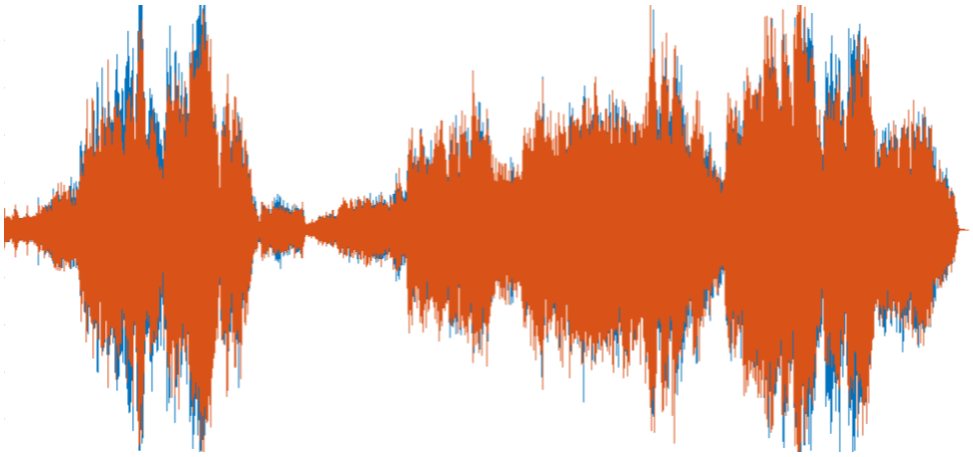
Digital Signal Processing using MATLAB



Electrical and Computer Engineering 5

Exploring Digital Signal Processing

Developed by:

Scott Zhao • Tina Kafel • Willy Ma

Joshua Jain • Professor Karcher Morris • Professor Truong Nguyen

Professor John Eldon | Professor Vikash Gilja | Professor Drew Hall | Professor Truong Nguyen

Introduction

Welcome to Lab 3! This assignment focuses on the field of Electrical Engineering known as Digital Signal Processing (DSP). DSP is a sub-discipline of the more general field of Signal Processing, a broad and highly important field which has applications in fields ranging from image and audio processing to financial services. In the infancy of signal processing, signals were mainly processed using analog amplifiers; this was largely due to the fact that it just was not feasible to perform the desired operations and manipulations on the current digital systems. Nowadays, huge advances in computing power and technology have made DSP a powerful alternative to analog signal processing. In addition, many modern systems can be classified as "mixed signal" systems as that they contain both digital and analog components. It is to your advantage as an engineer to know how to work in both of these domains.

What You Will Need

Materials:

- Arduino, photoresistor, and jumper wires.

Equipment:

- Desktop computer / Laptop

Software Tools:

- MATLAB
  - Instrument Control Toolbox
  - Digital Signal Processing Toolbox
  - Image Processing Toolbox
  - Add-ons: Arduino Hardware, Webcam

Objective

The DSP application we will be exploring in this lab follows three general steps:

1. First, a measurement from the world is captured as an analog signal. This analog signal is *digitized*, or *sampled*, at a specific frequency (represented in Hertz (Hz)), and the values of the samples are converted to a number of binary bits (in a process called quantization). These bits form the *digital representation* of the original analog signal. This entire process is known as Analog to Digital Conversion (ADC).
2. These bits form *digital signals* which can be manipulated and processed by a digital computer. The useful information these signals contain can be extracted at this stage.
3. Depending on the application, the processed signals can be converted back to an analog signal in a process known as Digital to Analog Conversion (DAC). This is often the case in digital audio processing, as a speaker system is driven by an analog voltage/current signal.

In this lab, you will have the opportunity to perform the first two of these steps. (The third step can be done in many ways. One could build a simple DAC circuit and speaker driver circuit for example. You will find many such projects online.) In doing this, you will gain exposure to many of the key concepts and tools used in DSP.

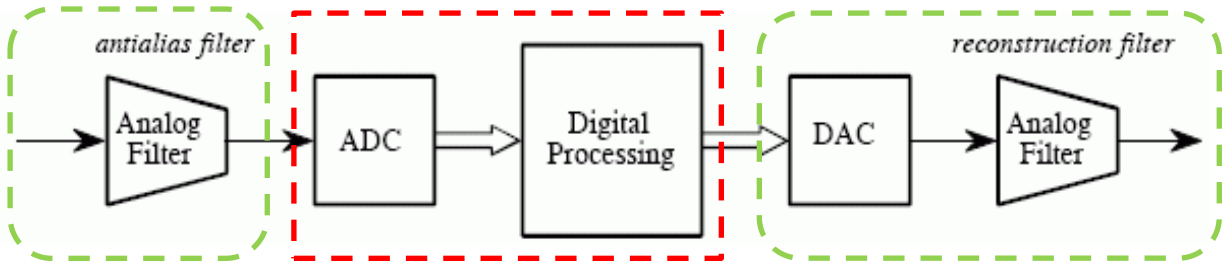


Figure 1: (Source: http://www.dspguide.com/graphics/F_3_7.gif)

Figure 1 shows the general DSP procedure mentioned previously. We will be focusing on the processes inside of the red, dashed rectangle. This will be done using MATLAB and Arduino. MATLAB is a powerful computing platform which is widely used by professional engineers in many different industries. The goal of this section of the course is to explore the concepts of sampling, filtering, and how signals change over time.

Part 1: Signal Processing using MATLAB & Arduino

When you first open MATLAB, you will notice three distinct windows (Figure 2: numbers added for clarity.)

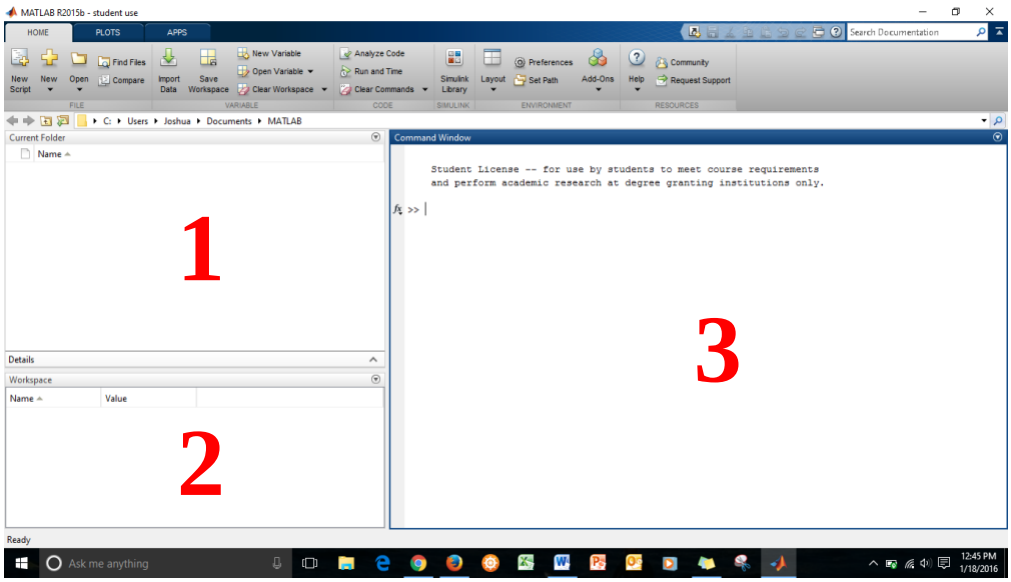


Figure 2: MATLAB home screen.

