Pulse Oximeter

In this lab we will build a pulse oximeter. The pulse oximeter is commonly used in hospitals. A small clamp is attached to the patient's finger and the device measures the pulse and the oxygenation of the blood. The device is often attached to a monitor so that if the patient stops breathing and the oxygen level starts to drop, an alarm goes off. You can read more about the pulse oximeter on the fountain of all knowledge, Wikipedia. The pulse oximeter has been around for at least 70 years.

The device works by shining light through the finger (or sometimes the earlobe). Two different wavelengths of light are used, red light (our LEDs have a peak emission at 624 nm) and infrared (our LEDs have a peak emission at 940 nm). The absorption of light at these wavelengths is vastly different between the oxygenated form of hemoglobin and the deoxygenated form. Therefore, by looking at the ratio of how much light passes through the finger at these two wavelengths you can get a measure of the oxygen content of the blood. You can also see the patients pulse as well.

We will build a simplistic version of this device. Our device is un-calibrated, therefore we can only see trends and our device cannot make any statement about how well oxygenated you are. However, as we will show later in this lab, you can clearly see the pulse as well as a general trend over a long time scale as you hold your breath. This experiment has no medical value and nothing should be interpreted from the data.

New components

The two new parts that we will use are LEDs (light emitting diodes) and photodiodes. The devices are the duals of each other. An LED converts an electrical current into light. A photodiode converts photons hitting the device into an electrical current. We will use two LEDs that emit light at different wavelengths and two photodiodes directly across the finger. We could actually use a single photodiode, but we found a better signal if the emitter and detector were pointed directly at each other. Wikipedia has a rather extensive overview of LEDs and photodiodes.

Synchronous detection – done in software

One of the new features that we will use is synchronous detection. This scheme can be implemented in hardware for real-time measurements, but we will implement it in software after the data is collected. The main idea will be explained and demonstrated in class. Briefly, the problem is that we need to extract the amount of light passing through the finger from the LEDs and need to distinguish that light from the ambient light. The photodiode simply provides a current, we cannot tell from the current directly what source the photon originated from. In practice, we don't want someone to turn off the lights and then we think our patient has stopped breathing. So a system that rejects the ambient light and only "detects" the light from the LED is needed.

The trick we use is that we modulate the amplitude of the LED with a sinusoidal source at a known frequency. This frequency is much higher than the ~1 Hz pulse that we are trying to detect. We then take the measured signal from the photodiode and in software after we have measured the signal we multiply the data by a sinusoidal function of the exact same frequency. We then take the average of the signal over time. Any constant multiplied by a sine and integrated (averaged) is zero, so this method will reject any constant ambient light. It will also reject any signals at other frequencies. The property of sines and cosines that we are using are,

$\int_0^1 \sin(2\,\pi\, nt)\sin(2\pi mt)\,dt = 0$ if n and m are integers that are not equal.

$\int_0^1 \sin(2\,\pi\, nt)\sin(2\pi nt)\,dt = 1/2$ if n is any integer.

$\int_0^1 \sin(2\,\pi\, nt)\cos(2\pi mt)\,dt = 0$ if n and m are any integers.

Averaging the product of sines only yields a non-zero result if the frequencies match. So by multiplying the experimental data by a sine of the known frequency of modulation, we can extract only the part of the signal of interest.

We will discuss this demonstration in class, but below is a simulation of this effect in MATLAB. You can copy the following script into MATLAB and try it for yourself. Basically, the variable "sig" is our made up measured signal which includes a constant, a signal at our driving of 500 Hz that represents the driven LED and some 60 Hz noise. Since in general don't know the phase (here it is set to 45 degrees) we multiply our measured signal by a sine and cosine of the known frequency. After integrating, taking the square root of the squares will provide the amplitude of the signal at the known frequency. You can play with this script and see that it returns the amplitude of the 500 Hz sinusoid no matter what the rest of the signal is. If you add random noise, then you will get some error from the actual amplitude.

```
N = 10000;
t = linspace(0,1,N);
sig = 1 + 0.03*sin(2*pi*500*t-pi/4)+0.15*sin(2*pi*60*t);  %%% made up data

y1 = sin(2*pi*500*t);  %%% sine at known freq of 500 Hz
y2 = cos(2*pi*500*t);  %%% cosine at known freq of 500 Hz

A = trapz(t,sig.*y1)*2;
B = trapz(t,sig.*y2)*2;
Amplitude = sqrt(A.^2+B.^2)
```

