ENGR 40M Project 4: Electrocardiogram

Prelab due 24 hours before your section, August 14–15
Lab due 11:59pm, Saturday, August 19

1 Introduction

In this project, we will build an electrocardiogram (ECG or EKG). This is a noninvasive and painless test that measures the electrical activity of the heartbeat between pairs of electrodes placed at certain points on the skin. The heart is a relatively large piece of tissue, so the flow of electrical current associated with (and immediately preceding) contraction produces detectable voltages (typically a few millivolts) on the surface of the body, that oscillate at a frequency of around 1.3 Hz or around 80 beats per minute.

Because electrodes will be connected to you, while you're viewing your heartbeat, you can't connect your circuit to anything connected to wall power. In particular, this means **you must not use an oscilloscope to view your heartbeat**. Instead, a circuit will prepare the signal to be digitized by the Arduino's analog input and sent to a laptop computer for display.

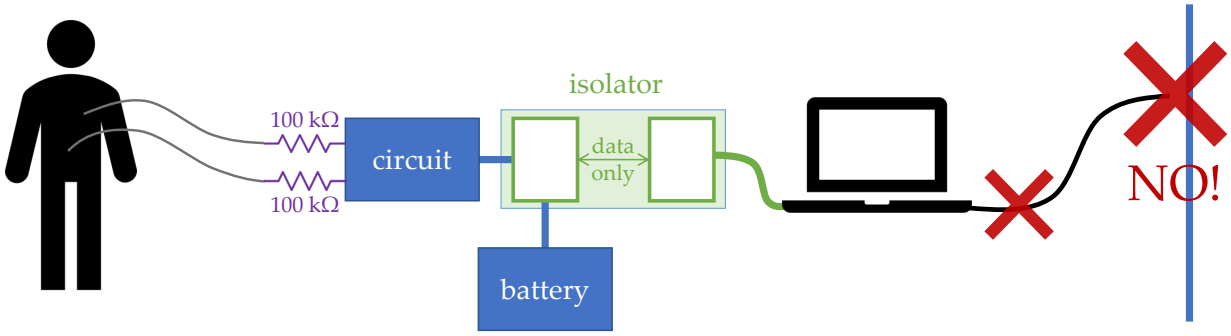
In practice, the signal from your heartbeat will be noisy and have a small DC offset, so this circuit needs to amplify the millivolt signal to something on the order of volts (since the Arduino's input range is 0–5 V), attenuate high-frequency noise and eliminate the DC offset. The objective of this lab is to design this circuit.

By completing this lab, you will:

- Enhance your skills in using the function generator and oscilloscope
- Gain experience debugging analog circuits
- Analyze filter circuits involving operational amplifiers
- Use an integrated circuit using information from its datasheet

2 Safety

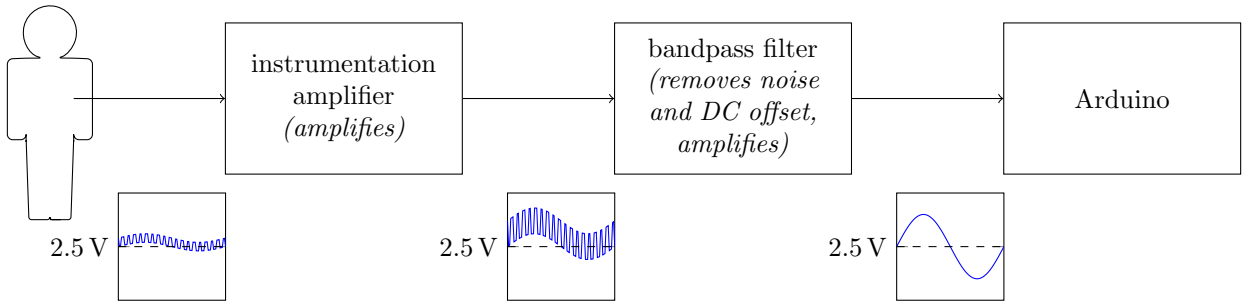
Since you will be connecting the ECG to yourself during this lab, we will stress safety and introduce each of the isolation schemes you will implement in this project. For this class we will use triple current limits, and double isolation to guarantee your safety while working with ECGs.



- Firstly, and most importantly, you should never be connected, through **any** devices, to the wall outlet. This includes a computer, function generator, or an oscilloscope that uses power from the wall. It follows that all computers should only run on batteries, and all other equipment should be disconnected when the ECG is connected to anyone's body. This, in turn, means that you must not use the oscilloscope (at all!) when you are going to measure your heartbeat.

- Secondly, we have a USB isolation board built using Analog Devices iCoupler technology. This board electrically isolates the Arduino circuit from the computer, so that you will not be directly connected to the laptop. Please always use the USB decoupler whenever you need to connect to the Arduino to a computer for this experiment.
- Finally, we will use current-limiting resistors to limit any potential current going through your body even if you are accidentally connected to a high voltage. Specifically, we place 100 kΩ resistors in series with each cable that connects to your body. Please ensure that these current-limiting resistors are implemented.

