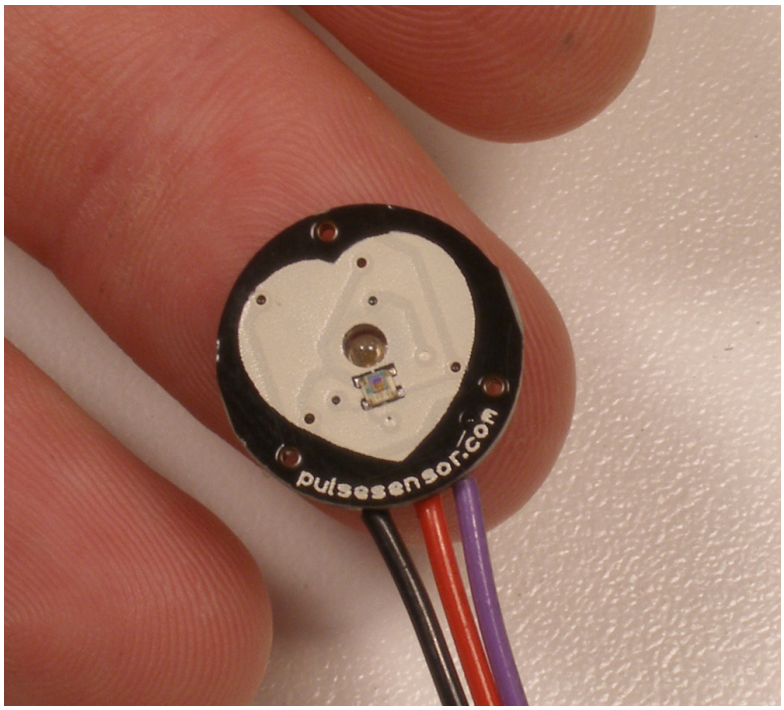
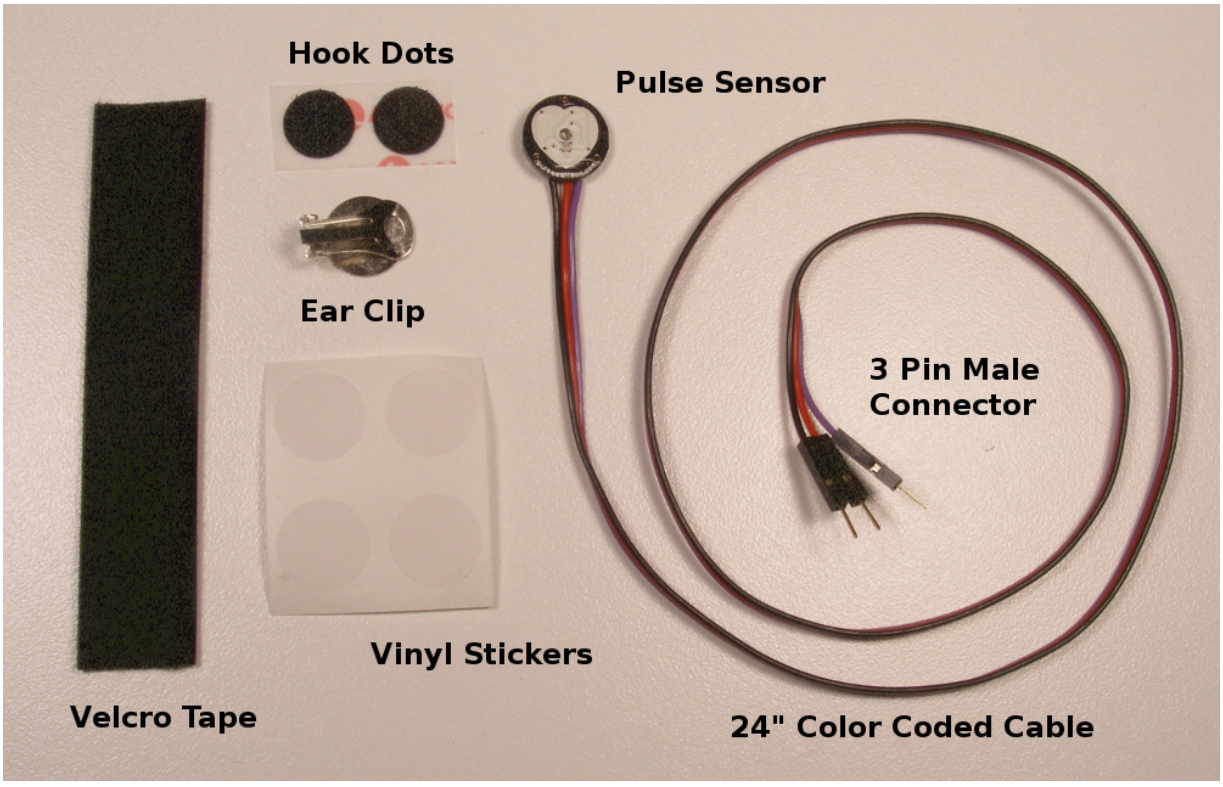
Pulse Sensor Getting Started Guide



Introduction:

Pulse Sensor is a well-designed plug-and-play heart-rate sensor for Arduino. It can be used by students, artists, athletes, makers, and game & mobile developers who want to easily incorporate live heart-rate data into their projects. The sensor clips onto a fingertip or earlobe and plugs right into Arduino. It also includes an open-source monitoring app that graphs your pulse in real time.



The Pulse Sensor Kit includes:

1) A 24-inch Color-Coded Cable, with (male) header connectors. You'll find this makes it easy to embed the sensor into your project, and connect to an Arduino. No soldering is required.

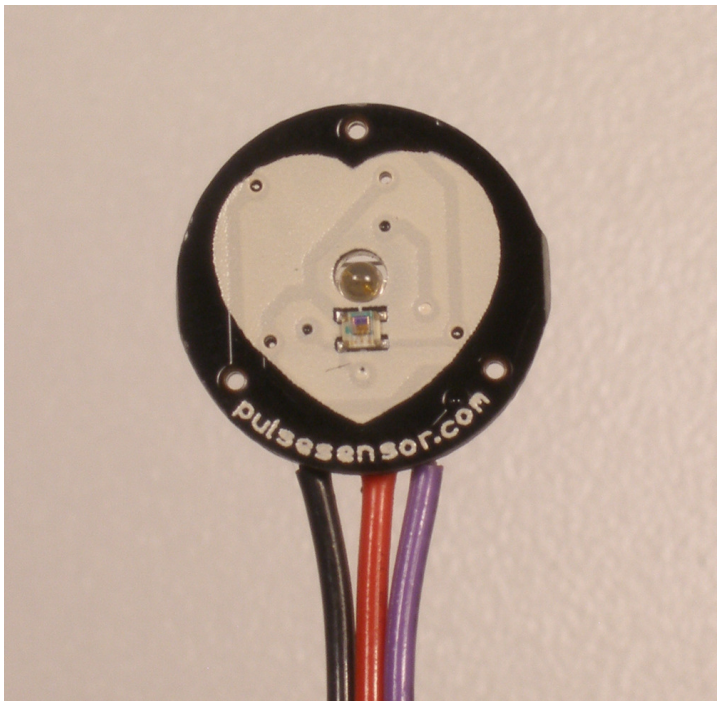
2) An Ear Clip, perfectly sized to the sensor. We searched many places to find just the right clip. It can be hot-glued to the back of the sensor and easily worn on the earlobe.

