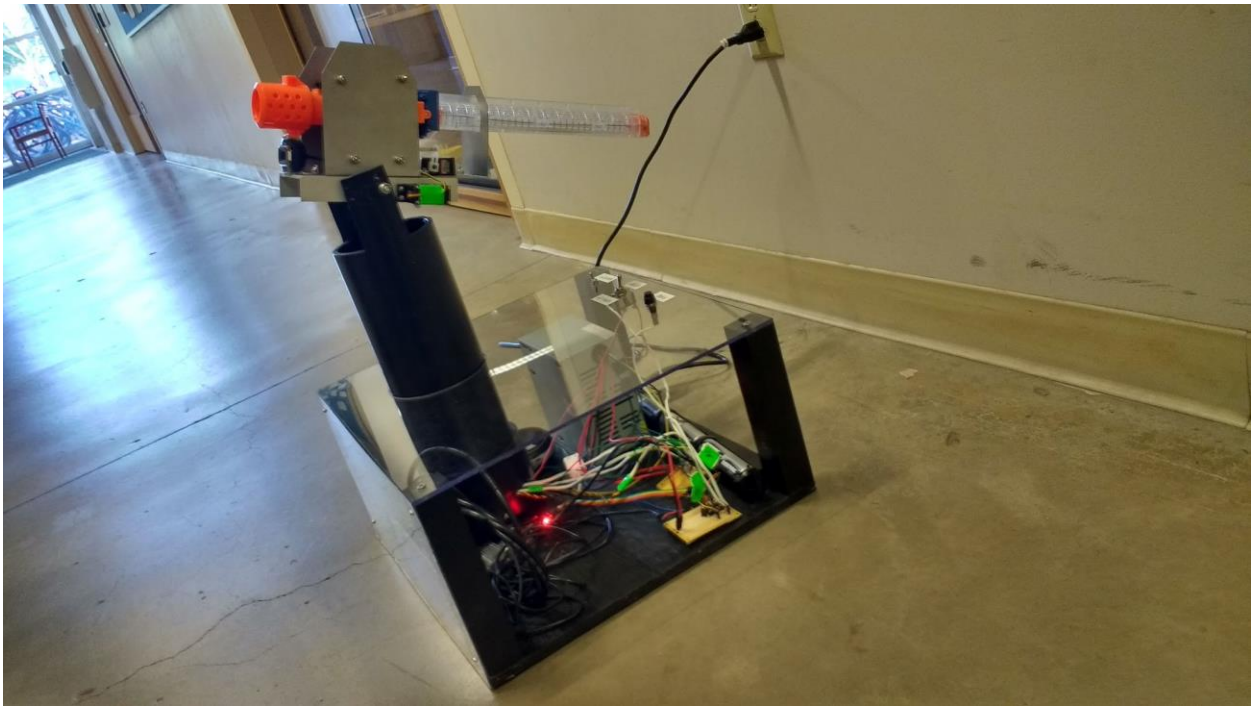
Self Aiming Nerf Turret Automaton (SANTA)



Group 54

Michael Cookson
Jared Greene
Gavin Miller
Alex Baughman

May 6th, 2016

Table of Contents

Design Summary:

SANTA is a robot designed to locate, aim at a target, and then fire Nerf balls at the designated target. The robot is largely autonomous, but requires human interaction to enable it and to fire. The system includes a camera (1), a ball accelerator (2), a motorized trigger system (3), a two-axis gimbal (4), a display (5), and a control base (6). The camera gives SANTA vision of his target. Once the camera finds and locks onto the target SANTA's ready to deliver his presents.

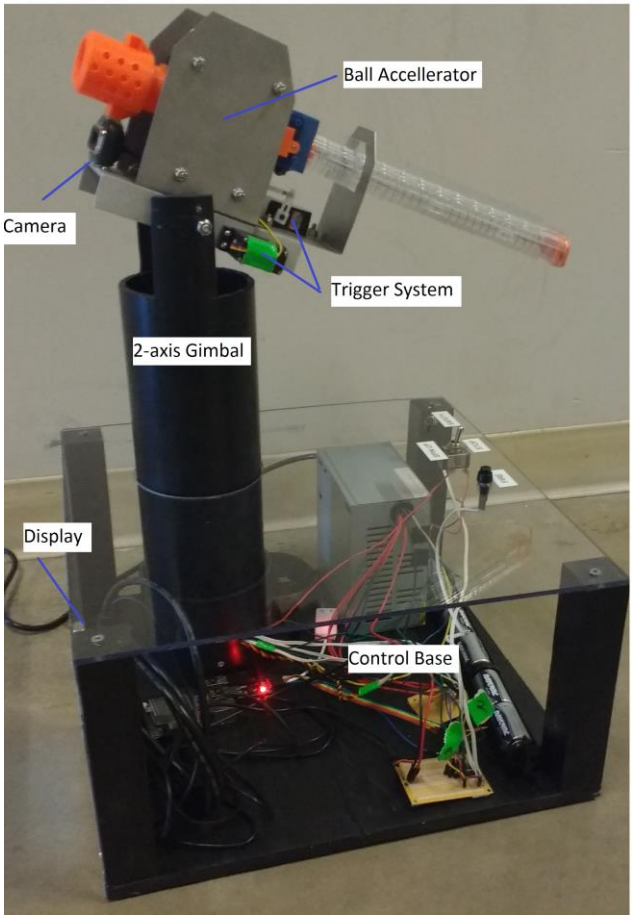


Figure 1: Side view of turret

Figure 1 (above) shows the final design of SANTA. The camera is a web camera attached underneath the barrel of the purchased Nerf Zeus launcher. It points in the same direction as the launcher, giving SANTA vision. The ball accelerator is a Nerf Zeus launcher that has been stripped down to just required parts; two motors, a barrel, and the trigger. The launcher and camera assembly is mounted to a metal shell, aluminum channel and formed sheet

metal. The motorized trigger system is two motors that separately pull the two triggers that allow the launcher to fire. These two motors are also mounted to the metal shell. The entire metal shell assembly is mounted to the gimbal. The gimbal is comprised mostly of a PVC pipe. A motor at the top of the PVC allows ~ 90° of vertical motion. The PVC pipe fits into a collar in which another motor gives SANTA ~180° of horizontal motion. The collar of the gimbal is mounted to the control base. The control base is a sheet of plywood with a 5V power source, battery power source, ODROID, speaker, and the PIC microcontroller all attached. The display is a 7" LCD mounted to a polycarbonate cover on the front of the control base.

SANTA works largely based off of what is seen via the web camera. Once powered up, the gimbal motors move to the last position the launcher was at before is was powered off. He will then begin to sweep back and forth to look for the target. Once the target is found, SANTA will track any movements the target makes. If the target is locked onto for 10 seconds the fire button will be unlocked. If you want to fire at the target, the spin-up switch needs to be flipped, this turns on the launcher motors. There is also a safety switch that can be activated killing all power to the motors, rendering SANTA safe. Now he is ready to launch. Press the launch button and he will fire a Nerf ball at the target.

System Details

Code name Santa consists of six parts: camera, a ball accelerator or luncher, a motorized trigger system, a two-axis gimbal, a display, and a control base.

Camera

The camera is mounted under the muzzle of the launcher and is aimed in the same direction. What the camera "sees" is what the launcher will shoot at.

Figure 2: Camera

Launcher

The "launcher" part of the turret consists of the internal workings of a Nerf Zeus ball launcher. The projectiles are Nerf balls, approximately 3/4 inch in diameter. It uses two gripped wheels, spinning opposite directions, to grab the available ball out of the magazine and propel it forward, up to 30 feet or more. The overall firing system works very similarly to a two-wheeled baseball pitching machine. The complete Nerf Launcher was ultimately disassembled, using only the magazine, firing mechanism, and trigger mechanism in the project. The magazine is tubular and spring loaded, forcing available loaded Nerf balls into the launching area.