3 System description



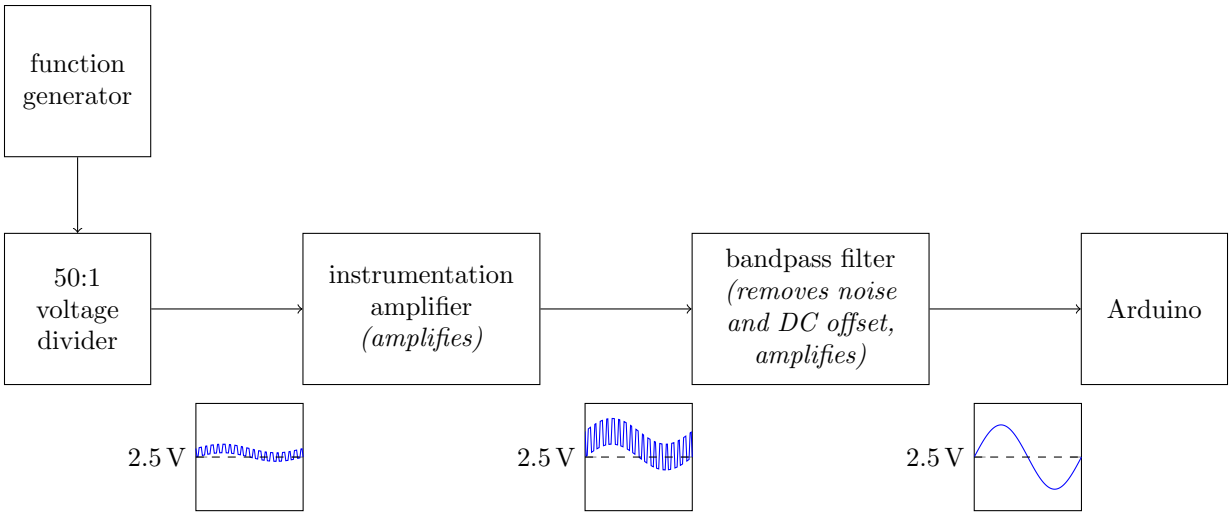
Our raw input signal—that is, your heartbeat's signal as measured via the electrodes—will be *small*, *noisy* and have a *DC offset*. The job of our circuit is to prepare the signal to be digitized by an Arduino analog input, which means combating all three of these properties.

- The **instrumentation amplifier** (IA) stage amplifies the signal from one on the order of millivolts, to an amplitude more amenable to our filter stage. It uses an *instrumentation amplifier*, which you analyzed in Problem 4 of Homework 6.
- The **bandpass filter** stage, which you analyzed in Problem 3 of Homework 6, removes noise and the DC offset. More precisely:
  - The **low-pass** part of the filter, which you considered in Problem 1 of Homework 6, removes noise, including the 60 Hz signal that couples in from wall power, as well as lots of other high-frequency noise that can enter our signal in many ways.
  - The **high-pass** part of the filter, which you considered in Problem 2 of Homework 6, removes the DC offset. (What "frequency" is DC?)

  The bandpass filter stage also amplifies the signal (or the part of the signal in the passband) a little more, so there is amplification in both the first amplifier and the bandpass filter stages.

The output of the bandpass filter goes directly to the Arduino analog input.

While we're building and testing the circuit, we won't use your heartbeat as the input signal. Instead, we will use the *function generator* on your lab bench to simulate a "heartbeat" (well, a sine wave). we'd like it to generate a signal that is about 0.4 mVpp, but sadly the generator can only go down to 20 mVpp. We'll therefore use a voltage divider on the function generator's output to produce a 0.4 mVpp input signal. So during building and testing, our system diagram will look more like this:



4 Prelab

P1: Before anything else, because of safety concerns, we want to use the solar-powered charger as your your power supply to your Arduino. Verify that yours still works. If it doesn't, please leave it in the sun so it has lots of charge in it.

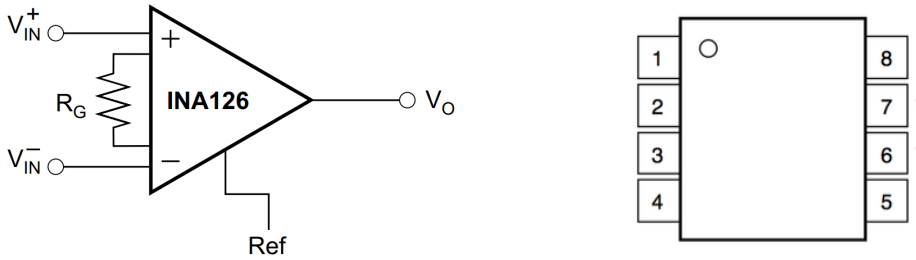
(If you happen to have an external battery pack with a USB port on it, this will also work.)

