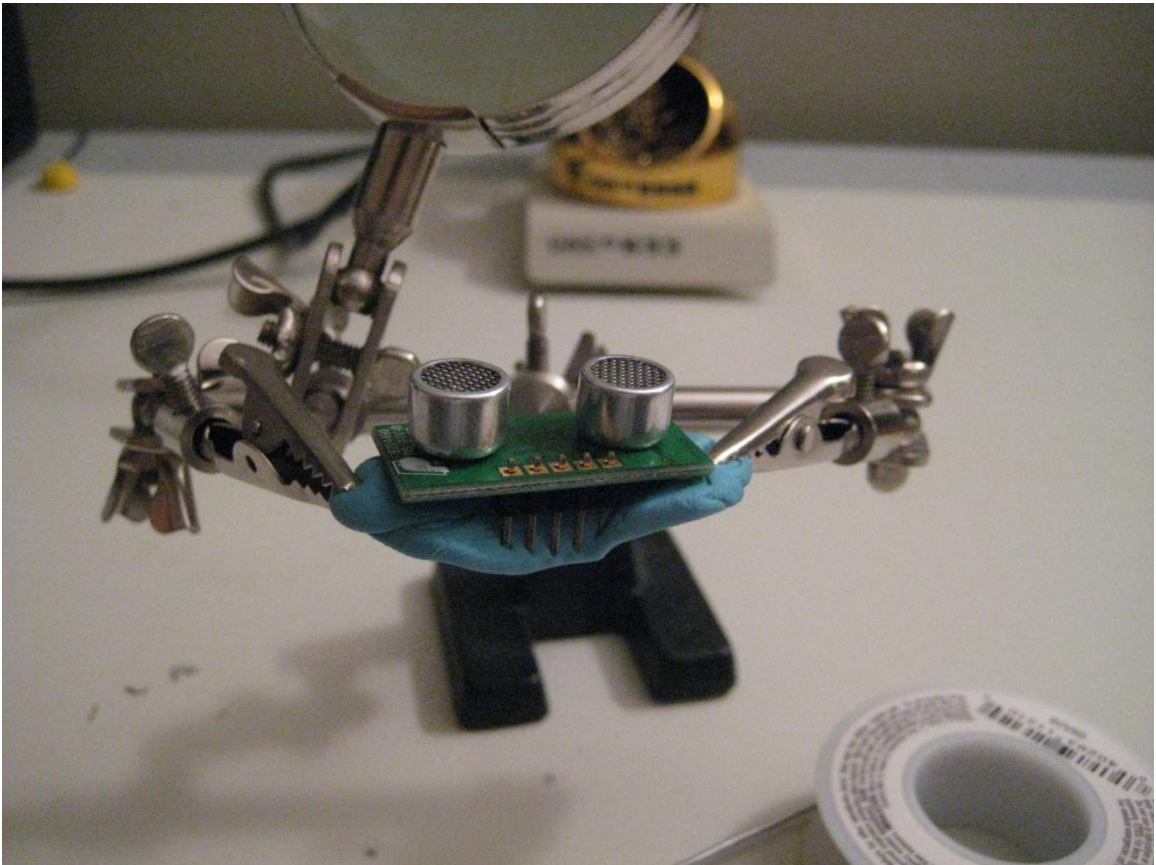
Ardino IDE	\$0.00	N/A	<a href="http://www.arduino.cc">http://www.arduino.cc</a>
Processing IDE	\$0.00	N/A	<a href="http://processing.org">http://processing.org</a>
X-CTU Configuration Software	\$0.00	N/A	<a href="http://www.digi.com">http://www.digi.com</a>
Libelium Xbee Shield	\$24.95	SKU# 51835	<a href="http://www.hvwtech.com">www.hvwtech.com</a>
Devantech SRF10 Tiny Ultrasonic Ranger	\$59.95	SKU# 40326	<a href="http://www.hvwtech.com">www.hvwtech.com</a>
Devantech SRF10 Mounting Kit	\$8.95	SKU# 40360	<a href="http://www.hvwtech.com">www.hvwtech.com</a>
Sub Micro Servo - NARO HP BB	\$15.95	SKU# 22150	<a href="http://www.hvwtech.com">www.hvwtech.com</a>
XBee Series 1 Pro (x2)	\$75.90	WRL- 08690	<a href="http://www.sparkfun.com">www.sparkfun.com</a>
XBee Explorer USB	\$24.95	WRL- 08687	<a href="http://www.sparkfun.com">www.sparkfun.com</a>
9V Switched Battery Case with Barrel Jack	\$5.00	N/A	<a href="http://www.sparkfun.com">www.sparkfun.com</a>
VoiceBox Shield	\$39.95	DEV- 09624	<a href="http://www.sparkfun.com">www.sparkfun.com</a>
Arduino Duemilanove	\$29.99	DEV- 00666	<a href="http://www.sparkfun.com">www.sparkfun.com</a>
ProtoShield Kit	\$16.95	DEV- 07914	<a href="http://www.sparkfun.com">www.sparkfun.com</a>
Mini Breadboard	\$3.95	PRT- 08800	<a href="http://www.sparkfun.com">www.sparkfun.com</a>
Jumper Wire Kit	\$6.95	PRT- 00124	<a href="http://www.sparkfun.com">www.sparkfun.com</a>
Jumper Wires Premium 6" M/F Pack of 100	\$24.95	PRT- 09139	<a href="http://www.sparkfun.com">www.sparkfun.com</a>
Break Away Headers - Straight	\$2.50	PRT- 00116	<a href="http://www.sparkfun.com">www.sparkfun.com</a>
Break Away Headers - Right Angle	\$1.95	PRT- 00553	<a href="http://www.sparkfun.com">www.sparkfun.com</a>
Arduino Stackable Header - 8 Pin (x2)	\$1.00	PRT- 09279	<a href="http://www.sparkfun.com">www.sparkfun.com</a>
Arduino Stackable Header - 6 Pin (x2)	\$1.00	PRT- 09280	<a href="http://www.sparkfun.com">www.sparkfun.com</a>
Loktite Mounting Putty	\$1.88	N/A	Wal-Mart
Parallax PIR Sensor Module	\$9.99	276-033	Radio Shack
Nylon Wire Ties	\$2.49	278-1632	Radio Shack
Coat Hanger	\$0.00	N/A	N/A
Assorted Wire Bundles	\$0.00	N/A	Junk computer
Small Speaker	\$0.00	N/A	Junk computer