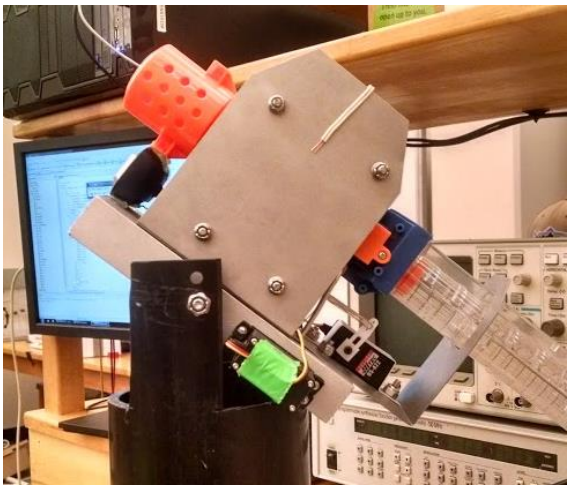


Figure 3: Launcher

Trigger System

The trigger mechanism is controlled by two electric motors, one releasing the ball from the magazine into the chamber, the second pushing the ball into the spinning wheels that fire the projectile. The motors are controlled through programming, using a “spin up” switch to activate the firing wheels, and a “fire” switch to launch the ball itself.

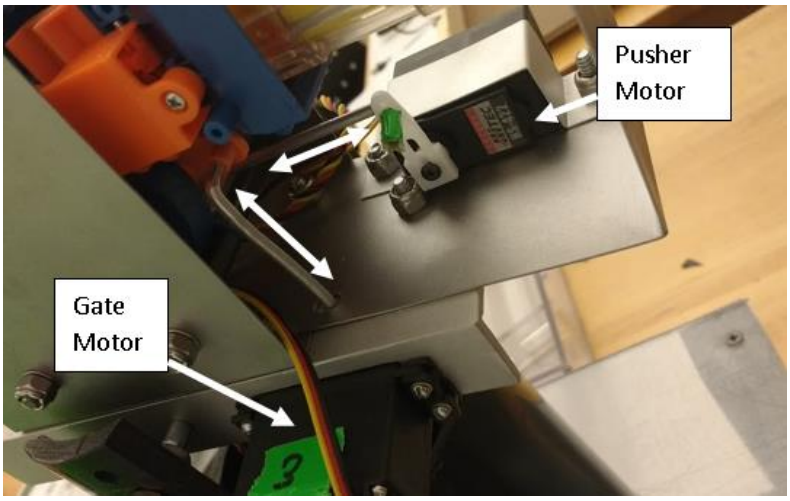


Figure 4: Triggering Motors

Two-axis gimbal

The launcher is mounted on top of a gimbal style mechanism. The turret has the ability to move up and down with approximately 90° of motion, and side to side with approximately 180° of motion. Two identical electric servos control these movements, both programmed to respond to the input from the sighting camera. The upper portion of the gimbal, moving up and down, is comprised of an aluminum channel, housing the launcher along with the camera and trigger motors. It pivots on an axle that is controlled by an electric motor. The axle runs through a 4 inch diameter section of PVC, approximately 1.5 feet tall. The PVC section can rotate left and right, using an identical electric motor to control movement. This combined gimbal mechanism allows the firing portion of the machine to move up and down, as well as side to side, tracking the target with the camera.

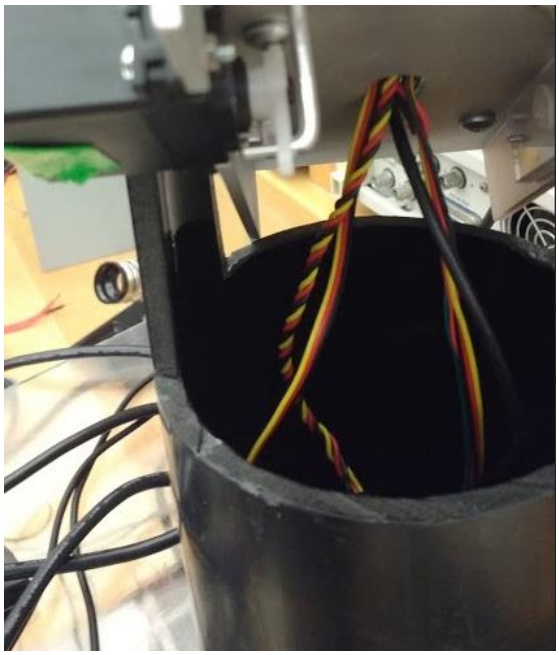


Figure 5:Two Axis Gimbal

Display

A screen is included on the side of the box, allowing the operator of the turret to see what the camera sees, and giving indication of a “target lock” situation so the turret may be effectively and accurately fired.

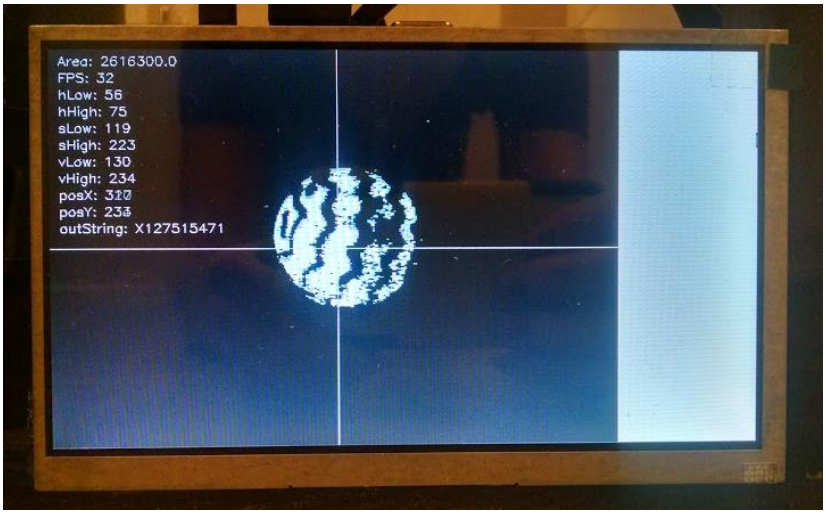


Figure 6:LCD Screen

Control Base

The entire launcher and gimbal portion of the machine is mounted in a box, constructed of plywood and clear polycarbonate. This box houses the computer and power supply components of the system. Polycarbonate was used so electronic hardware would be visible and accessible. The top of the box houses three control switches, “safety”, “spin-up”, and “fire”. The “safe” toggle allows the machine to be deactivated, the “spin-up” toggle activates the firing motors in the launcher after the camera is locked on target, and the “fire” button activates the trigger motors, feeding a Nerf ball from the magazine into the chamber and pushing it into the firing wheels. The computer system is powered by 110 Volt A/C, along with a bank of C cell batteries to operate the launching mechanism of the Nerf Launcher.

