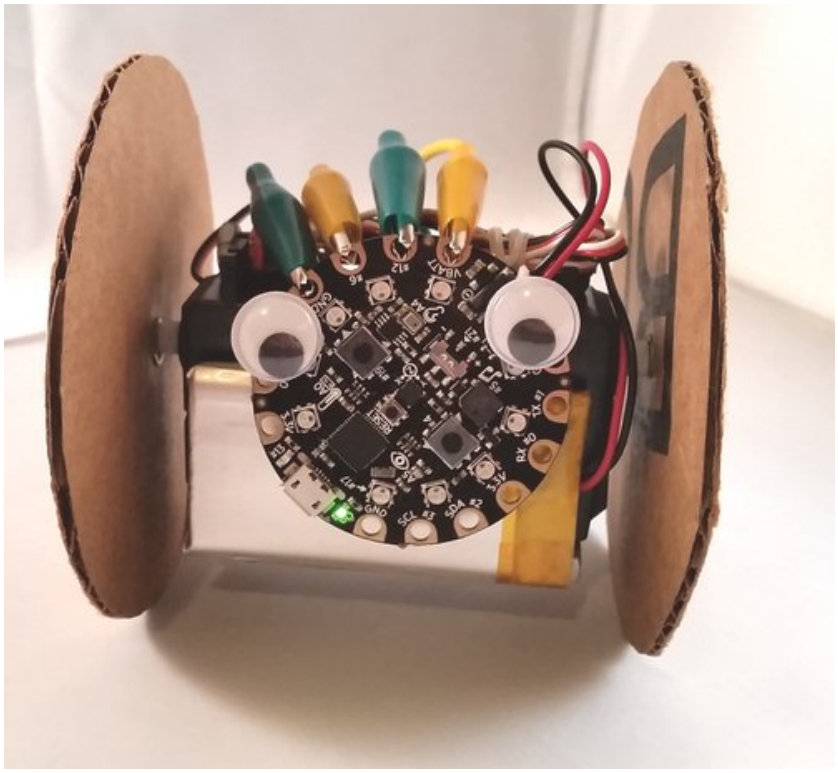
adafruit learning system

Circuit Playground Sound-Controlled Robot

Created by Mike Barela



Last updated on 2017-07-14 11:53:19 PM UTC

Guide Contents

Guide Contents 2
Introduction 3
Robot Mechanics 5
- Body 5
- Wheels 6
- Servo Cables 7

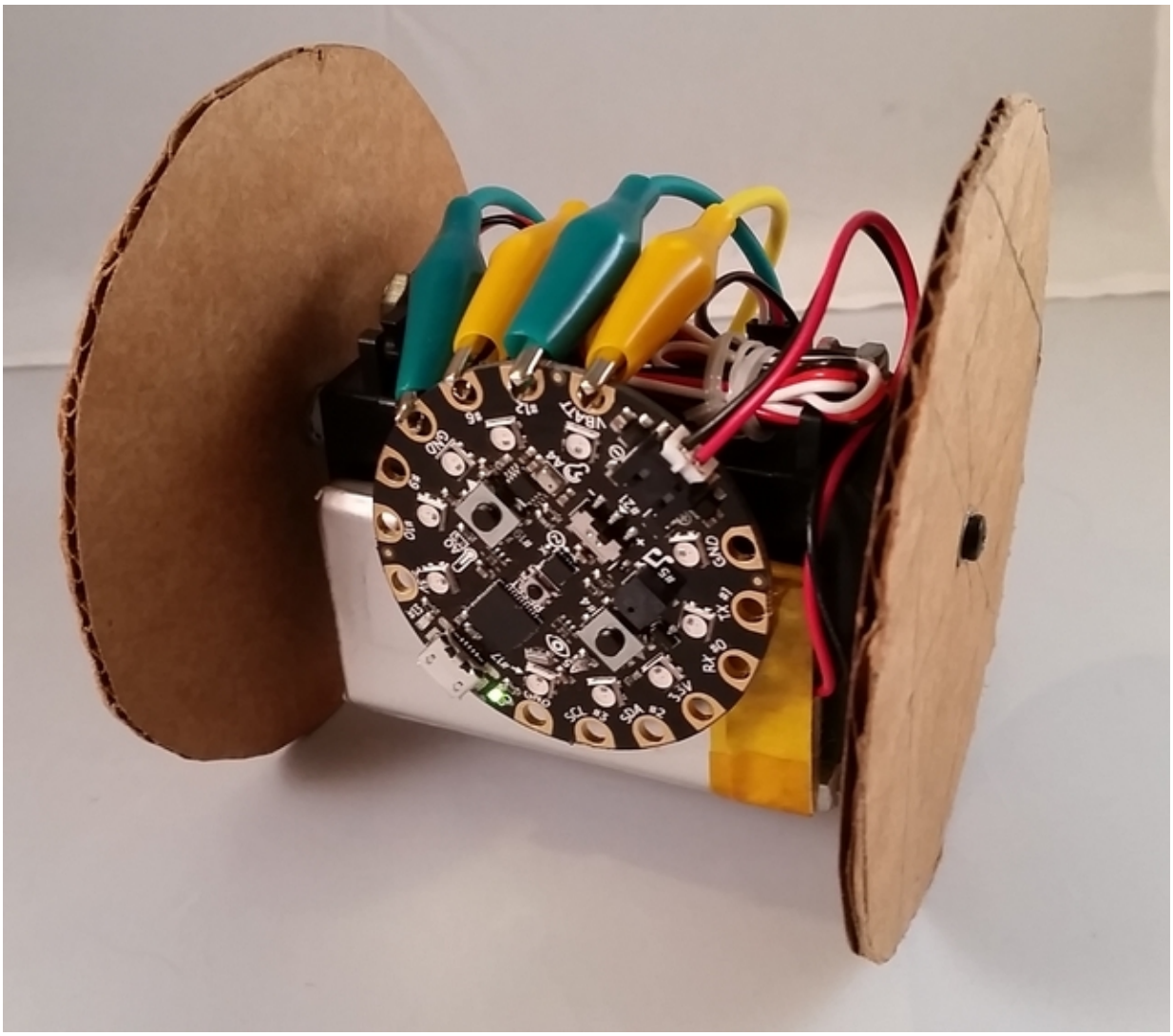
Add Circuit Playground 9
Using the Microphone 13
Continuous Rotation Servos 17
Whistle Bot 20
- Using the Whistle Bot 24
- Variations on the Whistle Bot 24

Tone Controlled Robot 26
Build the Controller 31
Use 37
Going Further 38
- Control 38
- Robotics 39

Introduction

Circuit Playground, Adafruit's multi-faceted experimenter board, may be used in many of the maker projects on the Adafruit Learning System and other tutorials on the Internet. This project extends Circuit Playground into the field of robotics.

With this project, you can make a robot with Circuit Playground that is low cost, easy to build and program. The techniques used are applicable to using continuous servos, sound processing, and robotics techniques.



There are two general methods of controlling robots: via a set of wires from a person to the robot or wirelessly in some fashion. While Circuit Playground contains a plethora of capabilities, a wireless radio is (currently) not onboard. Looking at the input sensors shows two which might allow non-wired control: the light sensor and the sound sensor. The light sensor could be a

challenge to use as a control, especially in bright light. I believe there is more flexibility with the microphone.

At present, use of a Alexa/Siri type voice control system takes more processing power than is available on the Circuit Playground's ATmega 32u4 microcontroller. Circuit Playground is very able to listen for tones at certain frequencies with the capabilities built into the Circuit Playground software library.

This project is intended to be educational, so there is more detail on how the various parts of the project work. This includes optimal sound values for controlling the robot and how to set the servo motors for no movement. The data gathered in those steps will be used in building the final robot.

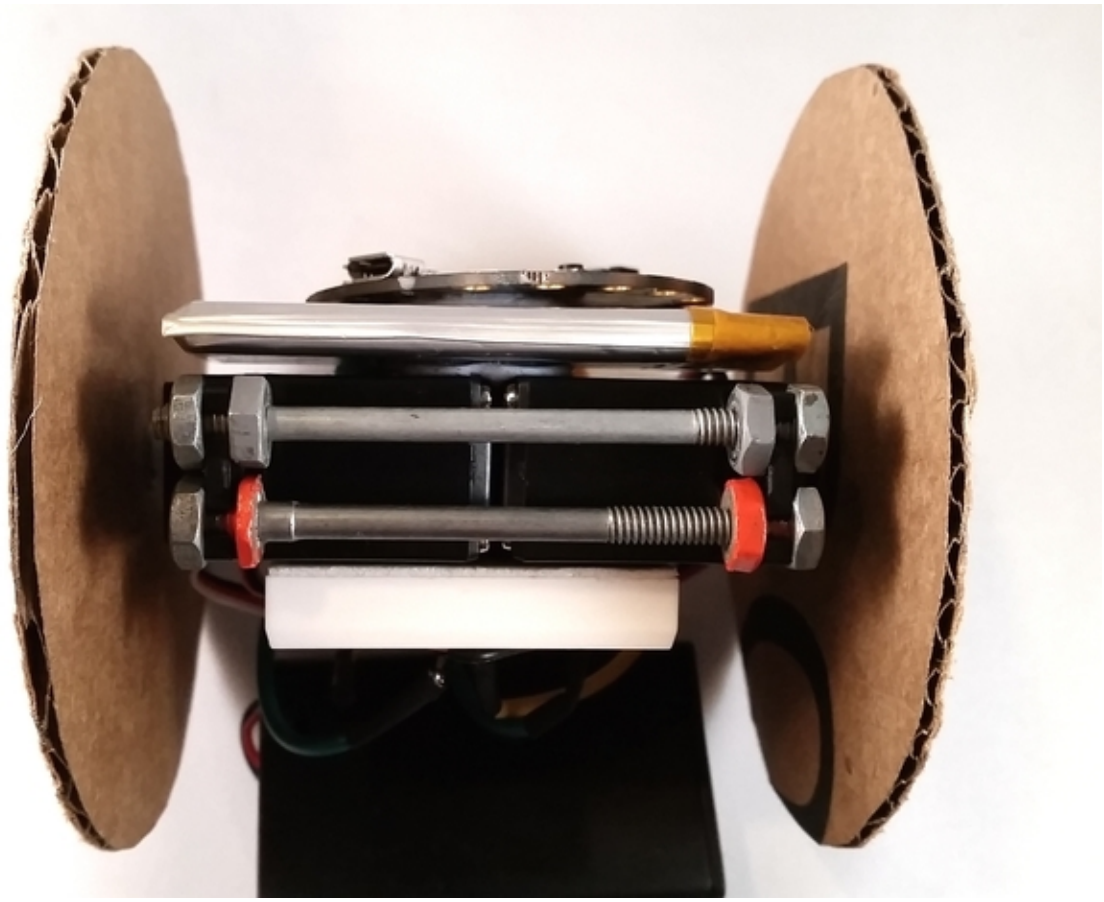
Robot Mechanics

Body

In guilding a simple body, we can just attach the servo motors back to back. This is done using three or four 2.5" / 6.5 mm long screws. You will need 3 nuts for each one so that the head and one nut grips one servo and other two nuts grip the other.

Alternatively you can cut two pieces of wood. Nominal measurements are 2 inches / 57 mm ling by 0.75" / 20 mm by 0.25 inch / 8 mm tall. You may want to measure the space between servos yourself to ensure a snug fit. You can also look to 3D print an appropriate piece. Measuring carefully, possibly with a caliper, is recommended if you design a custom piece. Align to the top and bottom of the servos where the mounting holes are and secure with wood or self tapping screws.

The secured servos serve as a rigid body for mounting other parts including a mini breadboard on one side and a LiPo battery and Circuit Playground on the other.