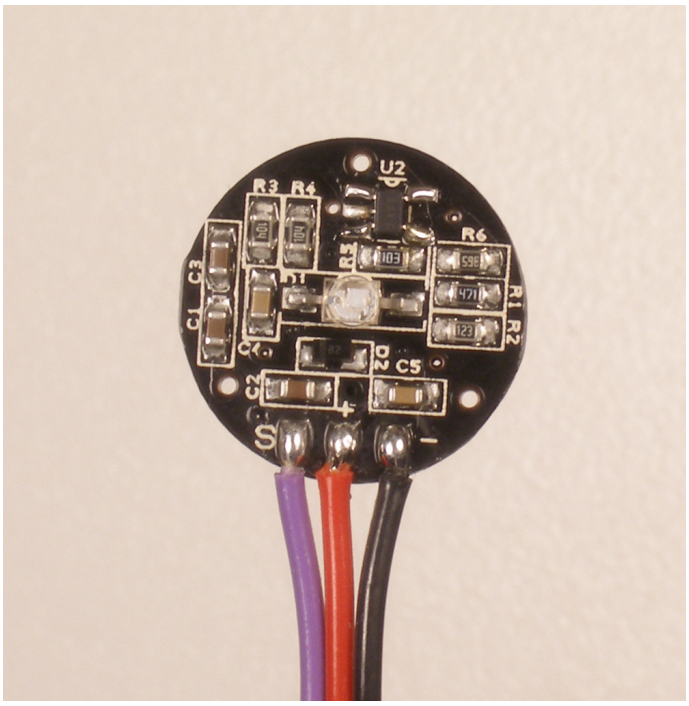
3) 2 Velcro Dots. These are 'hook' side and are also perfectly sized to the sensor. You'll find these velcro dots very useful if you want to make a velcro (or fabric) strap to wrap around a finger tip.

4) Velcro strap to wrap the Pulse Sensor around your finger.

4) 3 Transparent Stickers. These are used on the front of the Pulse Sensor to protect it from oily fingers and sweaty earlobes.

5) The Pulse Sensor has 3 holes around the outside edge which make it easy to sew it into almost anything.



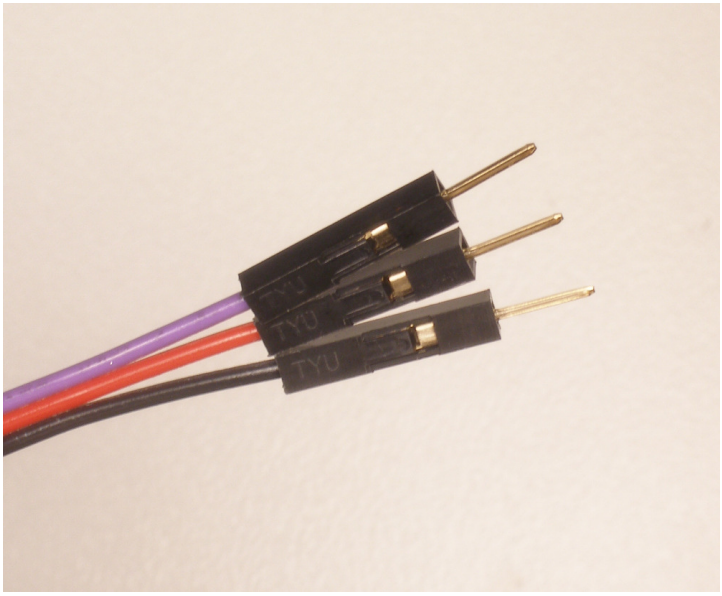


Let’s get started with Pulse Sensor Anatomy

The front of the sensor is the pretty side with the Heart logo. This is the side that makes contact with the skin. On the front you see a small round hole, which is where the LED shines through from the back, and there is also a little square just under the LED. The square is an ambient light sensor, exactly like the one used in cellphones, tablets, and laptops, to adjust the screen brightness in different light conditions. The LED shines light into the fingertip or earlobe, or other capillary tissue, and sensor reads the amount of light that bounces back. The other side of the sensor is where the rest of the parts are mounted. We put them there so they would not get in the way of the of the sensor on the front. Even the LED we are using is a reverse mount LED. For more about the circuit functionality, check out the Open Hardware page.

The cable is a 24” flat color coded ribbon cable with 3 male header connectors.

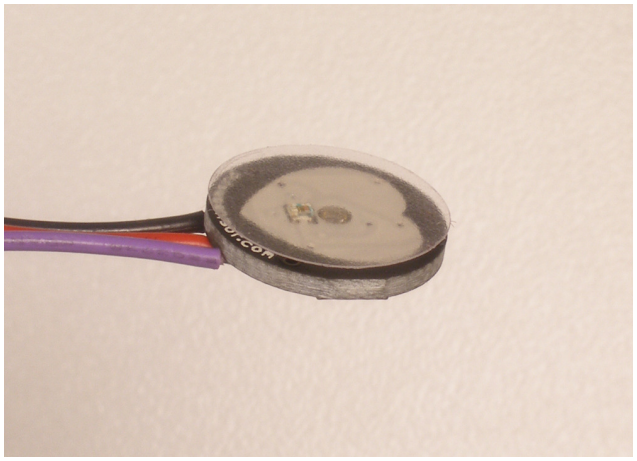
RED wire = +3V to +5V
BLACK wire = GND
PURPLE wire = Signal

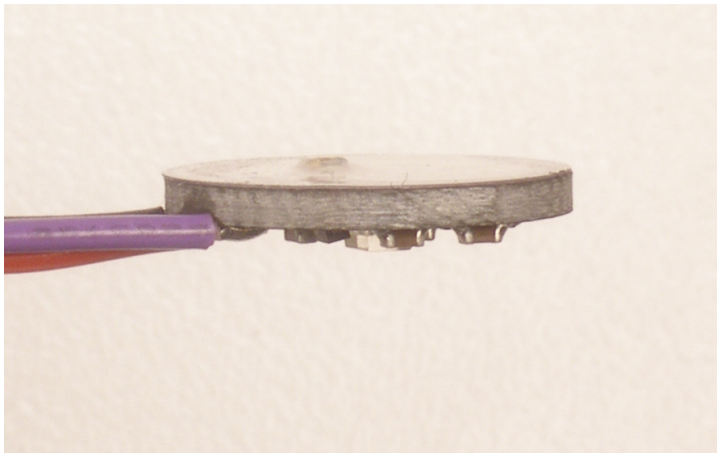


The Pulse Sensor can be connected to arduino, or plugged into a breadboard. Before we get it up and running, we need to protect the exposed circuitry so you can get a reliable heartbeat signal.

Preparing the Pulse Sensor

Before you really start using the sensor you want to insulate the board from your (naturally) sweaty/oily fingers. The Pulse Sensor is an exposed circuit board, and if you touch the solder points, you could short the board, or introduce unwanted signal noise. We will use a thin film of vinyl to seal the sensor side. Find the small card with four clear round stickers in your kit, and peel one off. Then center it on the Pulse Sensor. It should fit perfectly.