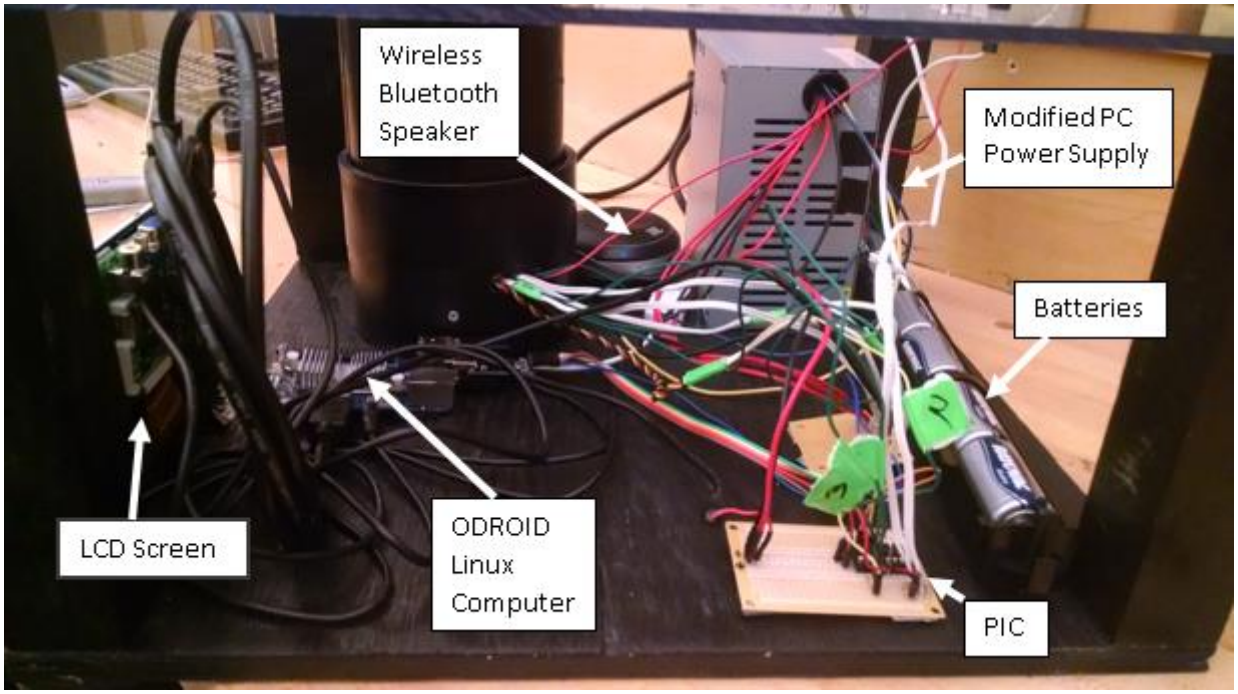


Figure 7: Control Base

Software:

The software for this project is split into two pieces: the Java code running on the ODROID, and the PICBasic Pro code running on the PIC16F88. The ODROID does computer vision and handles sounds, and then tells the PIC where to move the motors and when it can fire. The PIC generates the PWM train for the servos and listens for switch interaction, telling the ODROID when it is firing.

