How To Make a Simple Arduino Alarm System

Written by James Bruce
August 26, 2013
(http://www.makeuseof.com/tag/author/jbruce/)

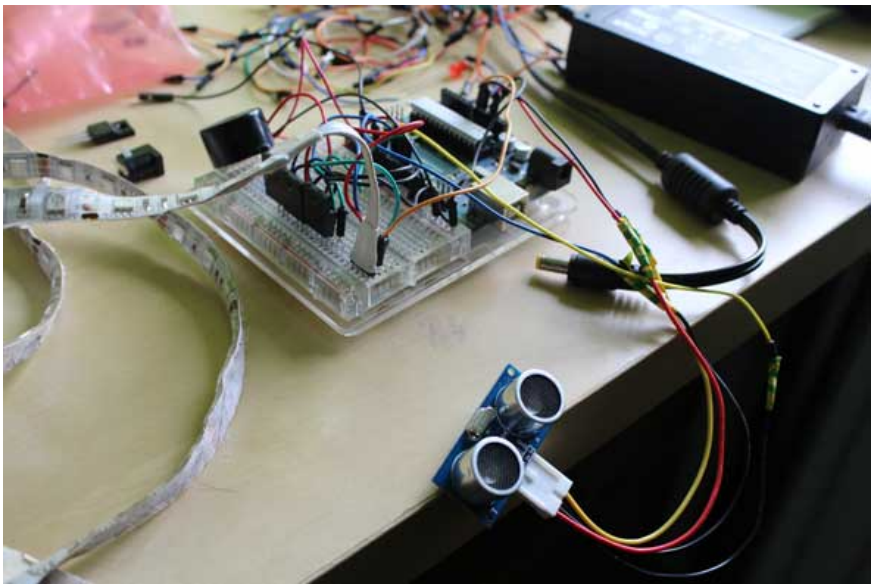
Detect movement, then scare the heck out of an intruder with a high pitched alarm sounds and flashing lights. Does that sound fun? Of course it does. That's the goal of today's Arduino project, suitable for beginners. We'll be writing completely from scratch and testing as we go along so you can hopefully get some idea of how it's all being done rather than simply installing something I've already made.

Disclaimer: this isn't going to actually protect your house. It might give your sister a nasty shock when she sneaks into your room though.

You'll need:

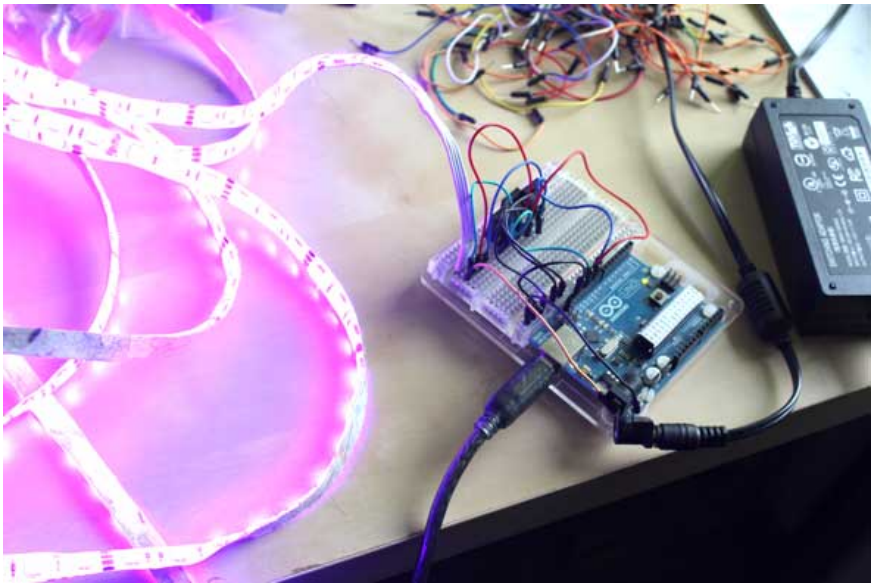
- An Arduino
- Ultrasonic "ping" sensor, I'm using HC-SR04 (http://www.amazon.com/dp/B004U8TOE6/ref=as_at?tag=mak041-20&linkCode=as2&) A PIR would be better, but those are expensive. A ping sensor can be placed surreptitiously in a doorway and still serve the same basic job, and is only $5
- A piezo buzzer
- LED strip light, with the same wiring we used back in this project (http://www.makeuseof.com/tag/build-your-own-dynamic-ambient-lighting-for-a-media-center/).

