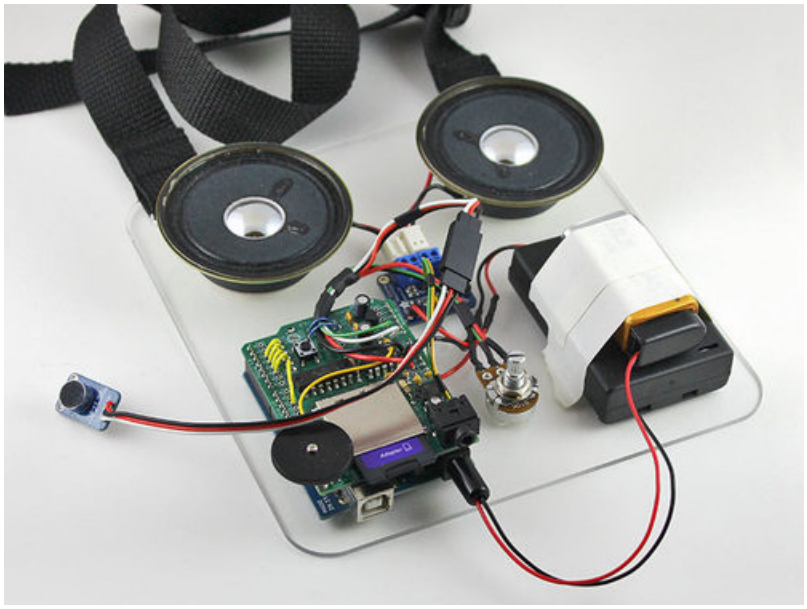
adafruit learning system

Wave Shield Voice Changer

Created by Phillip Burgess



Last updated on 2019-06-10 04:34:26 PM UTC

Overview

The Wave Shield for Arduino (http://adafru.it/94) is one of Adafruit's earliest shield kits and remains a perennial favorite. And for good reason — it's among the easiest and most flexible means of adding quality sound effects to an Arduino project!

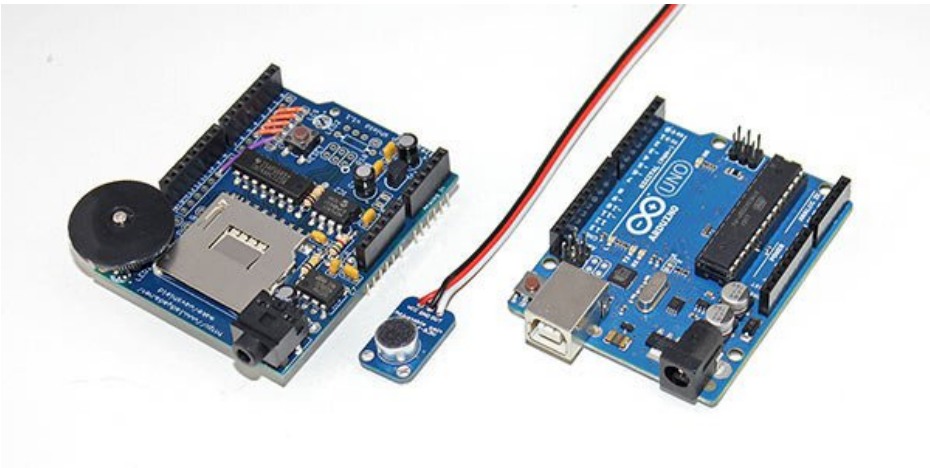
Like a fine wine, open source projects improve with age. We've taught this classic shield a new trick: a *realtime voice changer!* Speak like everyone's favorite baritone Sith lord or sing along with the Lollipop Guild. The Wave Shield has long been a staple among makers' Halloween projects. This latest addition really cinches it!

Core Parts List

There are three central components to this project:

- Adafruit Metro 328 or Arduino Uno (https://adafru.it/METROXMETR) (an older Arduino Duemilanove or "328" Diecimila can be used as well...but not an Arduino Mega nor Leonardo, sorry).
- Adafruit Wave Shield (http://adafru.it/94).
- Adafruit Microphone Amplifier Breakout. (http://adafru.it/1063)

You'll also need basic soldering tools, wire and bits & bobs.