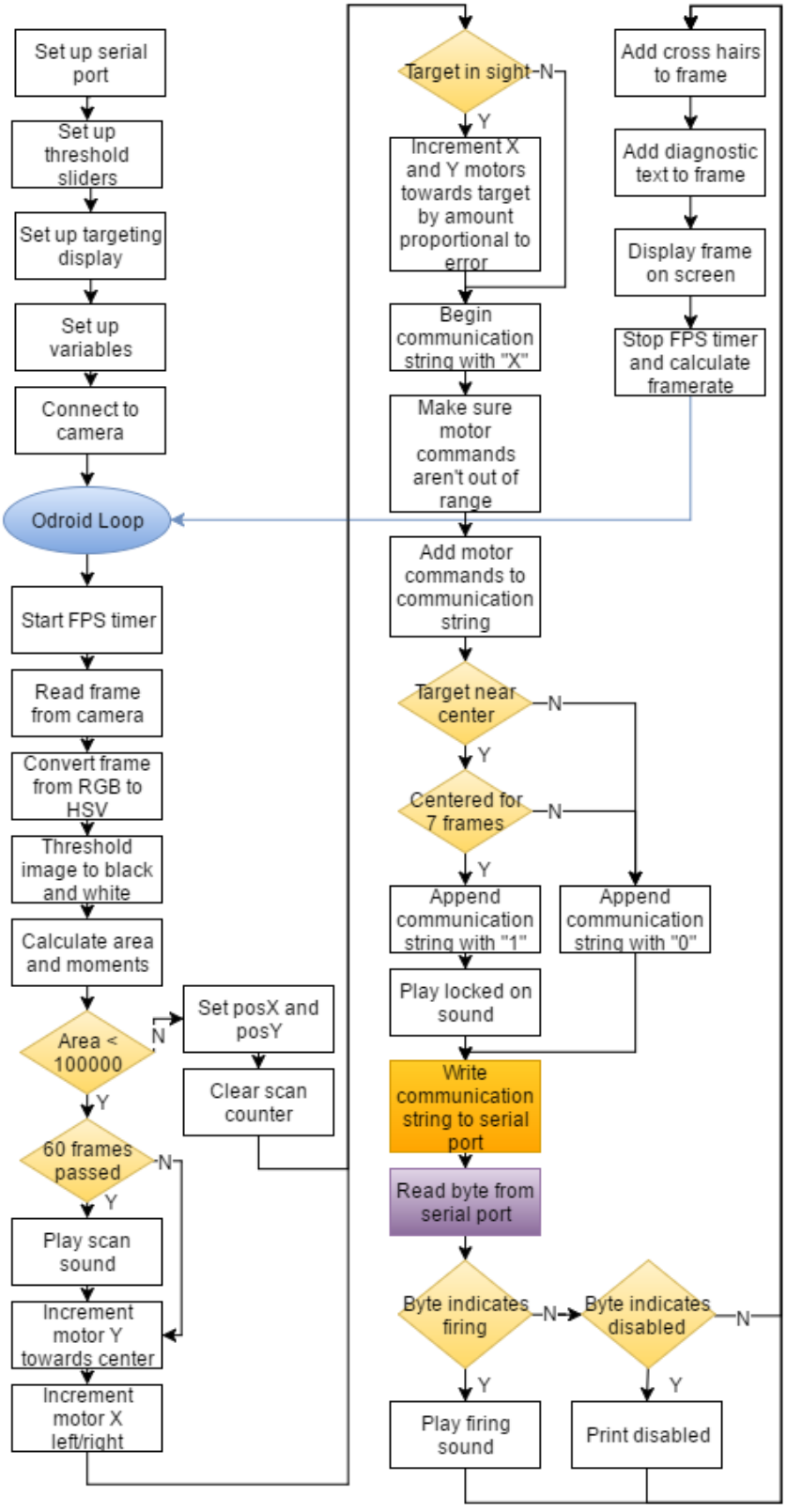


Figure 8: ODROID Code Flow Chart

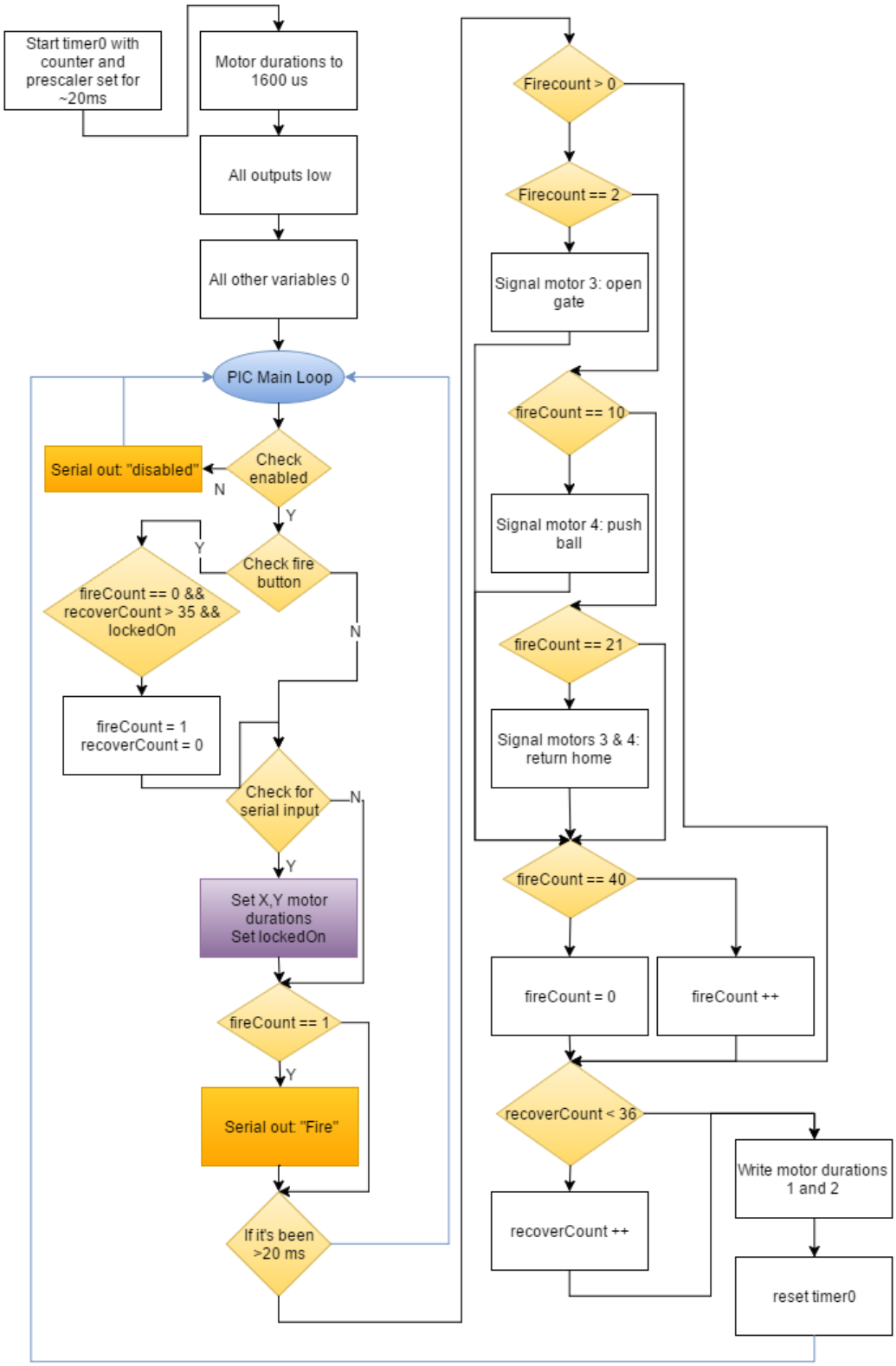


Figure 9: PIC Code Flow Chart

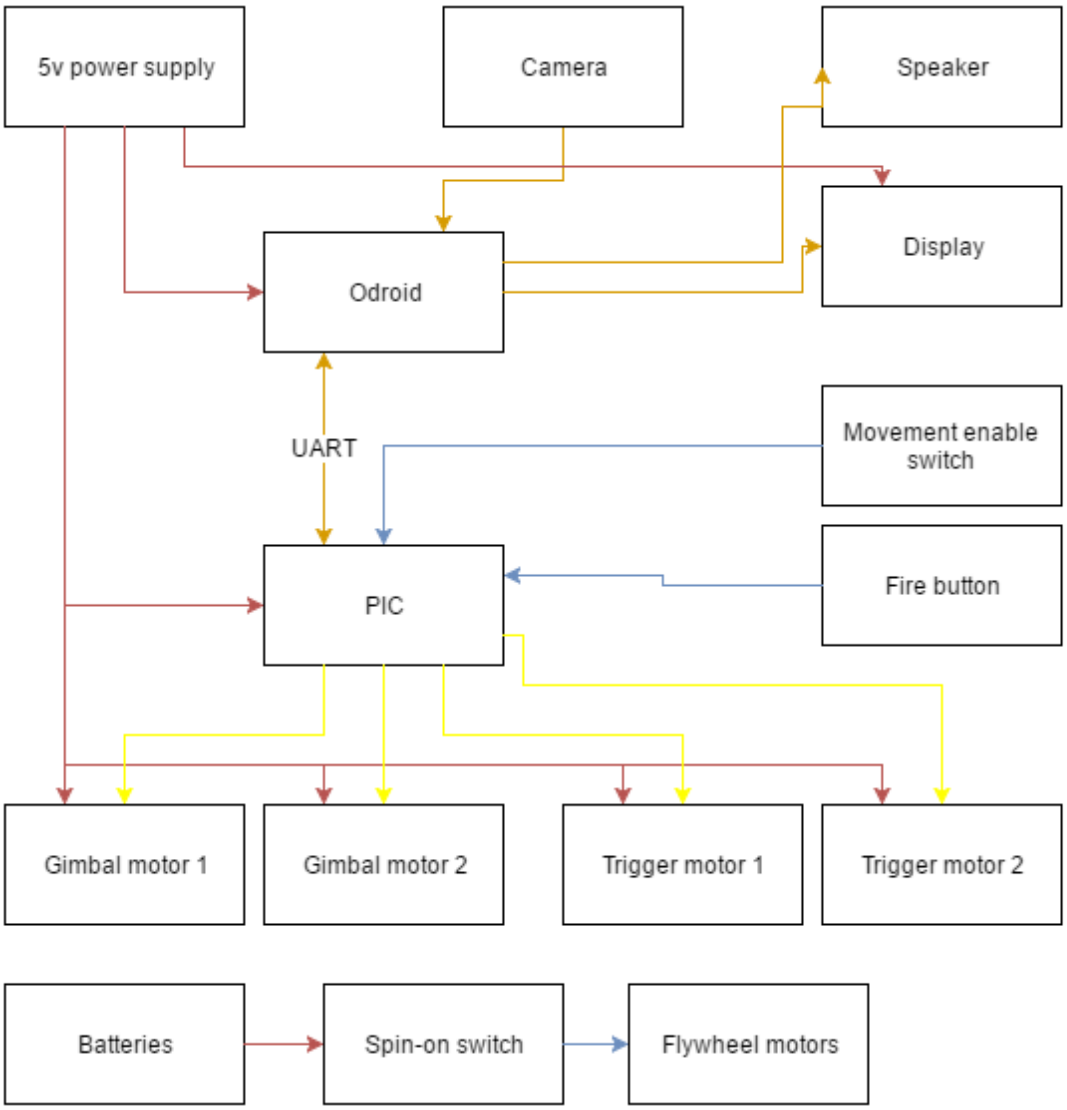


Figure 10: Functional Diagram

Design Evaluation:

The Following scores for each of the six categories are based off the following scoring criteria:

- 0 = Nothing
- 2 = Something but nothing working
- 5 = Something works but not as intended
- 10 = Something works repeatability but didn't require much research
- 15 = Something works repeatability and required significant effort (In-class knowledge)
- 20 = Something works repeatability and required substantial effort (Out of class knowledge)

1. Output Display - LCD
    - For the purpose of debugging and aiming the turret we installed a 7" LCD screen. The LCD screen is used to output the camera feed. The camera is programed to only see one small range of wavelengths of color which shows up as white, all other light comes up as black. The LCD Screen also displays debugging info such as coordinates, if the camera has found a lock (stable image). The LCD screen works every time with its intended purpose to display information from the camera. Getting the screen power and getting it to display desired debugging stats took a few hours of work.
    - Est Score: 15
        - Works Repeatably
        - Works as Intended
        - Required research for Implementation

Figure 11: 7” LCD Screen

2. Audio Output Device - Battery powered speaker
    - When the turret completes an action the computer plays sound to the speaker. This feature works well with the system and adds feedback when the turret is locked on, ready and has hit its target.
    - Est. Score: 10
        - Works Repeatably
        - Works as Intended

Figure 12: JBL Wireless Bluetooth Speaker

3. Manual User Input (for interaction with the user) - Switches

- The turret uses three manual Switches for the user to interface with the rest of the system: a power button, a safety button and a fire button. Each button was carefully considered for its purpose. The Power button which powers up the two fly wheels on the Nerf Launcher needed to handle a large current at startup (up to 30Amps). A large Toggle switch was most appropriate here. The Safety button disables the targeting and firing motors. Again a toggle switch was most appropriate here to keep the motors on while the turret is launching. Lastly was the fire button, for which a toggle wouldn't work. The toggle would cause all of the balls to fire at once which wasn’t in our original design. Instead we decided to use a temporary button. A temporary button will allow for a single shot or multiple (long press) if necessary.
- Est. Score: 12
    - Works Repeatably
    - Works as Intended
    - Each button is strategically designed for its purpose
    - User can move the target to any place in the turrets range and the turret will follow

