How things work: FMCW RADAR

Frequency **M**odulated **C**ontinuous **W**ave Radio Detection and Ranging

Useful References:

At the end of this exercise you will be able to:

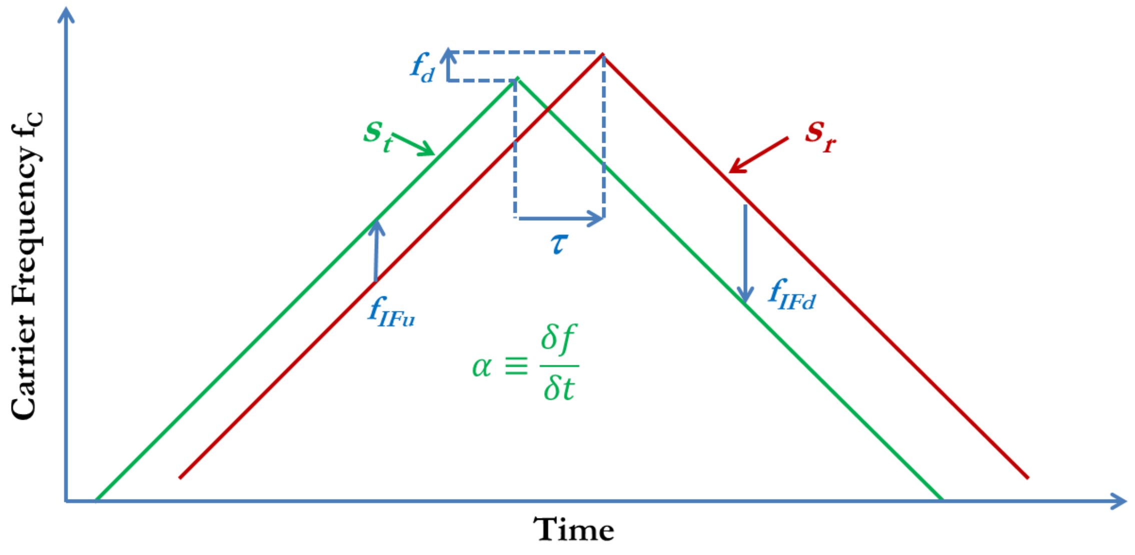
- Read data from an audio file
- Format and process data in a known format
- Plot data with defined scales
- Understand the concept of spherical spreading of signals and compensation for it
- Understand the concept of a noise floor
- Understand how to take a FFT
- Plot 3D images

Christian Hulsmeyer invented radar to avoid ship collisions in fog. During the run up to World War II development proceeded rapidly in countries on both sides of the engagement. The term RADAR was coined by the US signal corps. Key development was the Magnetron which generated microwaves. The Magnetron development in Russia and the UK in 1940 made RADAR small and practical. The biggest issue had to be that electronics to build systems and process data was in its infancy.

There are two major classes of RADAR, pulsed and continuous wave. Pulsed RADAR is designed to send out a pulse and then detect pulses reflected back from targets down range. The time delay would indicate distance, and the frequency content of the pulse the ~velocity and material of the target.

Continuous wave RADAR can be used to accurately determine velocity and range of a target as well using a chirp signal. We will use a Frequency Modulated Continuous Wave (or FMCW) RADAR. The idea of FMCW is to ramp the frequency over time and then compare the frequency generated to the frequency that is reflected back from targets. This might seem at first glace to be ideal for automotive collision avoidance RADAR system. Several commercial systems use FMCW.