The main idea is that if our measured signal from the photodiode has a small trace of the frequency component from the driven LED buried with a lot of other noise, we can extract the piece of the signal we want.

Software low-pass filter

In practice in our lab, we will use a low-pass filter which provides the same effect as integration in the above example. The low pass filter provides a continuous trace of the data on a "slow" time scale. We want to see changes on the time scale of your pulse and over a period of time where you hold your breath. We do not want to average over the entire experiment. We essentially want a running average to smooth out the high frequency 500 Hz driving of the LEDs. If our implementation of the detection above was in hardware, we could use a simple RC circuit as we have always done. Instead, we will implement the low pass filter in software, after the experiment is run.

The method to apply a software low pass filter on the experimental data is easy to derive from standard RC low-pass analog filter that we have used. For the standard RC circuit model, the output voltage of the filter is related to the input voltage through the following differential equation (which you should be able to derive if it seems unfamiliar):

$$RC\frac{dV_{out}}{dt} = (V_{in} - V_{out})$$

We approximate the left side of the derivative as

$$RC\frac{V_{out}(t+dt) - V_{out}(t)}{dt} = (V_{in}(t) - V_{out}(t)).$$

Where dt would be the time between measured samples. This equation can be rearranged to read,

$$V_{out}(t+dt) = \left(1 - \frac{dt}{RC}\right)V_{out}(t) + \frac{dt}{RC}\,V_{in}(t).$$

The equation simply relates the filtered value of the voltage, Vout, to the previous value of the filtered voltage and the current measurement, Vin.

This software filter is easy to implement in MATLAB with a simple for loop as follows,

```
%%% data_lp is the vector of low pass filtered data
%%% data is the original vector of data
%%% alpha is the value of dt/RC

data_lp = zeros(size(data));  %% initialize the vector for the LP filter
for i = 2:length(data)
    data_lp(i) = data_lp(i-1)*(1-alpha) + alpha*data(i);
end
```

Driving the LEDs

In our lab we will drive the LEDs at different known frequencies using the sound card on your laptop. You have stereo output so we can output two channels at two different frequencies. We use the sound card because the analog output on the DAQ cannot run at very high frequencies. We will monitor the photodiodes with the DAQ.

To send a signal out to the sound card, the following matlab script will do. This script will send a sine wave at the standard audio rate of 44100 samples per second to the two stereo channels. The frequency of the signals should be changed later in the experiment so the two channels drive two LEDs at different frequencies. We will use a standard audio cable plugged into the headphone jack to route the signal to your protoboard.

```
Fs = 44100;  %%% sample rate for the output
t = linspace(0,duration,Fs*duration )';
y1 = sin(2*pi*500*t);
y2 = sin(2*pi*500*t);
soundsc([y1 y2 ],Fs);
```

Procedure

We will have you build the system in stages so that you can test each piece as you go along.

Before starting, realize a little planning will make your final system neater and easier in the end. You will need 4 op-amps and thus can use one quad chip. We recommend putting the op-amp in the center of your breadboard, place your audio cables output to the right of the op-amp and your DAQ input to the left. Build the audio amplifier on the right side of the op-amp close to the audio input and the detector on the left. This will keep things relatively neat. You can look at the layout of the demo circuit that we have built.

