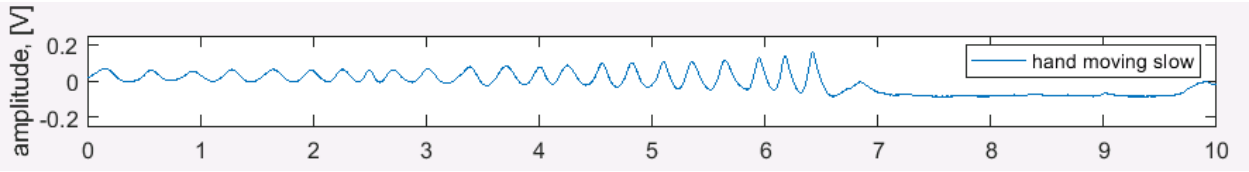
Lab 4 Report

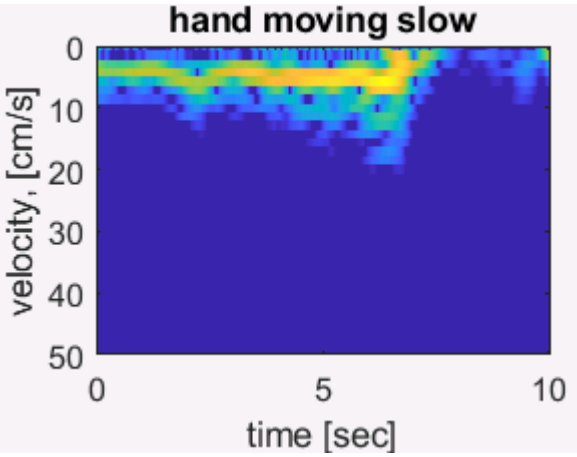
Ryan Nand
ECE332L

First I would like to talk about the overall program before I start explaining the plots and the code. So this program is essentially a Doppler radar system. A Doppler radar system is a system where a signal is generated and bounced off a target which returns to a sensor within the radar system. The system then takes that bounced signal analyzing the changes to the signal which can then explain characteristics of the target object, such as velocity or distance.

In this case, a sensor was placed at an arbitrary origin where the transmitted signal is a 10.525 GHz signal and the object is a person's hand. The hand was placed 30cm away from the sensor. The hand moved toward the sensor in three different velocities (slow, medium, fast) recording the data. The program then takes that data and computes the stfft (short time fast fourier transform). This means that the program takes the bounced signal and computes the fast fourier transform of that signal while only looking at a short time frame within that bounced signal (not the whole signal). Then finally, the program creates two subplots containing information for each velocity type.

Here I would like to explain the plots by velocity type, which may help grasp the differences and explain what information can be dissected. Starting with the hand moving slow.



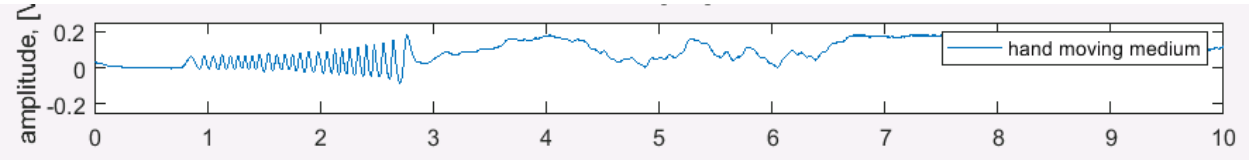


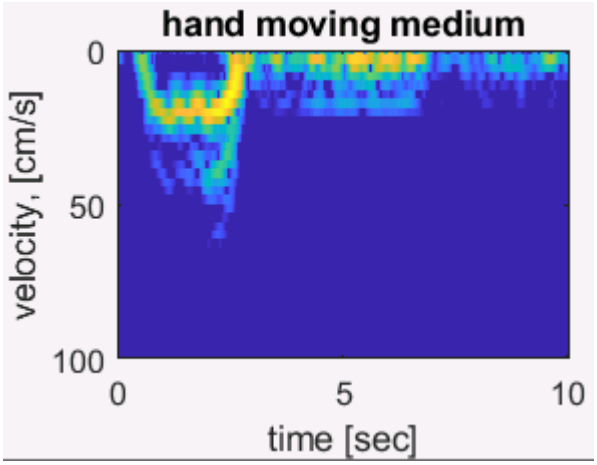
These two plots can tell pretty much the same information, it's just another visual that might help grasp the system. That being said, both of these can tell valuable information. The first one is amplitude[V] vs. time[sec] the second one is velocity[cm/s] vs time[sec] There are four major factors of information that the plots above can tell. I will be explaining using the first of the two plots above.