When you are happy with the way it's lined up, squeeze it onto the face all at once! The sticker (made of vinyl) will kind of stretch over the sensor and give it a nice close fit. If you get a wrinkle, don't worry, just press it down really hard and it should stick. We gave you 4, so you can replace it if necessary.

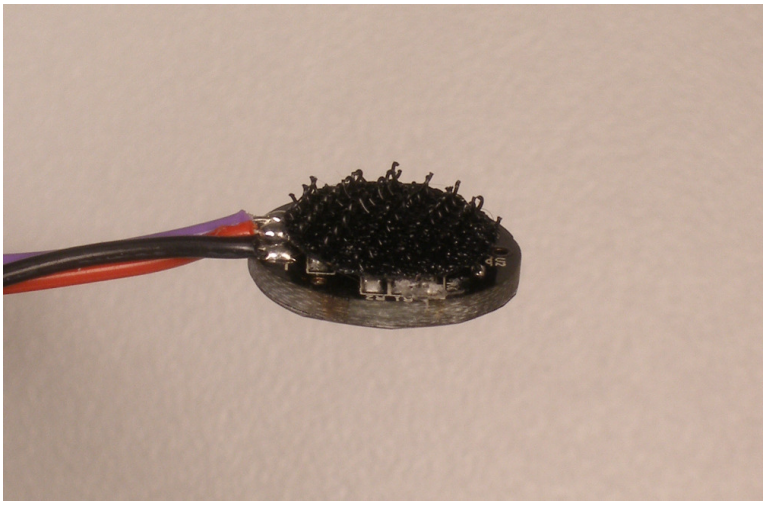
That takes care of the front side. The vinyl sticker offers very good protection for the underlying circuit, and we rate it 'water resistant'. meaning: it can stand to get splashed on, but don't wear it in the pool!

If this is your first time working with Pulse Sensor, you're probably eager to get started, and not sure if you want to use the ear-clip or finger-strap (or other thing of your design). The back of the Pulse Sensor has

even more exposed contacts than the front, so you need to make sure that you don't let it touch anything conductive or wet.

The easiest and quickest way to protect the back side from undesirable shorts or noise is to simply stick a velcro dot there for now. The dot will keep your parts away from the Pulse Sensor parts enough for you to get a good feel for the sensor and decide how you want to mount it. You'll find that the velcro dot comes off easily, and stores back on the little strip of plastic next to the other one we gave you.

Notice that the electrical connections are still exposed! We only recommend this as a temporary setup so you can get started. We show you how to better seal the Pulse Sensor later in this document.



Running The Pulse Sensor Code

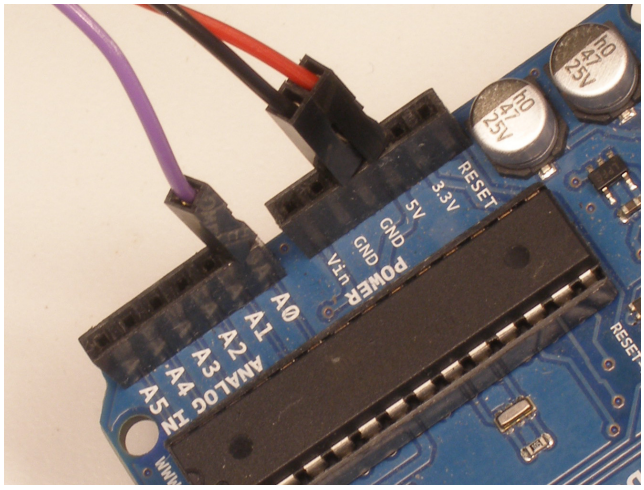
Get the latest Arduino and Processing Pulse Sensor software here http://pulsesensor.com/downloads/
Arduino code is called "PulseSensorAmped_Arduino-xx"
The Processing code is called "PulseSensorAmped_Processing-xx"
For Arduino and Processing programming environments: www.arduino.cc www.processing.org

We strongly advise that you DO NOT connect the Pulse Sensor to your body while your computer or arduino is being powered from the mains AC line. That goes for charging laptops and DC power supplies. Please be safe and isolate yourself from from the power grid, or work under battery power.
Connect the Pulse Sensor to: +V (red), Ground (black), and Analog Pin 0 (purple) on your favorite Arduino, or Arduino compatible device, and upload the 'PulseSensoAmped_Arduino-xx' sketch.

note: If you want to power Pulse Sensor Amped with low voltage (3.3V for example), make sure you have this line of code in the setup()

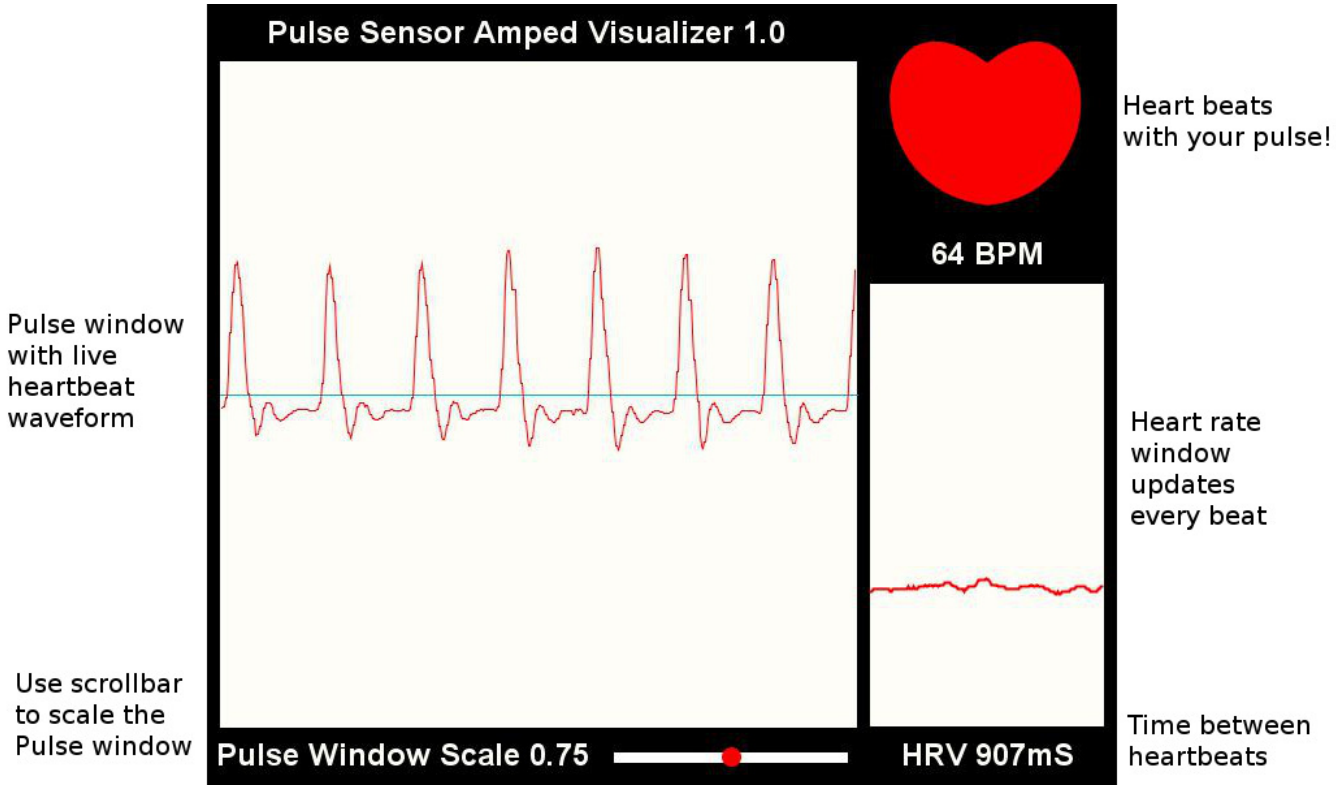
analogReference(EXTERNAL);

Also, make sure that you apply the lower voltage to the Arduino Aref pin (next to pin 13).