Window "1" shows the current file path which MATLAB is using to open and save files. Changing the path can be accomplished by either specifying the path in the search bar or clicking the "browse for folder" icon located at the top left of the window.

Window "2" is known as the "Workspace". MATLAB stores all variables and objects which you create in the workspace, it is a convenient location to access your variables and to check for errors in your programs. Window "3" is called the "Command Window", and here is where you can execute your MATLAB Commands. Executing a command is as simple as typing in the command and pressing enter. For example, type in the MATLAB command *"ver"* (omit quotation marks) and press enter. This command will list the version numbers for all the features of your MATLAB suite. To clear your old commands from the Command Window, type in "*clc*". For help with any command or MATLAB function, type "*help*" followed by the name of the function or command in question. For example, if I wanted to know more about the MATLAB absolute value function, I would type the following into the command window: "*help abs"*.

The command window will only execute one command at a time, so it is difficult to execute long programs requiring multiple lines of code. So instead, we create what is called a "script", a collection of commands which can all be executed at once, just like a typical computer program. To create a script, click on the "New Script" button at the top left corner of the MATLAB home screen. You may find that performing the following exercises using MATLAB scripts is much easier than entering each line of code individually in the command window.

Challenge #1: MATLAB OnRamp

Working with MATLAB for the first time can be challenging. Fortunately, MathWorks (the MATLAB company) has created an interactive tutorial for first time users that covers a wide range of topics in approximately 2 hours. These topics help with basic syntax, programming logic, and share the many hints that truly make MATLAB a powerful engineering tool, not just another language. Complete the MATLAB OnRamp tutorial provided in the link below:

https://matlabacademy.mathworks.com/en/selfpaced/gettingstarted/

You may need to create a MathWorks User ID and should with your university email. It is not necessary to download and install MATLAB in order to complete this tutorial (but maybe you can do this in parallel). At the end of this tutorial you will earn a certificate of completion. More importantly you will be much more prepared for the challenges ahead.

Challenge #2A: Creating a Signal in MATLAB

This exercise will show you how signals can be created using MATLAB. In the following example, you will be shown how to create and plot a signal represented by a sine wave.

First, create a new script in MATLAB and save it as {last_name}_(first_name}_Exercise_{#}. (We would also recommend creating new scripts for each exercise in this lab with a similar name structure.) To begin, we will be creating a basic sine wave. In further courses you will see that a variety of real world periodic signals can be represented using the sine wave. Normally, a sine wave would be represented digitally as a vector of repeating data points. However, instead of typing in all of these data points by hand, we can use the MATLAB *sin()* function to generate the data points we need. To do this, all we need to do is specify the *frequency* of the sine wave, the *sampling frequency*, and the *length of time* over which we wish to plot.

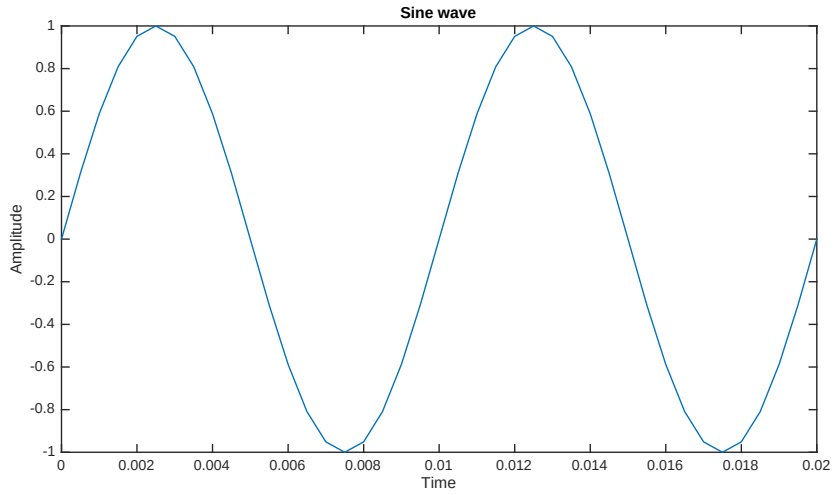
For example:

```
clear all; close all; clc;

f = 100;          % This is the frequency of the generated sine
                  % wave, not be confused with the sampling frequency
a = 1;            % maximum amplitude

fs = 20*f;        % Sample 20 points per period
dt = 1/fs;
t = 0:dt:0.02;    % Plot 2 periods (1 period = 1/100Hz = 0.01)

y = a*sin(2*pi*f*t);    % Calculate samples
figure;
plot(t,y);
```

When ran, the code should generate a plot of a sine wave with a frequency of 100Hz and a maximum amplitude of 1. Before you run the code, it is good practice to label your axes on your plots and to also give them a title. In order to do this, we will use the MATLAB commands "*xlabel*", "*ylabel*" and "*title*." (Use the "*help*" command to find out how to use these features.) Enter the following code segment into your previous script directly after you call the plot command.

```
xlabel('Time (s)');                 %Specify label for x axis.
ylabel('Amplitude');                %Specify label for y axis.
title('Sine wave');                 %Specify title for figure.
```

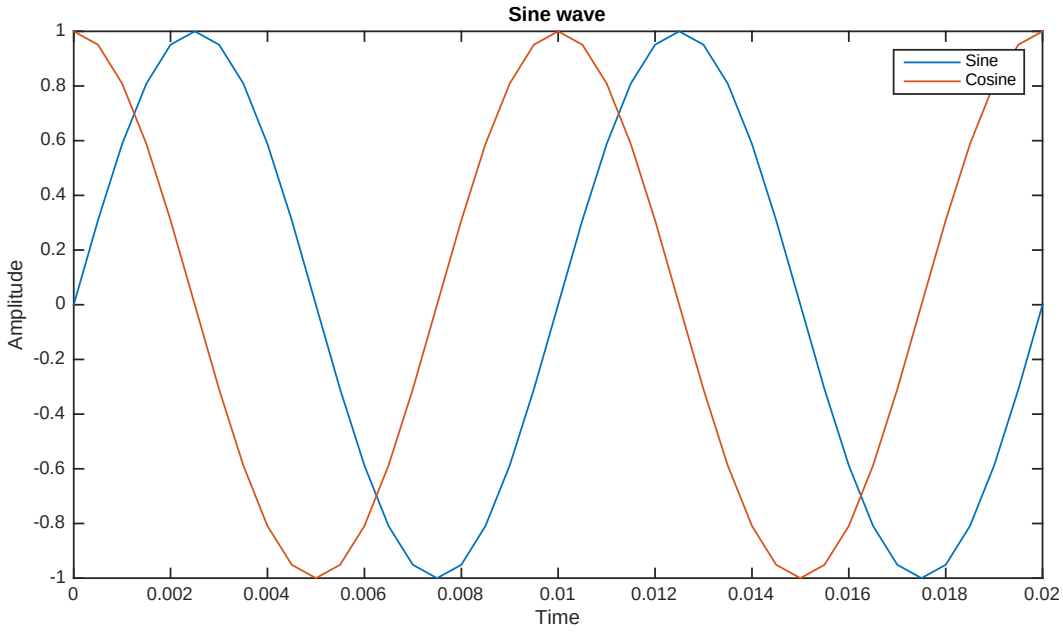
If done properly, you will see a labeled figure which looks like this:



Oftentimes, we wish to plot multiple functions on the same graph as this is a quick and easy way to make comparisons (by default, MATLAB will create a separate plot each time you call the plot command). Let's say we wish to plot the cosine wave on the same plot as our sine wave. We want the cosine wave to have the same characteristics of the sine wave (same time base, frequency, sampling frequency, and amplitude). To do this, we will add the following code to our script:

```
z = a*cos(2*pi*f*t);            % same w, t, and a as in sine wave
hold on;                        % tell MATLAB to use the same plot
plot(t,z);
legend('Sine', 'Cosine');       % create a plot legend.
```

We first use the "*cos()"* function, passing the same arguments to it as the "*sin()"* function to generate the cosine wave data. In order to plot this data on the same plot as sine wave, we use the line *"hold on"*, which tells MATLAB to use the same figure to plot the new data. The *"legend()"* command creates a legend with the specified labels. Running the entire script will show a figure which should look like this:



Using the above example, perform the following exercises.

1. Set the sampling frequency (F_s) **5** times the sine signal frequency, plot the following three sine waves from time **0 to 0.005**. Make sure all 3 curves are in one figure (hint: use commands *hold on* and *legend*):
    a. Frequency 500 Hz, Amplitude 1
    b. Frequency 1000 Hz, Amplitude 0.5
    c. Frequency 3000 Hz, Amplitude 3
2. Choose one of the three sine waves you plotted above, plot it on the same figure three times **with different sampling frequencies**. To plot each of these resampled signals in the same figure window, make use the "*subplot*" command. If unsure how to use this, type "*help subplot*" in the command window or check out this link. Be sure to title and label each of these subplots and specify the frequency at which they were sampled in the titles.
    a. 3 times the sine signal frequency
    b. 5 times the sine signal frequency, same frequency / amplitude
    c. 10 times the sine signal frequency, same frequency / amplitude
3. Comment on how the shape of the sine wave changes as you change the sampling frequency. Do certain plots look better than others?

Challenge #2B: Receiving Signal Through Serial

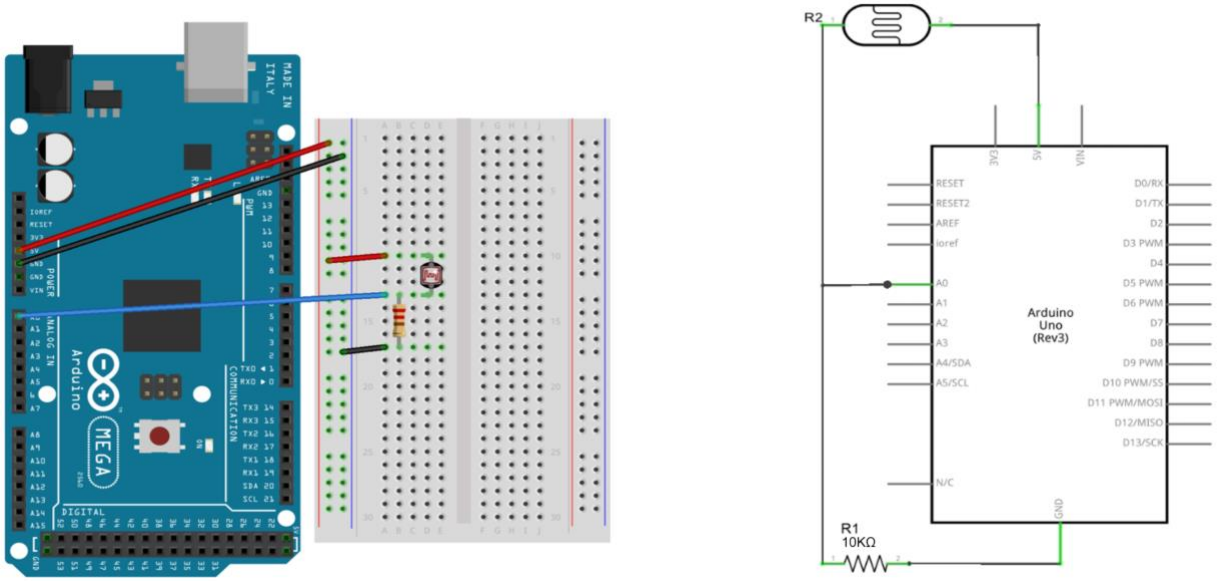
Now that you've learned how to generate sine waves in MATLAB, let's try and generate sine waves from the "real" world using a photoresistor and Arduino. In this exercise, we will utilize Arduino and MATLAB to plot the sine and square waves generated by your hand motions detected by a photoresistor. To record samples at a high and stable sampling rate, we will read the sensor values through the analog pin A0 and Arduino will send the digitized values through the Serial Communication. MATLAB will read the values through serial and plot the data as desired.

The following code has been developed to upload to your Arduino. (Download on *Canvas* ***SamplePhotoresistor50Hz.ino***) Read the code carefully as special care was taken to collect and report the data through serial at 50Hz. Interesting questions should arise, i.e., what does the function `millis`() output?, what does it mean to have an unsigned long declaration for a specific variable?, why was unsigned long used?, How and why is the if statement used?... etc.

SamplePhotoresistor50Hz.ino

```
// Outputs analog reading at A0 and system time to Serial at 50 Hz.
      unsigned long ii = 0; // Counter
      int sensorValue; // Stores value read from the sensor
      unsigned long millinum;
      void setup() {
        Serial.begin(9600); //Initialize Serial connection
      }

      void loop() {
        millinum = millis();
        if ((millinum) > 20 * ii) { //Make the following code execute every 20ms
          millinum = millis();
          sensorValue = analogRead(A0); // read the input on analog pin A0
          // Print to serial the value you read:
          Serial.print(millinum); // Time
          Serial.print(' ');
          Serial.println(sensorValue); // Sensor reading
          ii = ii + 1; // Increment counter
        }
      }
```

First start with building a voltage divider circuit. Connect a 10k Ohm resistor in series with photoresistor, the same as what you did in Lab 0. Connect the output to analog input A0 on Arduino.