4.1 Instrumentation amplifier (INA126P)

This amplifier provides most of the gain to the small heartbeat signal that comes from the electrodes. You studied how the IA works in Problem 3 of Homework 5; the last thing we need to do is take note of its pinout (which pins correspond to which terminals).

P2: Using the datasheet on the class website, label the corresponding pins in the figure below, and on the schematic symbol of the instrumentation amplifier on its right.

Note, that confusingly, V+ and V− on the datasheet refer to the power supply terminals of the instrumentation amplifier, which (like for an op-amp) are not typically explicitly drawn on schematics.



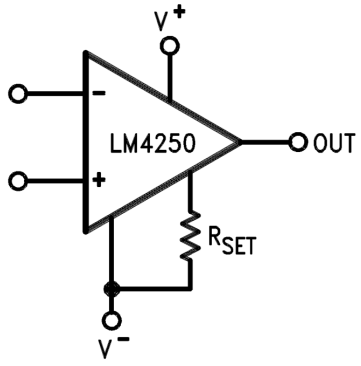
We will connect the 'Ref' pin to 2.5 V, so that the output will be halfway between ground and 5 V. The voltage between the electrodes connected to your body will contain your heart's signal, but will often also contain a DC offset. You studied the consequences of this phenomenon in your homework.

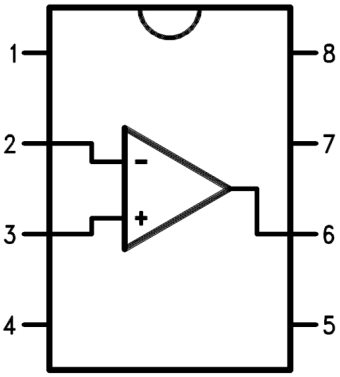
4.2 Operational amplifier (LM4250CN)

We use the op-amp to implement our band-pass filter, which removes the DC offset and some of the 60 Hz noise, as well as providing additional amplification in the passband. Again, to allow this op-amp to remain between 0 V and 5 V, we connect the op-amp so that its output will be centered at 2.5 V.

P3: Using the datasheet on the class website, label the corresponding pins in the figure below, and on the schematic symbol of the operational amplifier on its right.

V^+ and V^- in the datasheet refer to the power supply terminals. The resistor R_{SET} is a resistor that sets the *quiescent current* of the op-amp. We'll use a 1 MΩ resistor here.[a]





[a] The quiescent current setting is a parameter that allows a circuit designer to choose how to trade off performance (*i.e.*, closeness to ideal) against power consumption.

4.3 Building the 2.5 V reference and IA circuit

You will need:

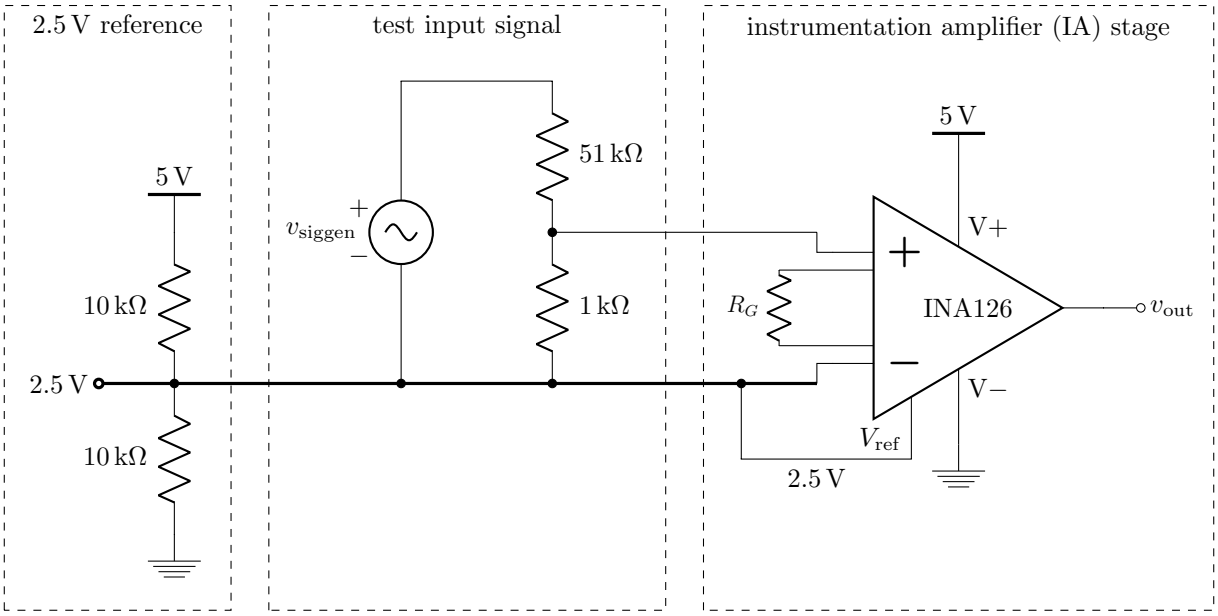
- Two 10 kΩ resistors
- 51 kΩ and 1 kΩ resistors
- The R_G resistor you found in Problem 3 of Homework 5
- Instrumentation amplifier (INA126P)