*Table 1: Parts List*

## ***Implementation & Construction***

### **Ultrasonic Sensor Assembly**

1. Carefully solder header pins on the SRF10. It helps to hold them in place using mounting putty.



2. Jam the rubber grommets into the SRF10 mounting bracket. I found that a pen was handy in squeezing them into a perfect circular shape.



3. Carefully slide the SRF10 into the rubber grommets.



4. Attach the SRF10 mounting kit to a servo mounting horn using small screws.
5. Attach four prototyping cables to the sensor. They can (and should) be secured using a wire tie as they will otherwise slide off as the servo pans.
6. Do not screw the servo horn into the servo yet. I found that the horn seated firmly enough onto the servo that this was not required, and this allowed me to easily tweak the sensor alignment to the rotation to which the Pico servo could be electronically commanded. This was limited to about 130 degrees or so.

### Shield Assembly

Arduino shields generally need some level of assembly. Even pre-built shields will likely soldering of header pins. Consult assembly documentation as needed.

1. Assemble the ProtoShield. Instructions can be found at: <http://www.ladyada.net/make/pshield/solder.html>.
2. Attach the mini breadboard to the ProtoShield using double-sided tape.
3. Solder female header pins and speaker connection pins to the VoiceBox shield which houses the SpeakJet and amplification circuitry. I used right angle connectors in order to save space. Mounting putty is very useful in holding these in place while you solder.

## Project Final Report

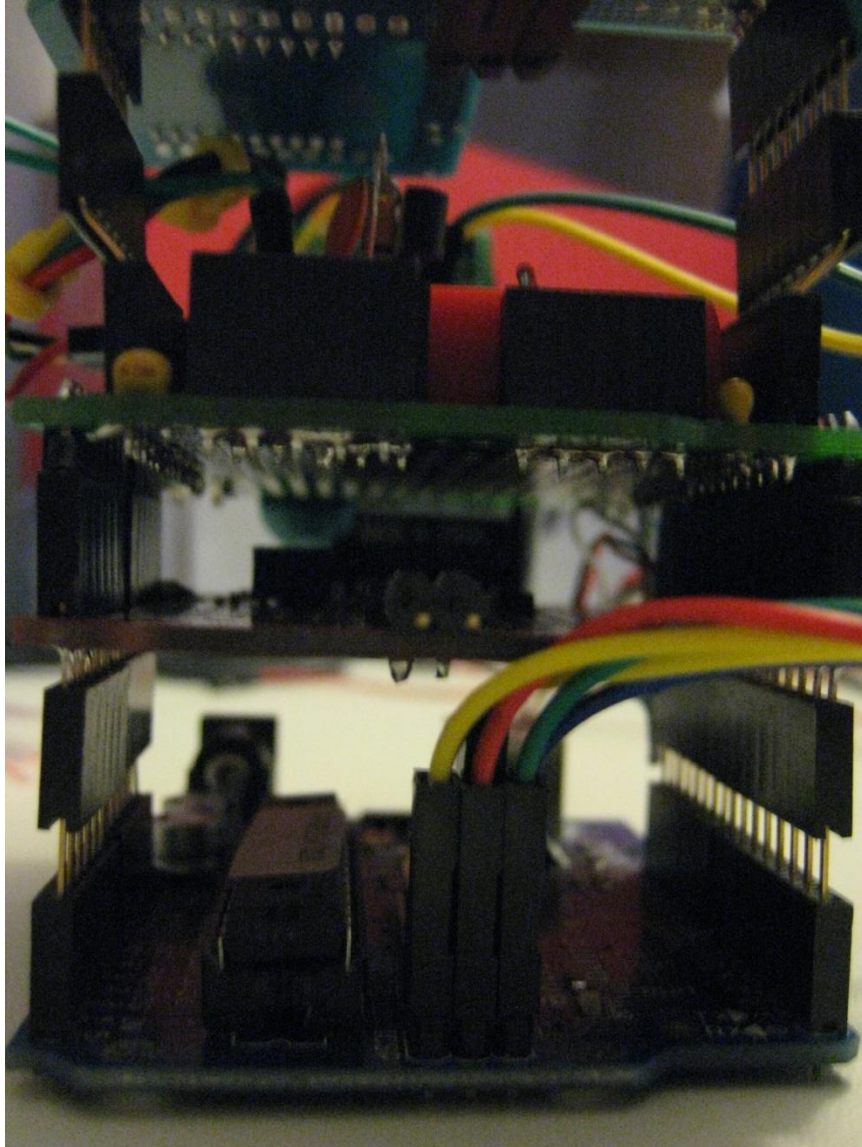
4. A small jumper placed between digital pins 0 and 2 allow you to command the SpeakJet from the PheaseALator software, if desired.