Then, upload the code to your Arduino and check to make sure the data is being output successfully by opening your Serial monitor. We record data for every 20 ms, which is equal to 50 Hz.

$$f = \frac{1}{T} = \frac{1}{20 \times 10^{-3}} = 50\ Hz$$

Now close your Arduino IDE and let's read the data sent through the serial port in MATLAB instead of through the Arduino Serial monitor. Download from *Canvas* and read through the code below, "***SerialLivePlot.m***".

SerialLivePlot.m

```
clear all; close all; fclose all; delete(instrfind); clc; % Start Fresh
PLOTTING_LENGTH = 10; % Change this to adjust max seconds to show on plot
Fs = 50; % Arduino Sampling Frequency

%% Connect to Arduino through Serial
ARDUINO_SERIAL_PORT = 'COM2'; % Change this (Win: 'COM?', MAC: '/dev/cu.usbmodem?')
arduinoSerial = serial(ARDUINO_SERIAL_PORT,'baudrate',9600,'InputBufferSize',5120); %
Create Serial Object a to connect to arduino (choose com port correctly)
fopen(arduinoSerial); % Open Serial communcation
pause(1) % Wait data on Serial
fgetl(arduinoSerial); % Take out broken first line in serial
%% Initialize Plotting axis
fig1 = figure();
set(fig1,'Units','normalized')
set(fig1,'Position',[0.1,0.1,0.8,0.8])
ii = 0; % Initialize counter

%% Main Loop
while ishghandle(fig1)
    while arduinoSerial.BytesAvailable %Read until serial is empty
        htic1 = tic;
        while toc(htic1)<0.05 %Read for 0.05 seconds before checking if serial is empty
            %% Read a line in serial and put data into arrays
            ii = ii + 1;
            [timestr,sensstr] = strtok(fgetl(arduinoSerial),' ');
            time(ii) = str2num(timestr) / 1000;
            sens(ii) = str2num(sensstr);
        end
    end
    %% Update plot
    cla() % Clear previous plot
    plot(time,sens,'-o','linewidth',2), xlabel('Time [s]')
    ylabel('Photoresistor Reading'), title('Photoresistor Sampled at 50kHz')
    xlim([time(max(1,ii-Fs*PLOTTING_LENGTH)),time(ii)+0.001]) % Adjust x-axis range
    pause(0.05) % Give MATLAB time to render plot
end
delete(arduinoSerial) % Close serial connection
```

First, change ARDUINO_SERIAL_PORT to match what you see from Arduino IDE – Tools – Port.
Then, using the above Arduino and MATLAB code, perform the following exercises:

1. Create sine waves using your hand moving up and down above the photoresistor at approximately 0.5 Hz and 1Hz. Save 2 separate plots showing these waves.
2. Create a square wave with your finger block & unblock the photoresistor (or phone flashlight on & off) at approximately 0.5Hz.
3. What was your amplitude for each wave?

Do not disassemble your photoresistor circuit! You are going to use them on Challenge 3 & 4!

Challenge #3A: Sample Signals via Functions

In this challenge, we will learn how to use functions to create signals. Functions are a group of statements packaged together to perform a certain task. In MATLAB, functions can be defined in separate m-files. **The name of the function must match the name of the file of the function.** Syntax of a function in MATLAB is:

```
function [returnParameters] = FunctionName(inputParmeters)
% Body of the function
end
```

Let's use this syntax to create a function that will help us generate sine waves and sample them. The block of code provided below will help us achieve this goal.

```
function [t,y] = genSine(frequency,________, time, numOfSamplesPerPeriod)
      f = _________;  % Frequency of the sine wave
      a = amplitude; % Amplitude of the sine wave
      fs = _________*f; % Sampling frequency
      dt = 1/__;
      t = 0:dt:time; % Length of time for plotting.
      y = a*sin(2*pi*f*t);
end
```

Now lets make use of this function. Complete the blanks above for the following exercises.

Generate and plot three sine waves using genSine() function you made with following configurations. **Make only one figure with three subplots**.

1. Frequency 0.5 Hz, Amplitude 2, Time 5, 10 SamplesPerPeriod
2. Frequency **2** Hz, Amplitude 2, Time 5, 10 SamplesPerPeriod
3. Frequency 2 Hz, Amplitude 2, Time 5, **5** SamplesPerPeriod

Challenge #3B: Sampling with Arduino/MATLAB

In this challenge, we're going to continue what we did in Challenge 2B, use an Arduino to sample sensory data from the "real" world (photoresistor) at 50Hz, send to MATLAB through Serial. To visualize the effect on different sampling frequencies, we used MATLAB to resample the photoresistor data at several lower frequencies. The serial port implements a buffer that is able to store data that arrives. This is useful in the case that MATLAB cannot process the incoming serial data before the next samples are sent by the Arduino.

The MATLAB script ***SerialResampledLivePlot.m*** is available on *Canvas* and in the appendix.

Perform the following tasks.

1) Connect the Arduino with the photoresistor setup in Challenge 2B. Your Arduino should already have "***SamplePhotoresistor50Hz.ino***" uploaded on Challenge 2B. Plug in your Arduino.
2) Run the "***SerialResampledLivePlot.m***" MATLAB code from *Canvas* or appendix. Make sure to change your serial port name. The program plots the sampling data recorded by Arduino at 50Hz and then resamples the data at 10Hz, 5Hz, and 1Hz.
3) By moving your hand over the photoresistor, try to create sine wave at 0.5Hz and 1Hz.
4) Compare the subplots sampling at different frequencies. What's the effect on having a lower sampling frequency? Why?

(Optional) Interesting questions to discuss about:

1) What is the maximum frequency that MATLAB can plot the photoresistor data?
2) What is the advantage of separating tasks of sampling to the Arduino and plotting to MATLAB?
3) If we were to use MATLAB to sample and plot at 50Hz, what could be some drawbacks?
4) If the serial buffer was infinite, and the data comes in faster than it can be plotted, what do you think is actually plotted?
5) If the serial buffer was very small, and the data comes in faster than it can be plotted, what do you think is actually plotted?

Challenge #4A: Time and Frequency Representation

You have seen from the previous exercise what the waveform of a photoresistor signal looks like, you saw how the signal moves as your hand moves with respect to time. However, even more information can be extracted from this signal if you can consider the signal from a different perspective. We will now consider the signal from the perspective of the ***Frequency Domain***. This is particularly useful when you are dealing with more complex signals, for example, an audio signal coming out of your microphone in Lab 2.

Most interesting audio signals, such as music, contain different sound components. There may be the low sounds from the bass guitar, higher, sharper sounds from vocals or other instruments, and then midrange sounds. We often ascribe to these different sounds a quality known as "pitch". In general, we consider higher frequency sounds to have higher pitch, and lower frequency sounds to have lower pitch. What if there was a way that we could look at a signal in terms of its "pitches". To do this, we need to establish a relationship between the time signal and the pitches/frequencies it contains.