Additional Parts

This is an "open ended" project and the exact components for completion will depend on where you want to take it. Read through the full tutorial for ideas and recommendations on specific parts.

- For sound output you'll want headphones, portable MP3 player speakers or our Class D Audio Amplifier (http://adafru.it/987).
- The example sketch uses a 12-button keypad (http://adafru.it/419) for triggering pre-recorded sounds. But your application might need just a few simple buttons (http://adafru.it/476)...or none at all, if you're only using the voice effect.
- If adding pre-recorded sounds, you'll also need an SD card (http://adafru.it/102) containing WAV files.
- A 10K potentiometer (http://adafru.it/562) is used for setting the voice pitch...or you can simply rig the code for a permanent setting.
- If you want to noodle around with wiring, an extra prototyping shield (http://adafru.it/51) and stacking headers can come in very handy (http://adafru.it/85) - solder the wave shield with stacking headers and put the proto shield on top

- For portable use (such as costumes and props), add batteries, battery holders (http://adafru.it/771), etc.

To reiterate, it's a very good idea to read through the full tutorial and firm up your own project concept before making a shopping list. We'll demonstrate a couple of examples, but these aren't the last word. That's really the essence of Arduino, isn't it? Make it your own!

First Things First...

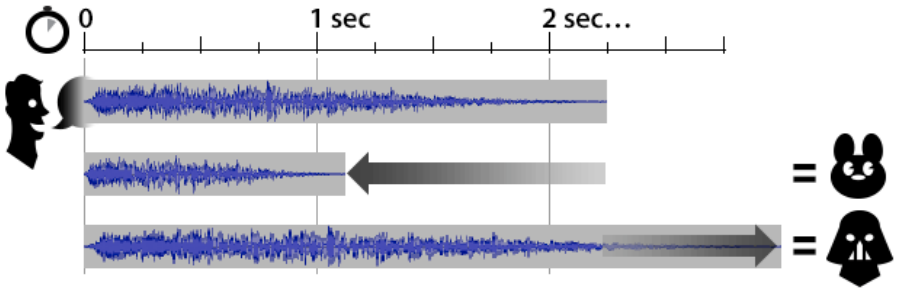
We also very strongly recommend...no, make that require...that you work through the original Wave Shield tutorial (https://adafru.it/cl8) before commencing with this project. It's a good way to verify the core pieces are working before adding extra layers of complexity.

Principles of Operation

Here we explain some of the geeky background theory stuff. If you just want to get into building the thing, you can skip ahead to the next page.

Graaains...

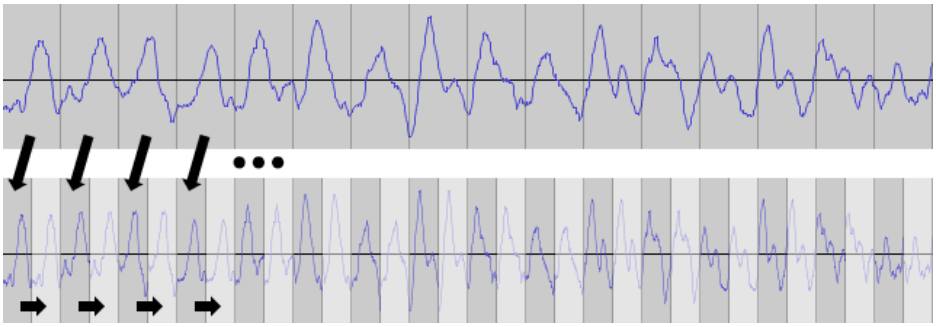
Regardless whether you're old enough to have played with Dad's LP turntable, or have dabbled in digital audio programs on the newest modern PC, you've likely experienced some version of this phenomenon: take an audio recording that's normally played back at one specific speed...and then change that speed, either compressing or expanding time...and the pitch of the audio changes along with it. Compress time and the pitch rises. Expand time and the pitch drops. *Frequency is inversely proportional to wavelength.*



That's easy with recordings...but with live audio, we don't really have that luxury. Realtime is *realtime*...we can't compress or expand it...it's happening as it happens. What's a would-be voice-changer to do?