We ask you to build the instrumentation amplifier (IA) part of the circuit as part of the prelab, to help save time during the lab. A schematic of the 2.5 V reference, test input signal generator and the IA stage is shown below. Some notes:

- The 2.5 V reference is used by many stages in the circuit, including the op-amp filter stage that you will add in lab.
- The test input signal generator comprises both the function generator, represented by the source v_{siggen}, and the 51:1 voltage divider. Sadly, the function generator can only generate signals as small as 20 mVpp, whereas your heartbeat signal will be less than a millivolt peak-to-peak, so we use a voltage divider to produce a much smaller input signal than the function generator can manage.

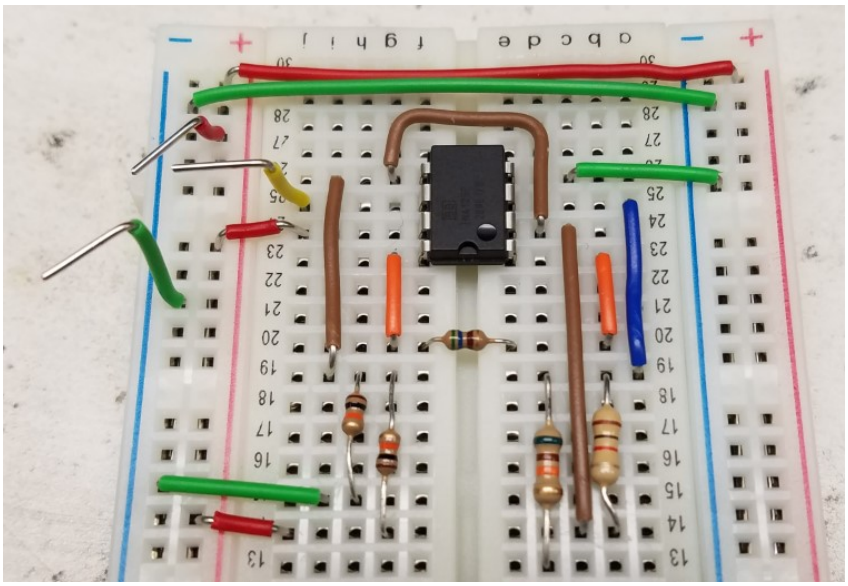
  You should just have loose wires available for the function generator—that is, short wires with one end in the breadboard, and with the other end sticking out for the function generator to clip to.
- You might need to refer to pinout for the IA you found above to get the right pins for the IA stage. Take note of the orientation of the IA—the little notch indicates the top of the chip in the pinout diagram above.

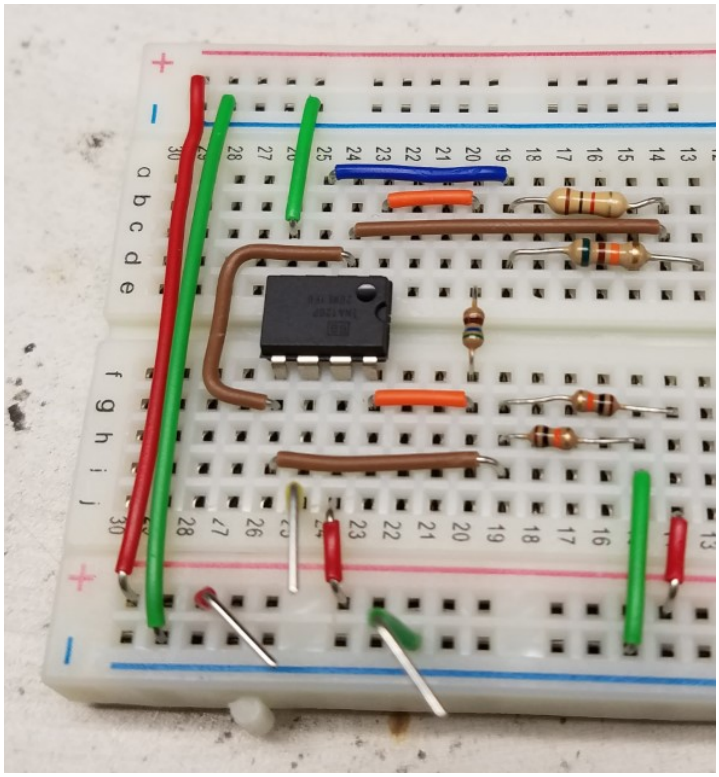
  You should use the value of R_G you found in Problem 3 of Homework 5, or something close to it.