After it's done uploading, you should see Arduino pin 13 blink in time with your heartbeat when you hold the sensor on your fingertip. If you grip the sensor too hard, you will squeeze all the blood out of your fingertip and there will be no signal! If you hold it too lightly, you will invite noise from movement and ambient light. Sweet Spot pressure on the Pulse Sensor will give a nice clean signal. You may need to play around and try different parts of your body and pressures. If you see an intermittent blink, or no blink, you might be a zombie or a robot.

To view the heartbeat waveform and check your heart rate, you can use the Processing sketch that we made. Start up Processing on your computer and run the 'PulseSensorAmped_Processing-xx' sketch. This is our data visualization software, and it looks like this.



note: If you get an error when starting this code, you may need to make sure you are selecting the right serial port. Check the Troubleshooting section below..

The large main window shows a graph of raw sensor data over time. The Pulse Sensor Data Window can be scaled using the scrollbar at the bottom if you have a very large or very small signal. At the right of the screen, a smaller data window graphs heart rate over time. This graph advances every pulse, and the Beats Per Minute is updated every pulse as a running average of the last ten pulses. The big red heart in the upper right also pulses to the time of your heartbeat. When you hold the Pulse Sensor to your fingertip or earlobe or (fill in body part here) you should see a nice heartbeat waveform like the one above. If you don't, and you're sure you're not a zombie, try the sensor on different parts of your body that have capillary tissue. We've had good results on the side of the nose, middle of the forehead, palm, and lower lip. We're all different, original organisms. Play around and find the best spot on you and your friends. As you are testing and getting used to the sensor. You may find that some fingers or parts of fingers are better than others. For example, I find that when I position the sensor so that the edge of the PCB is at the bottom edge of my earlobe I get an awesome signal. Also, people with cold hands or poor circulation may have a harder time reading the pulse. Run your hands under warm water, or do some jumping-jacks!

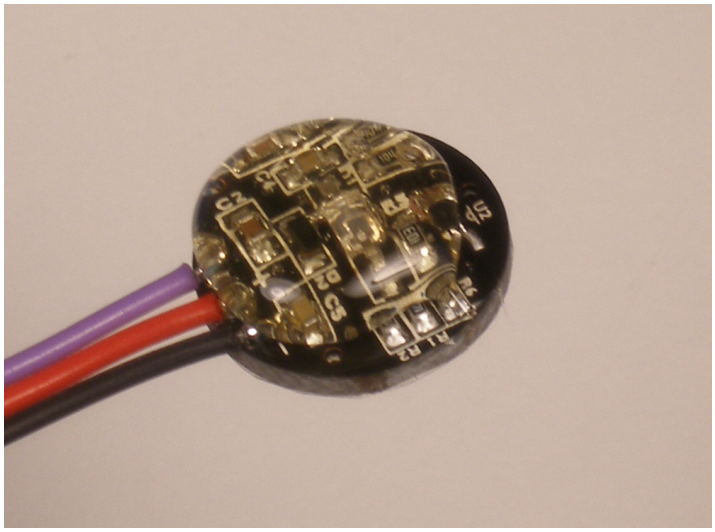
The Pulse Sensor signal will settle at the midpoint of the Pulse window when there is no heartbeat, or if it's just sitting still on your desk. If you are still seeing only a small pulse waveform, check the troubleshooting section below for more tips and tricks.

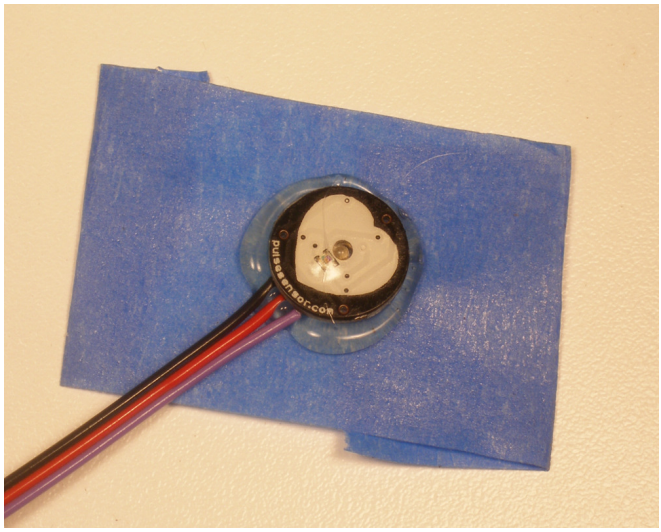
Sealing the Back Side of Pulse Sensor

Basic protection for the back of the Pulse Sensor
and prep for attaching Velcro Dot.

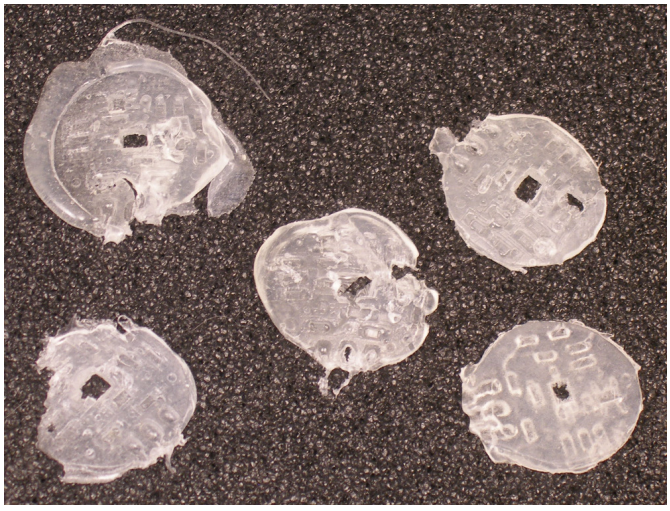
It's really important to protect the exposed Pulse Sensor circuitry so the sweat of your fingertips or earlobe (or wherever) doesn't cause signal noise or a short circuit. This How-To uses hot glue, which can be removed easily or re-worked if you want to change where you've stuck your Pulse Sensor. We love hot glue!
The other things you'll need are:
Hot Glue Gun
Blue Tape (any tape should do ok)
Nail Trimmers (or your favorite trimming device. Flush-cut wire snips work well too)
Read these instructions all the way through before you start!