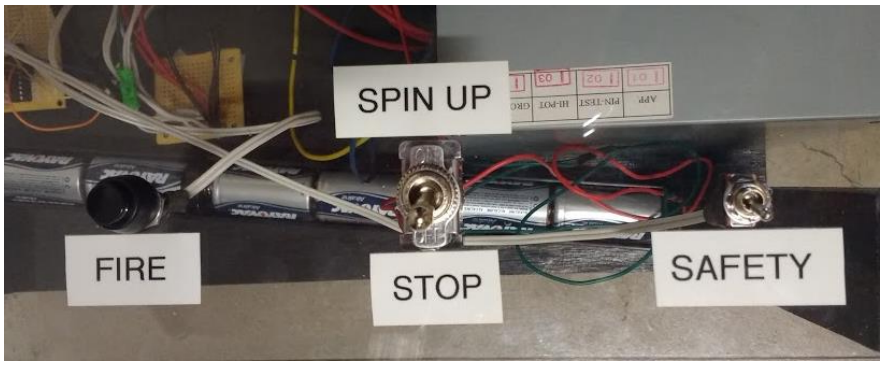


Figure 13: Manual input Switches

4. Automatic Sensor (for response without user input) - Camera/Light Sensor

- We used a video camera to get information about the lighting. The camera records an image which is thresholded to find an object of specific hue, saturation, and value. The color chosen was chosen deliberately to be different than any other lighting present in the environment. That color was bright green, which (subjectively) is not commonly found in our testing environments. The camera looks for that particular color of green and makes an image in the shape and size of it. From there it calculates the center of mass (assuming that everything is weighted the same) of the image and a lines the turret to that center of the mass.
- Est. Score: 20
    - Works Repeatably
    - Works as Intended
    - Considerable amount of time spent working with computer vision which was not covered in any of the class material.

Figure 14: Camera and Bright Green Target

5. Actuators, Mechanisms & Hardware - Servo Motors/Gimbal
    - The Turret uses six motors in total, and required a fair amount of construction to make the pieces fit.
    - Actuators:

- Flywheel Motors (Nerf Launcher)
    1. Launch the specially designed balls
- Small Servo Motors
    1. Recreates originally intended firing mechanism which push a ball from clip and opens gate to let only one pass through at a time
    2. Uses linkages to operate hardware
- Large Servo Motors
    1. Allows for rotation of the turret around X and Y axes.

- Mechanical/Hardware Systems:
    - Gimbal
        1. The gimbal allows for the turret to move to hit its target without having to get any closer or farther away. It was made from 4” pipe and a 3” C-channel connected to the gimbal servos
    - Launcher case
        1. The launcher case holds the launcher and mounts it to the gimbal. It also acts as support for the magazine which holds the specially made balls.
    - Gating mechanism to allow balls through
    - Lever to push ball into the Fly wheels.
- Est. Score: 15
    - Works Repeatably
    - Works as Intended
    - Interesting use of existing materials

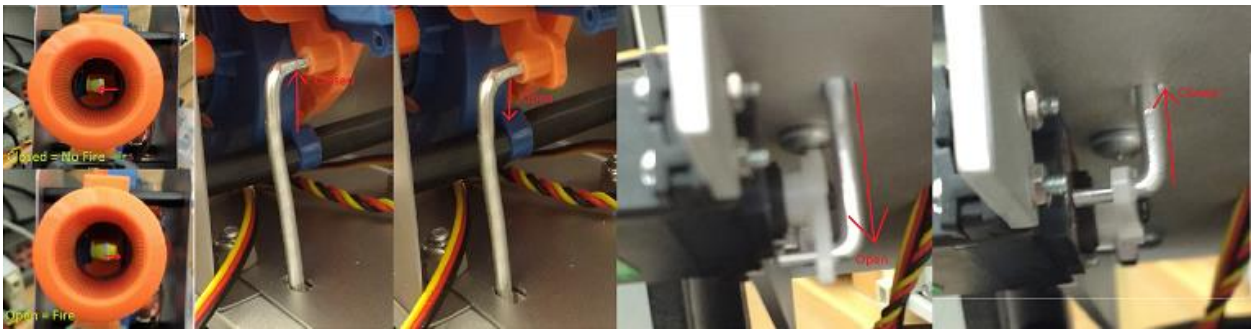


Figure 15: Gate Motor

6. Logic, Processing, and Control; AND Miscellaneous (functional elements not covered in the categories above)
    - Programmed logic
    - Interfaced PIC microcontroller
    - Closed-loop feedback control
    - Computer vision
    - Advanced timing code implemented to enable the use of only a single PIC
    - Components not included in other categories
    - Est. Score: 20
        - Works Repeatably
        - Works as Intended
        - Required substantial time and research outside of class scope

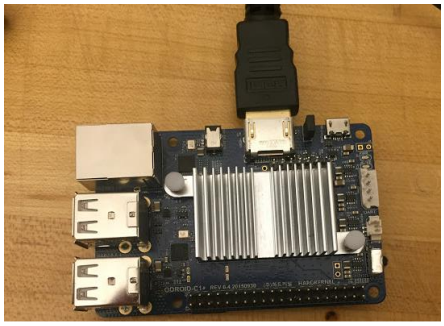



Figure 16: ODroid, Modified PC Power Supply,

Est Total Score (max 175):

Output Display	15
Audio Output Device	10
Manual User Input (for interaction with the user)	12
Automatic Sensor (for response without user input)	20
Actuators, Mechanisms & Hardware	15
Logic, Processing, and Control; AND Miscellaneous	20
Early bird	10
Presentation (look)	5
Apparent and Meaningful Effort	0

● Construction Cost	-5
● Poor level of integration of components	0
● Performance in Presentation	0
● Excessive use of Microcontrollers	0
● Inappropriate Power source	0
Total	= 102%

Partial Parts List

Power Supply -

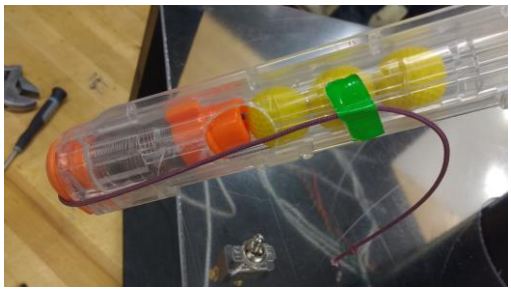
- Description: Provides power to all electronics except for the Nerf launchers which requires high current in the form of six D batteries
- Model: 300W PC Power Supply
- Source: Already owned
- Price: Est. $30

Gimbal -

- Description: Provides two axis of rotation for the launcher to move and hit its targets
- Model: N/A
- Source: ACE Hardware
- Price:
    - Pipe: $7

Nerf Launcher -

- Description: Parts taken for the pre existing fly wheels and specially designed balls to pushed through them. The Team decided that we shouldn't waste our time building something when a commercial product could help us get around the potential problems. These were likely to include sizing the motor and balls, finding some mechanism to push them into the fly wheels and the special design of the balls that allows them to compress as they go through the wheels.
- Model: Nerf Rival Zeus MXV-1200 Blaster (Blue)
    - Item Model Number: B1593000



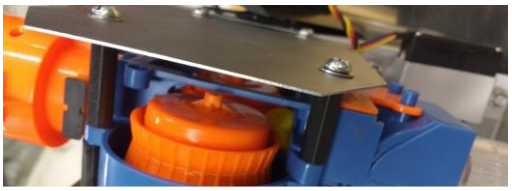


Figure 17: Launcher Parts used

    - URL: http://www.amazon.com/Nerf-Rival-Zeus-MXV-1200-Blaster/dp/B00VX9F5G6/ref=sr_1_2?ie=UTF8&qid=1457469257&sr=8-2&keywords=nerf+zeus
- Source: Amazon.com
    - Screws, Nuts and Bolts: ACE Hardware
    - Sheet Metal: ACE Hardware
- Price:
    - Screws, Nuts and Bolts: $10
    - Sheet Metal: $20

Servo Motor (Small) x2 -

- Description: Small hobby servos to manipulate gating and firing system
- Model: HS-422
- Source: Amazon.com
- Price: $15.97

Servo Motor (Large) x2 -

- Description: High-torque hobby servos to motivate the gimbal
- Model: HS-755HB
- Source: www.robotshop.com
- Price: $31.00

Speaker -

- Description: 4 Ohm 40 Watt speaker
- Model: Unknown, found in thrift store bin
- Source: CSU surplus store
- Price: $3

ODROID-

- Description: ARM-based single board computer, similar to a Raspberry Pi 2, but with better performance
- Model: ODROID C1+
- Source: Amazon.com
- Price: $45

Camera -

- Description: Webcam with automatic exposure adjustment and white balance. 30 fps frame rate, being sampled at 640X480 resolution.
- Model: Microsoft LifeCam HD3000
- Source: Already Owned
- Price: $23

LCD Display -

- Description: HDMI-driven backlit LCD panel for computer output. Shows camera output and program information
- Model: “7 inch 800*480 LCD Display Driver Board”
- Source: Ebay.com
    - URL: http://www.ebay.com/itm/321988152298
- Price: $27.54

Lessons Learned

1. Originally, the plan was to power all of the launcher from the computer PSU. At the first test, the PSU shut off immediately upon turning on the motors. We estimate that the motors for the Nerf Launcher need almost 30A at startup.
    a. Solution: Just use the batteries intended with the product.