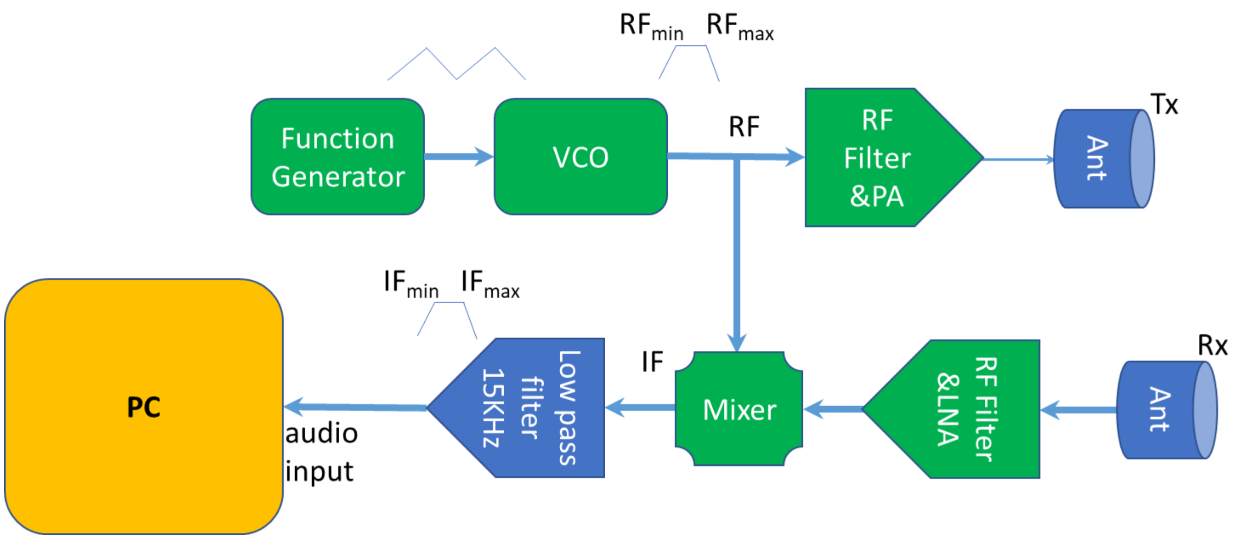
Let's look at the method. First the system slowly ramps the transmitted frequency up and then down, this is the green curve. Remember this is a plot of frequency versus time not amplitude. The ramps used are linear with a ramp rate of alpha Hz/sec. A receiver detects the signal reflected back from targets. First let's consider a stationary target. The received signal is in red and note it is delayed in time by tau which is the transit time to and from the target, which gives range.



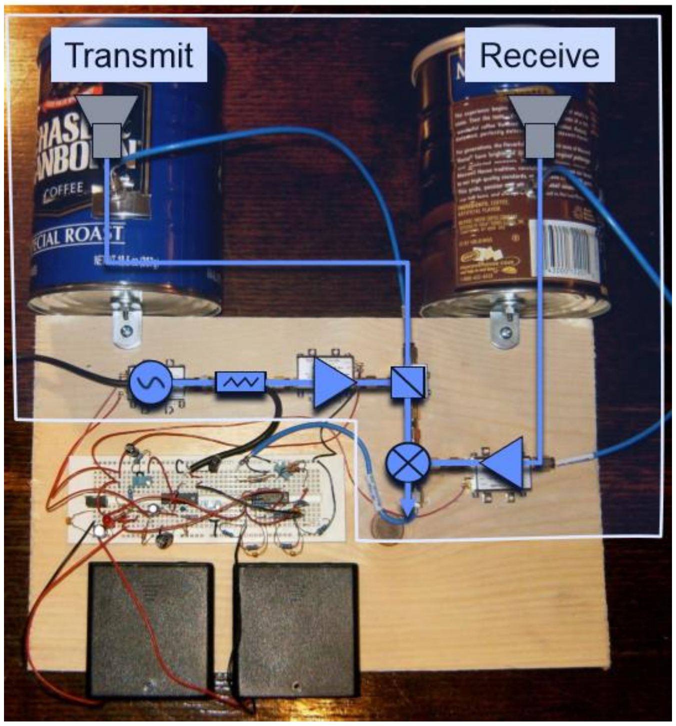
The frequency of the signal transmitted at a point in time and received at that point in time are different by the time associated with the travel from the transmitter to the target and back. Taking these two signals into a mixer we can get the difference in the frequency because of the delay in time. f_{IFu} is the difference in frequency as the frequency is ramped up as a function of time and f_{IFd} is the difference as the frequency is ramped down.