There's a complex technique called a *Fourier transform* that converts a function (or, say, a stream of audio samples) into its frequency spectrum. The resulting frequency values can be altered and an *inverse transform* applied to turn this back into audio. This is all mathematically good and proper...but it's a very demanding process and way beyond what our little Arduino can handle. A fairly potent CPU or DSP is usually required. We'll need a shortcut or some hack...

In digital music circles, *granular synthesis* is a technique of joining and layering lots of very short audio samples (or "grains") — on the order of one to a few milliseconds — to build up more complex sounds or instruments. Now picture just a single "grain," 10 milliseconds or so...and we continually refresh this one grain from a live microphone. By time-compressing or -stretching this one tiny loop, repeating or dropping short segments to keep up with realtime, we have the basis for a realtime pitch shifter. It really seems like this shouldn't work...but it does! Speech waveforms tend to repeat over the very short term, and we can drop or repeat some of those waves with only minor degradation in legibility.



This approach is totally suited to the Arduino's limited processing power and RAM. The result isn't going to be

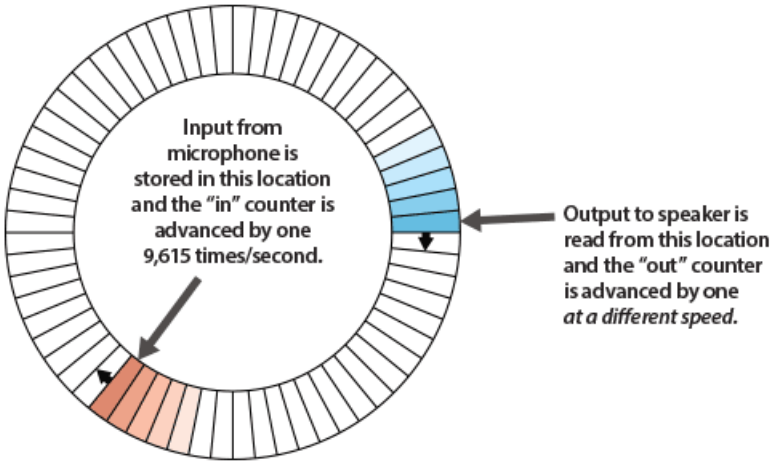
Hollywood quality, but it's still vastly better than the majority of voice-changing toys and masks on store shelves. And you get to make it yourself...how cool is that?

Sampling Audio

The frequency range of human voice covers about 300 Hz to 3,500 Hz (and harmonics may extend above this). The Nyquist sampling theorem (https://adafru.it/vCL) states that a minimum 2X sample rate is needed to faithfully reconstruct a signal. For human voice, that means 7 KHz sampling...but a little more wouldn't hurt.

Repeatedly calling the Arduino's standard analogRead() function in a loop is way, WAY too slow for this. We need to get deeper into the works of the Arduino's analog-to-digital converter, fiddling directly with special registers and modes. A capability called *free-run mode* collects analog samples at a fast, fixed interval without repeated polling in our code. An interrupt handler is automatically called each time a new sample is ready, which happens like clockwork. Running full tilt, a 16 MHz Arduino can capture 9,615 10-bit samples per second. More than enough for sampling voice!

The audio samples are stored in a *circular buffer,* which is really just big fancy computer science words for "when you reach the end of the buffer, roll back around to the beginning and write over it." But conceptually, it helps to think of it as a literal circle:



The frequency of recorded sound will seldom match the buffer length exactly, and audio samples are stored and read at different rates. This can produce a sharp discontinuity — a popping noise — each time the "in" and "out" points cross. A small extra buffer is used to store some of the prior audio samples, and the code cross-fades the audio over this boundary to reduce the "pop."