- The first factor of information is when there are changes in the system/signal or rather when the hand is moving.
- Therefore, the second factor is when there are no changes within the system/signal or rather when the hand does not move.

- The third factor is velocity. Knowing the fact that when the hand reaches the sensor the hand must stop, we know that there will be little to no changes to the signal afterwards. Looking at the first plot we see that movement stops around 7 seconds. Taking the distance and dividing by the time taken, we get a velocity of about 4.3 cm/s. The second plot confirms the velocity found from the first plot.
- Lastly, the fourth major factor is the distance of the object. Notice as the hand gets closer to the sensor as time increases the amplitude increases as well. This complies with the inverse-square law. As the distance from the sensor and hand become smaller the amplitude increases. Therefore, we can tell the distance of the object/hand from the origin. This can also be reflected in the second plot by the color intensity.

Now here are the plots for the system when the hand is moving at a medium velocity.

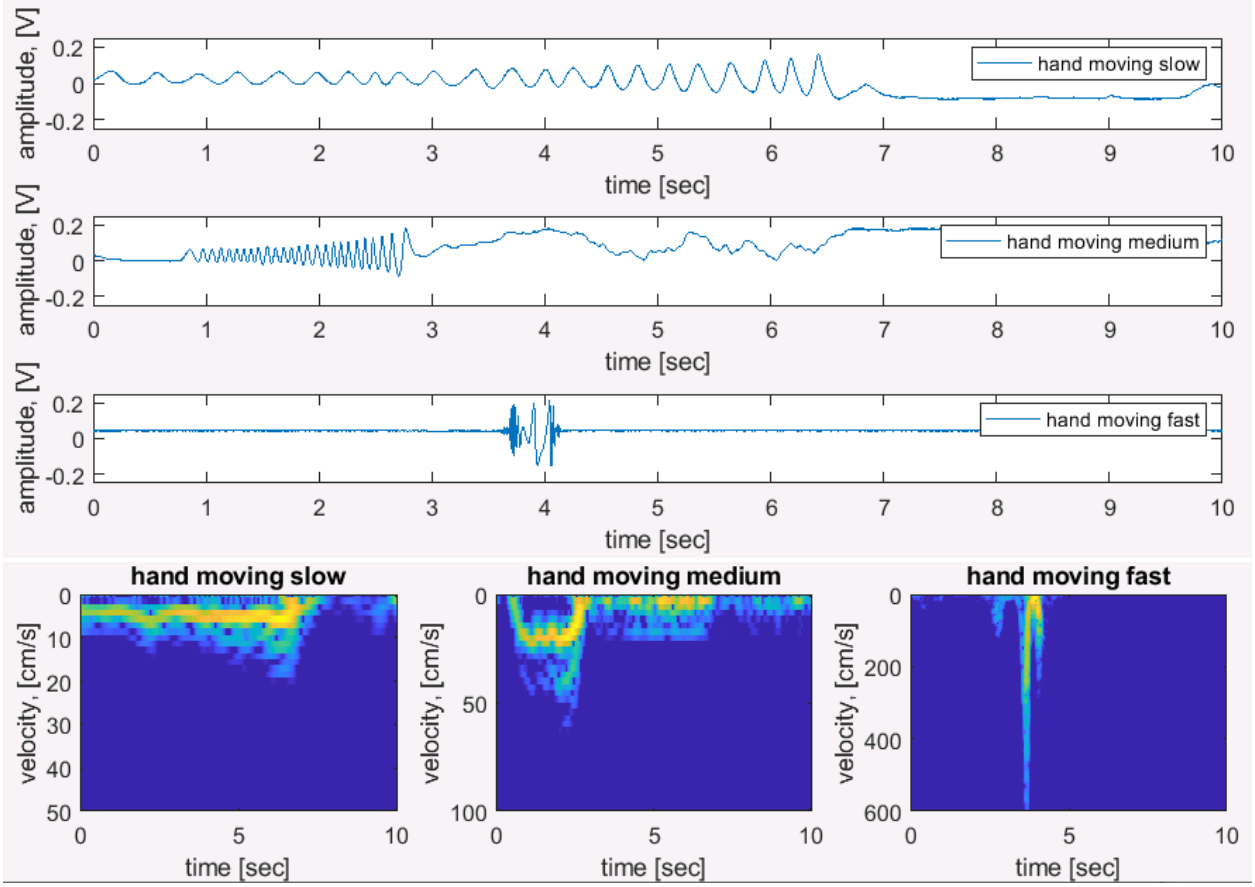




Here we can see that the movement of the hand is between 1 and 3 seconds. A velocity about 15 cm/s. The second plot does not show this well with the y and x axis bounds, however, we can see an increase in velocity in that region of time. We can also see the other three major factors as well.

The big difference between this system and the first system (other than the velocity) is that the hand seems to be kept there for a few seconds where in the first system the hand must have been removed. We can see slight changes in the system between 3 and 7 seconds, which can lead to the earlier conclusion. Vibrations from the hand in trying to keep it still in front of the sensor. Notice that there is an offset from 7 seconds and on. This is from being able to keep the hand essentially still in front of the sensor.

Here are the subplots as the program originally generated including the last velocity (hand moving fast).



Looking at the third velocity, we can see that there was indeed a very fast movement around the four second region. We can also make out a period where there is an amplitude difference to show the distance involved. The two clusters on the ends of the movement might display the hand moving in and out of the system or very rapid hand movement even faster than the intended hand velocity. We can tell the system/sensor is not very accurate with fast movement and would need to record data at a faster rate.