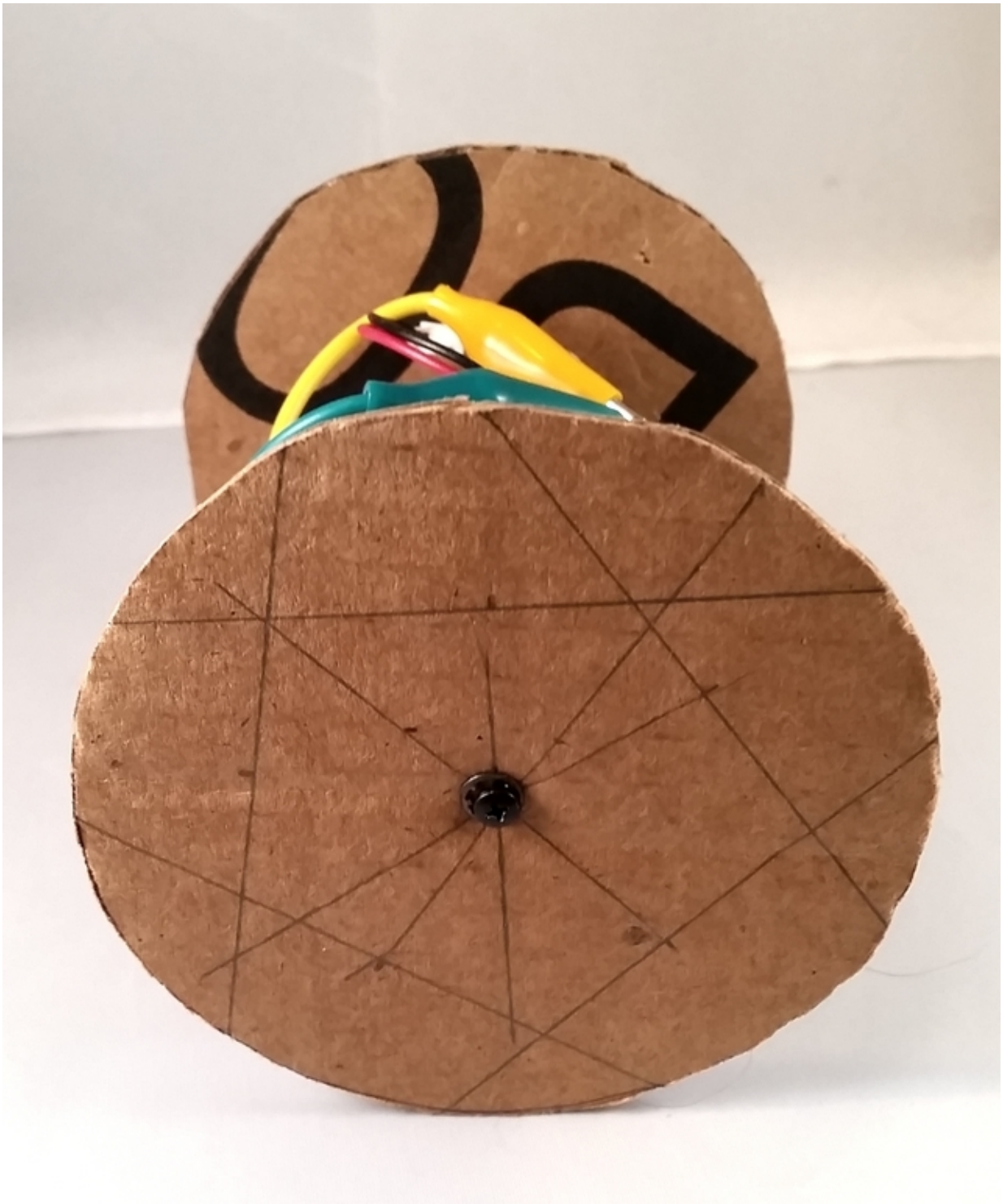


Wheels

I designed the robot to sit pretty close to the ground but this is a design decision you can change without any consequences. I could not source wheels approximately 100 mm in diameter with a Futaba/Parallax servo mount, so I made my own. You can source wheels if you like from a shop specializing in robotics.

Using good stiff cardboard, I made circles with a diameter of 3.75 inches / 95 mm. You can find a can or other round shape at least this size to trace around. You can also draw a circle with a compass. Be sure the circle is nice and round when you cut it out as any flat spots will not allow the robot to roll well. Cut out two circles.

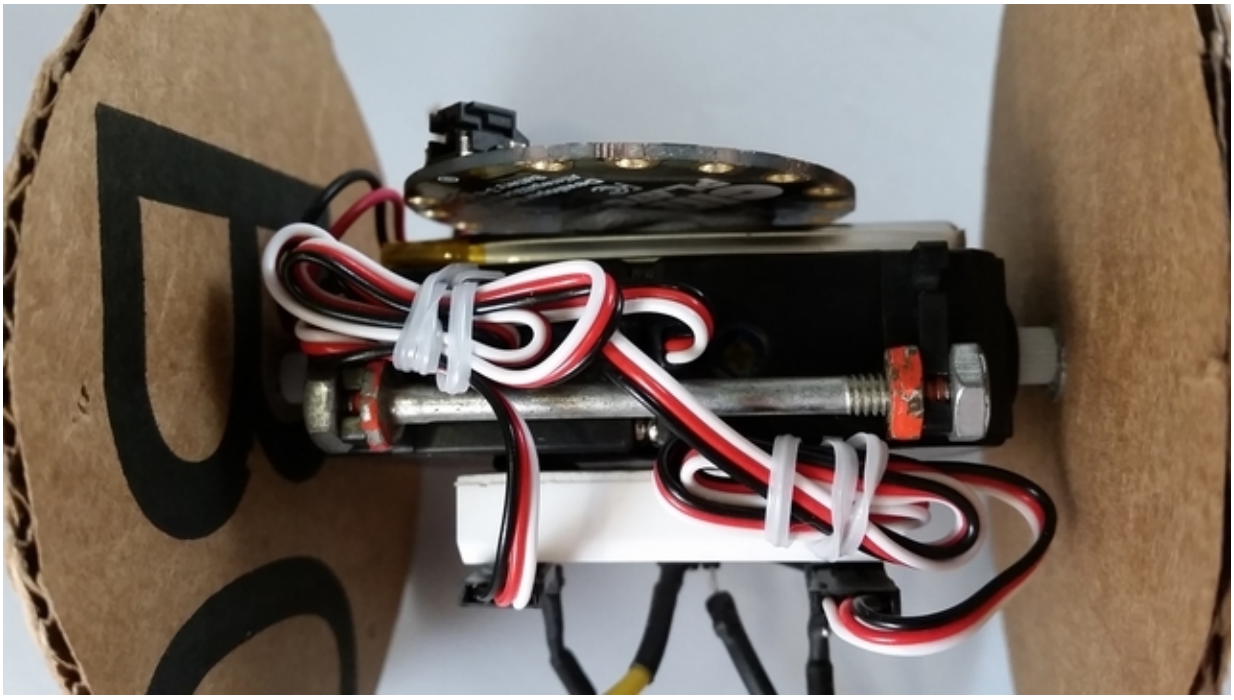
To find the center of the circle, I suggest followingthis tutorial on instructables.com (http://adafru.it/rMf) or you may Google "Find center of a circle". Using a servo screw to carefully poke through and make a hole at the center point only large enough for the screw. If you have some tiny lock washers, they can help keep the wheels screws tight. If the robot is not moving when the servos spin, check the wheel center screws to ensure they are tight.



Servo Cables

As this is an educational build, I did not cut the generous length of servo cable on each motor. Rather the excess is folded and secured with a wire tie and tucked it onto the top of the servo.

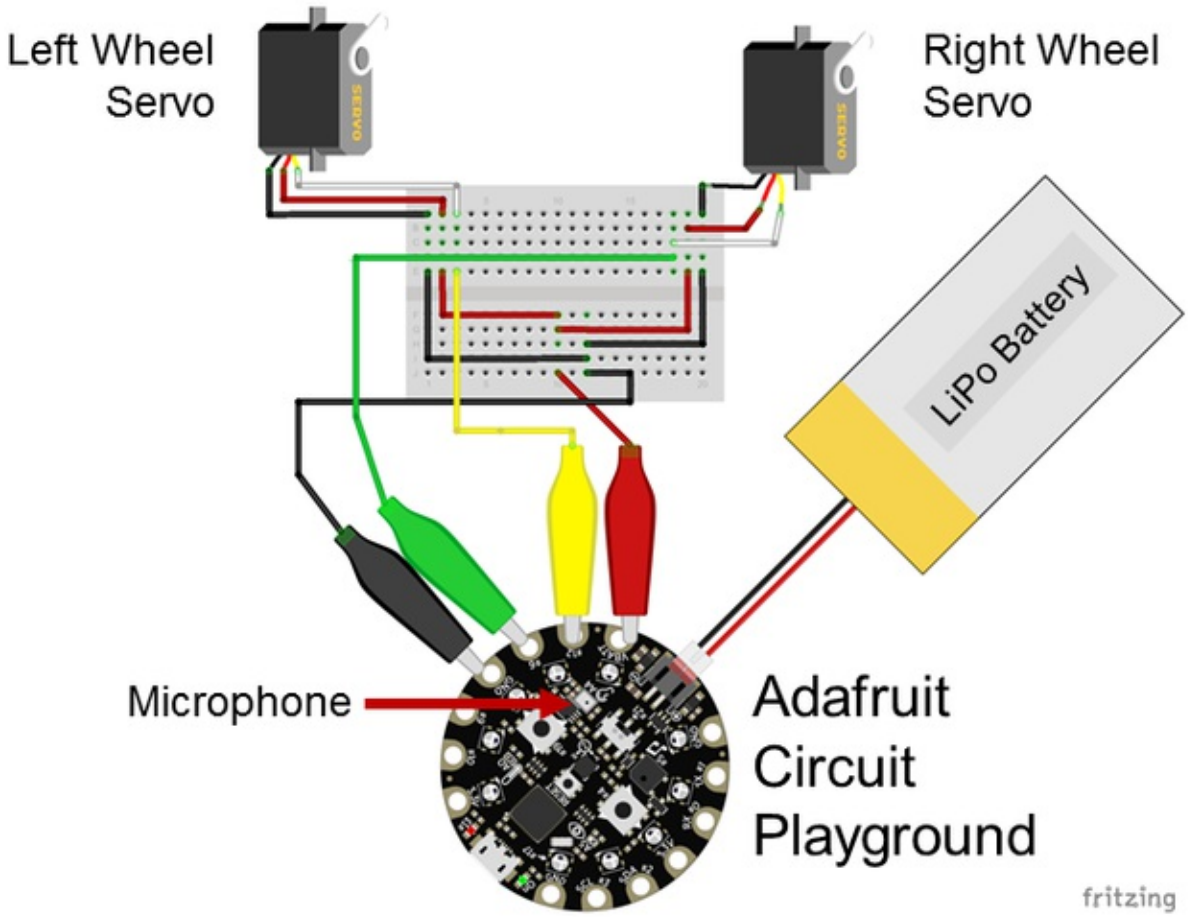
While *technically* these servos require 5V power to run, you can get away with running them from 4V power like the LiPo, and they'll work OK! They'll just run a little weak/slow.



Leave enough slack at the end to just touch the mini breadboard. The ends of the servo cables will be mounting to one side of the breadboard.