NOTE: Be careful that the female headers go on straight, or stacking the shields will be difficult.

### **Arduino Assembly**

At this point, we are ready to stack the shields and begin constructing the prototype.

1. Begin by placing the Arduino on a flat surface.
2. Add jumper wires to all of the ICSP header pins. These will be connected to the XBee shield later on. Use zip ties to keep these in a neat bundle.
3. Add stackable header pins onto the 6 and 8 pin rows. These are required to allow clearance for the jumper wires which you added in the previous step.
4. Add the VoiceBox shield to the stack.
5. Add the ProtoShield to the stack.
6. Depending on the model of ProtoShield you have, you might notice that the pins on the top do not align right for adding an additional shield to the stack. This can be corrected by connecting two stackable headers and carefully bending the pins on one of them to correct for the offset.



7. The stack can be completed by placing the XBee shield onto the last layer of stackable header pins and connecting the wires to ICSP header on the XBee shield. Using right angle connectors allows you quickly remove these connections to program over USB.

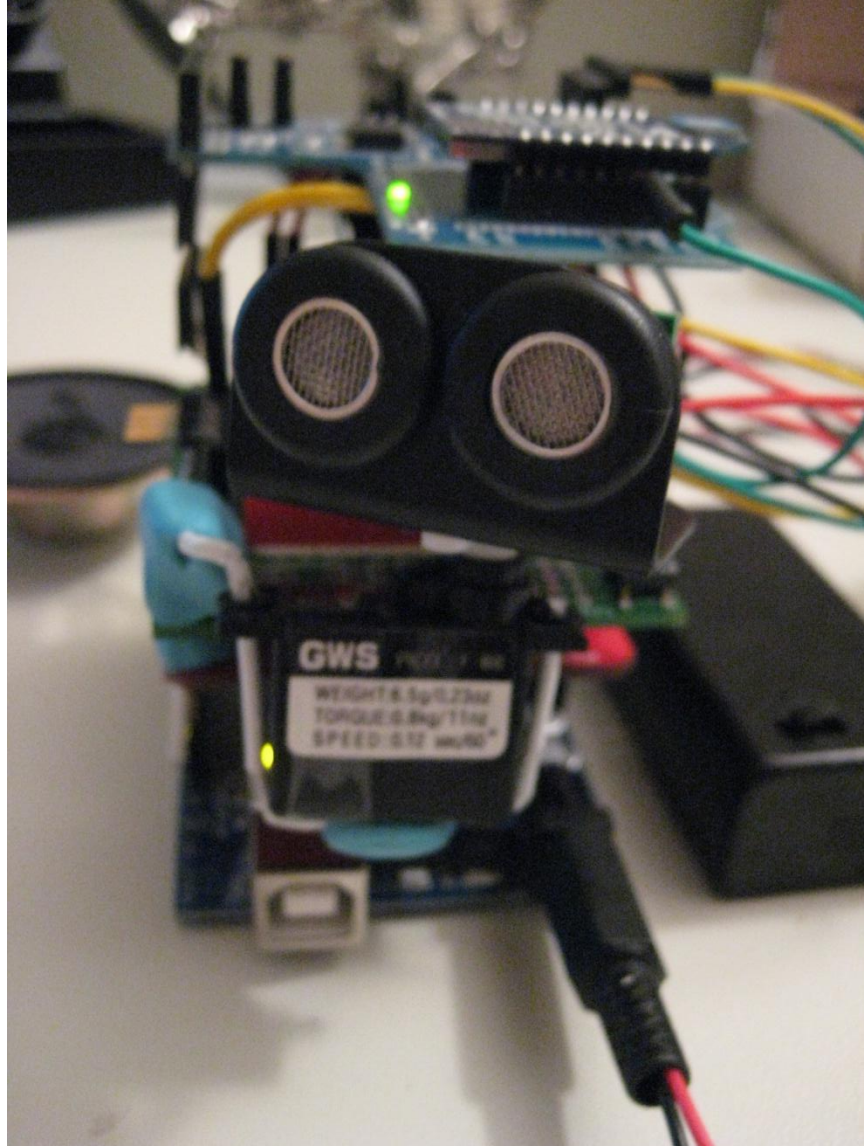
## **Servo Mount**

This part is subject to creative interpretation, but these steps describe the design I improvised for this prototype.

1. Place the rubber grommets included with the servo into their mounting slots.
2. Cut a length of coat hanger to about a foot or so.