If there the target is not stationary, then there is also a frequency shift f_d (doppler shift).

Let's look at a diagram of the components of this system



Block diagram of a FMCW radar system with the data collected through the PC audio input along with the modulation data. We can use a computer program to organize and process this data.



Picture of our system showing how these components match to the block diagram.

Based on this arrangement we can calculate the range and velocity from the frequencies f_{IHu} and f_{IHd}, and the other parameters of the system.

R is the range, BW the band width

T_s is the sampling time,

c is the speed of light, v_r velocity

$$BW = F_{\max} - F_{\min},\ \lambda = {}^{2c}\!\big/\!\left(F_{\max} + F_{\min}\right)$$

$$f_b = \frac{BW}{T_s}\frac{2R}{c}$$

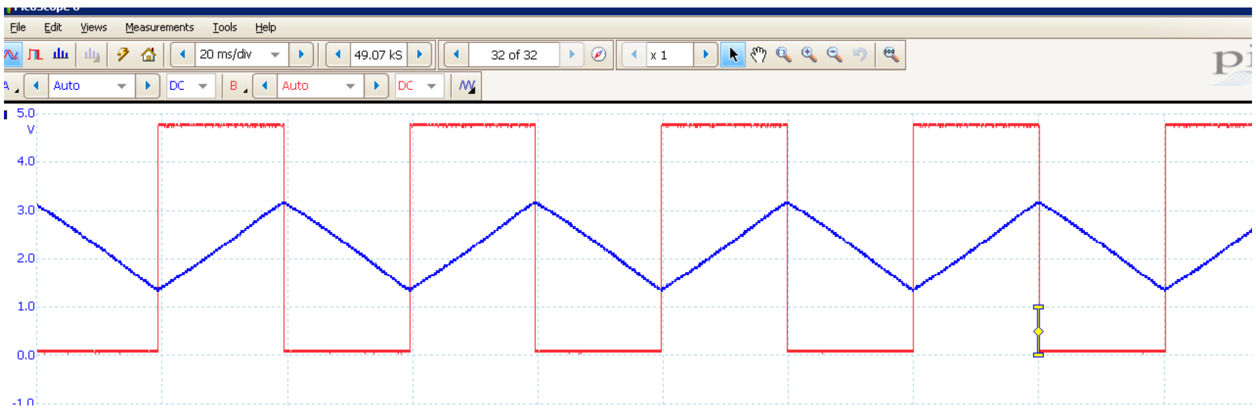
$$f_d = \frac{2v_r}{\lambda}$$

$$f_{IHu} = f_b - f_d,\ f_{IHd} = f_b + f_d$$

$$R = \frac{cT_s}{4BW}\left(f_{IHu} + f_{IHd}\right)$$

$$v_r = \frac{\lambda}{4}\left(f_{IHd} - f_{IHu}\right)$$

This is the signal from the modulator, the red curve is recorded through the audio input and the blue curve goes to the VCO generating the frequency dependent signal. The output of the VCO (Voltage Controlled Oscillator) is called a chirp signal. For our dataset try Fmax = 2.47 GHz, and Fmin = 2.34 GHz.



You may read the data in using these commands

```
import scipy.io.wavfile as wavfile

## read wav file

def read_wav(filename):

   [Fs,Y] = wavfile.read(filename)

   print( 'Fs %d', Fs)

   return [Fs,Y]

[Fs,Y] = read_wav('Rec_2ch_4_away_back.wav')
```

Fs is the frequency of sampling of the data, which is based on the audio system and is typically 44.1 kHz. Y is the data from the .wav file inside the quotes. This is just an example.