First, attach the clear vinyl sticker to the front of your Pulse Sensor, as shown above. Then put a blob of hot glue on the back, right over the circuit. Size can be difficult to judge sometimes. What I meant was put a hot glue blob about the size of a kidney bean on the back side of the Pulse Sensor.



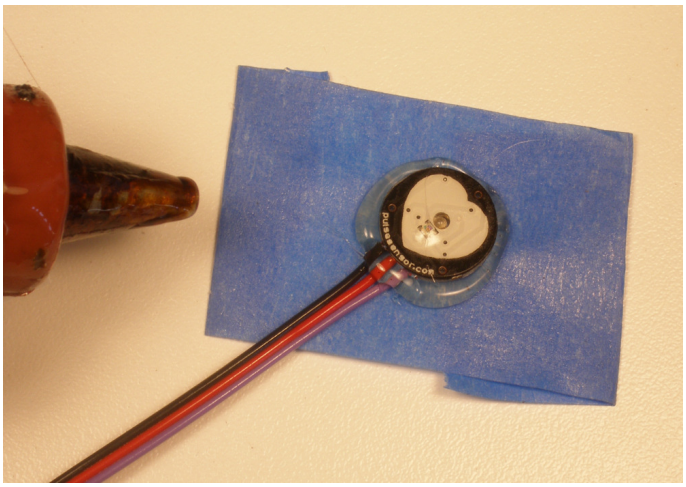


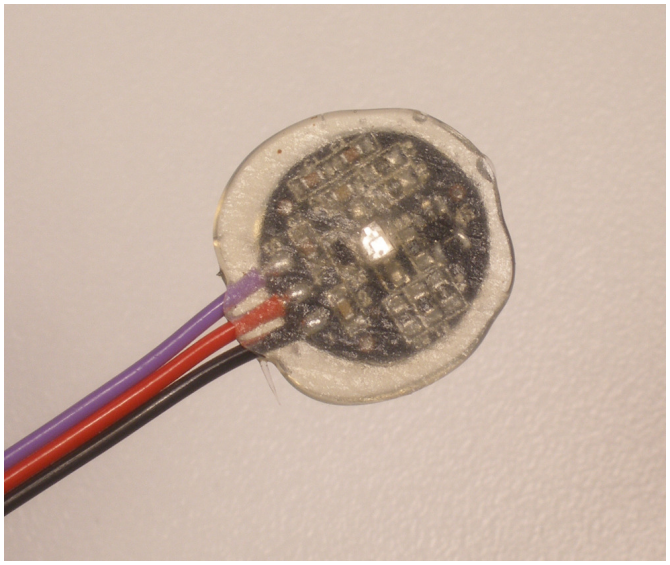
Then, while the glue is still very hot, press the Pulse Sensor glue-side-down onto the sticky side of a piece of blue tape (I believe that blue tape has magical properties, but if you don't have blue tape other kinds of tape will work just as well).

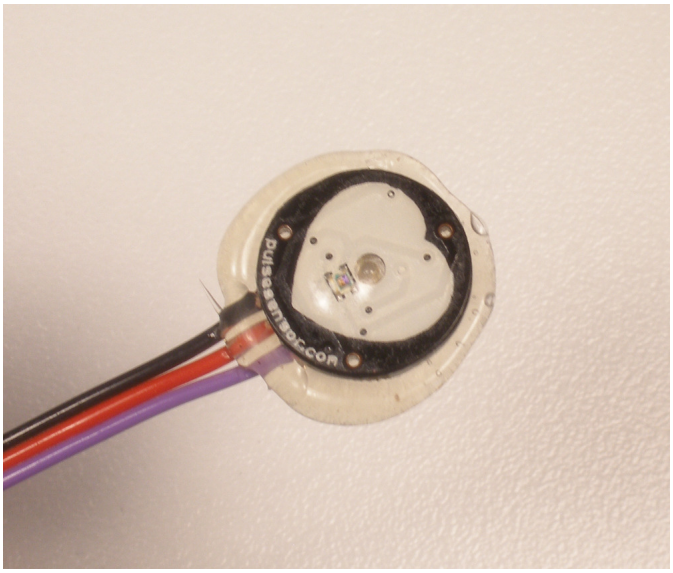


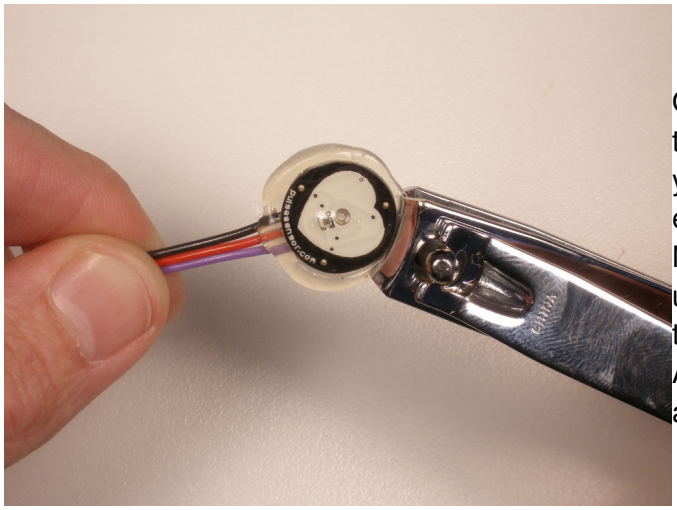
note: The tallest thing on the back of the Pulse Sensor is the green LED housing right in the middle. Use it to make the hot-glue seal thin and strong. When you press evenly until the back of the LED touches, all the conductive parts will be covered with hot glue. If the glue doesn't ooze out all around, let it cool down, then peel if from the Pulse Sensor and try again. Once the glue has cooled down and has some body, you can peel it off easily. Here's some pics of hot glue 'impressions' that I took during the making of this guide.

Next put a small dab of hot glue on the front of the cables, where they attach to the Pulse Sensor circuit board. This will bond to the other glue that's there and act as a strain-relief for the cable connection. This is important because the cable connection can wear out over time.