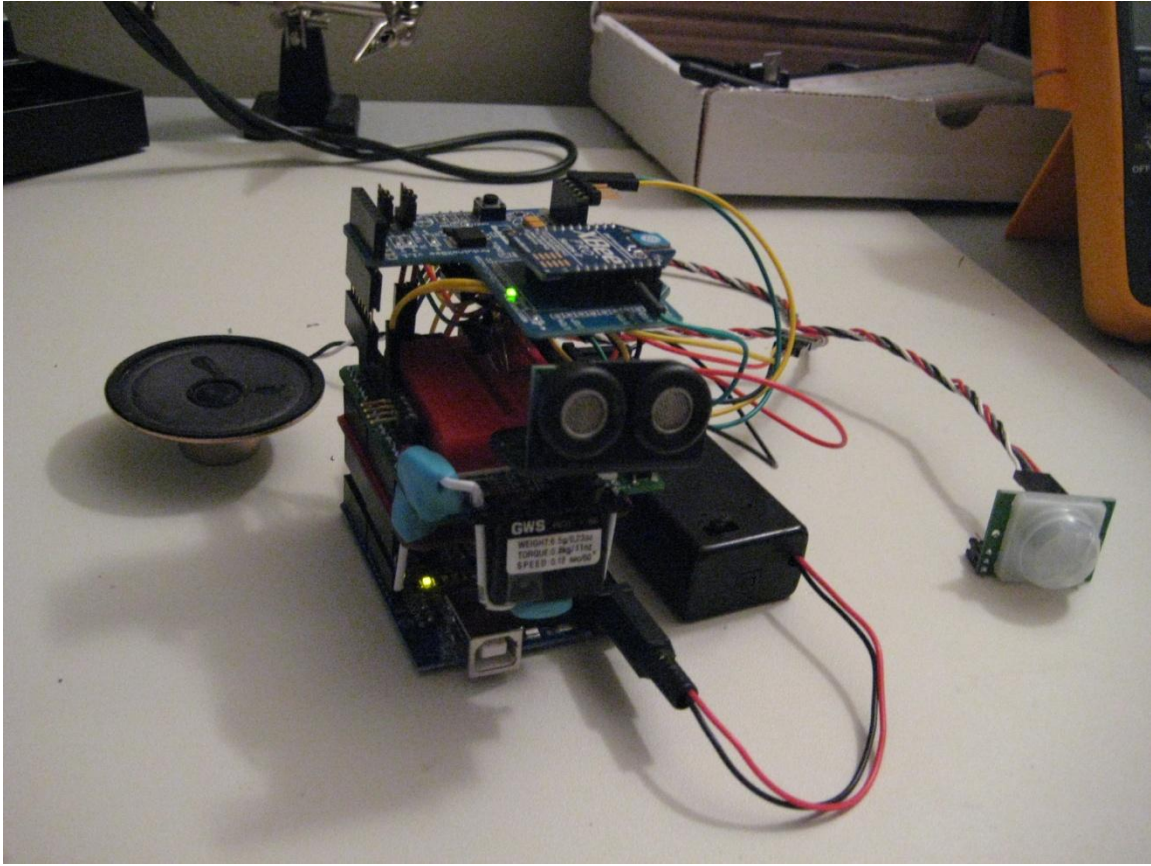
## Project Final Report

3. Using a pair of pliers, bend the hanger wire so that it slides into the mounting grommets and holds the servo. I found that a slight upward angle helped in preventing the SONAR sensor from ranging on the ground instead of outward.
4. The ProtoShield and VoiceBox shield have pre-drilled holes for PCB offset pads. You can use these holes, along with some mounting putty or hot-glue to hold the servo securely in place. This is important, as a loose servo mount will sacrifice a lot of accuracy.