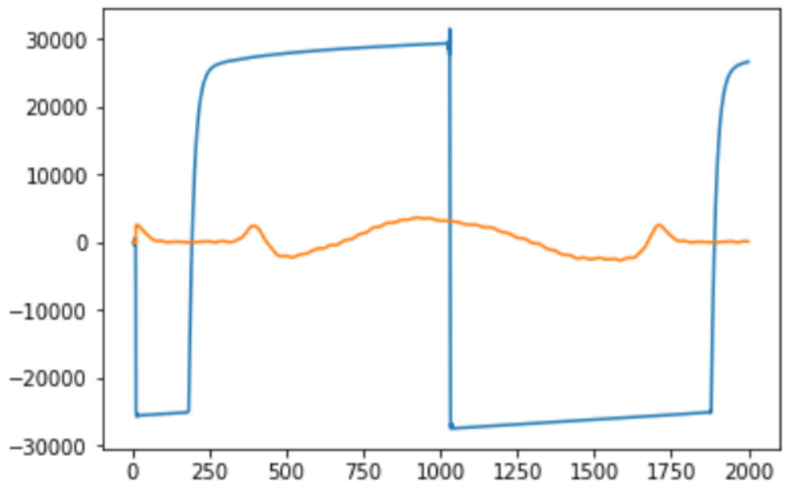


Figure 1 Example data

First 2000 samples of example data, blue is the sync pulse and the red the signal data. The positive pulse is when the ramp up in frequency occurs, and the negative curve is when the ramp down in frequency occurs. If we want to find when each sweep up occurs get the index when the blue curve crosses through 0 going to a positive number. If we want to find when each sweep down occurs get the index when the blue curve crosses through 0 going to a negative number.

Using this place, the signal in a 2D matrix, the even rows are sweeps up, the odd are sweeps down. Take the Fourier transform of each row. Note that if you want to take the fast Fourier transform (FFT) for a sequence of data point you need to pad it to 2^n points (with zeros) which can be done with numpy's pad routine. The FFT algorithm assumes 2n datapoints. The numpy library also has a FFT algorithm named fft, and will pad out to that with zeros automatically saving you a step. For best results you should also window the data using a Hamming or Hanning window (also in numpy).

Remove spherical spreading, this is the reduction in amplitude due to the signal spreading out. This is done by Sig[i] = Sig[i] * pow(i,alpha_) where alpha_ is about 2.0 where i is the index which is proportional to the range. This will return the signal to an amplitude that is not decaying, but when the noise is larger than the signal the amplitude to increase rapidly. Fortunately, this indicates the location of the noise floor.

This plot shows how the data blows up when you go below the noise floor. This is because the signal drops off due to spherical spreading, but noise does not drop off due to spherical spreading, so it just blows up.

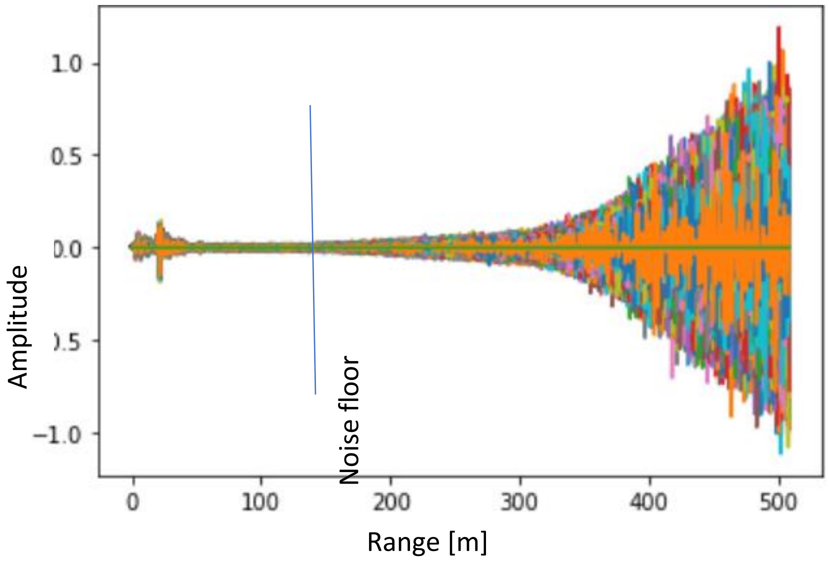


Figure 2. Example data

Produce a plot similar to this one, your colors may be different.