As you're wiring up this project, don't remove everything each time — just keep building on the last block. By the time you get to "Coding The Alarm System" section, you should have all the bits and pieces wired up, looking something like this:



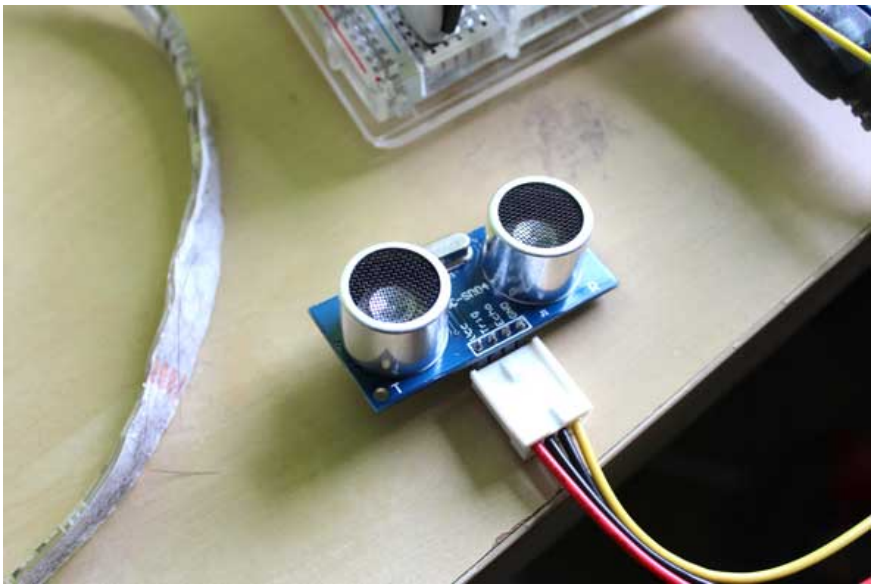
Flashing Lights

Use the wiring diagram from this project (http://www.makeuseof.com/tag/build-your-own-dynamic-ambient-lighting-for-a-media-center/) to hook up your LED strip; don't change the pins, as we need PWM output. Use this code to quickly test your wiring. If all goes well, you should have this:

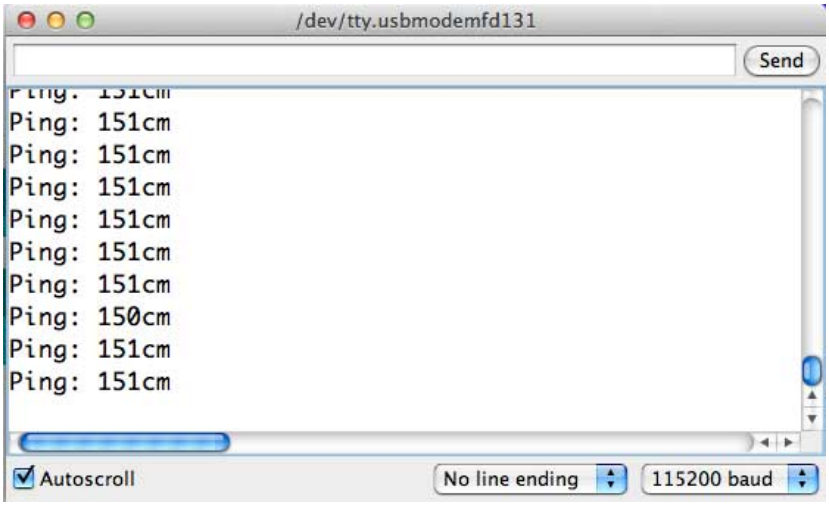


Distance Sensor

On the SR04 module, you'll find 4 pins. **VCC** and **GND** go to +5V rail and ground respectively; **TRIG** is the pin used to send a sonar signal, put this on pin 6; **ECHO** is used to read the signal back (and therefore calculate the distance) — put this on 7.



To make things incredibly simple, there's a library we can use called NewPing (http://code.google.com/p/arduino-new-ping/). Download and place in your Arduino's **Library** folder and restart the IDE before continuing. Test using this code (http://pastebin.com/tgjAKySw); open up the serial monitor and make sure the speed is set to 115200 baud. With any luck, you should see some distance measurements being send back to you at a pretty high speed. You may find a variance of 1 or 2 centimeters, but this is fine. Try running your hand in front of the sensor, moving it up and down to observe the changing readings.



The code should be fairly simply to understand. There are a few declaration of relevant pins at the start, including a maximum distance – this may vary according to the exact sensor you have, but as long as you're able to get less than 1 meter readings accurately, you should be fine.

In the loop of this test app, we use the **ping()** function to send out a sonar ping, getting back a value in milliseconds of how long it took for the value to return. To make sense of this, we use the NewPing libraries built in constant of **US_ROUNDTRIP_CM**, which defines how many microseconds it takes to go a single centimeter. There's also a 50 ms delay between pings to avoid overloading the sensor.

Piezo Alarm

The Piezo crystal sensor is a simple and cheap buzzer, and we can use a PWM pin 3 to make different tones. Connect one wire to pin 3, one to ground rail – it doesn't matter which.

Use this code (http://pastebin.com/7eeQPS9h) to test.

The only way to kill the rather obnoxious and loud alarm is to pull the plugs. The code is a little complex to explain, but it involves using sine waves to generate a distinctive sound. Tweak the numbers to play with different tones.

Coding The Alarm System

Now that we have all the pieces of this puzzle, let's combine them together.