P4: Build the above schematic on your breadboard. An example picture of the circuit is shown below, which you're welcome (but not obliged) to follow.

Please attach a photo of your completed breadboard containing the above three stages, and please label on your picture what each component of the breadboard is, and where you intend for the following five external connections to connect to your breadboard circuit: V_{DD}, ground, both sides of the function generator v_{siggen}, and the output of the IA stage.





4.4 Reading and viewing a signal

To visualize the analog waveform on your computer, we will use a software called Processing. Processing is a simple programming environment very similar to the Arduino programming environment.

We'll first program the Arduino to send values to the computer via the serial connection; this will look very much like the "ReadAnalogVoltage" example. Then, we'll slightly modify an example Processing code to plot a sequence of numbers on the screen.

For now, our Arduino will naturally be sending nonsense data to Processing, because you haven't built a circuit yet. But that's okay, because we're just looking to get the communication link between the Arduino and Processing going.

P5: Go to www.processing.org and download the software. Open Processing. The interface should look very similar to the Arduino development interface.

P6: Download `ecg.ino` from the class website. This is the simple code we need to run on the Arduino. Make sure you understand how it works.

Set the input pin to the analog pin you intend to use to read the output of the ECG. Any of pins A0–A5 will do.

(This might seem odd that you're not connecting anything to the input pin in question, but it's okay for now that the Arduino's sending nonsense data, as noted above.)

```
void setup() {
  // initialize the serial communication:
  Serial.begin(9600);
}
```

```
    void loop() {
      // send the value of the analog input
      Serial.println(analogRead(inputPin));
      // wait a bit for the analog-to-digital converter to
      // stabilize after the last reading:
      delay(2);
    }
```

P7: Download `ecg.pde` from the class website. This is the Processing code that will display the signal on your computer.

The first few lines of `ecg.pde` set up the variables that the program will use to run. You will need to change `SERIAL_PORT` to the right number for your Arduino. If set to `-1`, it will print out the list of ports, which makes it easy to find the right port. The ports are printed in the status bar under the main screen and look like `/dev/tty.usbmodemfd1331` and `/dev/cu.usbmodemfd1331`. You should connect to the `tty` port that corresponds to your Arduino. If you're not sure which port that is, try unplugging your USB cable to see which port disappears. It is often the first port (port 0), but you can run it with -1 first to double check.

Like the Arduino code, Processing also has a setup routine that runs once to setup the rest of the code. In addition to setting up the serial link, it also created the graphics window with the `size(x,y)` command. This command creates a graphics window that is `x` pixels wide and `y` pixels tall. You should change these numbers to make sure the window fits on your computer screen.

```
// TODO replace this with the appropriate serial port index.
// You can run this code with SERIAL_PORT = -1 to get a
// list of available serial ports.
static final int SERIAL_PORT = -1;

Serial myPort;        // The serial port
int xPos = 1;         // Horizontal position of the graph
float heightOld = 0;  // The old vertical position

void setup () {
  // Set the window size
  // TODO: Adjust this to fit your computer screen
  size(1200, 800);

  if (SERIAL_PORT == -1) {
    println("Please␣set␣SERIAL_PORT␣to␣an␣appropriate␣value.");
```

The main draw loop is shown below. Since we are drawing lines that are one pixel wide, we can erase the previous line (written on the previous scan line by simply drawing a vertical black line. This gives a screen display that updates each position as it passes it instead of clearing the screen at the end of the scan.

The function `stroke(r, g, b)` sets the color of the line; `stroke(c)` is equivalent to `stroke(c, c, c)`. The function `line(x0, y0, x1, y1)` draws a line from `x0, y0`, to `x1, y1`. The only tricky part is that in processing 0,0 is the top left part of the screen, so you need to plot height-y to get a graph that looks correct.

So the first line call draws a black line to erase the previous drawing, and then it draws the next segment. If you would prefer a display that blanks when the scan is through, you should comment out the code that draws this black line, and uncomment the command that sets the background to black when the scan line finishes.

```
    float outHeight = map(output, 0, 1024, 0, height);
```