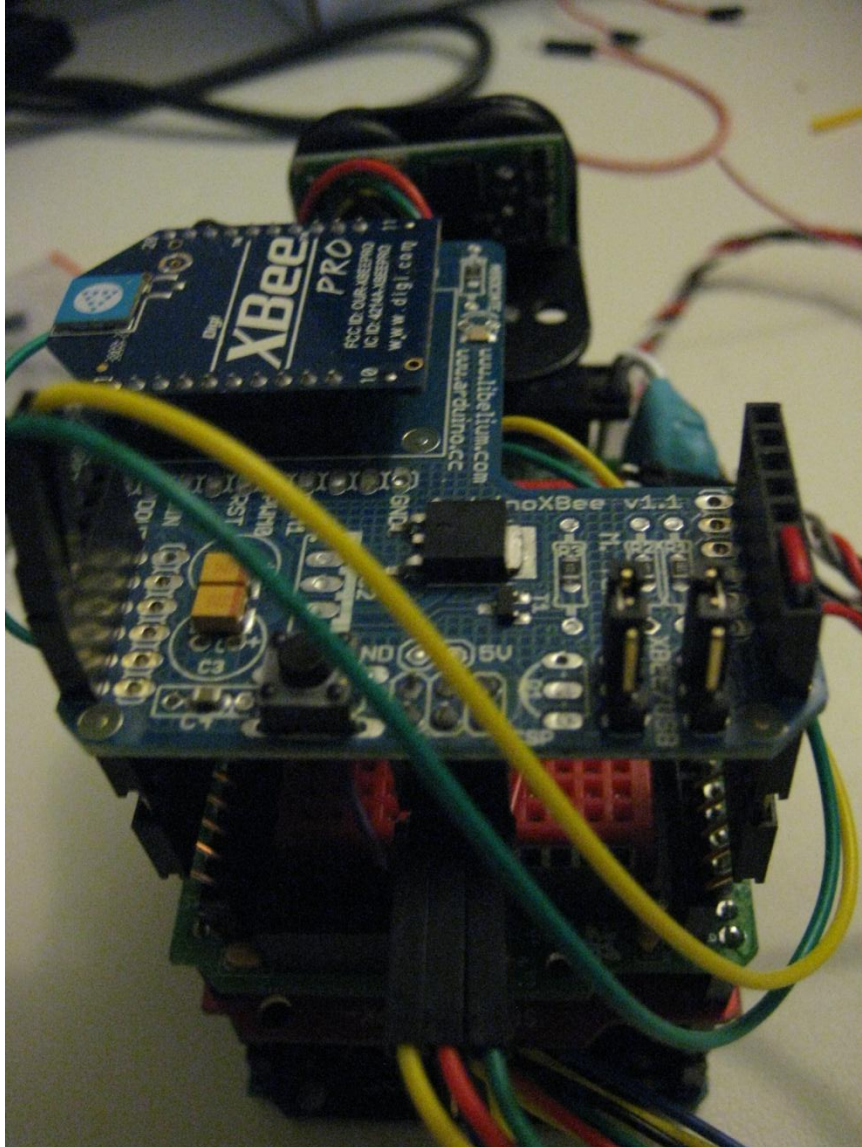


### **Breadboard Detail**

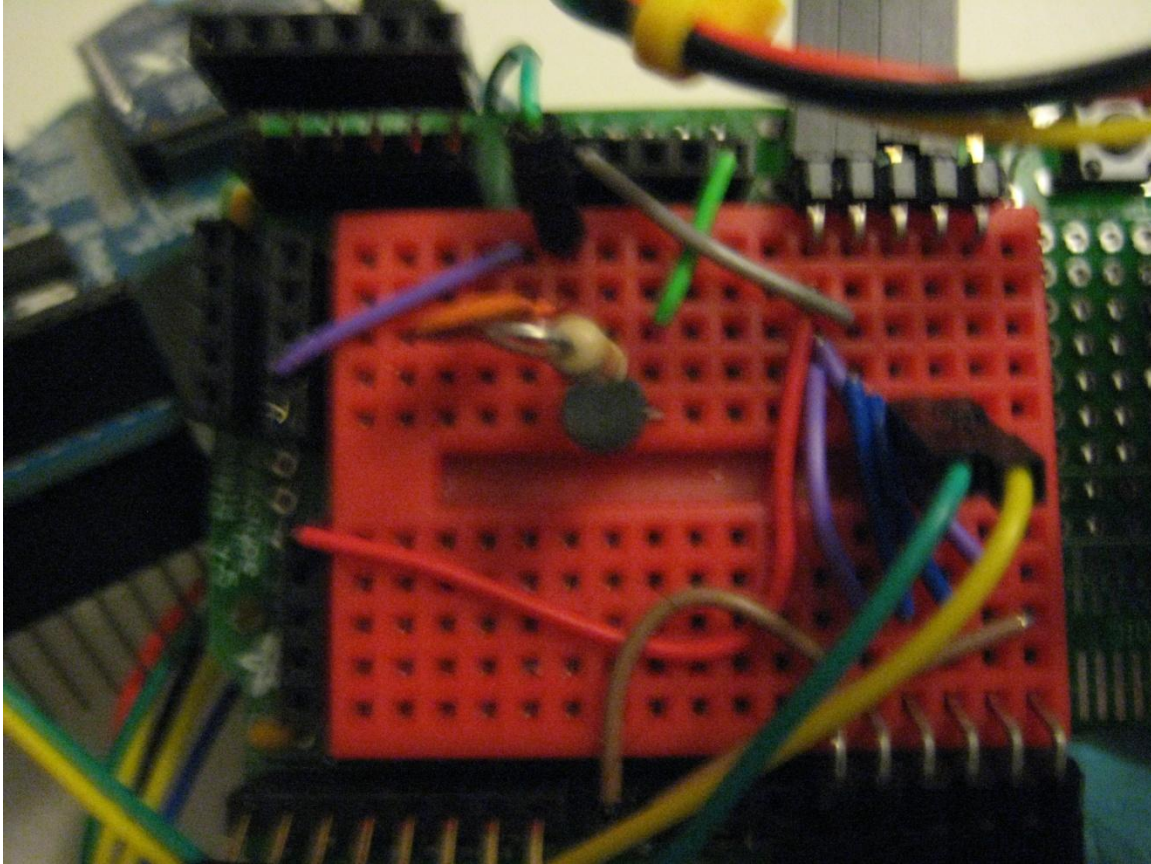
This part is not necessarily needed, but it will lead to a cleaner package with less stray wires to interfere with the servo rotation.



1. Use 90 degree header pins to add connectors for the pir, servo, and i2c connections.
2. Use flat jumper wires to route the appropriate power, ground, and GPIO pins to each connector.
3. Double check all of the connections.
4. Check everything again, and make the connections to the header pins. If you have any old computers lying around, you may be able to salvage some lengths of multi-strand wires with headers already attached.



5. At this point, you may want to attempt to breadboard the XBee wireless programming circuit found at: <http://www.ladyada.net/make/xbee/arduino.html> I did not have much luck getting this to work with the XBee shield, but it seems possible, and I intend to revisit this.



### **XBee Configuration**

The aforementioned link has a good tutorial on how to configure the XBee adapters themselves so that they can talk to each other. Here are some hints:

1. Connect the XBee Explorer USB to the PC and install FTDI drivers if needed. Always be careful that you are seating the XBee in the right direction. You might damage it otherwise.
2. Using the X-CTU tool downloaded from the Digi support site, make the documented changes. I generally have learned to make parameter, firmware, and baudrate changes separately. The XBee chip can be a bit finicky otherwise.
3. If you find that the tool is unable to connect to the XBee at any point, do not panic. It is highly unlikely that you have caused any permanent damage through misconfiguration... Verify that your baudrates in X-CTU match what has been programmed into the XBee. You might have success enabling API mode and reflashing the firmware. Another trick is to short the ground and RST pins and force the XBee to restart. Google is your friend...
4. The Funnel IO project has released an XBee configuration which works well, if your XBee chips are running the specific supported firmware version:  
<http://funnel.cc/Hardware/FIO>
5. Wireless programming is best accomplished using an XBee Series 1 chip. The Series 2 (now 2.5) chips do not support transparent IO line passing, which is

needed to pass the reset to the Arduino. An alternative is to accomplish the reset in software, but this is not optimal as I have read that it is kind of tricky to time it correctly.