Because our audio "grain" is relatively short (about 10 milliseconds), the RAM requirements should be fairly modest, a few hundred bytes. Problem is, we'd also like to continue doing those things that the Wave Shield was designed for — namely, playing back WAV files. That requires reading files from an SD card, and that in turn consumes lots of RAM. Fortunately the design of the WAV-playing code lets us gain access that library's memory and recycle it for our own needs.

The technical details are all well-commented in the source code. So if you're curious about the specifics of this implementation...use the source, Luke!

Limitations

When introducing new users to Arduino, I often describe it as “just enough computer to do any one thing really well.” Walking while chewing gum is a challenge. And so it goes with this project as well. Keep the following limitations in mind:

- It can process the voice effect or play back WAVs (and can do both within the same sketch), but you can’t do both simultaneously.
- You can’t read other analog inputs when the voice effect is running (case in point, you can’t alter the pitch continually with a potentiometer). If using analog sensors as sound triggers (e.g. force-sensing resistor pads in shoes), consider work-arounds such as using a carefully-trimmed voltage divider to a digital input, or a second MCU to process analog inputs, forwarding triggers over a serial or I2C connection.

Building It

Phase 1:

Follow the original Wave Shield tutorial

We can't emphasize this one enough: work through the original Wave Shield tutorial (https://adafru.it/cl8) before moving on to the voice changer!

This project has many separate parts, and a misstep with any one of them can stop the whole system from working. It would be tricky to debug the point of failure among all the possibilities. Invest a little time now to get the basic Wave Shield examples working — especially the "Pi speak" demo. This lets you know that the shield is properly assembled, the SD card properly formatted and so forth. *Then* we'll add the extra features.

Start by downloading the WaveHC library for Arduino (https://adafru.it/kAa)...not only for WAV playback, but the voice changer relies on this code too. We have a tutorial explaining how Arduino libraries are installed (https://adafru.it/aYG). Download this ZIP file containing WAV files (https://adafru.it/cml) for the digits of pi. Then proceed through the tutorial until your Wave Shield is speaking them.

Phase 2:

Adding voice effects and a sound trigger keypad

With the basic Wave Shield working, now we can add the voice changer and a sound-triggering keypad. You can complete this phase on your workbench using a breadboard...we'll make it portable later, after confirming that it works.

Download the Adavoice sketch for Arduino (https://adafru.it/E-S). And you should already have the WaveHC (https://adafru.it/aQ7) library installed from the prior phase.

https://adafru.it/E-S

https://adafru.it/E-S

The GND and 3.3V lines from the Arduino need to connect to several points, so you may want to a breadboard's power rails for this. 3.3V from the Arduino should connect to the Electret Mic Amp VCC pin, one outside leg of a 10K potentiometer, and the Arduino's AREF pin. GND from Arduino should connect to GND on the Mic Amp and the opposite outside leg of the potentiometer.