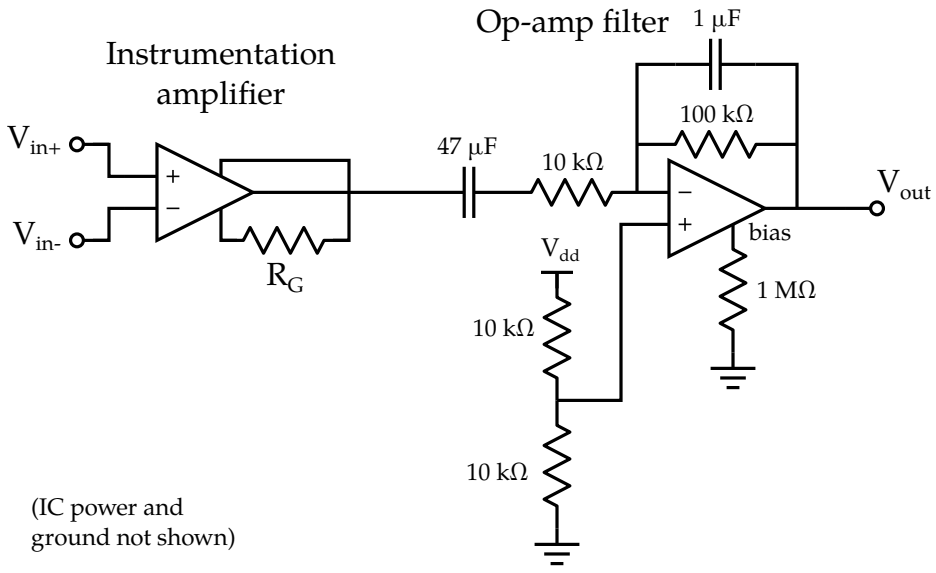
```
  //erase the next position to be written
  stroke(0);
  line(xPos, height, xPos, 0);

  // Draw the line - the numbers specify the RGB color, pick one you like.
  // Remember that processing draws things upside down.

  stroke(255, 255, 255);
  line(xPos-1, height-1 -heightOld, xPos, height-1 -outHeight);
  heightOld = outHeight;


  // If at the edge of the screen, go back to the beginning
  if (xPos >= width) {
    xPos = 1;      // Not zero, since I the line starts at x-1
    // background(0);  // Clear the screen for the next scan
  }
  else {
    xPos++;
  }
```

P8: In `ecg.pde`, modify `SERIAL_PORT` to be the port corresponding to your Arduino, and change the dimensions in the `size()` call to suit your computer screen. Verify that, when the Arduino is connected, the Processing code continually draws a waveform across the screen.

If your Processing code looks good, we're ready to hit lab!

5 Lab procedure

Below is the full circuit schematic. There are two main parts, the IA circuit that you built in the prelab and the bandpass filter that you will build now.



5.1 Testing the instrumentation amplifier

First off we will test the instrumentation amplifier stage that you built in the prelab.

Connect a 5 V power supply to your circuit (either your Arduino or the power supply on your lab bench). Connect the function generator (red to the side of v_{siggen} labelled +), and set it to 20 mV at 1 Hz. Double-check that the reference of your IA is connected to your 2.5 V reference.

Then, work through your circuit:

- Check that your 2.5 V reference works.
- Check that the function generator does what you expect.
- Check that the divided 0.4 mVpp signal looks as you expect.
- Check that the output of the IA looks like you think it should.

  Don't continue before you're satisfied with the amplitude of the IA signal! Remember: We build step-by-step so we can iron out bugs as we go. Experienced engineers don't magically avoid making mistakes, they just learn to catch them quickly!

L1: Take a picture of the output on the oscilloscope *when the function generator's amplitude is* 20 mV and include it in your lab report. How is this compared to what you expected? Is your waveform clipped?

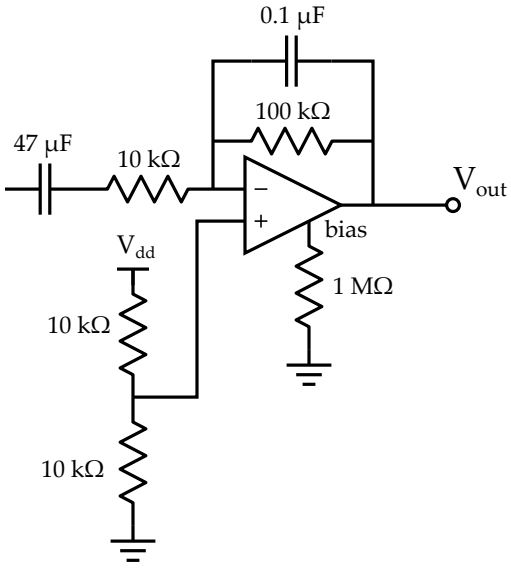
L2: Measure the output of your instrumentation amplifier (again referenced to 2.5 V) and compute for the gain of your circuit *when the function generator's amplitude is* 100 mV.