## **Functional Testing**

### ***Power Issue***

Once I had assembled everything, I disconnected the ICSP pins from the XBee shield and continued to develop code using USB. I noticed that the speaker tended to output garbage when the servo moved and the ultrasonic sensor would sometimes crash while trying to get a reading. This was due to an inability to power the apparatus reliably using power from the USB connector.

I tried to use capacitors to fix this, but in the end the only solution I found was to use a separate AC transformer when testing. I found that a 9V battery is not a good choice for this application. It barely has enough current to power the device when fresh, and this performance degrades quickly. I am planning on doing some more research on battery options and purchasing a more powerful rechargeable battery kit.

### ***Ultrasonic Sensor Accuracy Issue***

I had a lot of problems getting reliable readings from the sensor at first. I put together some functional tests which took multiple sonar sweeps and calculated absolute value deltas between them. I dumped these sensor readings and delta calculations into a spreadsheet. I also had a test case where I took multiple sensor ranging samples at each step. It was then that I noticed the deltas were greater between the first and second ranges than they were between the second and third readings. There were also larger deltas at either end of the sweep.

The data demonstrated very plainly that I wasn't giving the servo enough time to move. Increasing the delay in between readings (to allow the servo more time to step) and adding a delay at the beginning and end of the sweep helped to have much more reliable results. I also noticed during these tests that decreasing the analog gain sensitivity seemed to decrease anomalous readings between passes.

### ***SpeakJet Command Issue***

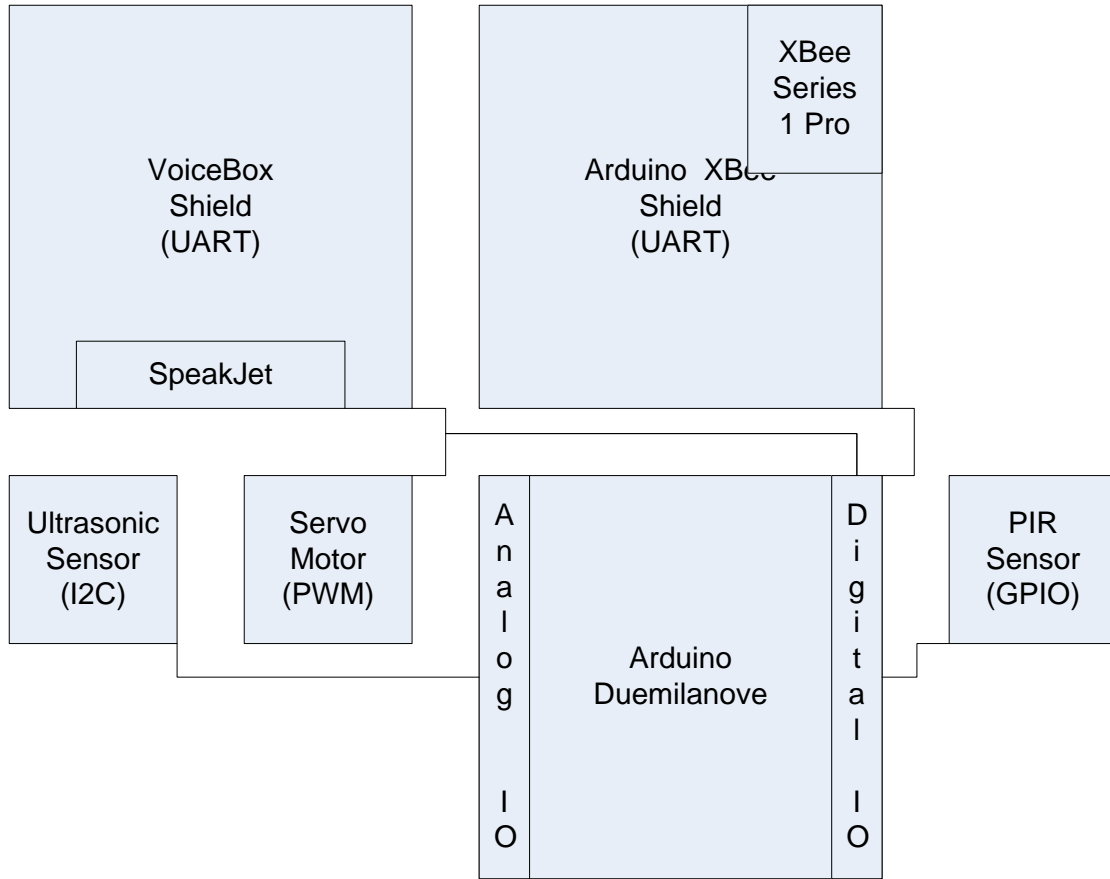
This chip seemed to have a mind of its own... It simply would say whatever it wanted, whenever it wanted to! I think it was designed more with the event pins in mind, rather than for continuous serial command. I tried switching to the NewSoftSerial library. I also tried to break sentences into smaller chunks and adding delays to the routine that was sending each chunk. This did not help at all.