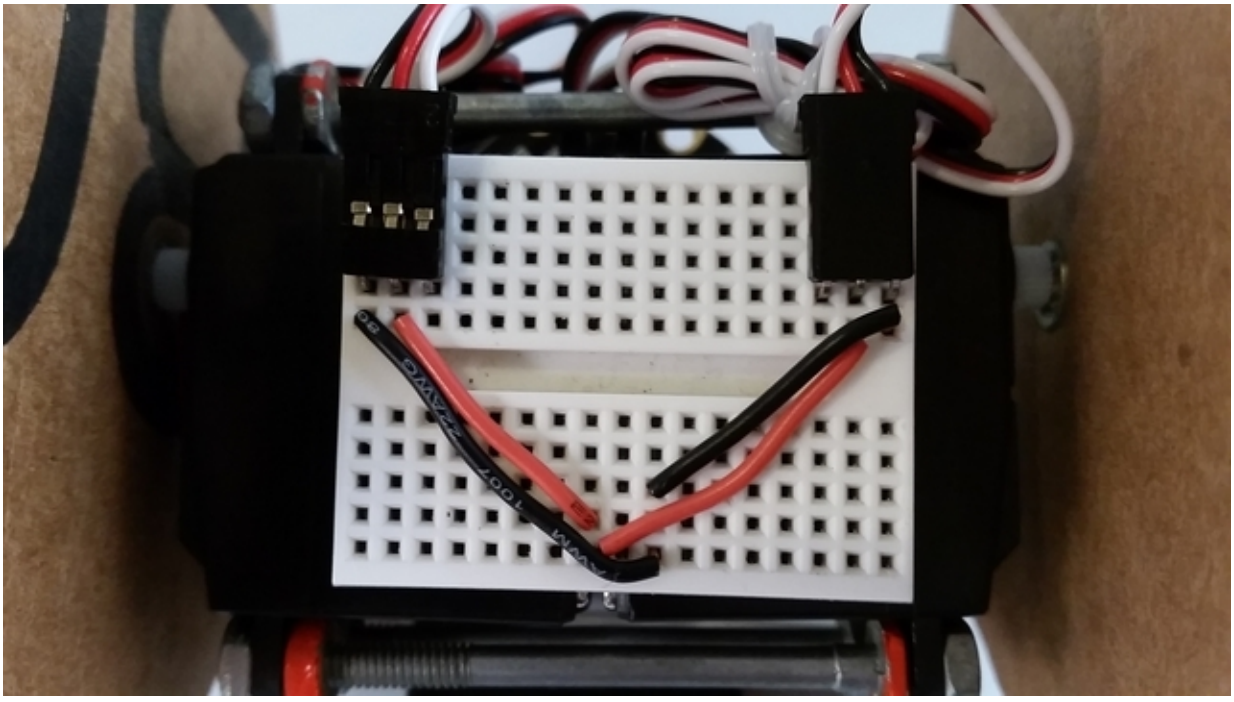
Add Circuit Playground

The Fritzing diagram for the robot is shown below. The number of parts and connections is fairly low. Hookup wire for the breadboard makes for sturdier connections than premade breadboard wires.



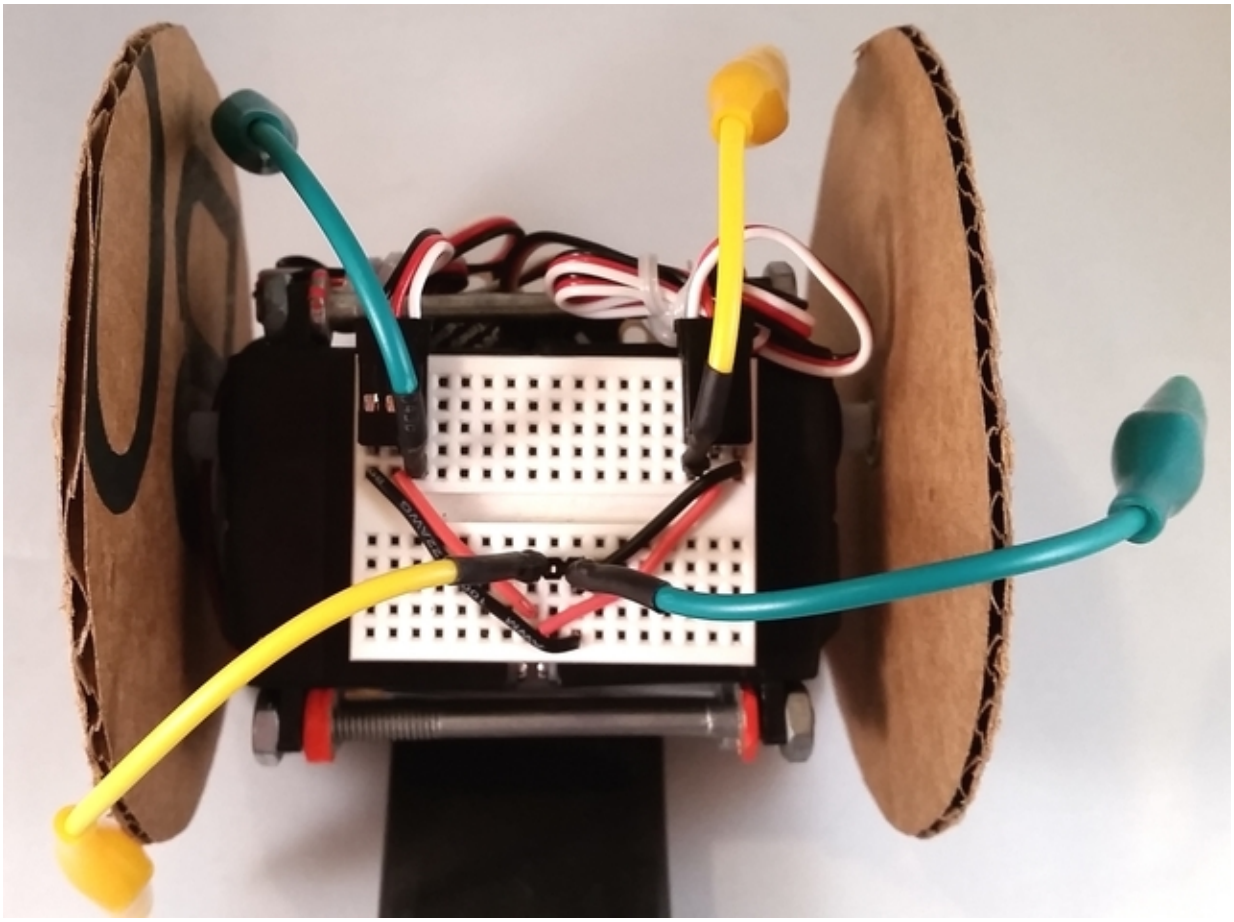
The alligator clip to male jumpers allow connection between the Circuit Playground and the robot. You can buy them from Adafruit as part #3255 or make them yourself from clips and pins.

Peal the adhesive off the mini breadboard (or put some adhesive on the back if the breadboard lacks it) and stick to one side of the robot body defined by the two servos. This will be our wiring side.

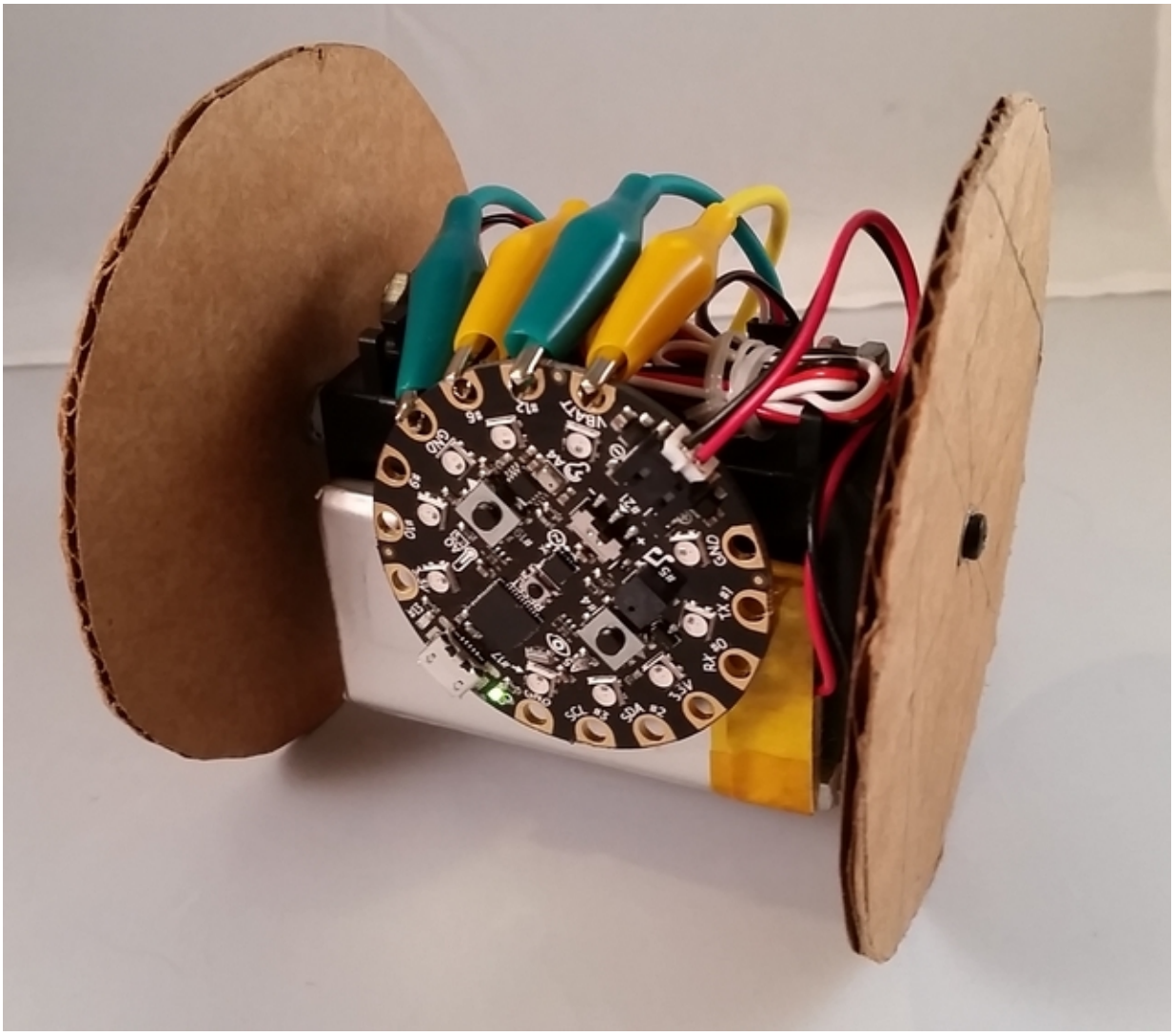


We need to attach the servo leads to the board as shown. Cut individual 90 degree header pins and take off the plastic square. Make 6 of them. Place them on the locations in the 4th row from the top with the ends pointing to the "top" defined as where the servo wires are at. This allows for good breadboard and servo cable connections. Add red and black solid wire to be able to power the servos off the same battery voltage.

Next add the alligator clips, connecting the two white servo control wires and the two joined power lines. I suggest Black for ground GND, red for power VBATT, and two other colors for the servo control wires. In the picture below, I did not have the newly stocked Adafruit multicolor cables.



Take the LiPo battery and stick it to the non-breadboard side of the robot body with the power lead sticking out towards the left wheel. Use Blu Tack, thick double-sided tape or other non-permanent adhesive. Place additional non-permanent adhesive on the back of the Circuit Playground board and carefully press it onto the battery at a 45 degree angle.



The Circuit Playground should be up a bit so that when we drape the alligator clips over from the breadboard side, they comfortably clip onto the 4 top pins (one reason we tilted the board at 45 degrees, the other two being good for the power and USB connectors).

The joint red connections on the breadboard go to the Circuit Playground VBATT connection. The joint black connections use the clip that is connected to the Circuit Playground GND connection. The right motor (looking from the breadboard side) goes to Pin #6 and the left motor connection to Circuit Playground Pin #12.