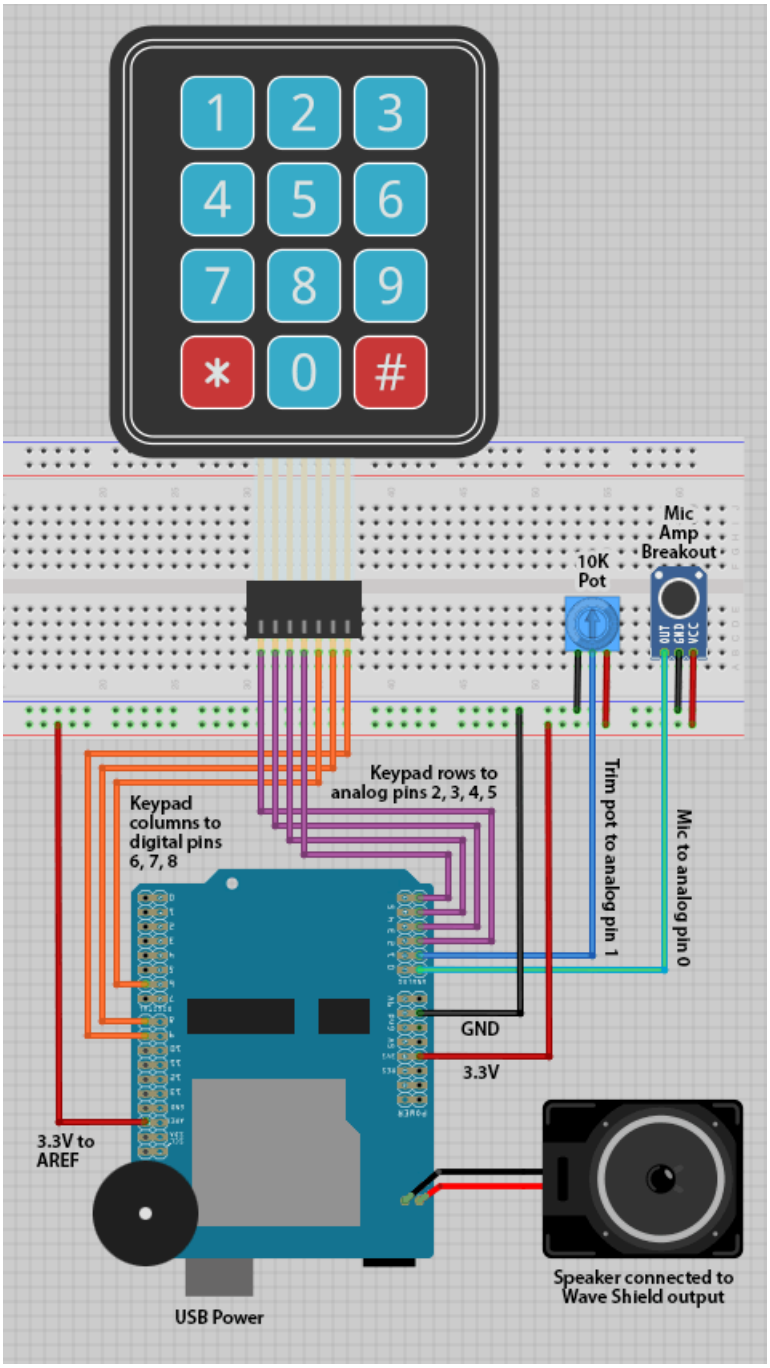
Don't forget the AREF connection...the circuit won't work without it!

The Mic Amp output connects to analog pin 0, and the center leg of the potentiometer connects to analog pin 1.

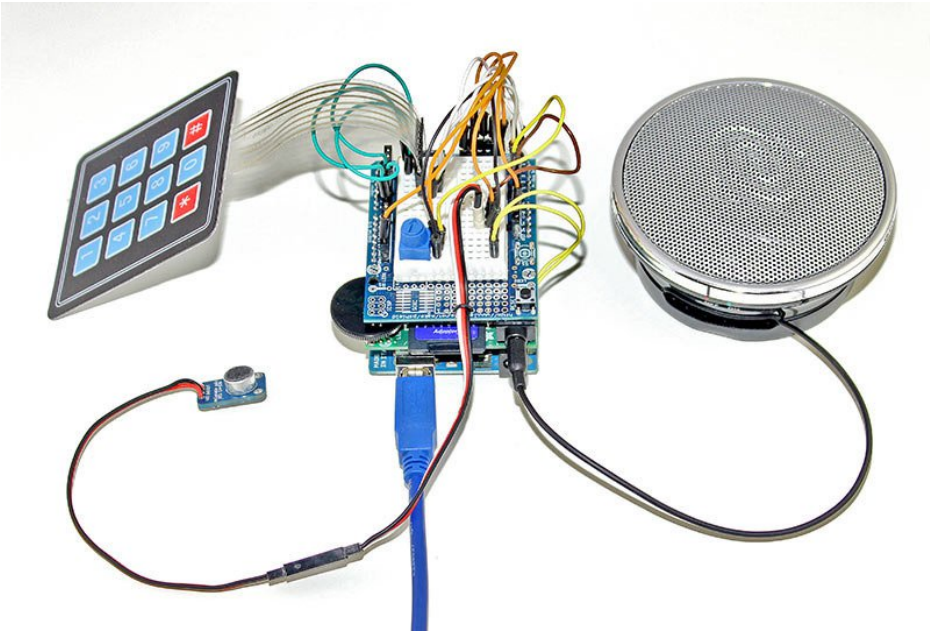
If you plan to use prerecorded sound effects (some examples are in the "wavs" folder included with the sketch), you'll need a FAT-formatted SD card with the files placed in the root directory (similar to how the "Pi speak" sketch worked). A 12-button keypad connects to digital pins 6, 7, 8 (columns) and analog pins 2, 3, 4, 5 (rows). But with some changes to the sketch, this can be adapted to use just a few buttons or other triggers. (The keypad is great for haunted house sounds, but too cumbersome for a costume.)