## Project Final Report

The final solution, which I never saw documented anywhere, was to add a SpeakJet command to place a 0 ms pause at the end of every character array before passing it to the SpeakJet. This had the effect of a string terminator, and each command wouldn't be randomly overlaid with previous commands anymore. It was an easy fix to a frustrating issue that took days to figure out.

Setup microcontroller and IDE	Completed
Configure XBee interface	Completed
Research sensor options	Completed
Research servo options	Completed
Order parts as needed	Completed
Build basic test apparatus	Completed
Interface with servo	Completed
Test servo range of motion	Completed
Interface with I2C sensor bus	Completed
Interface with SpeakJet	Completed
Interface with PIR	Completed
Write <code>pir_controller</code>	Completed
Write <code>servo_controller</code>	Completed
Write <code>sonar_controller</code>	Completed
Write <code>speakjet_controller</code>	Completed
Assemble travel-friendly prototype	Completed
Write Processing sketch	Completed
Calibrate ultrasonic sensor output	Completed
Refine <code>sonar_controller</code>	Completed
Refine <code>speakjet_controller</code>	Completed
Test completed code	Completed

*Table 2: Milestone Chart*



*Figure 2: Hardware Block Diagram*

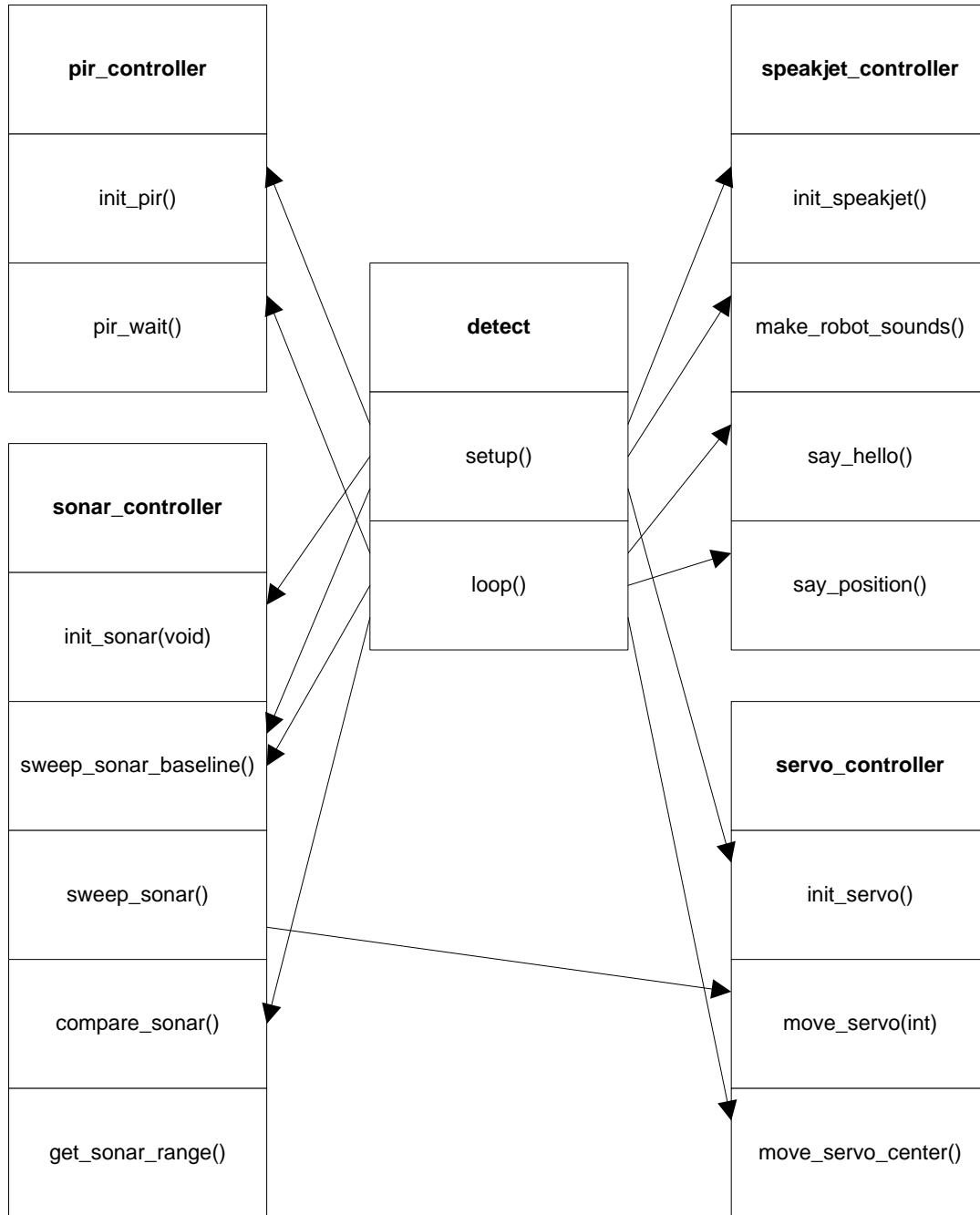


Figure 3: Software Block Diagram

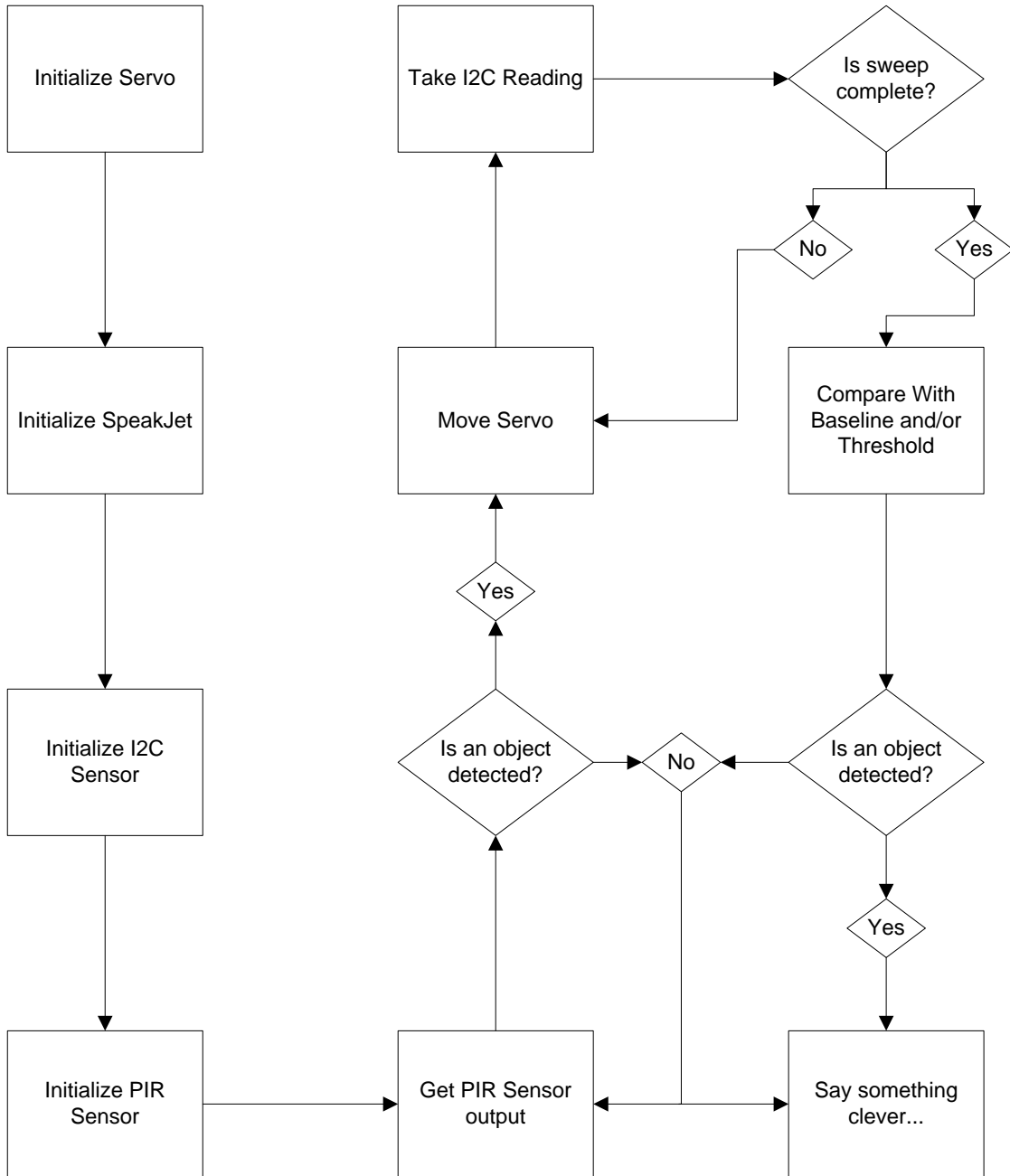


Figure 4: Flowchart of sensor analysis logic

### Retrospective

I am glad that I chose to complete this project on the Arduino. It was my first real coding experience on this platform, and I can say that compared to writing C for the Zilog ZNEO, writing Wiring libraries for Arduino makes for a much more fun and productive experience. I am grateful that my time on the ZNEO taught me a lot about what is happening behind the scenes, but quiet honestly it is nice to not have to worry about it so much.