First build an amplifier for 1 channel of the audio output. **Turn up the volume on your computer and unmute it after you plug in the audio cable.** As a first test of the software you can download the matlab starter script for this lab and directly hook the audio output to the DAQ analog input. You should get a sine wave at about ± 1.5 V and 500 Hz. We want to maximize the brightness of the LED so we need to amplify this signal by 5/3 (since the voltage difference of ± 1.5 V is 3 V and the maximum op-amp output is 5 V). We also want our sine wave centered at 2.5 V since our op-amp is single sided. Build the circuit shown in Figure 1. Send the output of the circuit to your DAQ analog input and confirm that you measure a sine wave of about 0 to 5 V. In your lab report, work out the equation that relates the output of this circuit to the input. Show a single plot confirming this stage of the circuit.

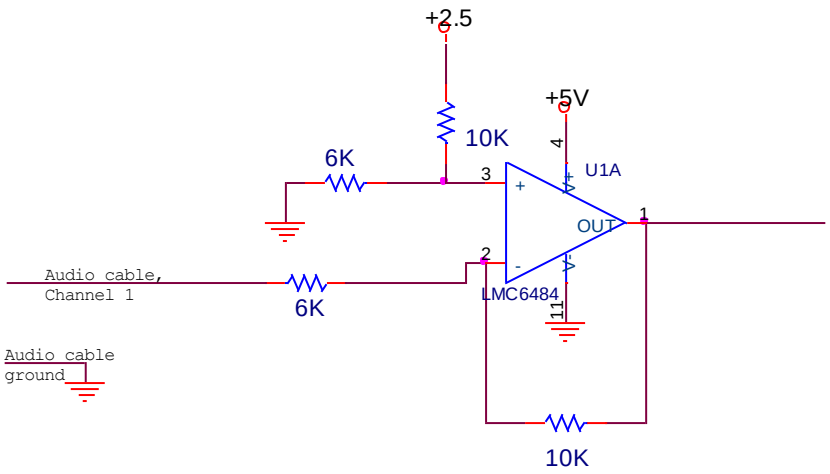

Figure 1: Audio amplifier circuit.

Now add a 60 ohm resistor and a LED in series as shown in Figure 2. Measure the voltage difference across the 60 ohm resistor using the DAQs analog input in differential mode. By measuring the voltage across the resistor you now know the current flowing. You should find that there is about 20 mA flowing through the LED. If you use the red LED and reduce the driving frequency to 5 Hz, you can confirm that it is working as your eye will see the LED blink. You don't need to include any plots for this part, just confirm the current through the LED. Note that the LED is directional. The short lead goes to ground. If you are using one of the soldered LED strips, we will denote which color should go to ground (either black or blue). The direction for the red LED is easy as you will see if it is on or not. To test the IR LED, you can use a cell phone camera which is sensitive to IR.

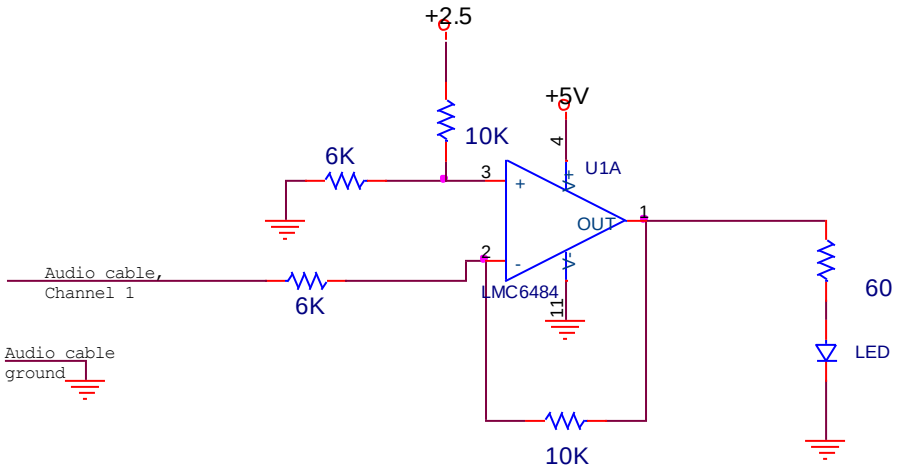

Figure 2: Audio amplifier driving the LED.