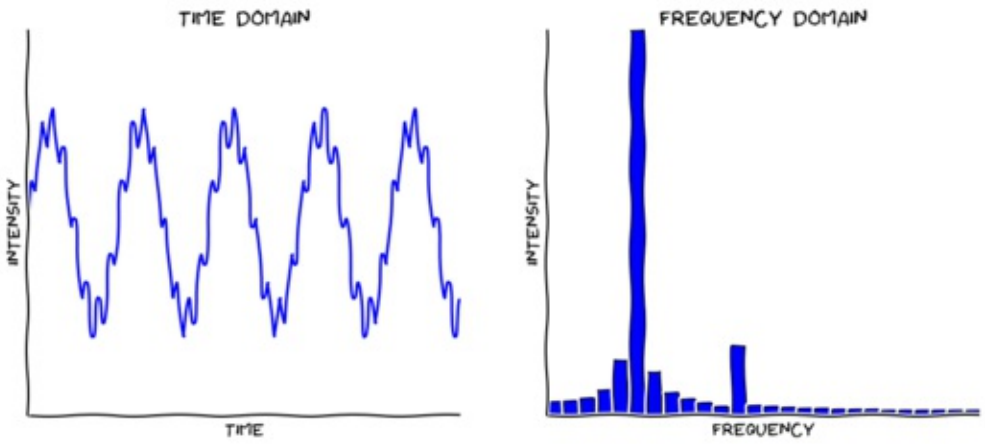
That's it. Check your connections against the diagram and pictures to be sure everything matches up.

Using the Microphone

It sounds like it would be an easy task: make a certain sound and have the microphone detect it and some software makes a decision based on what the microphone just heard. But most sounds are polyphonic, made up of many different sounds at multiple frequencies. This is very hard to match up in software.

But our microcontroller can match monophonic, single frequency sounds. It still is not so easy. You need to take into account a sound occuring over time. So to simplify a sound's identification, we want to convert the microphone input to the frequency domain. If we know the likely frequency of a sound, we can more easily compare it to the sound frequency you expect.

What is the method we transform sound to frequency? There are several methematical transformations that take intensities sensed over time and calculate a set of frequencies. The main method is called the Fast Fourier Transform (commonly abbreviated as FFT). For more in-depth information, see this Adafruit Learning Center tutorial (http://adafru.it/qMB) and Wikipedia (http://adafru.it/qMC).



XKCD-style graph generated by http://matplotlib.org/users/whats_new.html#xkcd-style-sketch-plotting (http://adafru.it/cM8) by Tony DiCola.

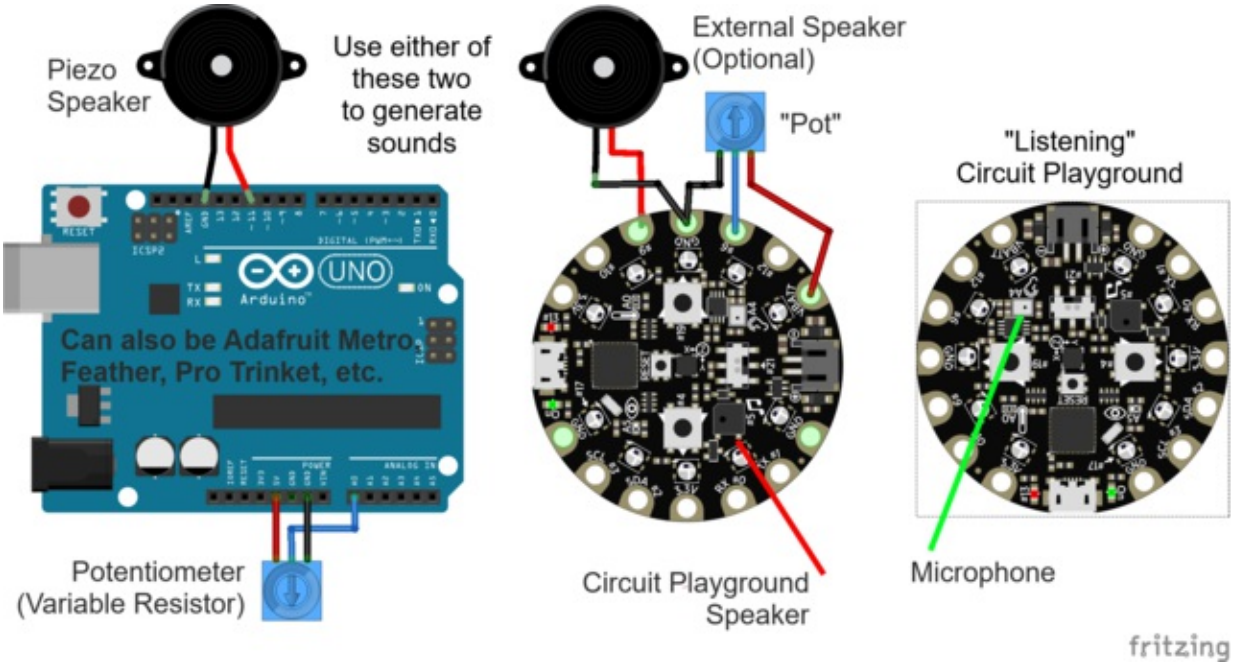
In the diagrams, you can see a continuous line graph transforms to a bar graph. That's part of the math. We look for the tallest bar and see where it lands on the x-axis which is the frequency of the sound.

The Circuit Playground library has a FFT routine built-in for our use. The routine CircuitPlayground.mic.fft takes in sound samples and puts the frequencies it finds into 32 frequency bins using some math. To make sure we find the sound we are looking for (as things can change over time), several samples are taken and averaged.

Which of the 32 frequency bins corresponds to sounds we might want to make? I used the list of musical notes in the tutorial Circuit Playground Sound and Music (http://adafru.it/qMD) as a start. But I found I was not getting good results.

The best frequencies to use are shown in the chart below. If you would like to perform sound tests or just have fun, here are the simple circuits and program you can use.

I grabbed an Arduino Uno (http://adafru.it/50) and created one of my "go to" circuits which reads a variable resistor and maps the values to a range of frequency tones to output to a piezo speaker. You can also use a second Circuit Playground as shown in the center diagram, either using an external piezo speaker or the internal speaker. The value of the potentiometer is not critical, any one from 1 kiloohms to 100 kiloohms or more will work fine as it acts as a volage divider to selectdifferent voltages on an analog pin which we translate to variable pitch on the speaker.




```
// Variable Tone Generator Circuit for Arduino Uno or Circuit Playground
// Mike Barela for Adafruit Industries    September, 2016

#define speakerPin 11 // For Circuit Playground change 11 to 9 for
                      //  external Piezo, change to 5 for onboard speaker
#define potPin A0     // For Circuit Playground change A0 to A7

void setup() {
  pinMode(potPin, INPUT);
  pinMode(speakerPin, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  uint16_t potValue;
  uint16_t freq;

  potValue = analogRead(potPin);
```

```
  freq = map(potValue,0,1023,100,8000);
  tone(speakerPin,freq);
  Serial.println(freq);
}
```

Test out your tone generator by turning the potentiometer and listening to the different tones. Hooking up a serial monitor to the USB port, you can see the frequency you have dialed in.

On the "listening" Circuit Playground, you use the onboard microphone and see which frequency bins "max out" (like the peak in the above bar graph) as it "hears" the sound from the sound generator circuit.

For the "listening" Circuit Playground, download the following code:

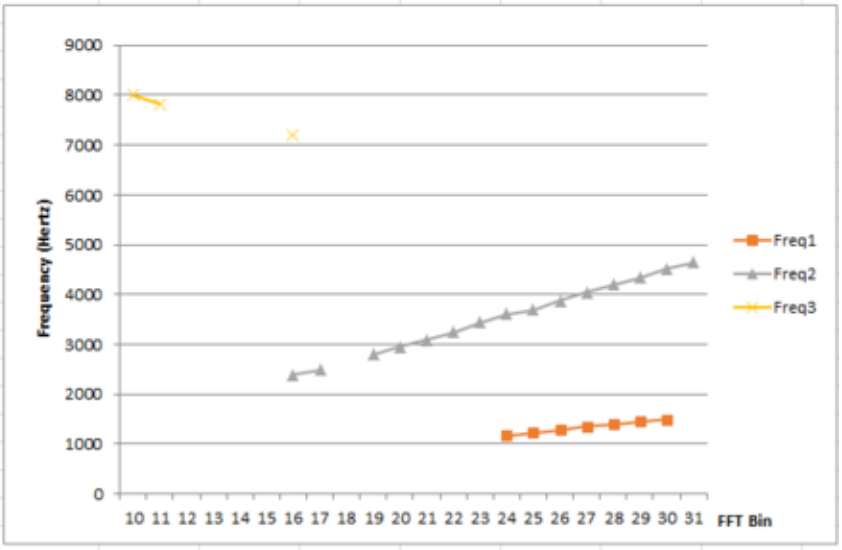
```
// Fast Forier Transform Test Program for Circuit Playground
// Mike Barela for Adafruit Industries    September, 2016

#include <Adafruit_CircuitPlayground.h>

#define BINS   32          // The number of FFT frequency bins
#define FRAMES 4           // This many FFT cycles are averaged

void setup() {
  CircuitPlayground.begin();  // Set up the board library and serial
  Serial.begin(9600);
}

void loop() {
  uint8_t i,j;
  uint16_t spectrum[BINS];     // FFT spectrum output buffer
  uint16_t avg[BINS];          // The average of FRAME "listens"

  for(j=1; j <= FRAMES; j++) {             // We gather data FRAMES times and average it
    CircuitPlayground.mic.fft(spectrum);  // Here is the CP listen and FFT the data routine
    for(i=0; i < BINS; i++) {              // Add for an average
      if(spectrum[i] > 255) spectrum[i] = 255; // limit outlier data
      if(i == 0)
        avg[i] = spectrum[i];
      else
        avg[i] = avg[i] + spectrum[i];
    }
  }
  for(i=0; i < BINS; i++) {              // For each output bin average
    avg[i] = avg[i] / FRAMES;            //  divide about the number of values aaveraged
  }
  int maxVal = 0, maxIndex = 0;
  for(i=0; i < BINS; i++) {              // For each output bin average
    if(avg[i] >= maxVal) {               //  find the peak value
      maxVal = avg[i];
      maxIndex = i;                     //  and the bin that max value is in
    }
  }
  for(j=0; j < 32; j++) {           // print spectrum 32 bins
    Serial.print(avg[j]);
```

```
    Serial.print(" ");
  }
  Serial.println("");              // and print the highest value and the bin it is in
  Serial.print("Max Value = "); Serial.print(maxVal);
  Serial.print(", Index of Max Value = "); Serial.println(maxIndex);
}
```

Find a room where people will not be annoyed by squeeky sounds and is quiet. Let the far right "listening" Circuit Playground listen to the ambient sounds with the second sketch and a computer to monitor the serial output. Then use your tone generator to make tones.

You can use the same computer with a separate USB port and serial terminal program to monitor the frequency you chose dial in. Adjust the potentiometer and note where a FFT bin reaches a maximum (hopefully 255, but maybe greater than 220 or so) and write that frequency and that bin number as a data point. Continue through the full range of the potentiometer. I recorded the following values and made a quick plot.