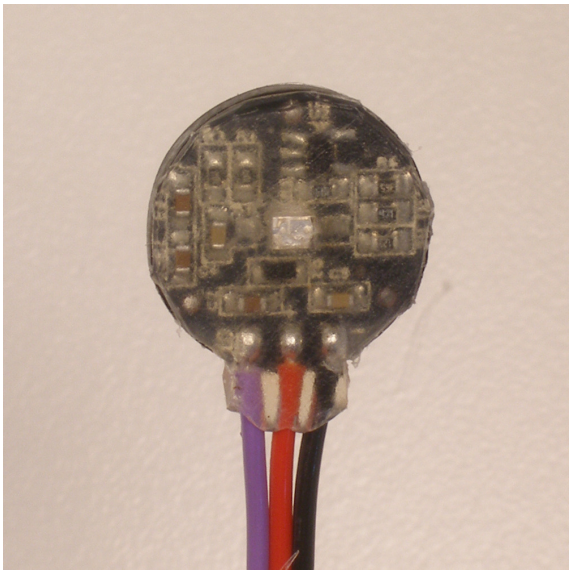




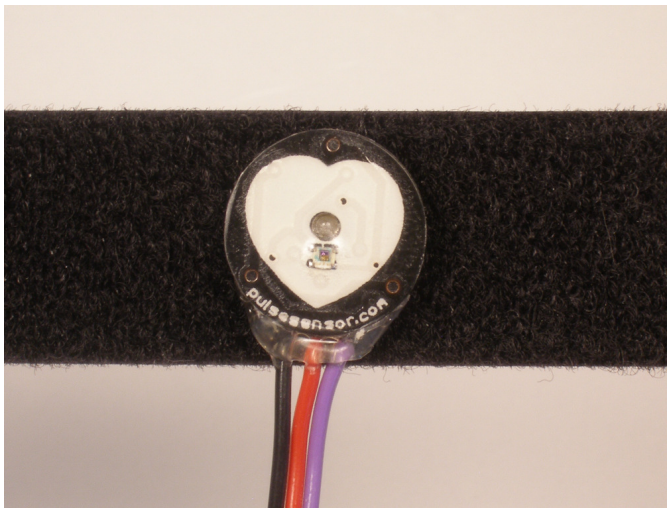


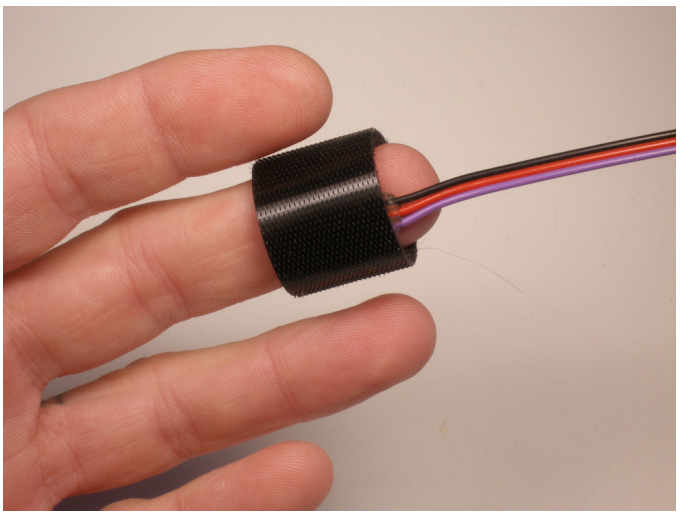


Once the hot glue has cooled (wait for it!) the blue tape will peel off very easily. Check your work to make sure that there are not exposed electrical connections!
Next is trimming. I find the easiest way is to use nail clippers. Flush cutting wire snips work too. Take care not to clip the wires!!!
And leave enough around the cable to act as a good strain-relief



This is the basic Pulse Sensor Hot Glue Seal, It's also got the clear vinyl sticker on the front face. We're calling this 'Water Resistant', ready to be handled and passed around from fingers to earlobes or whatever. It is not advised to submerge or soak the Pulse Sensor with this basic seal.
Now you can stick on the velcro dot (included) and make a finger strap with the velcro tape (included)!





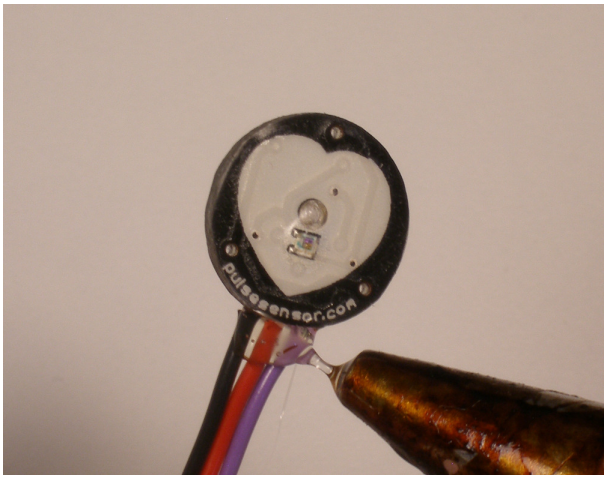
Attaching the Ear Clip

We looked all over, and were lucky enough to find an ear clip that fits the Pulse Sensor perfectly. The earlobe is a great place to attach Pulse Sensor. Here's some instruction on how to do it.

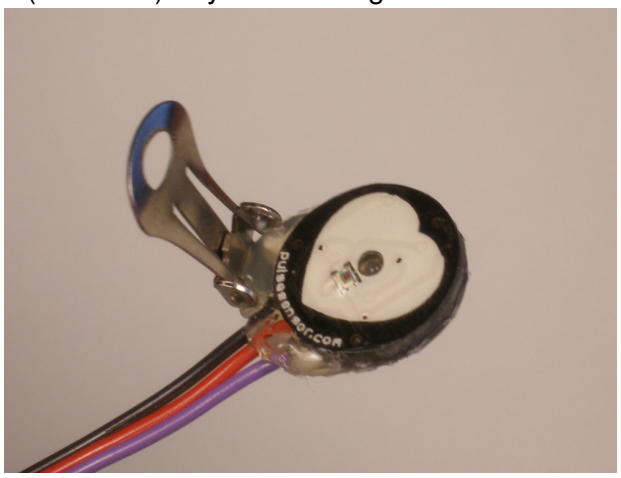
It is important to apply some strain relief to the cable connection where it meets the Pulse Sensor PCB. The little wire connections can wear out and break (or short on something) over time. We can do this with hot glue, like we did in the previous example.

First, attach a clear vinyl sticker to the front of the Pulse Sensor if you have not already. Then, put a small dab of hot glue on the front of the cables right where they meet the PCB. Get some on the edge of the PCB too, that will help. Remember, if you don't like the blob you've made for any reason, it's easy to remove once it cools down.

Next place the Pulse Sensor face down, and put a dab of glue about the size of a kidney bean on the back as illustrated above. Center the round part of the ear clip on the sensor and press it into the hot glue. The tallest component on the back is the plastic body of the reverse mount LED, and if you press it evenly it will help keep the metal of the ear clip from contacting any of the component connections.



Allow the hot glue to ooze out around the ear clip. That will ensure good coverage. Take care not to let the hot glue cover around the ear clip hinge, as that could get in the way of it working. Trimming is easy with nail clippers (as above) or your trimming tool of choice. Don't trim the wires by mistake!!!