Build one detector circuit as shown in Figure 3. This circuit is very simple. The photodiode creates a current related to how much light it receives. Since the input impedance of the op-amp is so large, this current must also flow through the 1M resistor. Thus the output voltage of the op-amp is 1,000,000 times the current generated by the photodiode. Move the input to the DAQ from testing the LED current

to reading the output of the detector circuit. You should send the output of the op-amp to AI0+ and AI0- should go to ground. You can test the circuit by pointing the photodiode up at ceiling lights, covering it with your hand, and aiming it at LED which you are driving at a known frequency. If you see the voltage saturate at 5V, it means you reached the limit of the op-amp. If you see nothing, maybe your photodiode is in backwards.

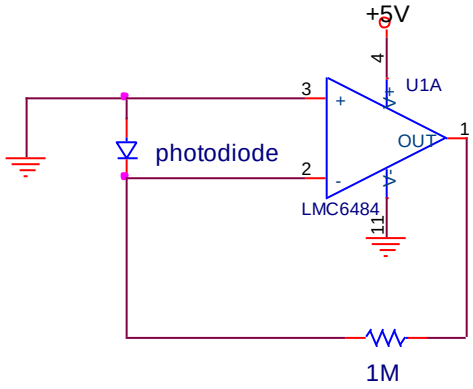

Figure 3: Detector circuit.

Now we will need to modify your code to do the detection at the known frequency. In the code we supply a blank function called detection. You will need to write this function, though we give you the essential pieces below. The inputs to the function are the measured data vector, the time vector, and the known frequency that your LED was blinking at (500 Hz). Your detection function should take the signal you measure from AI0, and multiply (using the MATLAB .* notation, it is element by element vector multiplication) by sine and cosine of same frequency. Now, take these two vectors of data and low pass filter them twice. Two passes through the filter seemed to work well for us and provide better data than a single more aggressive filter. You should fill out the blank low_pass function in your MATLAB template program, though we give you the essential code above. For the values we used, we found that alpha = dt/RC = 0.005 works well with DAQ sample frequency of 22050. What is the effective time constant of this filter (answer this in your lab report)?

```
%%% data and time are vectors of time and data from AIO
%%% f is the known frequency.
data_c = data.*cos(2*pi*f*time);  %% data times the cosine
data_s = data.*sin(2*pi*f*time);  %% data times the sine
data_c = low_pass(data_c,0.005);  %% low_pass is a function you write
data_c = low_pass(data_c,0.005);
data_s = low_pass(data_s,0.005);
data_s = low_pass(data_s,0.005);
data = sqrt(data_c.^2+data_s.^2);
```

Once you have the software written, you should test your program. While running the program, aim the photodiode at the room lights and at the LED. The raw data from the LED should make sense to you based on where you pointed the photodiode. The output from the detection function should only show an increase from the blinking LED. See Figure 4 for an example.

At this point, you can get most of the pulse oximeter working with only a single LED. If you are running out of time this week, you can obtain your results for the pulse oximeter with the single red LED. This will be able to measure your pulse and should show a change as you do the breath test (see below). The full oximeter relies on the ratio of the red to IR light.

If you are doing good on time, build second amplified on other side the op-amp (again, see our setup) for the infrared LED. You should test everything and build everything exactly as you did for the other LED driver circuit. Change the software to drive the two LEDs at different frequencies. We used 500 Hz and 327, which was an arbitrary choice. We choose 500 Hz as we want to average many cycles over the time scale of a few tenths of a second and we want to be higher in frequency than the ambient noise (120 Hz) of the room lights. We chose 327 for the IR LED such that it wasn't a multiple or too close to 500. Other values would work fine and you are free to explore this if you want. What you want to avoid is using a frequencies which are simple integer multiples (harmonics) of each other. Now collecting data from the photodiode with the DAQ, run the program and move the photodiode alternately over the IR and red LED. Modify your program such that your detection function runs on the two frequencies, i.e.

```
Data_500 = detection(data,time,500)
Data_327 = detection(data,time,327)
```

Plot the filtered data on the same plot. You should see that the raw signal spikes when the photodiode is over either LED or aimed at some bright ambient light. For example see Figure 4. The upper figure shows the raw signal and the lower one is the output of the detection function. The annotation on the raw data shows when the photodiode was pointed up at the lights, when a desk lamp was turned on and when the photodiode was pointed at the two LEDs. The detection signal clearly rejects the ambient light and the detection at the two frequencies shows that we can extract from a single photodiode, the signal from the two LEDs. If you understand this result and how this is working, this is the key to this lab. Include a test such as this in your lab report.