5.2 Building the bandpass filter

You will need:

- operational amplifier (LM4250CN)
- 47 µF and 0.1 µF capacitors
- 100 kΩ and 10 kΩ resistors
- 1 MΩ bias resistor

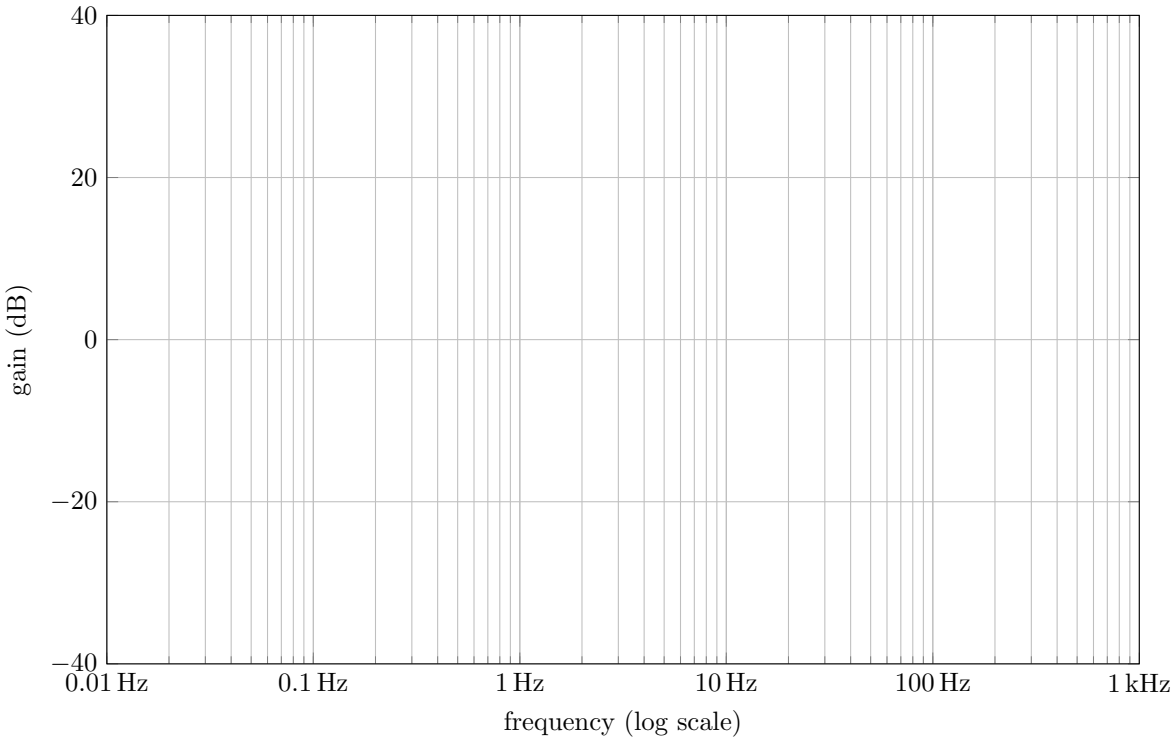
Build the op-amp circuit shown in the schematic. **Make sure to also connect the supply pins on the op-amp to 5 V and GND.** The 1 MΩ resistor connects to the quiescent current set pin.



As analyzed in Homework 5, this circuit is a bandpass filter. Using a test signal of around 500 mV, we will test the amplitude of the output at input frequencies ranging from 0.1 Hz to 80 Hz. Since we are using a 400 mV input, we don't need the attenuator that we built (but leave it on your board since we will need it later). Instead connect the signal output of the signal generator directly to the 47 µF capacitor. Remember to leave the common (black) lead connected to your 2.5 V reference.

L3: Using a test signal of 400 mV, find the amplitude of the output at 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, and 50 Hz. (These are roughly equally spaced on a logarithmic scale.) Use these data points to draw the Bode magnitude plot of your filter.

What is the approximate upper corner frequency of this bandpass filter?



L4: Put in a test signal at 400 mV, 1 Hz and a 1 V DC offset *(Note: This offset should be added to the signal generator, as opposed to in the circuit - keep the black lead at* 2.5 V*.)*. Take a picture of this. What happens to the offset at the output? Are these last two results consistent with the bode plot analysis from the prelab? *Hint: What is the frequency of a DC signal? What should your circuit do to signals at this frequency?*

5.3 Hooking up both amplifier circuits

Now, test both circuits by inputting a 1 Hz, 0.4 mVpp signal into the voltage divider before the IA. Attach the output of the IA to the bandpass filter input, and measure the output of the op amp.

L5: Take a picture of your final simulated "heartbeat" output and verify that both stages work.

5.4 Testing your viewing code

Now we will program the Arduino to take in this 0–5 V signal and output it on your computer screens using a program called Processing. By using an isolator board, we can safely hook ourselves up using actual electrodes and be able to view the output on our computer screens.