Now I would like to move onto talking about the code, I'll be using snippets of code at a time and progressing as would the processor/compiler.
Starting with the main file:

```
close all
clearvars
clc
```

Closes figures, clears variables, and clears command line history.

```
% fontsize for plots
fs = 12;
%--------------------------------------------------------------------------
% options for doppler function (stfft). The doppler function is a
% short-time fft which calculates the velocity output vector.

windowSize(1) = 10000; % window size used for "hand_moving_slow.mat" data
windowSize(2) = 5000; % window size used for "hand_moving_medium.mat" data
windowSize(3) = 2500; % window size used for "hand_moving_fast.mat" data

shift = 11;     % how much to shift the window between sucessive fft's

doMeanSubtract = 1; % removes dc component (mean) from each chunk of data
% processed in doppler function (stfft)

doWindowData = 1;    % applies hanning window to each chunk of data
                     % processed in doppler function (stfft)

zpad = 2^10;         % zero pad length for stfft
Ft = 10.525e9;       % sensor tranmit freqeuncy [GHz]
```

This is the preallocation of global variables used throughout the program. Some are self-explanatory like the font size or transmit frequency. Some are better explained when used later, however, the program creator added summaries to them which helps.

```
% load data and process and plot raw data results
load('hand_moving_slow.mat', 'data','timeStamps')
```

The program creator starts with loading the data that was stored in ‘hand_moving_slow.mat’ into the variable ‘data’ and a variable ‘timeStamps’. The program creator has formatted the .mat file in a way where the data variable holds the information from the sensor while the timeStamps variable holds the values of time for each data point.

```
[freq_slow, v_slow, data_slow] =...
    doppler(timeStamps,Ft,data,windowSize(1),shift,doMeanSubtract,doWindowData, zpad);
```

This is where the program creator calls the doppler function to compute stfft. This doppler function was created by the program creator as well. Here is the syntax for the function:

```
function [Fd, v, data] = doppler(tm,Ft,sig,win_size,shift,wind_flag,mean_remove_flag, zpad)
```

The function returns three variables, ‘Fd’(a frequency vector), ‘V’(a velocity vector), and ‘data’(stfft data). The arguments are ‘tm’(range of time for data points), ‘Ft’(transmit frequency), ‘sig’(data from sensor), ‘win_size’(size for windowing), ‘shift’(shifting of the window), ‘wind_flag’(apply hann window), ‘mean_remove_flag’(removes dc component from data), ‘zpad’(zero padding of window).