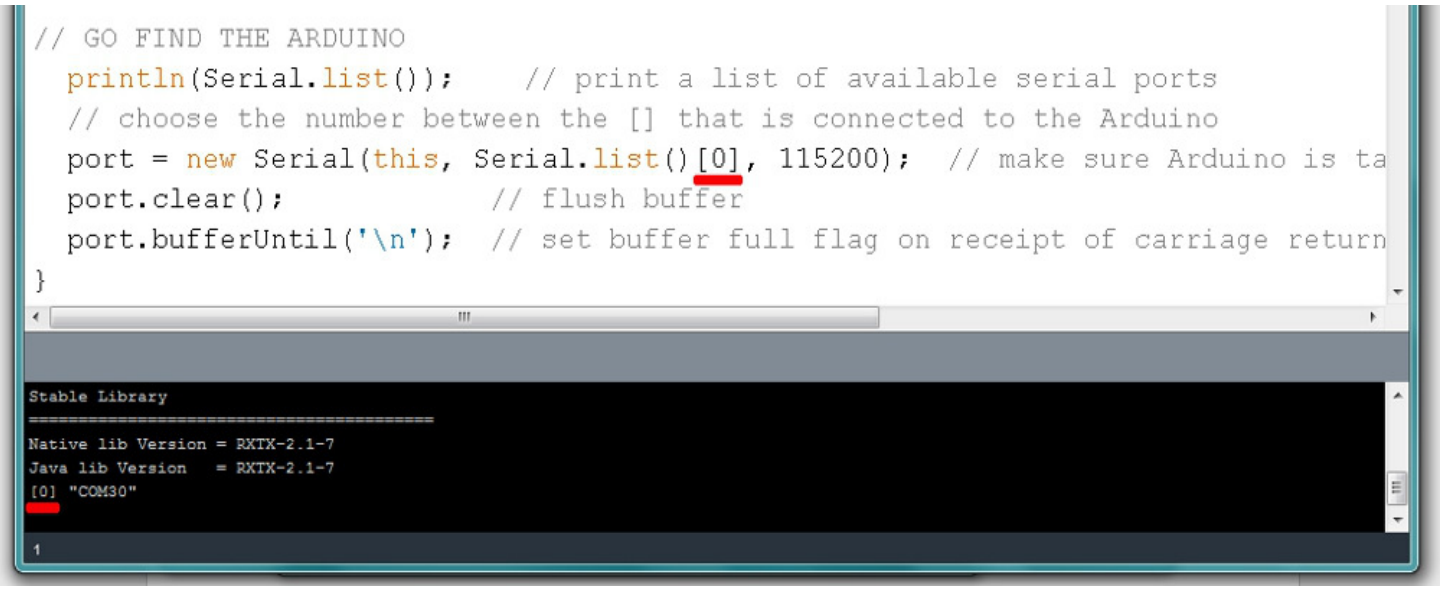
Take a good look at your work with a magnifying glass to be sure that you haven't made contact with any of the solder joints, then plug it in and test it. Hot glue is also great because it is easy to remove or re-work if you need to.

Troubleshooting:

Processing Sketch gives me a COM port error at startup.

Make sure you are plugged into an Arduino board, that it is working correctly, and running our firmware.

Check to see if you have the right serial port. The code underlined in red should match the correct port number in the terminal window at the bottom of Processing IDE.



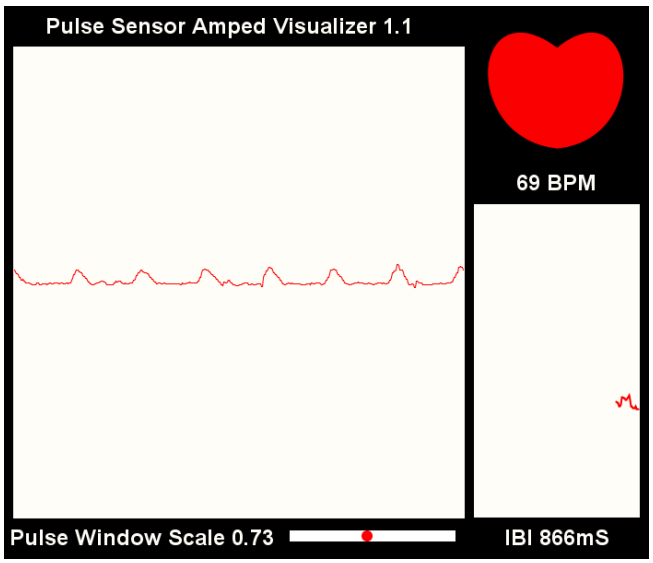
Processing gives an RXTX mismatch warning, then nothing happens

The RXTX mismatch problem can be resolved by making sure you are running the latest version of Processing and Arduino.

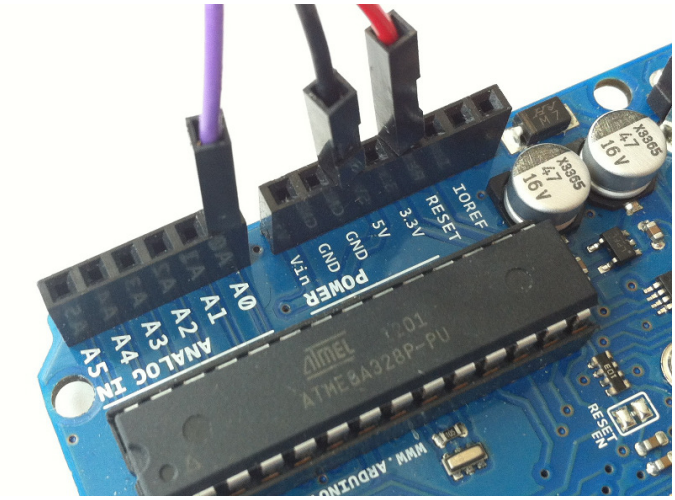
If you're still having trouble, go to http://processing.org/reference/libraries/serial/index.html for more info.

You are running the Pulse Sensor at 5V and only get a very small pulse waveform, even though you know you're not a zombie last time you checked.

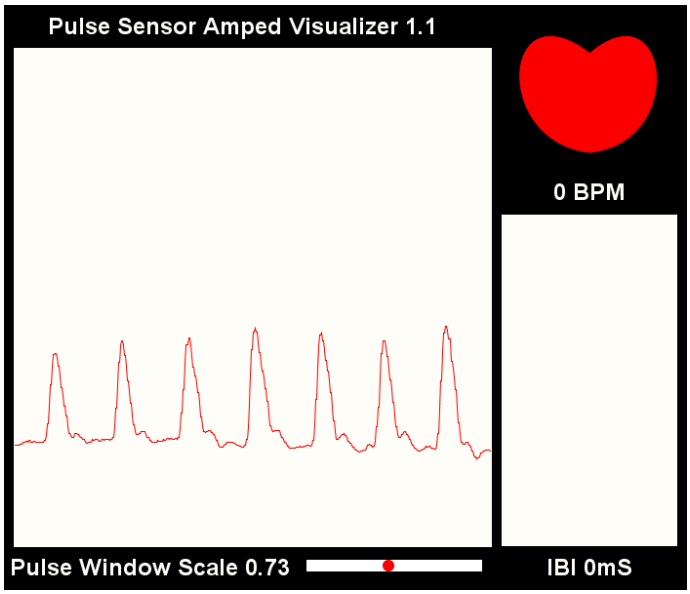
Make sure you have all your connections correct, and that you're not squeezing too hard on the Pulse Sensor. If you see a waveform that looks something like this,



then you may get better results by powering the Pulse Sensor with 3.3V. Here's a simple test and fix to get you on your way. First, unplug the RED wire from the 5V pin, and plug it into the 3.3V pin as shown here.