The relationship between the Time and Frequency Domain is usually described in terms of a mathematical operation known as a ***Transform***, which maps a function in one domain to the other. There are several such transforms that perform this mapping, and there are rules that dictate when each of these can be used. For our purposes, the *Fourier Transform* and the *Fourier Series Representation* are the best choices. We will not go into the mathematics of this transform operation; those interested can find more information in the lecture or online. Because we will not cover these mathematics, we will rely on software tools to do this for us. There are methods which exist that enable digital computers to perform these mathematics very accurately and efficiently, so these operations are almost always performed using a computer tool. What is important at the present time is an understanding of how these operations can aid us in our DSP applications.

In essence, the Fourier transform and series are important because they make the analysis of complicated signals much easier. The idea, developed by French mathematician Joseph Fourier, is to take a complicated signal and break it up into simpler pieces. These smaller pieces are simple waveforms which are represented by sine and cosine functions. These sinusoidal functions are periodic, which means that they repeat themselves. They have a property ascribed to them known as frequency, a quantity which specifies the number of times the sinusoid repeats per unit time. Take a look at the following figure.

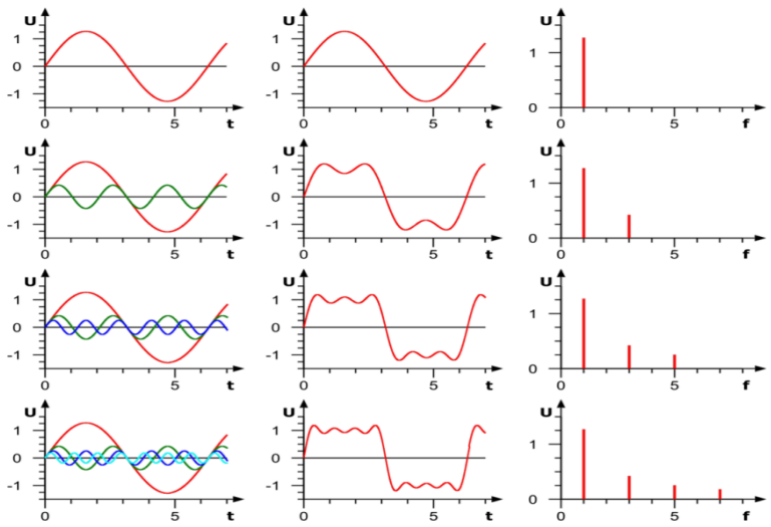


Figure 3: Time domain signals and their spectral components. Source: http://opticalengineering.spiedigitallibrary.org/article.aspx?articleid=1891707

This figure has three columns with three different types of graphs in each column. The first column shows individual sinusoids of different frequencies and amplitudes plotted together. By themselves, they are not very interesting. However, the second column shows the resulting signals when all the individual components of the first column are added together. As you add more and more sinusoidal signals together, you begin to see how the graph in column three changes. Joseph Fourier (who developed the Fourier series) showed that you can create any signal by just adding together a series of sinusoids. In essence, this is the main idea behind the Fourier Series.

Getting to column three from the information in columns one and two is a little more complicated. It involves taking mathematical transform, the Fourier Transform of the signal in column two. The plot in column three is the result of such an operation plotted on an x axis representing frequency. These types of plots are known as *Power Spectrum* plots. The red lines on the plots in column three correspond to different sinusoids of a particular frequency and amplitude. You can interpret the taller lines on the plot as those frequency components of the time signal which have higher power, that is the sine waves in column one with the highest amplitudes. Basically, each of the red lines on the graph represents one component from the graph in column one.

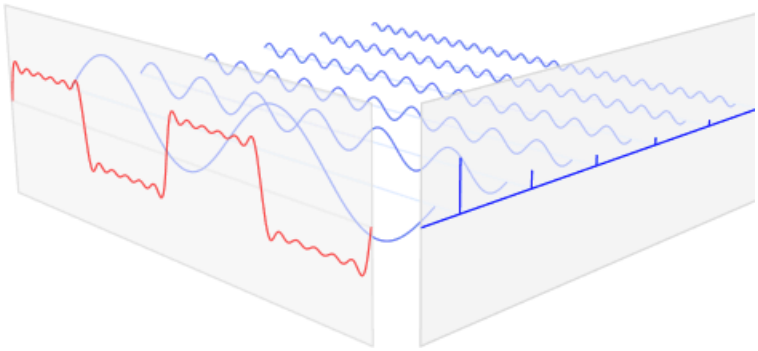


Figure 4: Time domain (red) vs Frequency domain (blue) https://i.stack.imgur.com/27HVo.gif (Animated Version)

The important concept for this lab is that you know how to construct and interpret the Power Spectrum (PS) plots, and you will be using MATLAB to do this.

Recall the three sine wave you generated with the genSine() function you completed in Challenge 3. In this challenge, we are reusing the genSine() function to generate three similar sine waves, at 1kHz, 2kHz, and 5kHz, but with **same sampling frequency** fs = 50kHz so that we could combine them later by simply adding the sample points.

As mentioned above, you don't need to worry about the *Fourier Transform* mathematics, as it is provided to you as a function plotFFT(). It takes three arguments: data array, sampling frequency, and the axis you are plotting on.

Perform the following task:
Run the script on next page to generate the three sine waves, combine them, and plot them both in **Time Domain** and in **Frequency Domain**. Make sure you downloaded the ***plotFFT()*** function and placed it in the same folder of the script. This script ***FFTGeneratedSine.m*** can be downloaded from *Canvas*.

Discuss in your write up:
1) What can you read from the PS plots? What does the spike mean?
2) What's the benefit on analyzing signals using PS plot (frequency domain)? Especially on the combined signal?
3) (**Optional**) What does the x-axis (frequency) range on the PS plots tell you? How does that relate to out sampling frequency 50Hz? Think about Nyquist Theorem (Sampling Theorem).

FFTGeneratedSine.m

```
clear; close all; clc; %clear previous data, command window, and all open plots
%% Generate sine waves (make sampling frequency consistent!)
[t1,y1] = genSine(1000,1,2/1000,50); %Generate Sine Wave at 1kHz
[t2,y2] = genSine(2000,0.7,2/1000,25);  %Generate Sine Wave at 2kHz
[t3,y3] = genSine(5000,0.3,2/1000,10);  %Generate Sine Wave at 5kHz

%% Combining the two signals
minLength = min([length(y1),length(y2),length(y3)]);
y = y1(1:minLength) + y2(1:minLength) + y3(1:minLength); % Trim the longer signals
to match signal length

%% Plot
fig1 = figure();
set(fig1, 'Units', 'normalized')
set(fig1, 'Position', [0.1,0.1,0.8,0.8])
hold on;
    %% 1kHz Sine Time Domain
    subplot(421)
    plot(t1,y1)
    title('1kHz Sine Time Domain'), ylim([-1,1])
    %% 1kHz Sine Frequency Domain
    fft1 = subplot(422);
    plotFFT(y1, 50000, fft1)
    title('1kHz Sine Frequency Domain'), ylim([0,1])
    %% 2kHz Sine Time Domain
    subplot(423)
    plot(t2,y2)
    title('2kHz Sine Wave Time Domain'), ylim([-1,1])
    %% 2kHz Sine Frequency Domain
    fft2 = subplot(424);
    plotFFT(y2, 50000, fft2)
    title('2kHz Sine Frequency Domain'), ylim([0,1])
    %% 5kHz Sine Time Domain
    subplot(425)
    plot(t3,y3)
    title('5kHz Sine Wave Time Domain'), ylim([-1,1])
    %% 5kHz Sine Frequency Domain
    fft3 = subplot(426);
    plotFFT(y3, 50000, fft3)
    title('5kHz Sine Frequency Domain'), ylim([0,1])
    %% Combined Sine Time Domain
    subplot(427)
    plot(y)
    title('Combined Wave')
    %% Combined Sine Frequency Domain
    fft4 = subplot(428);
    plotFFT(y, 50000, fft4)
    title('Combined Sine Wave')
```