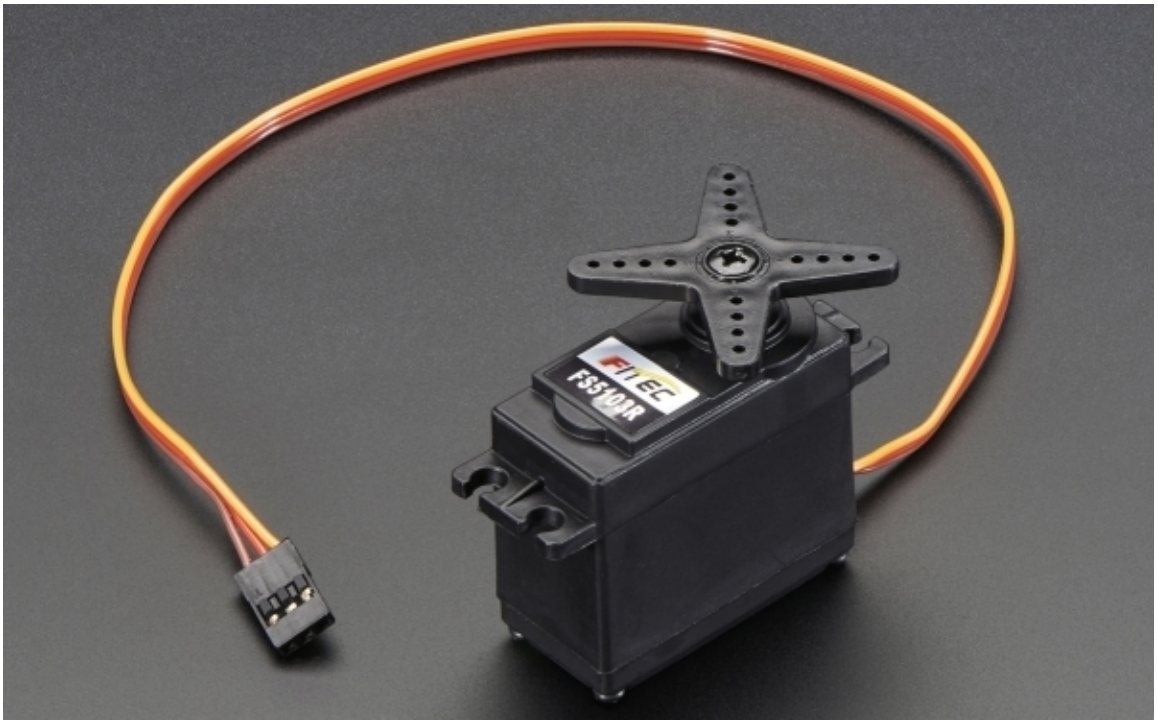
Bin	Freq1	Freq2	Freq3	Picked
10			8000	
11			7822	
12				
13				
14				
15				
16		2393	7200	
17		2493		
18				
19		2795		#1
20		2957		
21		3094		
22		3250		#2
23		3436		
24	1170	3605		
25	1235	3700		#3
26	1281	3876		
27	1351	4046		
28	1405	4192		#4
29	1451	4339		
30	1497	4517		
31		4640		#5



Several different frequencies maxed out an FFT bin. The data shows that you can get the same bin filling at different frequencies. Those frequencies are about 3 times a lower value (a third harmonic in musical/frequency terms). But not all frequencies will max out a bin. That's why my first inclination to pick nice musical notes didn't work like I thought. Several candidate frequencies (#1 to #5) are chosen to use for the robot. Those bins are spaced out so one tone that might be close to another does not get so close that the code cannot differentiate one for another.

Continuous Rotation Servos

The robot uses two continuous rotation servos (http://adafru.it/154) to move. These motors are a bit different than normal servos. See the Adafruit Motor Selection Guide on continuous servos (http://adafru.it/qME) for a great explanation for how they work.



The two servos we are using are rotated 180 degrees to each other (back to back) so one side will be running opposite the other. For example to go forward, run one servo forward, the other in reverse.

Using the Servo.write function, a setting of 180 is full forward, 0 is full reverse.

The value for no motion would usually be 90 degrees but each servo might be a bit different. We need to do a bit of testing to find the actual angle for a stop value for the servos you get.

The program below uses the Circuit Playground to find the values for two Servos connected back to back. If the slide switch is on "+", the right motor is calibrated, "-" for the left motor. The two pushbuttons adjust the right motor by 0.1 degree (left minus, right plus). The angle is printed out on the Arduino serial monitor.

Use the circuit shown on the page "Use Circuit Playground" and the program below.

```
// Circuit Playground Robot - Continuous Servo zero/no movement calibration program
// Mike Barela for Adafruit Industries  September, 2016
```

```
#include <Adafruit_CircuitPlayground.h>
#include <Servo.h>

Servo servoLeft;              // Define left servo
Servo servoRight;             // Define right servo
float speedAngleLeft = 90.0;  // Assume 90 degrees as a start
float speedAngleRight = 90.0;

void setup() {
  servoLeft.attach(12);         // Set left servo to digital pin 12
  servoRight.attach(6);         // Set right servo to digital pin 6
  CircuitPlayground.begin();    // initialize the Circuit Playground library
  Serial.begin(9600);           // to output servo angle values
  Serial.println("Robot Continuous Servo Zero Movement Calibration");
}

void loop() {                   // Loop through motion tests
  Serial.print("Speed Left = ");    Serial.print(speedAngleLeft);
  Serial.print(", Speed Right = "); Serial.println(speedAngleRight);
  if(CircuitPlayground.slideSwitch()) {  // + = Calibrate Right, - = Calibrate Left
    if( CircuitPlayground.rightButton() ) {
      speedAngleRight += 0.1;
    }
    if( CircuitPlayground.leftButton() ) {
      speedAngleRight -= 0.1;
    }
  }
  else {
    if( CircuitPlayground.rightButton() ) {
      speedAngleLeft += 0.1;
    }
    if( CircuitPlayground.leftButton() ) {
      speedAngleLeft -= 0.1;
    }
  }
  servoLeft.write(speedAngleLeft);
  servoRight.write(speedAngleRight);
  delay(50);
}
```

To use the program, upload it to your Circuit Playground with the right servo control wire connected to Pin #6 and the left servo control motor connected to Pin #12.

When you power on the project, both wheels will probably turn slowly. Maddening that they do, as at an angle of 90 degrees we'd think they should be stopped. But we'll fix that.

Move the slide switch to + and open the Arduino IDE serial monitor to view the right motor angles. Press the Circuit Playground push buttons to increment or decrement the angle given to the right servo. The switches are not debounced so the values might be a bit jumpy. Eventually the motor will stop at a certain value displayed on the serial monitor. Note that value for right.

Move the slide switch to - and do the same angle adjustment for the left motor. Record the left value which results in no movement. For my two servos, the angles were 96.2 for the left servo,

95.3 for the right.

You can hard code these into the main robot program. If you leave the values I found instead of using your own, you'll probably still have slowly turning wheels at a stop. All continuous servos are a bit different.

Whistle Bot

When working with what frequencies the robot responds to, the temptation is to whistle into the microphone and see what it does. This is what I did in uploading the Circuit Playground sample program vu meter (http://adafru.it/qPb) which uses the microphone as a sound level meter. I did get some multicolor sounds with different whistles.

But do whistles allow you to control the robot? Maybe, if you are good at your whistles. Through testing the various bins my whistling fell in, I could get decent start and stop whistles. If I whistle in a frequency filling bins 5, 6, or 7, the robot moves forwards. A different, higher whistle filling one of the bins 8, 9, or 10, the robot stops. My whistles are not varied enough to get left and right turns. If you have some range of tones, you can change which bins make which motion in the program's switch statement.

For the most flexible sound control, consider the tone controlled robot on the next page.

Here is the code for a two tone whistle bot. A lower tone starts the robot moving forward, a higher whistle stops the robot.

```
// A sound controlled robot for the Adafruit Circuit Playground board
//  This uses two continuous rotation servos for movement and listens to
//  commands via the built-in mic.
// This version is designed to respond to two whistle tones for
//   start and stop.
// By Mike Barela for Adafruit Industries September, 2016

#include <Adafruit_CircuitPlayground.h>   // Circuit Playground library
#include <Servo.h>                        // Audiono servo control library

// Global Definitions ------------------------------------------------------------
//  Fourier Transform
#define BINS       32    // FFT output is output in this many bins
#define FRAMES      4    // This many FFT cycles are averaged for leveling
#define THRESHOLD  150    // Max bin value to say a tone was detected

//  Servo objects
Servo servoLeft;            // Define left servo
#define leftStopAngle  96.2  // Angle that left servo will stop (calibrated)
Servo servoRight;           // Define right servo
#define rightStopAngle 95.3  // Angle that right servo will stop (calibrated)
uint8_t moving = 0;         // Is the robot currently moving value (0 = no, 1 = yes)

// SETUP FUNCTION - this runs once to set the robot up when it is powered on

void setup() {
  servoLeft.attach(12);               // Set left servo to digital pin 12
  servoRight.attach(6);               // Set right servo to digital pin 6
  CircuitPlayground.begin();          // Initialize the CP library
```

```
  CircuitPlayground.setBrightness(20);  // NeoPixels are another visual que on what is happening
  CircuitPlayground.clearPixels();      // Ensure all NeoPixels are off at start
  Serial.begin(9600);                   // Set up the USB port for serial information for users
  Serial.println("\n\nAdafruit Circuit Playground Robot using Whistle Commands");
  stopRobot();
}

// LOOP FUNCTION - runs over and over to check what to do ----------------------

void loop() {
  uint8_t  i,j;                         // loop index values
  uint16_t spectrum[BINS];              // FFT spectrum output buffer
  uint16_t avg[BINS];                   // The avaerage FFT values over FRAMES iterations
  int16_t  maxVal = 0, maxIndex = 0;   // The maximum value and the FFT bin of that value
  int8_t   maxBins;

  if( !CircuitPlayground.slideSwitch() ) {  // if slide switch is moved to "-", shut robot down
    if(moving) {
      stopRobot();                      // stop the robot
      CircuitPlayground.clearPixels();    // turn all pixels off in low power mode
    }
    Serial.println("\nRobot stopped due to slide switch, move to '+' to resume");
    return;                             //  and go back to top of loop
  }

  for(j=1; j <= FRAMES; j++) {          // Gather FRAMES samples of audio from the microphone
    CircuitPlayground.mic.fft(spectrum);  // This function gathers an audio sample and does FFT
    for(i=0; i < BINS; i++) {           // Add values to perform a simple average
      if(spectrum[i] > 255) spectrum[i] = 255; // in library, sometimes values get "huge"
      if(i == 0)
        avg[i] = spectrum[i];
      else
        avg[i] = avg[i] + spectrum[i];
    }
  }
  for(i=0; i < BINS; i++) {       // For each output bin average
    avg[i] = avg[i] / FRAMES;    //   calculate the average (unweighted)
  }

  maxBins = 0;
  for(i=0; i < BINS; i++) {   // Search for the highest value and the FFT bin it's in
    if(avg[i] > 255) avg[i] = 255;   // HACK 4 NOW
    if(avg[i] >= maxVal) {
      maxVal = avg[i];       // Important note: If there is an equal max value in higher bins
      maxIndex = i;          //   note the later index (helps when recognizing a sound)
    }
    if(avg[i] >= 254) maxBins++;
  }
  // avg[] is now FRAME averaged FFT output, 32 bins.

    if( CircuitPlayground.leftButton() ) {    // use onboard buttons to change behavior
      maxIndex = 9; // stop
      maxVal = THRESHOLD;
    }
    if( CircuitPlayground.rightButton() ) {
```

```
      maxIndex = 7; // forward
      maxVal = THRESHOLD;
    }

  // A detection is defined as a bin reaching a value of at least THRESHOLD
  if( maxVal >= THRESHOLD ) {

    // For visual review of the values the FFT has produced, print the FFTs 32 bins
    for( uint8_t j=0; j < 32; j++) {
      Serial.print(avg[j]);
      Serial.print(" ");
    }
    Serial.println("");
    // maxVal is the biggest bin, maxIndex is the index of that value
    Serial.print("\nMax Value = "); Serial.print(maxVal);
    Serial.print(", Index of Max Value = "); Serial.println(maxIndex);

    CircuitPlayground.clearPixels();  // clear old pixel values
    switch( maxIndex ) {  // based on which bin had the detection, act on it
      case  0:
      case  1:
      case  2:
            CircuitPlayground.strip.setPixelColor(0,0,240,0);
            // you can put a movement function call here
            break;
      case  3:
      case  4:
            CircuitPlayground.strip.setPixelColor(1,0,240,0);
            // you can put a movement function call here
            break;
      case  5:
      case  6:
      case  7:
            CircuitPlayground.strip.setPixelColor(2,10,225,10); // Low whistle (3rd LED Green)
            forward();   // forward
            break;
      case  8:
      case  9:
      case 10:
            CircuitPlayground.strip.setPixelColor(3,225,00,10); // higher whistle (4th LED red)
            stopRobot();
            break;
      case 11:            // 7822 hertz
            CircuitPlayground.strip.setPixelColor(3,0,240,0);
            // you can put a movement function call here
            break;
      case 18:
      case 19: CircuitPlayground.strip.setPixelColor(4,0,240,0); // 2795 hertz
            // you can put a movement function call here
            break;
      case 20:            // 2957 hertz
      case 21:            // 3094 hertz
      case 22: CircuitPlayground.strip.setPixelColor(5,0,240,0); // 3250 hertz
            // you can put a movement function call here
            break;
```

```
      case 23:            // 3436 hertz
      case 24:            // 3605 hertz

      case 25: CircuitPlayground.strip.setPixelColor(6,0,240,0); // 3700 hertz
            // you can put a movement function call here
            break;
      case 26:            // 3876 hertz
      case 27:            // 4046 hertz
      case 28: CircuitPlayground.strip.setPixelColor(7,0,240,0); // 4192 hertz (motor noise?)
            // you can put a movement function call here
            break;
      case 29:            // 4339 hertz
      case 30:            // 4517 hertz
      case 31: CircuitPlayground.strip.setPixelColor(8,0,240,0); // 4640 hertz (often goes here)
            // you can put a movement function call here
            break;
      default: CircuitPlayground.strip.setPixelColor(9,200,0,0); // if any other bin, light NeoPixel #9 red
            break;
    } // end switch
    CircuitPlayground.strip.show();  // show the neopixel assigned to the bin
  } // end if
} // end function loop

// Servo motion function routines for robot movement forward, reverse, turns, and stop

void stopRobot() {
  moving = 0;                    // let program know we are stopped
  servoLeft.write(leftStopAngle);    // Get these values from a calibration routine
  servoRight.write(rightStopAngle);  //  as continuous servos don't stop at exactly 90.0
  Serial.println("Stopped");
}

void forward() {
  if( moving == 1 ) {  // if the robot currently is moving  NEEDED?
    stopRobot();       // stop it
  }
  else {
    moving = 1;        // flag we are going to move
  }
  servoLeft.write(0);
  servoRight.write(180);
}

void reverse() {
  if( moving == 1 ) {  // if the robot currently is moving
    stopRobot();       // stop it
  }
  else {
    moving = 1;        // flag we are going to move
  }
  servoLeft.write(180);
  servoRight.write(0);
}

void turnRight() {
```

```
  if( moving == 1 ) {  // if the robot currently is moving
    stopRobot();       // stop it
  }
  else {
    moving = 1;        // flag we are going to move
  }
  servoLeft.write(180);
  servoRight.write(180);
}

void turnLeft() {
  if( moving == 1 ) {  // if the robot currently is moving
    stopRobot();       // stop it
  }
  else {
    moving = 1;        // flag we are going to move
  }
  servoLeft.write(0);
  servoRight.write(0);
}
```