2. Servos require a PWM train with a period of 20ms (50hz). The minimum frequency for the hardware PWM on the PIC16F88 was higher than that, and even with a supported frequency there were only two hardware PWM channels on the PIC, when four were needed. Rather than crawl through data sheets to find a PIC with the features needed, we created our own PWM timing system using the timer 0 counter overflows.
    a. Originally, the timer started an interrupt routine every 20ms, to create the motor signals. In the end, this interfered too much with serial communication, so another method was implemented.
    b. The timer was configured to set a flag every 20ms. In the main loop, this flag was checked. If the flag was raised, the motors were given signals. This allows the serial communication to happen in line with motor communication, preventing function collision. The servos are tolerant to the changes in timing, responding well to signals from about 13 to 25 ms apart.

3. Getting the small ARM-based ODROID computer working correctly took a lot of work. The saying goes “Linux is free if your time is worth nothing”, and it was certainly true for this project. Compiling the OpenCV library from source took 46 hours, and had to be performed multiple times due to various issues. Getting the audio working required several hours of searching the internet and modifying configuration files. Getting the device to login automatically was a hole that sunk a couple days, without any resolution. Working with single board computers is very frustrating, and if the more powerful Raspberry Pi 3 had been available before the project started, we would have chosen it for the better community support.

4. In the beginning, the camera vision code was developed on a desktop PC. The code involved several steps to recognize faces and circles in the image, and it was working well. However, once this code was migrated to the Odroid, the execution speed dropped dramatically, with a frame being processed almost 2.5 seconds after it was captured in real life. This was unacceptable, so the image processing code was stripped down to a bare minimum, with the code now running at a full 30fps (as fast as the camera can supply images). While the object recognition is not quite as reliable, careful use of thresholding and selecting a particular target has allowed the responsiveness of the code to increase to an acceptable level.

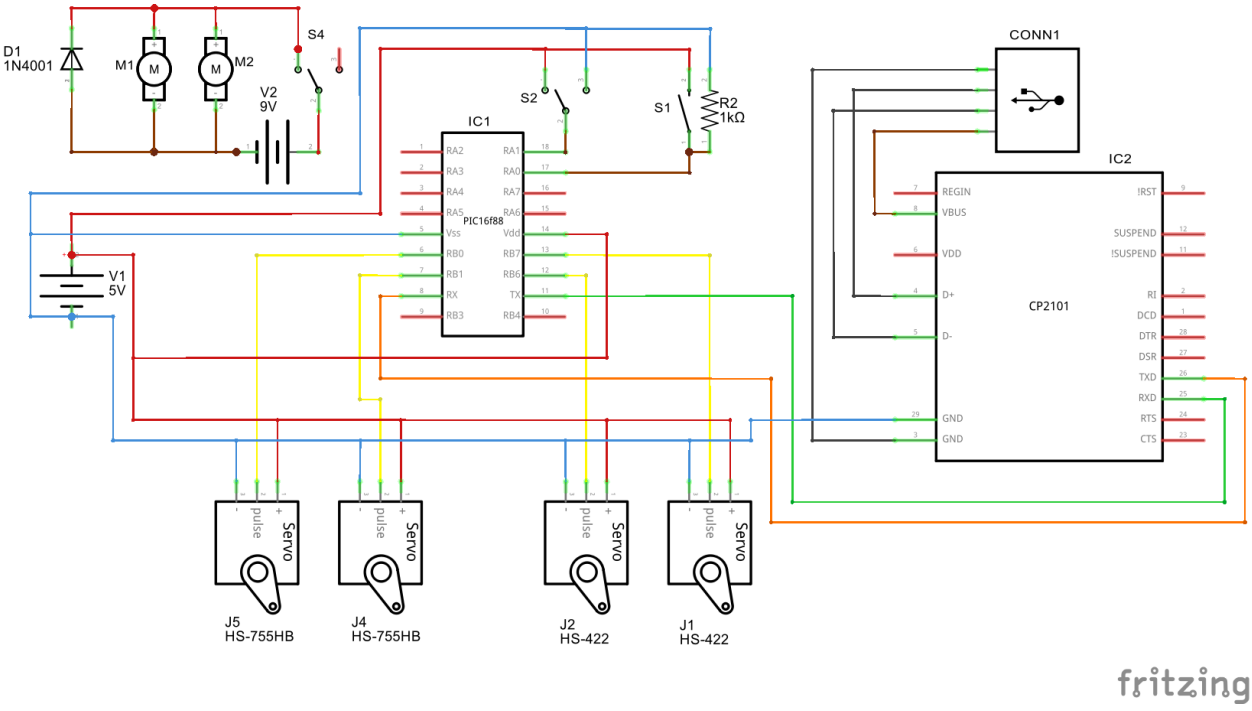
5. When working with Op amp make sure you understand what your max gain is and max current supplied. We tried to amplify the signal to a 40W speaker but started with a gain of 400 (max was 200). We tried another one and it still didn't work, turns out the 741 Op amp couldn't provide enough current for properly amplify the signal.

Appendix:

Decision Matrix

	Motorized Longboard	Inverted Pendulum	Pong table	Maze Solver	Turret	Multipliers
Cost	2	2	3	1	3	-3
Complexity	2	4	1	3	4	-10
Audio	1	1	2	1	1	5
Actuators/Hardware	3	4	1	2	5	7
Control systems	3	4	1	2	5	8
Environmental input	3	2	1	4	5	6
Human input	4	2	5	1	1	5
Display	3	1	5	2	4	4
Totals	74	45	57	39	82	

Detailed Wiring Diagram



Bill of Materials

Part Name	Part Cost (each)	Part Count	Part Cost (total)	Total Cost
Odroid C1+	$48.99	1	$48.99	314.8
FTDI adapter	$6.99	1	$6.99	
Nerf Zeus blaster	$52.46	1	$52.46	
Microsoft LifeCam HD3000	$23.00	1	$23.00	
7" HDMI screen	$27.54	1	$27.54	
Giant scale servo	$31.00	2	$62.00	
Standard servo	$15.97	2	$31.94	
Scrap metal	$10.00	1	$10.00	
Polycarb sheets	$12.00	1	$12.00	
Hardware/pipe	$25.00	1	$25.00	
Switches	$12.00	1	$12.00	
Target Ball	$2.88	1	$2.88	