Challenge #4B: Time and Frequency Representation

Recall in Challenge 3B we did some live plotting with the photoresistor data in time domain. In this part of the challenge, let's do the same plot, but in **frequency domain**. We will use a MATLAB script to read sampled photoresistor readings from serial, plot in **Time Domain**, then do *Fourier Transform* and plot a PS plot over 20 seconds in **Frequency Domain**. Lastly, we generate a "3D" spectrogram, which contains time, frequency and amplitude data of the whole signal, so that you can better visualize the frequency information in the signal as it varies over time. The spectrogram is basically a combination of PS plots at different time.

Plug in your Arduino with the same program uploaded in Challenge 3 and run the MATLAB code on next page (***FFTRealtime.m*** on *Canvas*). Don't forget to change the Serial port name. Keep the plotting window open, try creating "sine waves" with your hand again at 0.5Hz and 1Hz, each for 20 seconds. *Take screenshot of the whole plot at the end of each 20 second period. There should be in total two screenshots*, each of which has a PS plot that shows a spike on your hand moving frequency, and a spectrogram containing frequency information of the entire time since you started.

Discuss in your write up:

1) What did you observe on the PS plot (second subplot) on your two screenshots? What does it mean?
2) What did you notice on the spectrogram? How is that related to your hand moving frequencies?
3) On top of the MATLAB script, change the **FFT_PLOT_DURATION** from 20 seconds to 5 seconds. What's the effect? Why? (Do not take screenshots)

(**Optional**) Also try creating "sine waves" with your hand at a decreasing frequency (move fast at first, then slower and slower). Observe what's on the spectrogram.

Last updated: 10/28/2020

FFTRealtime.m

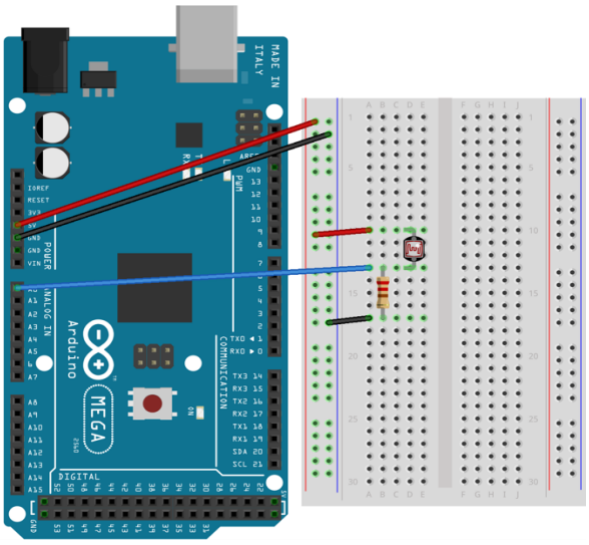
```
clear all; close all; fclose all; delete(instrfind); clc; % Start Fresh
%% Settings
SECONDS_TO_PLOT = 10;
ARDUINO_SERIAL_PORT = 'COM16'; % Change this (Win: 'COM?', MAC: '/dev/cu.usbmodem?')
FFT_PLOT_DURATION = 20;
Fs = 50; % Sampling frequency is 50 Hz

%% Initialize Serial Connection and variables
% Create Serial Object a to connect to arduino (choose com port correctly)
arduinoSerial = serial(ARDUINO_SERIAL_PORT,'baudrate',9600,'InputBufferSize',5120);
fopen(arduinoSerial); % Open Serial communcation
pause(1) % pause 1 second
fgetl(arduinoSerial); % Took out first line
timeArray = 0; sampleArray = 0; % Arrays to store time and sample
i = 0; % Initialize Counter
htic1 = tic; % Initialize timer

%% Setup Figure
fig1 = figure();
set(fig1, 'Units', 'normalized')
set(fig1, 'Position', [0.1,0.1,0.8,0.8])
timeDomainAxis = subplot(311); timeDomainPlot = plot(timeArray, sampleArray); title('Real Time Photoresistor Reading')
frequencyDomainAxis = subplot(312); frequencyDomainPlot = plot(0); title('20 second FFT'),
xlim([0 5]) % Change xlim to zoom in
spectrogramPlot = subplot(313); title('Spectrogram'), xlabel('Time(s)'),
ylabel('Frequency(Hz)')

%% Main Loop
while(ishghandle(fig1)) % loop until the figure windows is closed
    %% Read data from Serial and record
    while arduinoSerial.BytesAvailable %Read until serial is empty
        htic1 = tic;
        while toc(htic1)<0.05 % Read for 0.05 seconds before checking if serial is empty
            %% Read a line in serial and scale the numbers
            [timestr,sensstr] = strtok(fgetl(arduinoSerial),' ');
            timeData=str2num(timestr) / 1000; sensorData=str2num(sensstr) * (5/1023);
            %% Record data into array
            i = i + 1; %Increase counter every time a sample is recorded
            timeArray(i) = timeData; sampleArray(i) = sensorData;
        end
    end

        %% Plot Time Domain
        set(timeDomainPlot,'XData',timeArray,'YData',sampleArray); xlim(timeDomainAxis,[0 timeArray(end)])

        %% Plot Frequency Domain for samples in recent 20s (change this in settings on top)
            %% FFT calculations
            Y = fft(sampleArray(max(1,(i-FFT_PLOT_DURATION*50)):i));
            L = length(sampleArray(max(1, (i-FFT_PLOT_DURATION*50)):i));
            P2 = abs(Y/L);
            P1 = P2(1:floor(L/2+1));
            P1(2:end-1) = 2*P1(2:end-1);
            f = Fs*(0:(L/2))/L;
        set(frequencyDomainPlot,'XData',f,'YData',P1);

        %% spectrogram realtime
        if(i > Fs * 5) % Only start if we have enough data (5s)
            spectrogram(sampleArray,Fs * 5,Fs * 2,i,Fs,'yaxis')
            set(spectrogramPlot, 'Ylim', [0 5]); % Zoom in
        end
```