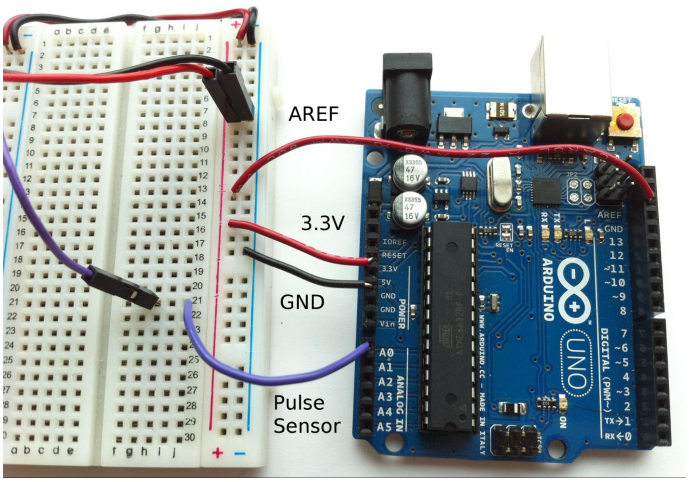
Then, run the processing sketch and see if you get a better wave shape.If you see a nice bright pulse waveform, then hooray! this is your fix! Notice, however that the pulse wave is in the lower portion of the pulse window. That's because of the difference in voltage between the power to the Pulse Sensor (3.3V) and the voltage reference to the Arduino (5V)



In order to make the Pulse Sensor work at 3.3V, we have to adjust either the hardware or the software to get the waveform up to the middle of the pulse window. Here's how to do that

Adjust the Hardware for 3.3V powered Pulse Sensor with Arduino running at 5V

The standard method for reading analog signals from a sensor that runs on 3.3V while the Arduino is running on 5V is to connect the AREF pin (near pin 13) to the lower voltage source. For this, you will need a breadboard, or some other way to split the 3.3V signal. Here's an image of the standard method.
You will also need to add or uncomment the line
analogReference(EXTERNAL);
in the void **setup**() routine Do Not Forget This!



Well, that was cumbersome, with the extra breadboard thing and all those wires. Isn't there an easier way to scale the input to work seamlessly with the rest of the Pulse Sensor code? Perhaps in the software, you ask? Well, yes there is.

Adjust the Software for 3.3V powered Pulse Sensor with Arduino running at 5V

The full range of the Arduino analog input is from 0 to 1023. The range of output from the Pulse Sensor running at 3.3V is only 0 to 675. We have to scale the Pulse Sensor signal so that it is sized for the larger expected range. Luckily, there's a handy-dandy function in the Arduino language called map(). Change the Pulse Sensor code in the Interrupt tab so that it looks like this:

```
// THIS IS THE TIMER 2 INTERRUPT SERVICE ROUTINE.
// Timer 2 makes sure that we take a reading every 2 miliseconds
ISR(TIMER2_COMPA_vect){                 // triggered when Timer2 counts to 124
  cli();                                // disable interrupts while we do this
  Signal = analogRead(pulsePin);        // read the Pulse Sensor
  Signal = map(Signal,0,675,0,1023);    // THIS WILL MAP A 3.3V SIGNAL TO 5V ANALOG RANGE
  sampleCounter += 2;                   // keep track of the time in mS with this variable
  int N = sampleCounter - lastBeatTime;  // monitor the time since the last beat to avoid noise
……..
```

And there you go. Now you're happy as a clam. A clam with a big thumping heartbeat!

The BPM values you are seeing seem to be way too high, or way too low.

You may need to change the interrupt settings in the Arduino code.

Our Pulse Sensor Amped Arduino code is based on the Arduino UNO, which uses the ATmega328, and has a system clock that runs at 16MHz. We set up Timer2, an 8 bit hardware timer, so that it throws an interrupt every other millisecond. That gives us a sample rate of 500Hz, and beat-to-beat timing resolution of 2mS. This will disable PWM output on pin 3 and 11. Also, it will disable the tone() command. This code works with Arduino UNO or Arduino PRO or Arduino Pro Mini 5V or any Arduino running with an ATmega328 and 16MHz clock.

```
void interruptSetup(){
  TCCR2A = 0x02;
  TCCR2B = 0x06;
  OCR2A = 0x7C;
  TIMSK2 = 0x02;
  sei();
}
```

The register settings above tell Timer2 to go into CTC mode, and to count up to 124 (0x7C) over and over and over again. A prescaler of 256 is used to get the timing right so that it takes 2 milliseconds to count to 124. An interrupt flag is set every time Timer2 reaches 124, and a special function called an Interrupt Service Routine (ISR) that we wrote is run at the very next possible moment, no matter what the rest of the program is doing. sei() ensures that global interrupts are enabled. Timing is important! If you are using a different Arduino or Arduino compatible device, you may eed to change this function.

If you are using a FIO or LillyPad Arduino or Arduino Pro Mini 3V or Arduino SimpleSnap or other Arduino that has ATmega168 or ATmega328 with 8MHz oscillator, change the line TCCR2B = 0x06 to TCCR2B = 0x05. That will cut the prescaler in half to 128.

If you are using Arduino Leonardo or Adafruit's Flora or Arduino Micro or other Arduino that has ATmega32u4 running at 16MHz, you need to replace the interruptSetup() with the following

```
void interruptSetup(){
    TCCR0A = 0x02;
    TCCR0B = 0x04;
    OCR0A = 0x7C;
    TIMSK0 = 0x02;
    sei();
}
```

Other devices that use a 32u4 and run under an 8MHz system clock, like the LilyPad Arduino USB, will need further correction to these settings. Change TCCR0B = 0x04; to TCCR0B = 0x03; Then change OCR0A = 0x7C; to OCR0A = 0xF9;

The only other thing you will need is the correct ISR vector that gets called. To do this change the line ISR(TIMER2_COMPA_vect){ to ISR(TIMER0_COMPA_vect)

Yes, we know this is tedious, and we're working on a Pulse Sensor Library that will handle all of these issues for you. ;]

You're trying to use the tone() command to make a sound when your heartbeats, and no sound comes out of your speaker even though you have it wired correctly.

We use the hardware Timer2 in our code, and so does the tone() command. That's a conflict. You will need to change the hardware timer that you're using. Here's how to use Timer2 instead of Timer1. In the Interrupt tab, comment out or remove the existing interruptSetup() routine, and paste in the following code.

```
void interruptSetup(){
  // Initializes Timer1 to throw an interrupt every 2mS.
  TCCR1A = 0x00; // DISABLE OUTPUTS AND PWM ON DIGITAL PINS 9 & 10
  TCCR1B = 0x11; // GO INTO 'PHASE AND FREQUENCY CORRECT' MODE, NO PRESCALER
  TCCR1C = 0x00; // DON'T FORCE COMPARE
  TIMSK1 = 0x01; // ENABLE OVERFLOW INTERRUPT (TOIE1)
  ICR1 = 16000;  // TRIGGER TIMER INTERRUPT EVERY 2mS
  sei();         // MAKE SURE GLOBAL INTERRUPTS ARE ENABLED
}
```

Next, change the name of the Interrupt Service Routine function from ISR(TIMER2_COMPA_vect) to ISR(TIMER1_OVF_vect). Note: this will disable PWM on pins 9 and 10, and also conflict with the Servo library. There's always trade-offs!

Now you're Arduino should should beep and boop to your heart's delight!