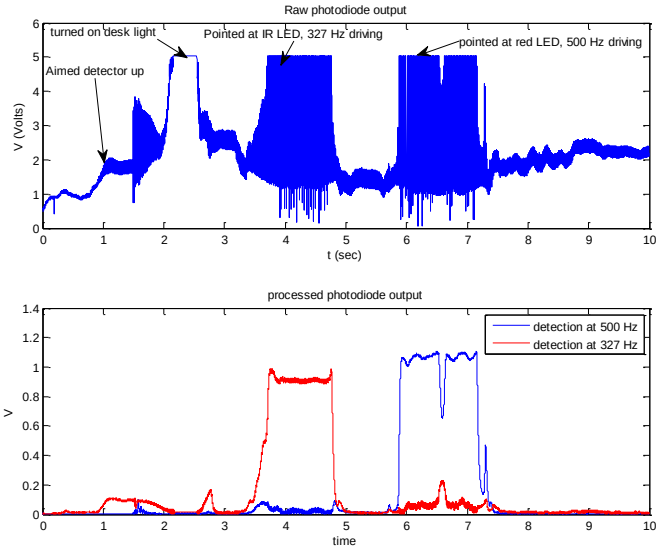

Figure 4: Detection of light from two different frequency LEDs, a red one blinking at 500 Hz and the IR one blinking at 327 Hz.

Now that everything is working, let's try to measure the light passing through your finger. Here we have found that things work a little better with two photodiodes aimed directly at the opposing LED, even though we see that we could get the result with a single detector and two lights. Build a second photodiode circuit on the last remaining op-amp of your quad package.

Now take one of the Lego assemblies that we pre-built (make sure you get a detector and an emitter), using a rubber band clamp your finger tightly (but not painfully) between the LEDs and the photodiodes. Arrange as best you can so it seems that the LED is pointed directly at the opposing photodiode. Try the experiment with your finger for about 10 seconds. You should see pulse. You really need to take care to align things and try to get LED/detector in good contact with skin. The pinky works well. It works better if the light is pushing on the fleshy side of your finger and the detector is above the finger nail. An example trace shown in Figure 5. In this figure we show the output of the detection function at the two frequencies from both photodiodes. You can see here that both photodiodes are able to pick up both LEDs and that the signal is stronger for the LED signal which is directly opposite the detector. While one photodiode works, we found that using two provided a stronger signal and was more likely to work.