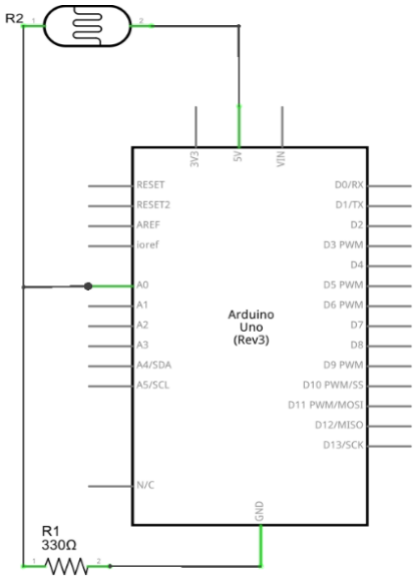
Appendix

SamplePhotoresistor50Hz.ino

```
// Outputs analog reading at A0 and system time to Serial at 50 Hz.
      unsigned long ii = 0; // Counter
      int sensorValue; // Stores value read from the sensor
      unsigned long millinum;
      void setup() {
        Serial.begin(9600); //Initialize Serial connection
      }

      void loop() {
        millinum = millis();
        if ((millinum) > 20 * ii) { //Make the following code execute every 20ms
          millinum = millis();
          sensorValue = analogRead(A0); // read the input on analog pin A0
          // Print to serial the value you read:
          Serial.print(millinum); // Time
          Serial.print(' ');
          Serial.println(sensorValue); // Sensor reading
          ii = ii + 1; // Increment counter
        }
      }
```

Photoresistor & Arduino Circuit





SerialLivePlot.m

```
clear all; close all; fclose all; delete(instrfind); clc; % Start Fresh
PLOTTING_LENGTH = 10; % Change this to adjust max seconds to show on plot
Fs = 50; % Arduino Sampling Frequency

%% Connect to Arduino through Serial
ARDUINO_SERIAL_PORT = 'COM2'; % Change this (Win: 'COM?', MAC: '/dev/cu.usbmodem?')
arduinoSerial = serial(ARDUINO_SERIAL_PORT,'baudrate',9600,'InputBufferSize',5120); %
Create Serial Object a to connect to arduino (choose com port correctly)
fopen(arduinoSerial); % Open Serial communcation
pause(1) % Wait data on Serial
fgetl(arduinoSerial); % Take out broken first line in serial
%% Initialize Plotting axis
fig1 = figure();
set(fig1,'Units','normalized')
set(fig1,'Position',[0.1,0.1,0.8,0.8])
ii = 0; % Initialize counter

%% Main Loop
while ishghandle(fig1)
    while arduinoSerial.BytesAvailable %Read until serial is empty
        htic1 = tic;
        while toc(htic1)<0.05 %Read for 0.05 seconds before checking if serial is empty
            %% Read a line in serial and put data into arrays
            ii = ii + 1;
            [timestr,sensstr] = strtok(fgetl(arduinoSerial),' ');
            time(ii) = str2num(timestr) / 1000;
            sens(ii) = str2num(sensstr);
        end
    end
    %% Update plot
    cla() % Clear previous plot
    plot(time,sens,'-o','linewidth',2), xlabel('Time [s]')
    ylabel('Photoresistor Reading'), title('Photoresistor Sampled at 50kHz')
    xlim([time(max(1,ii-Fs*PLOTTING_LENGTH)),time(ii)+0.001]) % Adjust x-axis range
    pause(0.05) % Give MATLAB time to render plot
end
delete(arduinoSerial) % Close serial connection
```

SerialResampledLivePlot.m Part1

```
clear all; close all; fclose all; delete(instrfind); clc; % Start Fresh
%% Settings
  SECONDS_TO_PLOT = 10;
  % Change this (Win: 'COM?', MAC: '/dev/cu.usbmodem?')
  ARDUINO_SERIAL_PORT = 'COM16';
%% Initialize Serial Connection and variables
  % Create Serial Object a to connect to Arduino
  arduinoSerial =
  serial(ARDUINO_SERIAL_PORT,'baudrate',9600,'InputBufferSize',5120);
  fopen(arduinoSerial); % Open Serial communcation
  pause(1) % pause 1 second
  fgetl(arduinoSerial); % Took out first line
  % Arrays to store time and sample
  sample_50hz=0;time_50hz=0;sample_10hz=0;time_10hz=0;
  sample_5hz=0;time_5hz=0;sample_1hz=0;time_1hz=0;
  % Arrays to store index
  counter_50hz=0; counter_10hz=0; counter_5hz=0; counter_1hz=0;

%% Setup figure to plot
fig1 = figure();
set(fig1,'Units','normalized')
set(fig1,'Position',[0.1,0.1,0.8,0.8])
axis_50hz=subplot(411);
plot_50hz=plot(time_50hz,sample_50hz,'-o','linewidth',2);
  title('Photoresistor Sampled at 50kHz')
  xlabel('Time(s)'),ylabel('Voltage(v)');
axis_10hz=subplot(412);
plot_10hz=plot(time_10hz,sample_10hz,'-o','linewidth',2);
  title('Photoresistor Sampled at 10kHz')
  xlabel('Time(s)'),ylabel('Voltage(v)');
axis_5hz=subplot(413);
plot_5hz=plot(time_5hz,sample_5hz,'-o','linewidth',2);
  title('Photoresistor Sampled at 5kHz')
  xlabel('Time(s)'),ylabel('Voltage(v)');
axis_1hz=subplot(414);
plot_1hz=plot(time_1hz,sample_10hz,'-o','linewidth',2);
  title('Photoresistor Sampled at 1kHz')
  xlabel('Time(s)'),ylabel('Voltage(v)');
```

SerialResampledLivePlot.m Part2

```
%% Main Loop
while ishghandle(fig1) % loop until the figure windows is closed
    %% Read data from Serial and record
    while arduinoSerial.BytesAvailable %Read until serial is empty
        htic1 = tic;
        while toc(htic1)<0.05 % Read for 0.05 seconds before checking if serial is empty
            %% Read a line in serial and scale the numbers
            counter_50hz = counter_50hz + 1;
            [timestr,sensstr] = strtok(fgetl(arduinoSerial),' ');
            timeData=str2num(timestr) / 1000; sensorData=str2num(sensstr) * (5/1023);
                %% Record data into array of different frequencies
                time_50hz(counter_50hz) = timeData;
                sample_50hz(counter_50hz) = sensorData;
                if(~mod(counter_50hz-1,5))
% Once per 5 loops (50Hz / 5 = 10Hz), record data at 10Hz into sample_10Hz
                    counter_10hz = counter_10hz + 1;
                    time_10hz(counter_10hz) = timeData;
                    sample_10hz(counter_10hz) = sensorData;
                end
                if(~mod(counter_50hz-1,10))
% Once per 10 loops (50Hz / 10 = 5Hz), record data at 5Hz into sample_5Hz
                    counter_5hz = counter_5hz + 1;
                    time_5hz(counter_5hz) = timeData;
                    sample_5hz(counter_5hz) = sensorData;
                end
                if(~mod(counter_50hz-1,50))
% Once per 50 loops (50Hz / 50 = 1Hz), record data at 1Hz into sample_1Hz
                    counter_1hz = counter_1hz + 1;
                    time_1hz(counter_1hz) = timeData;
                    sample_1hz(counter_1hz) = sensorData;
                end

        end
    end
    %% Update Plot
updatedXlim = [time_50hz(max(1,counter_50hz-50*SECONDS_TO_PLOT)),time_50hz(counter_50hz)+0.001];
    set(plot_50hz,'XData',time_50hz,'YData',sample_50hz);
    xlim(axis_50hz,updatedXlim);
    set(plot_10hz,'XData',time_10hz,'YData',sample_10hz);
    xlim(axis_10hz,updatedXlim);
    set(plot_5hz,'XData',time_5hz,'YData',sample_5hz);
    xlim(axis_5hz,updatedXlim);
    set(plot_1hz,'XData',time_1hz,'YData',sample_1hz);
    xlim(axis_1hz,updatedXlim);
    pause(0.05) % Give MATLAB time to render plot
end

delete(arduinoSerial); % Closes Arduino Serial connection
```