Place your Arduino on the proto-board that contains your circuit. Connect your circuit's 5 V and GND connections to the Arduino, so the Arduino powers your circuit (if you have not done this already), and then connect your op-amp output to the analog input you chose in your Arduino code, with your input still coming from the function generator. In your Arduino code, if you leave the delay to be every 2 ms, and your screen is greater the 1000 pixels wide, you will get a plot that is about two seconds of data. You should see several cycles of a sinewave centered on your screen. Leave your front-end circuit connected to the Arduino.

L6: Take a screenshot of the output on your screen.

5.5 Testing the isolation board

Now that you have everything working, your next step is to connect the USB isolator between your Arduino and your laptop computer. These isolation boards should have three USB connectors that are labeled on the board. It shouldn't be possible to connect them up backwards, but please double check just to be sure. Remember that since the Arduino is isolated you will need to power it from a different power supply. That is where the solar chargers come in.

Once you have connected your Arduino to your laptop through the isolator boards, and connected your solar chargers to supply the power to the Arduino, please repeat the previous experiment, and make sure you are still getting the output waveform you got before. If this doesn't work, you should ask your TA to help you.

5.6 Viewing your heartbeat

Now, you are ready to view your heartbeat!

You will need:

- The circuit you just got working above, with the isolation board and solar charger
- 3 100 kΩ resistors
- 3 foam electrodes to connect to your body.

1. Disconnect the function generator from your circuit, and remove the voltage divider from the input of the instrumentation amp. Make sure there is no lab equipment connected to your circuit (no power supply, lab DMM, oscilloscope, etc.), and turn all machines on the bench off for safety. Your breadboard & Arduino should be powered by your solar charger battery and should only be connected to your computer through the USB isolator board. **The computer should not be connected to the wall (unplug your laptop)**. Once you have checked all these steps, you may proceed.

2. Without connecting anything to yourself, use your handheld DMM to check voltages on important nodes. You should have 5 V at V_{DD}, 0 at GND and 2.5 V at the IA's V_{ref}.

3. Run your Processing code to display the output from your circuit (which is not connected to anything right now). You should see a flat line somewhere in the middle of the window. Don't be surprised if the value moves around a little bit. There is noise in the circuit.

4. Add a 100 kΩ resistor at each input pin of the IA. Also add a resistor at Vref. These resistors will be in series with the leads connected to your body and will serve as current limiting resistors.

5. Now, connect test leads to yourself. Using banana cables and alligator clips, attach two electrodes to your left and right inner wrists and use these as the inputs to your IA (through the 100 kΩ resistors). Be careful not to short anything. Attach the third electrode somewhere on your torso or leg and use this as the input to V_{ref} (also through the 100 kΩ resistor). Its position is not critical.

   At this point you should see a signal that is about 1 Hz that is your heartbeat. If you want to see a more impressive signal, you can put the two electrodes closer to your heart.

   > **L7:** Take a screenshot of the output on your screen.

6. **Optional:** You may notice that there is still a lot of noise. If you have time, you can implement a simple average filter in your Processing code. The basic idea is to use a sliding window and plot the average of n values instead of just the nth value. Show screen capture of a less noisy plot and your processing code.

   > **L8:** (*Optional.*) Take a screenshot of your the now less noisy output on your screen.

6 Analysis

> **A1:** We can still see some noise at the output of this circuit. Suppose we wanted to try to reduce this noise by changing the filter stages. Could we reduce this noise further? Which components would we need to add or change? Since the heart beat is not a sine wave (it has a spike in it), do you think there might be a downside of doing this?

7 Reflection

Individually, answer the questions below. Two to four sentences for each question is sufficient. Answers that demonstrate little thought or effort will not receive credit!

R1: What was the most valuable thing you learned, and why?

R2: What skills or concepts are you still struggling with? What will you do to learn or practice these concepts?

R3: If this lab took longer than the regular 3-hour allotment: what part of the lab took you the most time? What could you do in the future to improve this?

8 Build Quality rubric

Your build quality grade in this project is based on the quality of your breadboarding and testing setup. You should make use of the oscilloscope probe clip-tips and BNC minigrabbers, and avoid stringing together jumper wires and alligator clips.

Plus

- Testing set-up is stable and consistent
- Breadboard layout is clean and organized
- Set-up is very stable with all components fitting comfortably into the breadboard
- Wires are color-coded and easy to trace
- Wire lengths are about right
- All power supply points are cleanly connected

Check

- Testing set-up is mostly stable, although some components or cables could be connected better
- Breadboard layout is organized, but could have been improved with some more careful planning
- Wires are mostly color-coded, and can be traced with minimal difficulty
- Wires lengths are mostly right
- Most power supply points are cleanly connected

Minus

- Testing set-up is not very stable, and several components are sub-optimally placed
- Breadboard layout doesn't follow a consistent pattern
- Lack of color-coding makes wires difficult to trace
- Wires lengths are off and lead to spaghetti wiring, which may cause shorting
- Many power supply points are not very cleanly connected