This function call uses the majority of the preallocated variables from before. The ‘wind_flag’ and ‘mean_remove_flag’ are either set or not. Set for go ahead and apply the intended action, such as, apply the hann window or remove the dc component. **Also, if looked at real closely, the program creator called the function in the main with the intended variables in each other's spot. However, since they are both set it does not matter.**

I'll talk about the other arguments as we proceed through the created function.

```
t = tic;
display('processing doppler data...')
```

This creates a variable t that starts recording elapsed time (tic is a matlab function that records the current time). The next line displays to the user that the program is processing the doppler data. The purpose of this is to show the user elapsed time of the program and when the program starts to process.

```
if shift >= win_size
    error('shift  must be less then win_size')
end
```

This is an error check to see if the shifting is greater than the window size or not. Sends out an error message if so. Shifting must be less than the window size, otherwise, there is no need to shift. The purpose for this is to compute the fft using chunks of data seen later in the program. So there is no need to shift the whole window, because that does not make sense. Essentially, we can think of the window as the length of the data intended to be plotted.

```
j = win_size;
k = 1;
m = 1;
steps = floor((length(sig)-win_size)/shift);
```

Created local variables to use in the function. The 'steps' variable creates equally spaced divisions of the difference between the signal length and the window size. This creates the divisions to perform the stfft with. The function 'floor' is a matlab function that simply rounds down to the next integer complex number. The 'j' and 'k' variables are used for indexing arrays and processing the 'for' loop.

```
if zpad < win_size
    zpad = win_size;
end
```

Made sure that the zero padding variable is the size of the window. Zero padding is a technique to make the visual of the signal look more "informational". It does not make the signal more accurate. There is no point of the zero padding range to be greater than the window (cannot see the effect outside the window), therefore, the size of the window should be the upper bound.

```
% calculate frequency vector
dt = abs(tm(1)-tm(2));
T = dt*zpad;
df = 1/T;
Fd = 0:df:df*(zpad-1);
```

The first line takes the magnitude of the difference between the first two timestamps. In this case, the result is 1e-4. Then multiplied by the zero padding constant which is essentially the window size because the code before this equates the two in all three cases. This is then used to create a constant that will be used to create an array of that size. The end result for all three cases is an array by their respective window sizes (1x10000 for slow) that contain data points which all add up to df*(zpad-1). This makes the data points of the frequency vector add by df. For example, slow adds by one ending in 9999 whereas the fast adds by four and ends in 9996, both starting at zero.

Not quite sure why this variable is returned to the main, because it is never used again afterwards. But it is used in the next snippet of code.

```
% calculate the velocity vector
v = doppler_velocity(Ft,Fd);
```

Another call to a function created by the program creator. This function returns a vector containing velocities. The following is the function syntax.

```
function  [v] = doppler_velocity(Ft,Fd,theta)
```

The theta is optional according to the program creator. The function contains the following code.

```
% electromagentic propogation speed in vacuum in [m/s]
c = 299792458;

if nargin == 2
    theta = 0; % angle between transmitter/receiver and target
end

v = Fd.*c./(2.*Ft.*cos(theta));
```

The local variable ‘c’ has a comment that explains it’s content. The line after checks the number of arguments in the function call. In this case, it is two and now theta is equal to zero. Therefore v = Fd*c/(2*Ft) where everything is a constant except for the frequency array (‘Fd’). ‘Ft’ is the transmit frequency of 10.525 GHz. For the three cases, the result is an array that starts at zero and ends near 142 give or take the same variation within the ‘Fd’ array in each case. This array is then returned to the calling program. If you notice the calling program then returns this array to the main program as well.