Software - ODROID Java Code

Uses libraries OpenCV and RXTX:
OpenCV computer vision library: http://opencv.org/
RXTX serial communication library: http://rxtx.qbang.org/wiki/index.php/Main_Page
Part of the image display code sourced from a StackOverflow answer by "dannyxyz22":
http://stackoverflow.com/questions/15670933/opencv-java-load-image-to-gui

```
package mech307Turret;

import org.opencv.core.*;
import org.opencv.core.Point;
import org.opencv.imgproc.*;
import java.awt.*;
import java.awt.image.*;
import javax.swing.*;
import org.opencv.videoio.VideoCapture;
import gnu.io.*;
import java.io.*;
import java.util.*;

public class Hello
{
   static Enumeration portList;
   static CommPortIdentifier portId;
   static SerialPort serialPort;
   static OutputStream outputStream;
   static InputStream inputStream;

  /*initialize the sliders for adjusting the threshold*/
```

```
public static JSlider hLow = new JSlider(JSlider.HORIZONTAL, 0, 180, 5);
public static JSlider hHigh = new JSlider(JSlider.HORIZONTAL, 0, 180, 5);
public static JSlider sLow = new JSlider(JSlider.HORIZONTAL, 0, 255, 5);
public static JSlider sHigh = new JSlider(JSlider.HORIZONTAL, 0, 255, 5);
public static JSlider vLow = new JSlider(JSlider.HORIZONTAL, 0, 255, 5);
public static JSlider vHigh = new JSlider(JSlider.HORIZONTAL, 0, 255, 5);

public static void main( String[] args )
{
        openSerialPort();

        /*Load the OpenCV shared library and make sure that it's working properly by
        **printing an identity matrix*/
         System.load("/usr/local/share/OpenCV/java/libopencv_java310.so");
         Mat mat = Mat.eye( 3, 3, CvType.CV_8UC1 );
        System.out.println( "mat = " + mat.dump() );


        //setup preview window
        addSliderFrame();
        JLabel output1 = makeWindow("Output1", 700, 525, 700, 0);

        //Initialize the variables for the FPS timer
        long oldTime=0;
        long newTime=1;
        int fps = 0;

        //Initialize matrices to hold image data
        Mat framein = new Mat(480, 640, CvType.CV_8UC1);
        Mat framegray = new Mat(480, 640, CvType.CV_8UC1);
        Mat framecvt = new Mat(480, 640, CvType.CV_8UC1);
        Mat frameout = new Mat(480, 640, CvType.CV_8UC1);
         VideoCapture capture = new VideoCapture(0);

        //Initialize other loop variables
        int posX = -1;
         int posY = -1;
         long area = 0;
        boolean leftRight = false; //false=left, true=right
         boolean scanning = false;
        int actualX = 1600;
         int actualY = 1425;
         String outString = "";
         int inByte;
         long lockCount = 0;
         int scanCount = 0;
         Moments mo = new Moments();
```

```
	/*Begin the loop!!!*/
	  while(true){
		oldTime = System.nanoTime();
		capture.read(framein);
		Imgproc.cvtColor(framein, framecvt, Imgproc.COLOR_RGB2HSV);
		Core.inRange(framecvt, new Scalar(hLow.getValue(), sLow.getValue(),
vLow.getValue()), new Scalar(hHigh.getValue(), sHigh.getValue(), vHigh.getValue()), framegray);
	//threshold image to black and white
		mo = Imgproc.moments(framegray); // m10/area=x, m01/area=y, m00=area
		area = (long)mo.get_m00();
		posX = 0;
		posY = 0;
		if(area>100000)
		{
			//object confirmed!
			 posX = (int)(mo.get_m10()/area);
			 posY = (int)(mo.get_m01()/area);
			 scanCount = 60;
		}
		else
		{
			//No object: start scanning
			 scanning = true;
			 if(scanCount > 60)
			 {
				scanCount = 0;
				try {
					Runtime.getRuntime().exec("aplay
/home/gavin/sounds/sentry_scan2.wav");
				} catch (IOException e) {
					e.printStackTrace();
				}
			 }
			 scanCount++;
			 if(leftRight == false)
			 {
				if(actualX == 2400)
				{
					actualX += -10;
					leftRight = true;
				}
				else
				{
					actualX += 10;
				}
			 }
			 else if(leftRight == true){
				if(actualX == 800)
				{
```

```
                        actualX += 10;
                        leftRight = false;
                }
                else
                {
                        actualX += -10;
                }
        }
        if(actualY > 1428)
        {
                actualY += -3;
        }
        else if(actualY < 1422)
        {
                actualY += 3;
        }
        //actualY = 1425;
}

if(posX != 0 && posY != 0)          //BEGIN motor duration set block
{
        actualX += (posX-320)/13;
        actualY += -(posY-240)/11;
}
outString = "X";
if(actualX > 2400){actualX = 2400;}       //Make sure motors are within bounds
if(actualX < 800){actualX = 800;}
if(actualY > 1700){actualY = 1700;}
if(actualY < 1150){actualY = 1150;}
if(actualX < 1000)
{
        outString += "0" + actualX;
}
else
{
        outString += actualX;
}
if(actualY < 1000)
{
        outString += "0" + actualY;
}
else
{
        outString += actualY;
}//END motor duration set block
//BEGIN lock-on block
if((posX > 280 && posX < 360) && (posY > 220 && posY < 260))
{
        lockCount ++;
```

```
                        if(lockCount == 7)
                        {
                                //Locked On!!!
                                outString += "1";
                                try {
                                        Runtime.getRuntime().exec("aplay
/home/gavin/sounds/sentry_spot_client.wav");
                                } catch (IOException e) {
                                        e.printStackTrace();
                                }
                        }
                        else if (lockCount > 7)
                        {
                                outString += "1";
                        }
                        else
                        {
                                outString += "0";
                        }
                }
                else
                {
                        lockCount = 0;
                        outString += "0";
                }//END lock-on block

                try {
          outputStream.write(outString.getBytes());          //SERIAL OUT
        } catch (IOException e) {e.printStackTrace();}

                inByte = 0;
                try{
                        if(inputStream.available() > 0)
                        {
                                inByte = inputStream.read();    //SERIAL IN
                                System.out.println(inByte);
                        }
                } catch (IOException e) { System.err.println(e);}
                if(inByte == 49)
                {

                        //FIRE!!!
                        try {
                                Runtime.getRuntime().exec("aplay
/home/gavin/sounds/sentry_shoot.wav");
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
```

```
                    }
                    if(inByte == 50)
                    {
                            //play disabled sound
                            System.out.println("DISABLED");
                    }

                   //draw the crosshairs
                    Imgproc.line(frameout, new Point(0, 240), new Point(640, 240), new
Scalar(255,255,255));
                    Imgproc.line(frameout, new Point(320, 0), new Point(320, 480), new
Scalar(255,255,255));
                   //add debugging text
                    addFrameText(frameout, "Area: " + Double.toString(area), 10, 20);
                    addFrameText(frameout, "FPS: " + Integer.toString(fps), 10, 40);
                    addFrameText(frameout, "hLow: " + hLow.getValue(), 10, 60);
                    addFrameText(frameout, "hHigh: " + hHigh.getValue(), 10, 80);
                    addFrameText(frameout, "sLow: " + sLow.getValue(), 10, 100);
                    addFrameText(frameout, "sHigh: " + sHigh.getValue(), 10, 120);
                    addFrameText(frameout, "vLow: " + vLow.getValue(), 10, 140);
                    addFrameText(frameout, "vHigh: " + vHigh.getValue(), 10, 160);
                    addFrameText(frameout, "posX: " + posX, 10, 180);
                    addFrameText(frameout, "posY: " + posY, 10, 200);
                    addFrameText(frameout, "outString: " + outString, 10, 220);
                    displayFrame(frameout, output1);

                    newTime = System.nanoTime();
                   //How long this loop took, frequency (framerate)
                    fps=(int)(1000000000.0/(newTime-oldTime));    }
  }

/*Initialize the serial port, baud 9600, 8n1*/
  public static void openSerialPort(){
            portList = CommPortIdentifier.getPortIdentifiers();
            System.out.println("serial ports? " + portList.hasMoreElements());
    while (portList.hasMoreElements()) {
             System.out.println("inside loop");
       portId = (CommPortIdentifier) portList.nextElement();
       if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {
         if (portId.getName().equals("/dev/ttyUSB0")) {
            try {
               serialPort = (SerialPort)
                  portId.open("TurretApp", 2000);
            } catch (PortInUseException e) {}
            try {
               outputStream = serialPort.getOutputStream();
            } catch (IOException e) {}
            try {
```

```
                inputStream = serialPort.getInputStream();
            } catch (IOException e) {}
            try {
                serialPort.setSerialPortParams(9600,
                    SerialPort.DATABITS_8,
                    SerialPort.STOPBITS_1,
                    SerialPort.PARITY_NONE);
            } catch (UnsupportedCommOperationException e) {}
        }
    }
  }
}

/*Code from stackoverflow. Gets an image ready for display*/
  public static Image toBufferedImage(Mat m){
            int type = BufferedImage.TYPE_BYTE_GRAY;
            if ( m.channels() > 1 ) {
                type = BufferedImage.TYPE_3BYTE_BGR;
            }
            int bufferSize = m.channels()*m.cols()*m.rows();
            byte [] b = new byte[bufferSize];
            m.get(0,0,b); // get all the pixels
            BufferedImage image = new BufferedImage(m.cols(),m.rows(), type);
            final byte[] targetPixels = ((DataBufferByte) image.getRaster().getDataBuffer()).getData();
            System.arraycopy(b, 0, targetPixels, 0, b.length);
            return image;
        }

/*Updates the display frame with the new image*/
 public static void displayFrame(Mat frame, JLabel panel)
 {
        panel.setIcon(new ImageIcon(toBufferedImage(frame)));
        panel.repaint();
 }

/*Initializes the display window*/
 public static JLabel makeWindow(String name, int sizex, int sizey, int posx, int posy)
 {
        JFrame jframe=new JFrame(name);
        jframe.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel label = new JLabel();
        jframe.setExtendedState(JFrame.MAXIMIZED_BOTH);
        jframe.setUndecorated(true);
        jframe.setContentPane(label);
        //jframe.setSize(sizex, sizey);
        //jframe.setLocation(posx, posy);
        jframe.setVisible(true);
```

```
		return label;
	}

/*Add text to the image specified*/
	public static void addFrameText(Mat frame, String text, int x, int y)
	{
		Imgproc.putText(frame,  text,  new Point(x,y), 0, .5, new Scalar(255,255,255));
	}

/*Initialize the frame with threshold sliders*/
	public static void addSliderFrame()
	{
		JFrame sliderFrame=new JFrame("sliderFrame");
		JPanel sliderPanel = new JPanel();
		sliderPanel.setLayout(new FlowLayout(FlowLayout.LEFT));
		sliderFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		sliderFrame.setVisible(true);
		sliderFrame.setSize(400, 400);
		hLow.setValue(56); //Slider defaults
		hHigh.setValue(75);
		sLow.setValue(119);
		sHigh.setValue(223);
		vLow.setValue(130);
		vHigh.setValue(234);
		sliderPanel.add(hLow);
		sliderPanel.add(hHigh);
		sliderPanel.add(sLow);
		sliderPanel.add(sHigh);
		sliderPanel.add(vLow);
		sliderPanel.add(vHigh);
		sliderFrame.add(sliderPanel);
	}
}
```