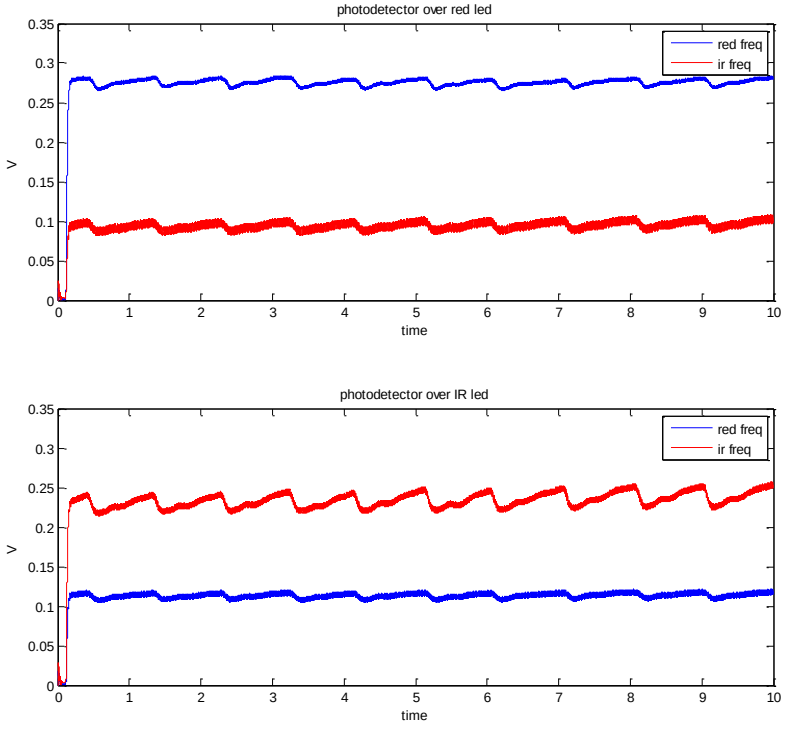

Figure 5: Example pulse. The upper figure is the photodiode which opposes the red LED and the lower one opposes the IR LED.

Next you can try a breath test. **Do not push yourself too hard on this test. If you do not want to do the breath test, one of the course instructors will be your patient.** Set the time for the experiment to run to

be about twice as long as you think you can hold your breath for. Take a deep breath and start the experiment. Hold as long as you can without pushing yourself too hard. Move as little as possible. Once you have reached your limit, take a deep breath and relax.

Sample results are shown in Figure 6. Here the patient took a deep breath at the start of the experiment and held their breath for 90 seconds. After about a minute the patient was beginning to feel a little anxious for air, this is the time the signal started to decrease in the red LED. At 90 seconds the patient came up for air and the signal immediately recovers. The patient takes several deep breaths to recover and we see the signal increase beyond the original level and then level out to the same state we started the experiment. The "noise" in this data is the patients pulse.

The reading for the oxygenation of the blood is the ratio of the two signals. To obtain a good result for the long term oxygenation, another low-pass filter is needed to smooth out the oscillations from your pulse. We will leave this for you to implement.

In your lab report, show a trace of your pulse and the breath test.

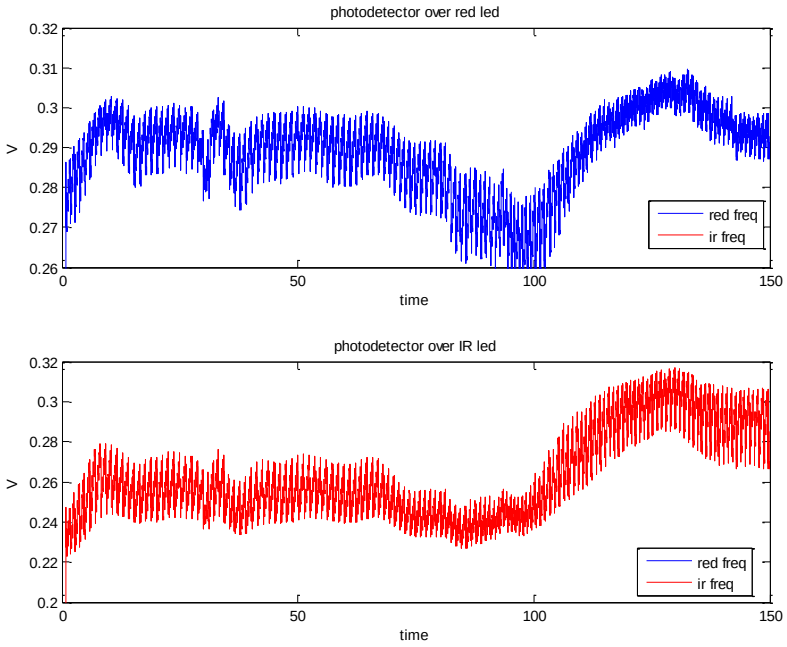

Figure 6: Signal from the red and IR LED over time while the patient was holding their breath. They took a big breath at the start (see the rise in the upper figure), and held their breath for 90 seconds.