Back into the calling program or the function ‘doppler’.

```
% allocate memory
data = zeros(zpad,floor(steps));
```

A preallocation of zeros using an array with the first argument being ‘zpad’ or in this case the window size. The second argument is the number of divisions rounded down to the nearest integer complex number. In the end, this creates an array of zeros with the intended size for preallocation.

```
% perform short time fft
for ii = 1:steps

    %-----------------------------------------------
    % select a chunk of data to perform fft on
    sig0 = sig(m:j);
    %-----------------------------------------------
    % remove dc (mean) offset from chunk
    if mean_remove_flag == 1
        sig0=sig0-mean(sig0);
    end
    %-----------------------------------------------
    % window data to reduce sidelobe ringing
    if wind_flag == 1
        wind = (hann(length(sig0)));
        sig0=sig0.*wind.';
    end
    %-----------------------------------------------
    % fft data and zero pad
    data(:,k) = ifft(sig0,zpad);
    %-----------------------------------------------
    % increment counters and shift window
    j = j + shift;
    m = m + shift;
    k = k + 1;

end
```

Here is the main computation for this ‘doppler’ function or rather when the stfft happens. This ‘for’ loop will execute for each division created earlier within the ‘steps’ variable. The variable ‘sig0’ takes the call argument ‘sig’ and indexes using ‘m’ and ‘j’ from earlier. Basically, taking chunks of data for each division from the data signal provided by the call in the main program. The next two ‘if’ statements checks to see if the ‘wind_flag’ and ‘mean_remove_flag’ variables are set. If so they will do their respective action, remove dc offset and apply the hann window. This is done by a predefined matlab function, ‘mean’. Also, by a created ‘hann’ function. For dc removal, simply eliminate the mean from the signal. For applying the window, use the length of the data chunk and multiply each data point by the result of using matlab ‘hann’ function.
Here is the syntax for the ‘hann’ function call.

```
function [ wind ] = hann(windowLength)
```

Here are the contents of the created function.

```
if nargin == 0
    windowLength = 100;
    N = windowLength - 1;
    num = linspace(0,N,windowLength);
    wind = 0.5*(1 - cos(2*pi*num/N));
    figure
    plot(wind)
    title('hann window')
else
    N = windowLength - 1;
    num = linspace(0,N,windowLength);
    wind =  0.5*(1 - cos(2*pi*num/N));
end
```

If there are zero arguments in the call, the function creates a predefined window and plots it. However, in this case there will always be an argument not equal to zero, therefore, the 'else' statement applies. The 'else' statement creates a variable that takes the window length and subtracts one and stores it in 'N'. Then creates equally spaced linear points from zero to 'N' with 'windowLength' rows. Lastly, the function uses the Hann window expression to compute the variable that is needed to apply the window to the data in the calling program, returning 'wind'

Back to the calling program in the 'for' loop within the 'doppler' function using the code above from before.
Proceed or finish the computation of the stfft by a predefined matlab function called 'ifft' (inverse fast fourier transform). The two arguments are the data itself and n for the n-point inverse transform (number of trailing zeros i.e. zero padding). Store the results in a data array which will be returned to the calling program.

Afterwards, proceed the 'for' loop by shifting the indexes (indexes for accessing data chunks and storing results in array).

```
display(sprintf('processing complete: %g [sec]',toc(t)))
```

Once the 'for' loop completes, the function displays to the user that processing is complete and the elapsed time using matlab's 'toc' function. The program then returns to the calling program. Here is the code after the 'doppler' function call in the main program.

```
figure
subplot(3,1,1)
plot(timeStamps,data)
legend('hand moving slow')
xlabel('time [sec]','fontsize',fs)
ylabel('amplitude, [V]','fontsize',fs)
set(gca,'fontsize',fs)
axis([0 10 -.25 .25])
drawnow
```

This is simply the code to plot the data and configure the plot to one's liking. Adding labels for the axes and including a legend while changing the font size. Also, plotting within certain limits. Limits defined by the program creator's preference.