plotFFT(…)

```
function plotFFT(data,maxSamplingFrequency, plottingAxis)
%Compute the Fourier Transform of X.
Y = fft(data);
L = length(data);
Fs = maxSamplingFrequency;
%Compute and plot the single sided PSD of X.
P2 = abs(Y/L);
P1 = P2(1:floor(L/2+1));
P1(2:end-1) = 2*P1(2:end-1);
f = Fs*(0:(L/2))/L;
plot(plottingAxis,f,P1)
```

FFTGeneratedSine.m

```
clear; close all; clc; %clear previous data, command window, and all open plots
%% Generate sine waves (make sampling frequency consistent!)
[t1,y1] = genSine(1000,1,2/1000,50); %Generate Sine Wave at 1kHz
[t2,y2] = genSine(2000,0.7,2/1000,25);  %Generate Sine Wave at 2kHz
[t3,y3] = genSine(5000,0.3,2/1000,10);  %Generate Sine Wave at 5kHz

%% Combining the two signals
minLength = min([length(y1),length(y2),length(y3)]);
y = y1(1:minLength) + y2(1:minLength) + y3(1:minLength); % Trim the longer signals
to match signal length

%% Plot
fig1 = figure();
set(fig1, 'Units', 'normalized')
set(fig1, 'Position', [0.1,0.1,0.8,0.8])
hold on;
    %% 1kHz Sine Time Domain
    subplot(421)
    plot(t1,y1)
    title('1kHz Sine Time Domain'), ylim([-1,1])
    %% 1kHz Sine Frequency Domain
    fft1 = subplot(422);
    plotFFT(y1, 50000, fft1)
    title('1kHz Sine Frequency Domain'), ylim([0,1])
    %% 2kHz Sine Time Domain
    subplot(423)
    plot(t2,y2)
    title('2kHz Sine Wave Time Domain'), ylim([-1,1])
    %% 2kHz Sine Frequency Domain
    fft2 = subplot(424);
    plotFFT(y2, 50000, fft2)
    title('2kHz Sine Frequency Domain'), ylim([0,1])
    %% 5kHz Sine Time Domain
    subplot(425)
    plot(t3,y3)
    title('5kHz Sine Wave Time Domain'), ylim([-1,1])
    %% 5kHz Sine Frequency Domain
    fft3 = subplot(426);
    plotFFT(y3, 50000, fft3)
    title('5kHz Sine Frequency Domain'), ylim([0,1])
    %% Combined Sine Time Domain
    subplot(427)
    plot(y)
    title('Combined Wave')
    %% Combined Sine Frequency Domain
    fft4 = subplot(428);
    plotFFT(y, 50000, fft4)
    title('Combined Sine Wave')
```

FFTRealtime.m

```
clear all; close all; fclose all; delete(instrfind); clc; % Start Fresh
%% Settings
SECONDS_TO_PLOT = 10;
ARDUINO_SERIAL_PORT = 'COM16'; % Change this (Win: 'COM?', MAC: '/dev/cu.usbmodem?')
FFT_PLOT_DURATION = 20;
Fs = 50; % Sampling frequency is 50 Hz

%% Initialize Serial Connection and variables
% Create Serial Object a to connect to arduino (choose com port correctly)
arduinoSerial = serial(ARDUINO_SERIAL_PORT,'baudrate',9600,'InputBufferSize',5120);
fopen(arduinoSerial); % Open Serial communcation
pause(1) % pause 1 second
fgetl(arduinoSerial); % Took out first line
timeArray = 0; sampleArray = 0; % Arrays to store time and sample
i = 0; % Initialize Counter
htic1 = tic; % Initialize timer

%% Setup Figure
fig1 = figure();
set(fig1, 'Units', 'normalized')
set(fig1, 'Position', [0.1,0.1,0.8,0.8])
timeDomainAxis = subplot(311); timeDomainPlot = plot(timeArray, sampleArray); title('Real Time Photoresistor Reading')
frequencyDomainAxis = subplot(312); frequencyDomainPlot = plot(0); title('20 second FFT'),
xlim([0 5]) % Change xlim to zoom in
spectrogramPlot = subplot(313); title('Spectrogram'), xlabel('Time(s)'),
ylabel('Frequency(Hz)')

%% Main Loop
while(ishghandle(fig1)) % loop until the figure windows is closed
    %% Read data from Serial and record
    while arduinoSerial.BytesAvailable %Read until serial is empty
        htic1 = tic;
        while toc(htic1)<0.05 % Read for 0.05 seconds before checking if serial is empty
            %% Read a line in serial and scale the numbers
            [timestr,sensstr] = strtok(fgetl(arduinoSerial),' ');
            timeData=str2num(timestr) / 1000; sensorData=str2num(sensstr) * (5/1023);
            %% Record data into array
            i = i + 1; %Increase counter every time a sample is recorded
            timeArray(i) = timeData; sampleArray(i) = sensorData;
        end
    end

        %% Plot Time Domain
        set(timeDomainPlot,'XData',timeArray,'YData',sampleArray); xlim(timeDomainAxis,[0 timeArray(end)])

        %% Plot Frequency Domain for samples in recent 20s (change this in settings on top)
            %% FFT calculations
            Y = fft(sampleArray(max(1,(i-FFT_PLOT_DURATION*50)):i));
            L = length(sampleArray(max(1, (i-FFT_PLOT_DURATION*50)):i));
            P2 = abs(Y/L);
            P1 = P2(1:floor(L/2+1));
            P1(2:end-1) = 2*P1(2:end-1);
            f = Fs*(0:(L/2))/L;
        set(frequencyDomainPlot,'XData',f,'YData',P1);

        %% spectrogram realtime
        if(i > Fs * 5) % Only start if we have enough data (5s)
            spectrogram(sampleArray,Fs * 5,Fs * 2,i,Fs,'yaxis')
            set(spectrogramPlot, 'Ylim', [0 5]); % Zoom in
        end
```