Go ahead and make a new sketch, called **Alarm**. Start by combining all the variables and pin definitions we've in the test examples until now.

```
#include <NewPing.h>

// Select which PWM-capable pins are to be used.
#define RED_PIN    10
#define GREEN_PIN   11
#define BLUE_PIN  9

#define TRIGGER_PIN  6  // Arduino pin tied to trigger pin on the ultrasonic sensor.
#define ECHO_PIN     7  // Arduino pin tied to echo pin on the ultrasonic sensor.
#define MAX_DISTANCE 100 // Maximum distance we want to ping for (in centimeters).

#define ALARM 3

float sinVal;
int toneVal;
```

Begin by writing a basic **setup()** function – we'll only deal with the lights for now. I've added a 5 second delay before the main loop is started to give us some time to get out of the way if needed.

```
void setup(){

   //set pinModes for RGB strip
   pinMode(RED_PIN,OUTPUT);
   pinMode(BLUE_PIN,OUTPUT);
   pinMode(GREEN_PIN,OUTPUT);

   //reset lights
   analogWrite(RED_PIN,0);
   analogWrite(BLUE_PIN,0);
   analogWrite(RED_PIN,0);

  delay(5000);
}
```

Let's use a helper function that allows us to quickly write a single RGB value out to the lights.

```
//helper function enabling us to send a colour in one command
void color (unsigned char red, unsigned char green, unsigned char blue)     // the color generatin
{
    analogWrite(RED_PIN, red);
    analogWrite(BLUE_PIN, blue);
    analogWrite(GREEN_PIN, green);
}
```

Finally, our loop for now is going to consist of a simple color flash between red and yellow (or, whatever you want your alarm to be — just change the RGB values).

```
void loop(){
   color(255,0,0); //red
   delay(100);
   color(255,255,0); //yellow
   delay(100);
}
```

Upload and test that to ensure you're on the right track.

Now, let's integrate the distance sensor to trigger those lights only when something comes within, say, 50 cm (just less than the width of a door frame). We've already defined the right pins and imported the library, so before your **setup()** function add the following line to instantiate it:

```
NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE); // NewPing setup of pins and maximum distance.
```

Underneath that, add a variable to store the state of the alarm being triggered or not, defaulting to false, of course.

```
boolean triggered = false;
```

Add a line to the **setup()** function so we can monitor the output on serial and debug.

```
Serial.begin(115200); // Open serial monitor at 115200 baud to see ping results.
```

Next, let's rename the current loop to **alarm()** – this is what's will be called if the alarm has been tripped.

```
void alarm(){
   color(255,0,0); //red
   delay(100);
   color(255,255,0); //yelow
   delay(100);
}
```

Now create a new **loop()** function, one in which we fetch a new ping, read the results, and trigger the alarm if something is detected within the meter range.

```
void loop(){
    if(triggered == true){
      alarm();
    }
    else{
      delay(50);// Wait 50ms between pings (about 20 pings/sec). 29ms should be the shortest delay
      unsigned int uS = sonar.ping(); // Send ping, get ping time in microseconds (uS).
      unsigned int distance = uS / US_ROUNDTRIP_CM;
      Serial.println(distance);
      if(distance < 100){
         triggered = true;
      }
   }
}
```

Let me explain the code briefly:

Start by checking to see if the alarm has been triggered, and if so, fire off the alarm function (just flashing the lights at the moment).

If it's not triggered yet, get the current reading from the sensor.

If the sensor is reading <100 cm, something has padded the beam (adjust this value if it's triggering too early for you, obviously).

Give it a trial run now, before we add the annoying piezo buzzer.

Working? Great. Now let's add that buzzer back. Add **pinMode** to the **setup()** routine.

```
pinMode(ALARM, OUTPUT);
```

Then add the piezo buzzer loop to the alarm() function:

```
for (int x=0; x<180; x++) {
    // convert degrees to radians then obtain sin value
    sinVal = (sin(x*(3.1412/180)));
    // generate a frequency from the sin value
    toneVal = 2000+(int(sinVal*1000));
    tone(ALARM, toneVal);
  }
```

If you try to compile at this point, you're going to run into an error — I've left this in deliberately so you can see some common issues. In this case, both the NewPing and standard tone library use the same interrupts — they are conflicting basically, and there's not a lot you can do to fix it. Oh dear.

No worries though. It's a common problem, and someone has a solution already — download and add this NewTone (http://code.google.com/p/arduino-new-tone/) to your Arduino Libraries folder. Adjust the beginning of your program to include this:

```
#include <NewTone.h>
```

And adjust the line:

```
  tone(ALARM, toneVal);
```

to

```
    NewTone(ALARM, toneVal);
```

instead.

That's it. Set your alarm up in the doorway of your bedroom for the next hapless would-be burglar.

Or, a dopey dog, which seemed completely unfazed by the alarm.

Having trouble with the code? Here's the complete app (http://pastebin.com/5kHgsHvi). If you're getting random errors, try pasting them below and I'll see if I can help.

Image credit: Fire Alarm (http://www.flickr.com/photos/magnera/3755006460/) via Flickr