Using the Whistle Bot

Use a lower tone whistle to start it forward, a higher whistle to stop. You'll probably need to experiment with your whistle tones to find the right ones. It currently is calibrated for more of a male whistle (I'm told by my wife).

Whistle and see which NeoPixels light up. For me it's the 4th and 5th pixels (#3 and #4). If you see something like the second and third (NeoPixels #2 and #3), you can change where the movement function calls are in the `switch..case` statement (there are comments where to put the calls in).

The left and right pushbuttons perform the same two functions (start & stop) if you would like to manually change the mode. If you change the `switch..case` statement, put the `case` numbers in the `if` statements that check the left and right buttons on Circuit Playground.

Here is a video of the whistle bot in action:

Variations on the Whistle Bot

You could use a first low tone for start and a second low tone for stop. You would test if variable `moving` is 1 then you's call `stopRobot` instead of `forward`. Then you have the higher tone to do something like calling `reverse`.

If you can whistle in three tones, you can probably call all of the control functions with some variables which record what state you are in and what state you want the robot to change to when a whistle is received.

On the next page you can build an inexpensive tone generator to save you from having to whistle

so precisely.

Tone Controlled Robot

The final program for the robot combines the building blocks we've worked on in the tutorial:

- Using the Circuit Playground library for microphone, NeoPixel, and switch functions
- Taking FFT information and looking for our command sounds
- Moving the robot based on the command sounds

You might want to review how toprogram Circuit Playground (http://adafru.it/rb4). And definitely use a good USB data cable and not a charging cable (even I have done this and said "doh!").

Here is the code to run on the Circuit Playground for the tone controlled robot:

```
// A sound controlled robot for the Adafruit Circuit Playground board
//  This uses two continuous rotation servos for movement and listens to
//  commands via the built-in mic. Sound processing is done with the
//  Circuit Playground FFT library function for AVR microcontrollers.

// The fast Fourier transform (FFT) algorithm converts a signal from the
// time domain to the frequency domain -- here, turning a specific audio
// signal into a command to the robot.

// Note the Circuit Playground slide switch when set to "-" stops the motors
//  It is not a battery off switch so unplug or turn off the battery to save
//  power when not in use.

// By Mike Barela for Adafruit Industries September, 2016  Version 1.0

#include <Adafruit_CircuitPlayground.h>   // Circuit Playground library
#include <Servo.h>                        // Audiono servo control library

// Global Definitions ------------------------------------------------------------
//  Fourier Transform
#define BINS       32    // FFT output is output in this many bins
#define FRAMES      4    // This many FFT cycles are averaged for leveling
#define THRESHOLD  150    // Max bin value to say a tone was detected

//  Servo objects
Servo servoLeft;            // Define left servo
#define leftStopAngle  96.2  // Angle that left servo will stop (calibrated)
Servo servoRight;           // Define right servo
#define rightStopAngle 95.3  // Angle that right servo will stop (calibrated)
uint8_t moving = 0;         // Is the robot currently moving value (0 = no, 1 = yes)

// SETUP FUNCTION - this runs once to set the robot up when it is powered on

void setup() {
  servoLeft.attach(12);               // Set left servo to digital pin 12
  servoRight.attach(6);               // Set right servo to digital pin 6
```

```
  CircuitPlayground.begin();            // Initialize the CP library
  CircuitPlayground.setBrightness(20);  // NeoPixels are another visual que on what is happening
  CircuitPlayground.clearPixels();      // Ensure all NeoPixels are off at start
  Serial.begin(9600);                   // Set up the USB port for serial information for users
  Serial.println("\n\nAdafruit Circuit Playground Robot");
  stopRobot();
}

// LOOP FUNCTION - runs over and over to check what to do ----------------------

void loop() {
  uint8_t  i,j;                       // loop index values
  uint16_t spectrum[BINS];            // FFT spectrum output buffer
  uint16_t avg[BINS];                 // The avaerage FFT values over FRAMES iterations
  int16_t  maxVal = 0, maxIndex = 0;  // The maximum value and the FFT bin of that value
  int8_t   maxBins;

  if( !CircuitPlayground.slideSwitch() ) {  // if slide switch is moved to "-", shut robot down
    if(moving) {
      stopRobot();                    // stop the robot
      CircuitPlayground.clearPixels();    // turn all pixels off in low power mode
    }
    Serial.println("\nRobot stopped due to slide switch, move to '+' to resume");
    return;                           //  and go back to top of loop
  }

  for(j=1; j <= FRAMES; j++) {        // Gather FRAMES samples of audio from the microphone
    CircuitPlayground.mic.fft(spectrum);  // This function gathers an audio sample and does FFT
    for(i=0; i < BINS; i++) {           // Add values to perform a simple average
      if(spectrum[i] > 255) spectrum[i] = 255; // in library, sometimes values get "huge"
      if(i == 0)
        avg[i] = spectrum[i];
      else
        avg[i] = avg[i] + spectrum[i];
    }
  }
  for(i=0; i < BINS; i++) {       // For each output bin average
    avg[i] = avg[i] / FRAMES;    //   calculate the average (unweighted)
  }

  maxBins = 0;
  for(i=0; i < BINS; i++) {   // Search for the highest value and the FFT bin it's in
    if(avg[i] > 255) avg[i] = 255;   // HACK 4 NOW
    if(avg[i] >= maxVal) {
      maxVal = avg[i];       // Important note: If there is an equal max value in higher bins
      maxIndex = i;          //   note the later index (helps when recognizing a sound)
    }
    if(avg[i] >= 254) maxBins++;
  }
  // avg[] is now FRAME averaged FFT output, 32 bins.

  // use onboard buttons to change behavior
    if( CircuitPlayground.leftButton() ) {
      maxIndex = 28; // stop
      maxVal = THRESHOLD;
```

```
  }
  if( CircuitPlayground.rightButton() ) {
    maxIndex = 11; // forward
    maxVal = THRESHOLD;
  }

// A detection is defined as a bin reaching a value of at least THRESHOLD
if( maxVal >= THRESHOLD ) {

 if( maxBins > 3 ) {  // Some loud broad spectrum sound, we don't want to act on that
  Serial.println("\nMore than 3 bins are maxed out");
  return;
 }

 // For visual review of the values the FFT has produced, print the FFTs 32 bins
  for( uint8_t j=0; j < 32; j++) {
   Serial.print(avg[j]);
   Serial.print(" ");
  }
  Serial.println("");
  // maxVal is the biggest bin, maxIndex is the index of that value
  Serial.print("\nMax Value = "); Serial.print(maxVal);
  Serial.print(", Index of Max Value = "); Serial.println(maxIndex);

  CircuitPlayground.clearPixels();  // clear old pixel values
  switch( maxIndex ) {  // based on which bin had the detection, act on it
    case 10:              // 8000  hertz
    case 11: unused();    // 7822 hertz
             break;
    case 18:
    case 19: CircuitPlayground.strip.setPixelColor(0,0,240,0); // 2795 hertz
             forward();   // forward
             break;
    case 20:              // 2957 hertz
    case 21: unused();    // 3094 hertz
             break;
    case 22: CircuitPlayground.strip.setPixelColor(1,0,240,0); // 3250 hertz
             turnLeft();
             break;
    case 23:              // 3436 hertz
    case 24: unused();    // 3605 hertz
             break;
    case 25: CircuitPlayground.strip.setPixelColor(2,0,240,0); // 3700 hertz
             turnRight();
             break;
    case 26:              // 3876 hertz
    case 27: unused();    // 4046 hertz
             break;
    case 28: CircuitPlayground.strip.setPixelColor(3,0,240,0); // 4192 hertz
             stopRobot();
             break;
    case 29:              // 4339 hertz
    case 30: unused();    // 4517 hertz
             break;
    case 31: CircuitPlayground.strip.setPixelColor(4,0,240,0); // 4640 hertz
```

```
            // reverse();
            break;
      default: CircuitPlayground.strip.setPixelColor(9,200,0,0); // if any other bin, light NeoPixel #9 red
            stopRobot();
            break;
    } // end switch
    CircuitPlayground.strip.show();  // show the neopixel assigned to the bin
  } // end if
} // end function loop

// Servo motion routines for robot movement forward, reverse, turns, and stop

void stopRobot() {
  moving = 0;                     // let program know we are stopped
  servoLeft.write(leftStopAngle);    // Get these values from a calibration routine
  servoRight.write(rightStopAngle);  //  as continuous servos don't stop at exactly 90.0
  Serial.println("Stopped");
}

void forward() {
  if( moving == 1 ) {  // if the robot currently is moving  NEEDED?
    stopRobot();       // stop it
  }
  else {
    moving = 1;        // flag we are going to move
  }
  servoLeft.write(0);
  servoRight.write(180);
}

void reverse() {
  if( moving == 1 ) {  // if the robot currently is moving
    stopRobot();       // stop it
  }
  else {
    moving = 1;        // flag we are going to move
  }
  servoLeft.write(180);
  servoRight.write(0);
}

void turnRight() {
  if( moving == 1 ) {  // if the robot currently is moving
    stopRobot();       // stop it
  }
  else {
    moving = 1;        // flag we are going to move
  }
  servoLeft.write(180);
  servoRight.write(180);
}

void turnLeft() {
  if( moving == 1 ) {  // if the robot currently is moving
    stopRobot();       // stop it
```

```
  }
  else {
    moving = 1;        // flag we are going to move
  }
  servoLeft.write(0);
  servoRight.write(0);
}

void unused() {   // Frequency not used at present
  CircuitPlayground.strip.setPixelColor(8,0,0,255); // display a blue light on pixel 8
}
```

Do you wheels turn when in the robot stop mode? You need to be sure the calibration calues for your servos are determined as on the previous page. I doubt the values I have will be the same as yours. They should be between 83 and 97.