Software - PIC PICBasic Pro Code

```
' robocode.bas

#CONFIG
   __CONFIG _CONFIG1, _INTRC_IO & _PWRTE_ON & _MCLR_OFF & _LVP_OFF
#ENDCONFIG

' Set the internal oscillator frequency to 8 MHz
DEFINE OSC 8
OSCCON.4 = 1
OSCCON.5 = 1
```

```
OSCCON.6 = 1

TRISB = %00000100       'Set serial input
TRISA = %11111111       ' Set PORTA to all input

ANSEL = 0

' Set TRM0 flag interrupts every ~20ms
OPTION_REG = %11000111
INTCON = %10100000
TMR0 = 102



motorDuration1                  var             word
motorDuration2                  var             word
motorDuration3                  var             word
motorDuration4                  var             word
fireFlag    var     bit
fireCount var word
recoverCount var word
lockedOn var word
enabled var word

input PORTA.0
input PORTA.1
Low PORTB.0
low PORTB.1
low PORTB.3
low PORTB.6
low PORTB.7
motorDuration1 = 1600
motorDuration2 = 1600
motorDuration3 = 1600
motorDuration4 = 1600
fireFlag = 0
fireCount = 0
recoverCount = 0
lockedOn = 0
enabled = 0

myloop:
   if(PORTA.1 == 1) then            'enabled
      enabled = 0
      if(PORTA.0 == 1) then         'Fire button pressed
         low PORTB.3                'Signal LED on
         if(fireCount == 0 && recoverCount > 35 && lockedOn == 1) then
            fireCount = 1           'Start the fire timer
```

```
        recoverCount = 0
      endif
    else
      high PORTB.3              'Signal LED off
    endif
    gosub charin                'Read data from the ODROID
  returnToLoop:                 'Re-entry point if serial comms time out
    gosub charout               'Write status to ODROID
    gosub motorcontrol          'Make the motors do things
  elseif(PORTA.1 == 0) then     ‘disabled
    gosub disabledReport
  endif
Goto myloop                     'LOOP

charin:
  serin2 PORTB.2, 84, 1, returnToLoop, [wait("X"), dec4 motorDuration1, dec4 motorDuration2, dec1
lockedOn]
  if(motorDuration1 > 2600)then motorDuration1 = 2600
  if(motorDuration1 < 500) then motorDuration1 = 500
  if(motorDuration2 > 2600)then motorDuration2 = 2600
  if(motorDuration2 < 500) then motorDuration2 = 500
return

charout:
  if(firecount == 1) then
    serout2 PORTB.5, 84, [dec 1]
    fireCount = 2
  endif
return

disabledReport:
  if(enabled == 0) then
    serout2 PORTB.5, 84, [dec 2]
    enabled = 1
  endif
return

motorcontrol:
  if(INTCON.2 == 1) then                    'Check if the 20ms flag has been set
    if(fireCount > 0)then                   ‘Firing?
      if(fireCount == 1 or fireCount == 2) then
        motorDuration3 = 1500               'motor3: 2100->1500->2100
        motorDuration4 = 1400               'motor4: 1400->2500->1400
      elseif(fireCount == 10) then
        motorDuration3 = 1500
        motorDuration4 = 2500
      elseif(fireCount == 21) then
        motorDuration3 = 2100
```

```
    motorDuration4 = 1400
  endif
  if(motorDuration3 > motorDuration4) then          'Make signals overlap to save downtime
    HIGH PORTB.6
    high PORTB.7
    pauseus motorDuration4
    low PORTB.7
    pauseus motorDuration3 - motorDuration4
    low PORTB.6
  elseif(motorDuration4 > motorDuration3) then
    HIGH PORTB.6
    high PORTB.7
    pauseus motorDuration3
    low PORTB.6
    pauseus motorDuration4 - motorDuration3
    low PORTB.7
  else
    HIGH PORTB.6
    high PORTB.7
    pauseus motorDuration3
    low PORTB.6
    low PORTB.7
  endif                                             'End motor 3&4 signals
  if(fireCount == 40) then                          'Increase firing timer
    fireCount = 0
  else
    fireCount = fireCount + 1
  endif
endif

if(recoverCount < 200) then                         'Make sure there's time to recover
  recoverCount = recoverCount + 1                   'before shooting again
endif

if(motorDuration1 > motorDuration2) then            'Motor 1&2 signals
  high PORTB.0
  high PORTB.1
  pauseus motorDuration2
  low PORTB.1
  pauseus motorDuration1-motorDuration2
  low PORTB.0
elseif(motorDuration2 > motorDuration1) then
  high PORTB.0
  high PORTB.1
  pauseus motorDuration1
  low PORTB.0
  pauseus motorDuration2-motorDuration1
  low PORTB.1
```

```
      else
        high PORTB.0
        high PORTB.1
        pauseus motorDuration1
        low PORTB.0
        low PORTB.1
      endif                                   'END motor signals
      INTCON.2 = 0                            'Clear timer0 flag
      TMR0 = 102                              ‘Set timer0 counter for another 20ms
    endif
return
END
```