## Project Final Report

One thing I learned from this project is that sonar is often a temperamental technology. Nuances in positioning, timing, and environmental texture can lead to all sorts of undesirable readings. I am a bit disappointed with the performance of the SRF10 in this particular use case, as its beam pattern is fairly broad. It required a lot of fine-tuning to get readings accurate as the servo rotated.

Knowing what I know now, I might have tried to find a sensor that had a narrower beam pattern. It also would have been nice to integrate an infrared range finder, as the time required to capture an ultrasonic range sample is fairly long. This being said, I think a broad beam pattern might be appropriate if the sensor is mounted in a stationary position or if the servo is only moved sporadically to face the direction of travel, so the SRF10 will still prove useful in future projects.

One area that will require a bit more effort is the integration of XBee with the Arduino. I initially purchased Series 2 Pro XBee chips some time ago, not knowing of their limitations with respect to wireless Arduino programming. When I finally got around setting the rig up, I realized my folly and ordered new Series 1 XBee chips. I still have not gotten wireless uploading of Arduino sketches working, and this would certainly be a nice feature to have.

I believe that a lot of my problems with programming the SpeakJet could have been helped by integrating a text-to-speech processor, so I am going to look into ordering a TTS256 or similar part and integrating it into the prototype area of the VoiceBox shield.

As stated previously, another area I need to look into is battery power. This project is a poor use case for a 9V battery. A better long term portable power supply would include a higher output rechargeable battery.

### ***Attachments***

1. Arduino Code
2. Processing Code
3. Hardware Block Diagram
4. Software Block Diagram
5. Process Flowchart