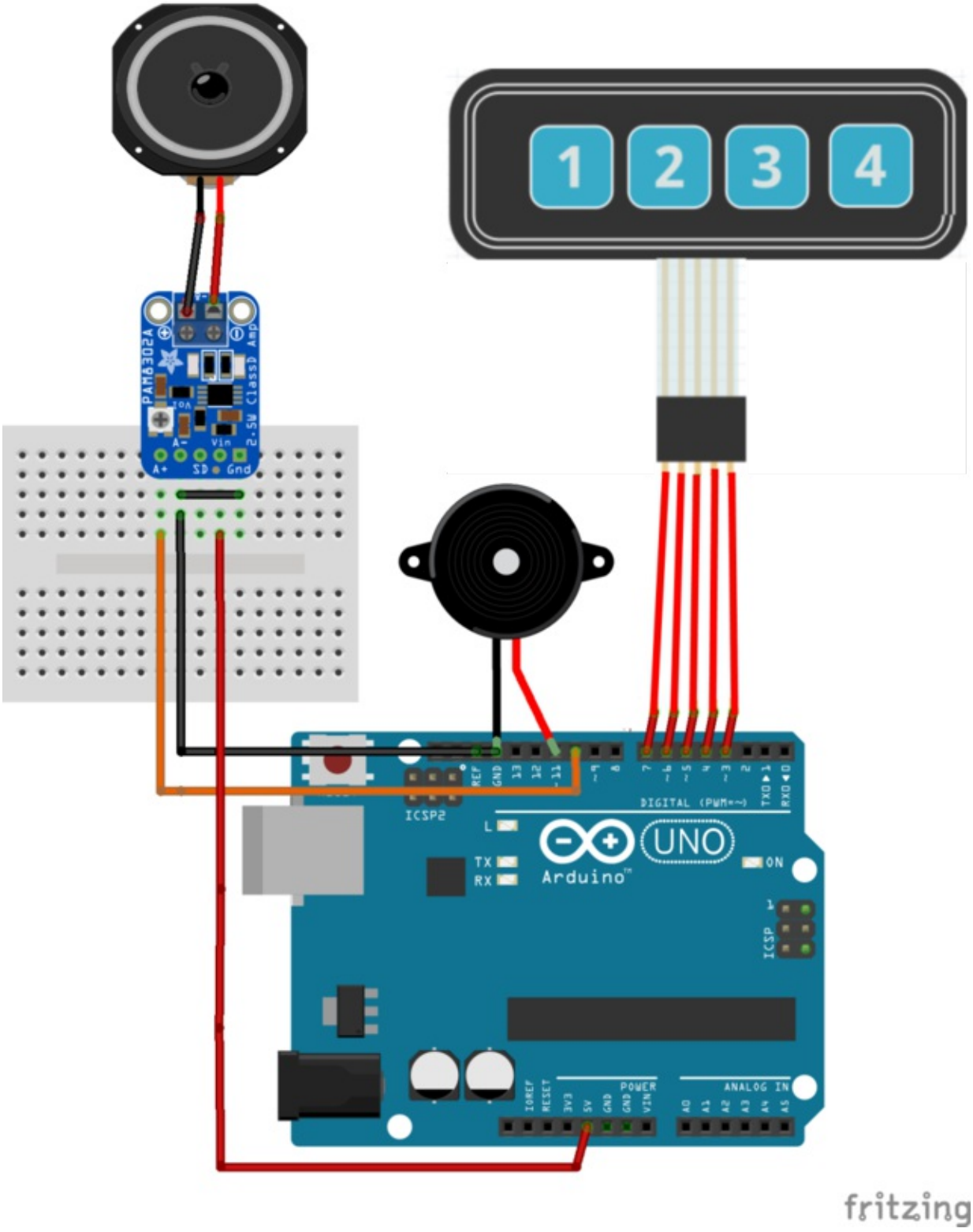
Build the Controller

An Arduino Uno (http://adafru.it/50) or Adafruit Metro (http://adafru.it/2466) or other handy microcontroller may be used to make a sound controller.

The controller will read some buttons and output a tone based on which button is pressed.

If you would like to build this based on the parts you have on hand, a Pro Trinket, a Feather, etc. then there are very few changes to the code other than the pin numbers that the switches and speaker are connected to. The check switches code sees if a switch has been pressed. It uses software switch debouncing (http://adafru.it/qPc) to ensure a positive push.

The design below contains an Adafruit audio amplifier (http://adafru.it/2130) on the tone output. This allows one to be further away from the robot and have the robot "hear" the command tones. You can try a piezo speaker (http://adafru.it/160) shown in the diagram without the amplifier and speaker. If you find your range is not as far as you like, you can add the amplifier and speaker. The tradeoff is you'll be making louder sounds possibly annoying those around you. You can adjust the amplification with the volume control.

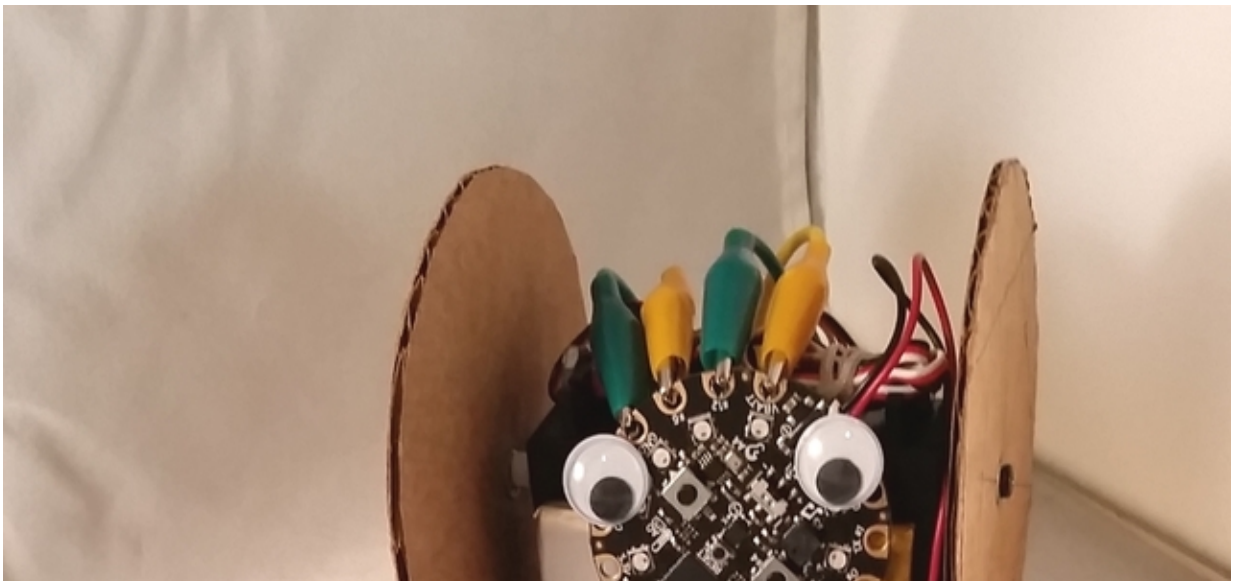


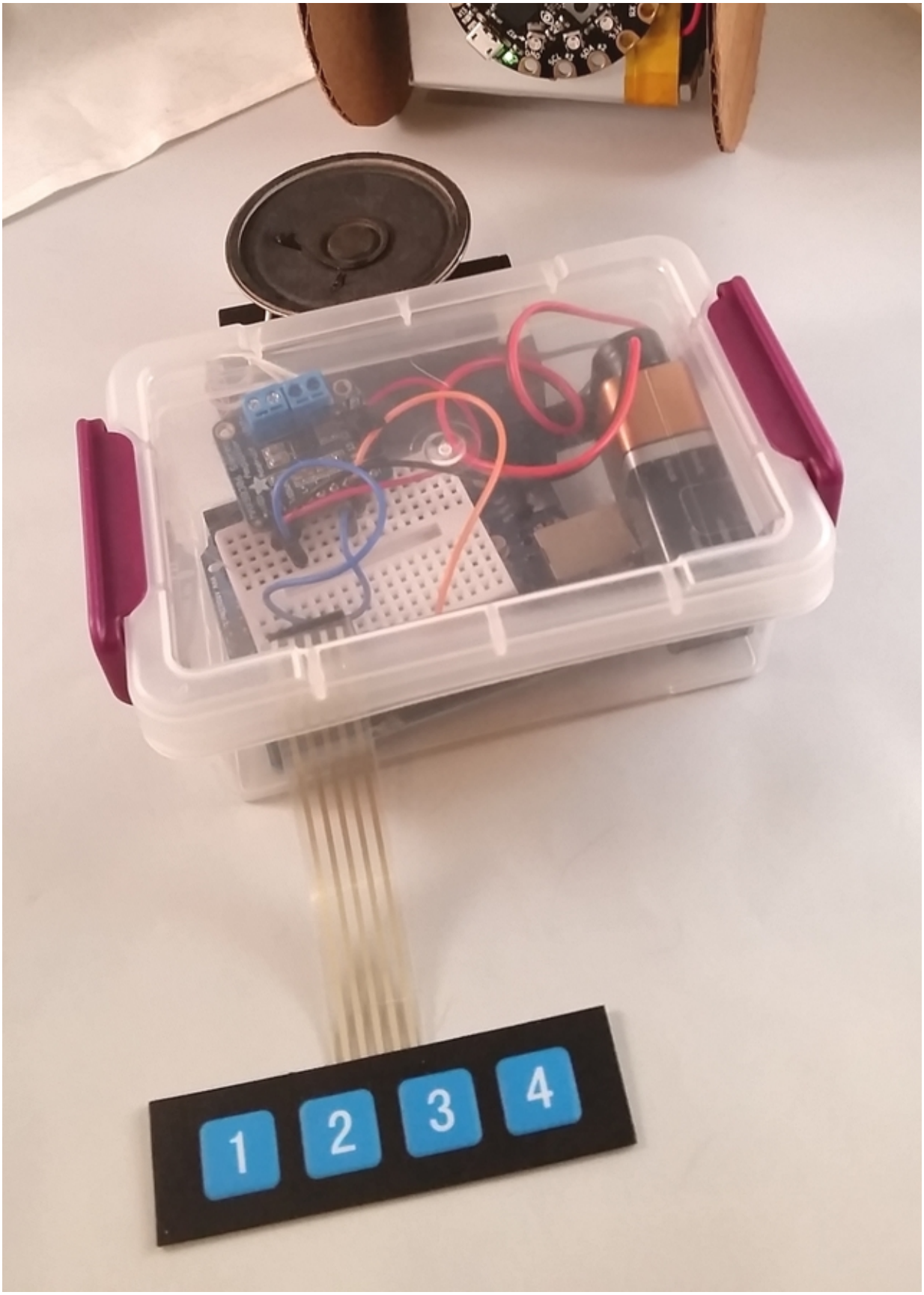
I had the 4 button keypad on hand, allowing for left, right, forward, and stop. You can use a larger keypad with a bit of decoding code and use buttons such as 2 for forward, 8 for reverse, 5 for stop, 4 for left, 6 for right. If you would like to add additional commands, you can look at the chart on the "Using the Microphone" page to select the frequencies best to use. Be careful, if you use two