A small speaker can be connected directly to the Wave Shield's amplifier output. For more volume, we recommend

using amplified speakers such as the portable type for iPods and MP3 players, or our Class D Audio Amplifier breakout.



Upload the Adavoice sketch to the Arduino if you haven't already done this. If everything is wired up and loaded correctly, you should head a startup chime when the sketch starts (if using an SD card with the sample WAVs). If there's no sound, use the Arduino serial monitor and watch for diagnostic messages.



Once up and running, you can then talk into the microphone and should hear the altered result through the speaker or headphones (keep the mic away from the speaker to avoid feedback). Pressing any of the keypad buttons will stop the voice effect to play the corresponding sound, then resume afterward.

Note that the pitch dial does not work in real time! This is normal and a limitation of the way we're running the analog-to-digital converter at full speed. To get a new pitch reading, you need to either play back a sound or press the reset button.

Phase 3:

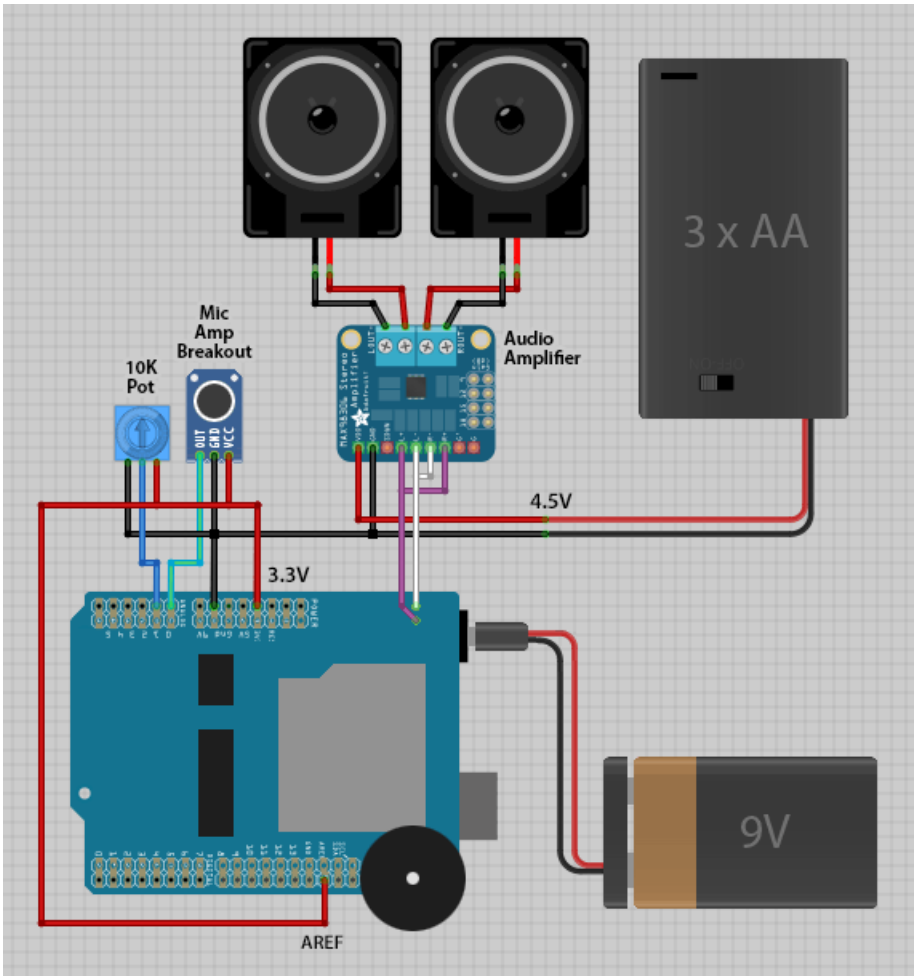
Making it battery-powered and portable

To simplify the wiring diagram, we'll illustrate this next section without the keypad. But you can still include it if you want! The connections are the same as above.

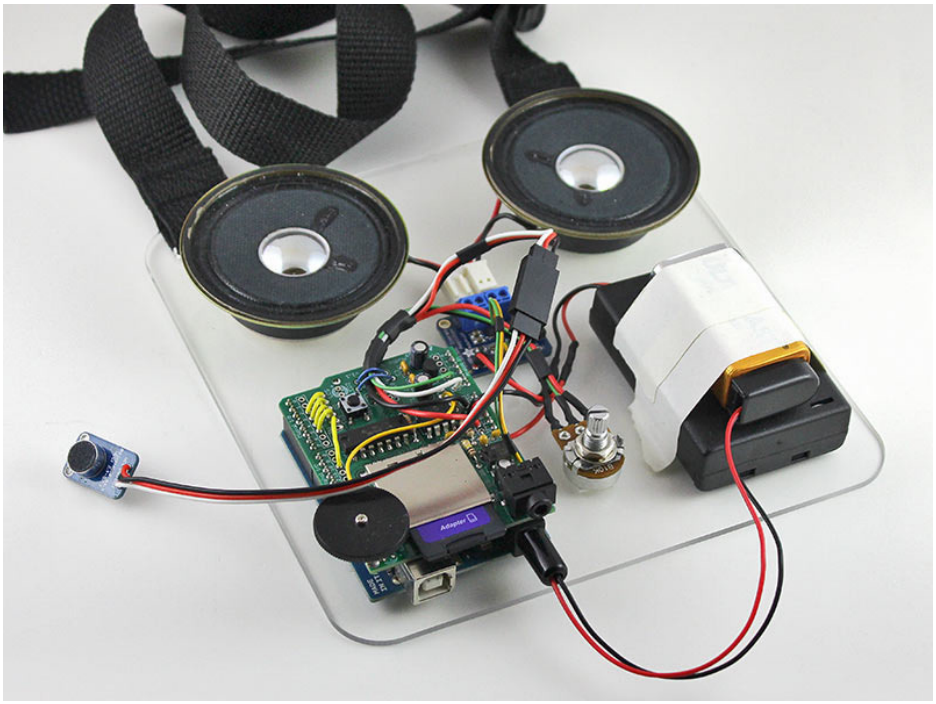
Because breadboard circuits are too delicate for portable use, we'll join components directly this time.

The Wave Shield can drive a small speaker on its own, but this doesn't provide a lot of "oomph." Parties and comic conventions are loud, so you'll probably want a boost! We're using our Class D Audio amplifier here with a pair of 4 Ohm speakers. Alternately, there are a lot of ready-to-go battery-powered speakers designed for iPods and other MP3 players that can plug right into the Wave Shield headphone jack. Using our own amp and speakers lets us custom-tailor the placement of all the parts.