This is the end for the slow hand portion. The program repeats everything for the medium and fast hand cases. The code is below.

```
% load data and process and plot raw data results
load('hand_moving_medium.mat')

[freq_medium, v_medium, data_medium] =...
    doppler(timeStamps,Ft,data,windowSize(2),shift,doMeanSubtract,doWindowData,zpad);


subplot(3,1,2)
plot(timeStamps,data)
legend('hand moving medium')
xlabel('time [sec]','fontsize',fs)
ylabel('amplitude, [V]','fontsize',fs)
set(gca,'fontsize',fs)
axis([0 10 -.25 .25])
drawnow
```

```
% load data and process and plot raw data results
load('hand_moving_fast.mat')

[freq_fast, v_fast, data_fast] =...
    doppler(timeStamps,Ft,data,windowSize(3),shift,doMeanSubtract,doWindowData,zpad);

subplot(3,1,3)
plot(timeStamps,data)
legend('hand moving fast')
xlabel('time [sec]','fontsize',fs)
ylabel('amplitude, [V]','fontsize',fs)
set(gca,'fontsize',fs)
axis([0 10 -.25 .25])
drawnow
% set plot size
set(gcf,'position',[504          567         1059          420])
```

This last line above configures the whole subplot. Making the figure into a decent size for a better visual.

Next the program moves on to plot image graphics or the plots with color intensity instead of amplitude.

```
figure
subplot(1,3,1)
imagesc([timeStamps(1) timeStamps(end)],v_slow*100,imnorm_db(data_slow),[-35 0])
title('hand moving slow','fontsize',fs)
xlabel('time [sec]','fontsize',fs)
ylabel('velocity, [cm/s]','fontsize',fs)
set(gca,'fontsize',fs)
ylim([0 50])
drawnow
```

This is for the slow hand case. As we can see there is only the plotting and plot configurations.

Since we already computed the stfft, not much else computation wise is needed. However, there is a function call within the matlab 'imagesc' function (a predefined matlab function that produces a colored image like the plots above with intensity color levels for amplitude) . The arguments for this matlab plot function is time interval (brackets containing the 'timeStamp' variables, the beginning and the end of the array), the velocity vector from the 'doppler' function, the data array from the 'doppler' function using created function call, and the last one is for limiting the color intensity range from -35 to 0. The created function call is 'imnorm_db' and the code is below.

```
function datan = imnorm_db(data)
datan = abs(data);
datan = datan-min(datan(:));
datan = datan/max(datan(:));
datan = 20*log10(datan);
```

The purpose of this function is simply to convert the data points to dB. First get the absolute value for each point. This is done by the predefined matlab 'abs' function. Then we must normalize the data. The next two lines do this action for each data point, I believe this normalization type is called "feature scaling". Then the last line is the expression to convert the data to the dB scale. The returned value is the normalized dB scaled data.

Back to the calling program. The program does this for the medium and fast hand cases as well.

```
subplot(1,3,2)
imagesc([timeStamps(1) timeStamps(end)],v_medium*100,imnorm_db(data_medium),[-35 0])
title('hand moving medium','fontsize',fs)
xlabel('time [sec]','fontsize',fs)
ylabel('velocity, [cm/s]','fontsize',fs)
set(gca,'fontsize',fs)
ylim([0 100])
drawnow

subplot(1,3,3)
imagesc([timeStamps(1) timeStamps(end)],v_fast*100,imnorm_db(data_fast),[-65 0])
title('hand moving fast','fontsize',fs)
xlabel('time [sec]','fontsize',fs)
ylabel('velocity, [cm/s]','fontsize',fs)
set(gca,'fontsize',fs)
ylim([0 600])
drawnow

%------------------------------------------------------------------------
% set plot size
set(gcf,'position',[506         231        1056         230])


display('processing completed.')
```

Notice that there are variations amongst the three cases to cater to the plot range/data. This is all done by the program creator's preference. In other words, the program creator added variations in the configurations of the plot for what was thought to put out the best results, no other logic. Also, things such as the title will be different for each case. The last two lines of the

program are the sizing of the whole subplot for a better visual and the print statement saying that the program has completed processing, respectively.

That is the end of the program.