tones that are close in frequency they might fall near the same bin on detection and the robot code would not interpret the tone correctly. This is why the 5 tones chosen are several bins away from each other.

```
// Tone Generator for the Circuit Playground Simple Robot
//
// Using a set of buttons, play predetermined tones to control the Circuit Playground simple robot
//
// Mike Barela for Adafruit Industries   September, 2016

const int speakerPin = 11;       // Uno pin 11, Circuit Playground pin 5

// here is where we define the buttons that we'll use. button "0" is the first, button "3" is the 3rd, etc
// uint8_t buttons[] = { 1, 0, 3, 2 };  // for a second Circuit Playground
uint8_t buttons[] = { 4, 3, 6, 5 };     // for Arduino Uno
#define DEBOUNCE 10  // button debouncer, how many ms to debounce, 5+ ms is usually plenty
// This handy macro lets us determine how big the array up above is, by checking the size
#define NUMBUTTONS sizeof(buttons)
// we will track if a button is just pressed, just released, or 'currently pressed'
int8_t pressed[NUMBUTTONS], justpressed[NUMBUTTONS], justreleased[NUMBUTTONS];

uint16_t notes[] = { 2795, 3250, 3700, 4192 };  // control tones in Hertz

void setup() {
  uint8_t i;

  pinMode(2, OUTPUT);   // Ground for
  digitalWrite(2, LOW); //  the 4 button keypad ground (eases wiring on Uno)
  // Make input & enable pull-up resistors on switch pins
  for (i=0; i < NUMBUTTONS; i++) {
    pinMode(buttons[i], INPUT_PULLUP);
  }
  pinMode(speakerPin, OUTPUT);  // The piezo speaker pin is set as an output
}

void loop() {
  check_switches();      // when we check the switches, we'll get their current state

  for (uint8_t i = 0; i < NUMBUTTONS; i++) {  // we check the state of the buttons
    if (pressed[i]) {
      //     digitalWrite(13, HIGH);
      // is the button pressed down at this moment
    }
    if (justreleased[i]) { // on button release . .
      tone(speakerPin, notes[i], 1000);  // Play the tone assigned to the button for 1 second
    }
  }
  for (uint8_t i=0; i < NUMBUTTONS; i++) {  // remember, check_switches() will necessitate clearing the 'just pressed' flag
    justpressed[i] = 0;
  }

}

void check_switches()
```

```
{
  static int8_t previousstate[NUMBUTTONS];
  static int8_t currentstate[NUMBUTTONS];
  static long lasttime;
  uint8_t index;

  if (millis() < lasttime){ // we wrapped around, lets just try again
     lasttime = millis();
  }

  if ((lasttime + DEBOUNCE) > millis()) {
    // not enough time has passed to debounce
    return;
  }
  // ok we have waited DEBOUNCE milliseconds, lets reset the timer
  lasttime = millis();

  for (index = 0; index < NUMBUTTONS; index++){ // when we start, we clear out the "just" indicators
    justreleased[index] = 0;

    currentstate[index] = digitalRead(buttons[index]);   // read the button

    if (currentstate[index] == previousstate[index]) {
      if ((pressed[index] == LOW) && (currentstate[index] == LOW)) {
          // just pressed
          justpressed[index] = 1;
      }
      else if ((pressed[index] == HIGH) && (currentstate[index] == HIGH)) {
          // just released
          justreleased[index] = 1;
      }
      pressed[index] = !currentstate[index];  // remember, digital HIGH means NOT pressed
    }
    //Serial.println(pressed[index], DEC);
    previousstate[index] = currentstate[index];   // keep a running tally of the buttons
  }
}
```





For short term use, you can use a 9 volt battery and 9 volt barrel adapter clip to power the tone

generator. If you plan on heavy use, a 6xAA battery pack (http://adafru.it/248) would be a better choice.

Use

You will want to use a flat surface for the robot and a quiet room without several of your friends also using sound to control their robots (unless you pick different control frequencies). I used both wood and low-pile carpet with good results. The servos have good torque to move. The robot is sensitive to bumps as a two wheeled robot turns wil slight changes in differential between wheels.

Put the robot on a good surface, plug in a charged LiPo battery and ensure the slide switch on the Circuit Playground board is on the "+" position. You can test if things are working well by holding the robot and pressing the right pushbutton on Circuit Playground. It should run forward. Pressing the left button stops it. If you pick the robot up and want the wheels to stop, press the left pushbutton and then turn the slide switch to "-". The slide switch is not a power switch though, when you want to have the robot off for a period of time, unplug the battery.

How close the control unit must be to the robot depends on a number of factors:

- Volume of the speaker (amplified or unamplified)
- If the microphone is facing the tone generator. If it is off center, the amount of sound energy getting to the microphone will be less.
- Ambient noise from other sources.

You can consider a cylinder or funnel on the front to focus sound to the microphone (where the ear symbol is on Circuit Playground).

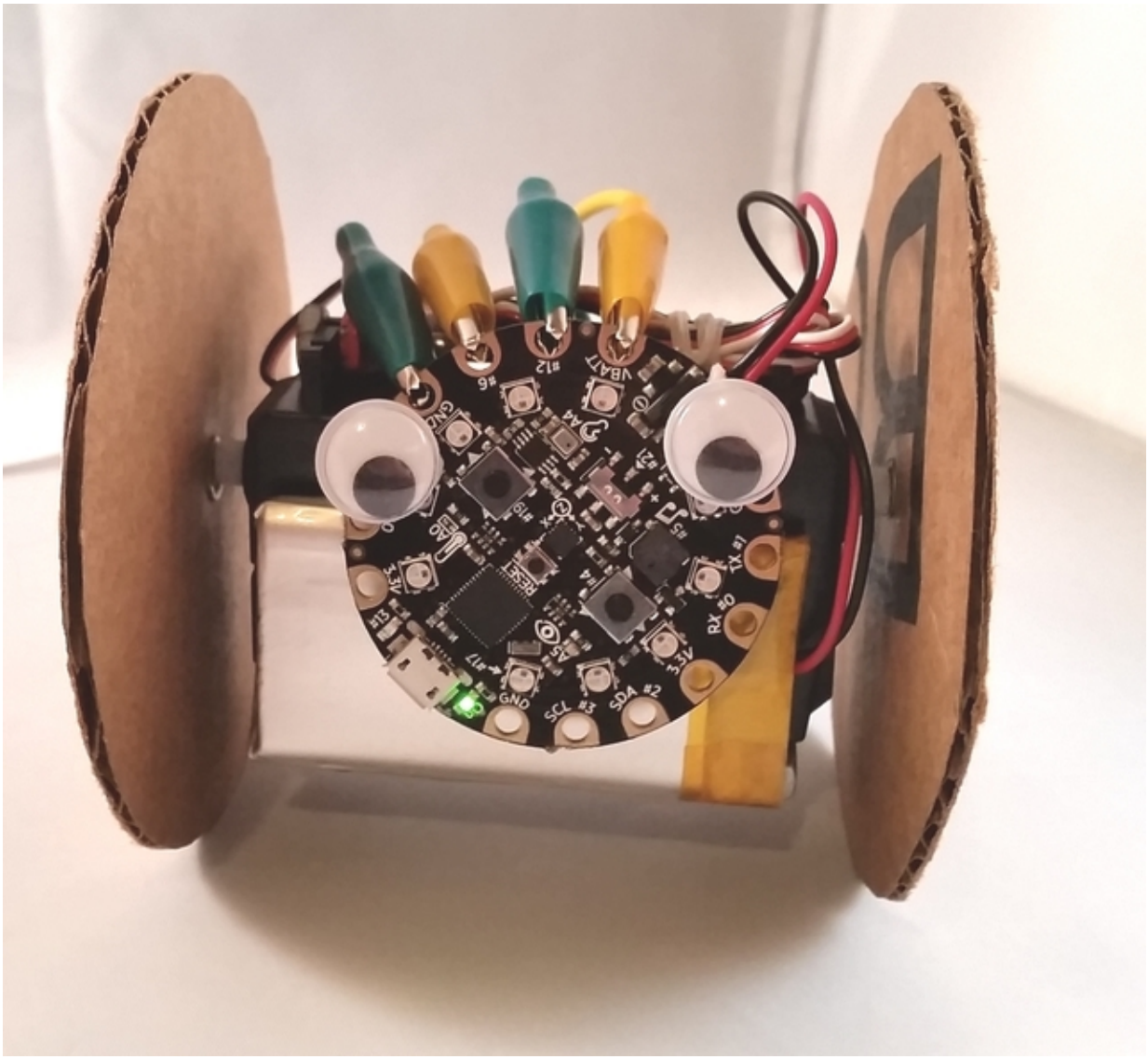
Going Further

Control

You can use sound control to do nearly anything you want. Instead of moving servo motors, you can use a sound to turn NeoPixels on or off. You can train your Circuit Playground to recognize your fire alarm and communicate via USB to the Internet to alert your mobile phone.

Using a wired controller instead of sound is a low cost, tried and true control method. You should have a keypad or buttons mounted on something like a small box (maybe a mint tin (http://adafru.it/97)). Find a wire that has several conductors, one that has the number of buttons plus one for a common ground. Flexible wire (stranded) is better for any twists and turns. The disadvantage to a wired controller is twisting of the wire and the length of the wire needed.

A more advanced control system could add a Wi-Fi orBluetooth radio board (http://adafru.it/qPd). Pins #2 and #3 would be good for a software serial connection (http://adafru.it/na2) as they are free and do not conflict with using Pins #0 and #1 (which are shared with the USB port). You may need bigger wheels than the 3.75" / 9.5 mm size shown.



Robotics

If you find the movements of the wheels are not quite what you would like (pivot on turn rather than a gradual turn), you can look to change motor angles for turn left and turn right.

You can move up to a real robotics frame rather than bolting the two servo motors together. Adafruit sells great frames (http://adafru.it/9720) (of course!) as do specialty robotics shops. This will add some cost. You can build your own robotics frame out of nearly any material. Think of materials that are sturdy and offer flexibility in attachments and drilling holes. If you have an erector set from a yard sale or are good in woodwork, you're in great shape. If you build a larger robot, I highly suggest a larger battery such as a 4xAA battery pack (http://adafru.it/830).

I suggest making your Circuit Playground removable. I am very much in favor of being able to take the main parts of a project (and the Circuit Playground is the heart and soul here) and reusing or borrow if I have a new idea. Teachers will also appreciate projects that do not take

hours to disassemble. Use of a non-permanent adhesive such as Blu Tack or similar (http://adafru.it/qPe) is one of the best methods I have found.