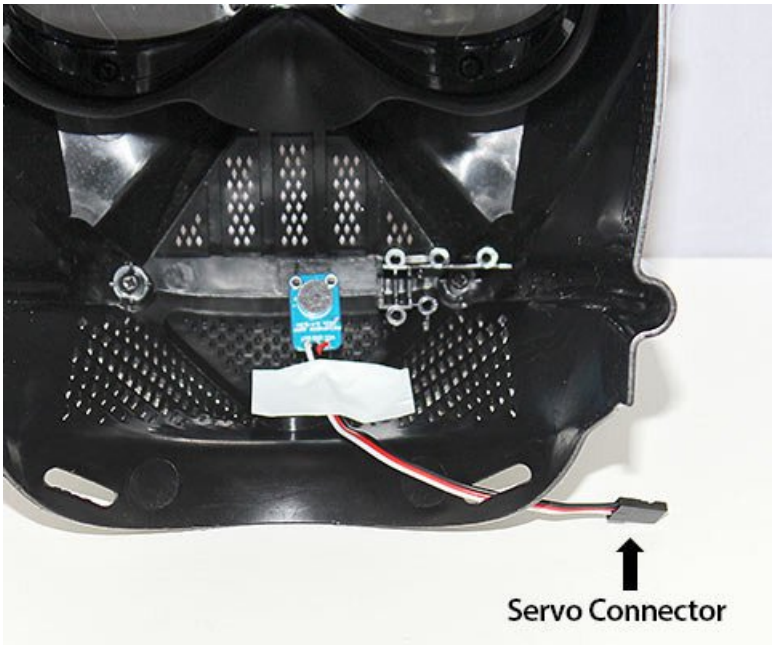
It's best to power the Arduino and audio amplifier separately. During particularly loud moments, the audio amp can draw a lot of current, resulting in a momentary voltage "sag" causing the Arduino to reset. Giving the Arduino it's own separate power supply prevents this. We're using a 9 Volt battery connected to the DC barrel jack, or a 6X AA battery pack will last considerably longer. In any case, the ground connection is common between the Arduino and audio power sections, as well as the 3.3V part of the circuit (for the mic amp and trim pot).



Here we've mounted all the parts on a sheet of acrylic using double-stick foam tape, then fastened this to a nylon strap so it can be worn over one's chest. We chose tape for expediency only...give some thought to making your rig more durable, using mounting screws, zip ties, etc.



You can run the microphone connection a couple feet to reach inside a mask or helmet. A servo extension cable (http://adafru.it/973) provides a very handy 3-conductor separation point, so you can pop your head and set it down! Cut the servo cable in half, soldering one end to the mic amp board and the other side to the Arduino circuit.



Tips for Use in Costumes

- Give your project a complete dry run well ahead of time...don't dash right off to the party! Know how long batteries will last. Check that wires aren't pulled and connections aren't strained. Make sure no components get uncomfortably or dangerously hot. Fine-tune audio levels and reduce feedback.
- Build it extra rugged and motion-proof. Stranded wire flexes much better than solid-core wire. Use beefy, NASA-style inline splices (https://adafru.it/aQ4). Implement strain reliefs to avoid cracked solder connections. Breadboards are fine for prototyping, but solder up your rig for deployment (http://adafru.it/591).
- Don't shout — speak softly and let the amplifier boost your voice. You want people to hear the "bent" sound, not your natural voice.
- Point speakers away from the microphone to avoid feedback. Even a few degrees can make a big difference.
- Sweat is horribly corrosive stuff! It's mostly salt water — and look what that does to ships at sea. Even worse, it's conductive! Seal *everything.* Heat-shrink all wire connections, and use plastic enclosures or epoxy for any electronics that are fully inside a costume. If using a microphone inside a mask (which may have both sweat *and* condensation from breath, ewww!), borrow an old audio pro trick and wrap the mic inside a balloon.
- Pack spare batteries and, space permitting, a minimal repair kit of safety pins, a few zip ties and a length of duct tape.
- Never let technical wizardry get in the way of a good performance! The example sketch uses a membrane keypad with many tiny buttons...that's fine for a tabletop "sound board" instrument, but a poor choice for a Godzilla suit (who really should be continually thrashing about leveling Tokyo, not standing still to hunt around for a specific button). More isn't always better...one or two buttons hidden in a glove may suffice. Practice until your performance is natural and your technology is discreet.