This axis is the time of the recording of each sweep

You can use an arbitrary color, amplitude scale for the result.

This is example data, your own data may be somewhat different

The range is the distance to each light color indicating a target reflection. It has been limited to about 30 meters after which in this case it was very noisy because of the low power used.

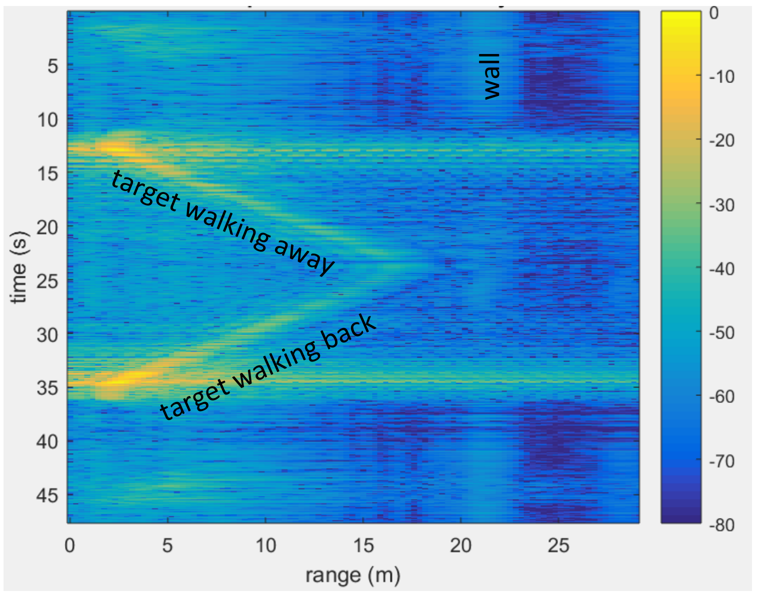


Figure 3 example data

Use the equations above. Now that you have the data in a 2D matrix, the sum of the odd and even rows should be used to calculate the range to each target, call this Figure 4, and the difference should be used to calculate the velocity of each target, call this Figure 5. These two plots should have half the rows as the one in figure 3.

Deliverables:

1) You should deliver your Python code

2) Your plot similar to Figure 1
3) Your plot similar to Figure 2
4) Your plot similar to Figure 3, along with your estimate of the distance to the noise floor.
5) Your plot Figure 4 along with estimates of range to each stationary target.
6) Your plot Figure 5 along with estimates of the velocity of each moving target.

For credit, code must be well commented and organized, everything must be clearly labeled.

Remember our coding standard

- **Write a description of what each script does in comments at the beginning**
    - **this description should describe**
        - **input variables and how to run the code**
        - **output variables and how to interpret them**
    - **each plot should be labeled with a title, and a axes labeled as well**
    - **each function should have**
        - **a comment describing what the function does,**
        - **a brief description of each input variable,**
        - **how to run the function, and**
        - **what the result is and how to interpret it**

an example heat map function I used, note that this is for a real matrix, you may want to take the absolute value for complex matrices. Nx and Ny are dimensions, vmin and vmax can be used to adjust the color scale.

```
def imagesc(data,Nx,Ny,maxtime,maxrange):
    fig, ax = plt.subplots()
    amax = int(np.max(data))
    amin = int(np.min(data))
    print('amp min ', amin, 'amp max ', amax)
    im = ax.imshow(np.transpose(data), cmap=plt.get_cmap('hot'),
                interpolation='nearest', vmin=amin, vmax=amax,
                aspect='auto',extent=[0,maxrange,maxtime,0])
    fig.colorbar(im)
    plt.xlabel('Range[m]')
    plt.ylabel('Time[s]')
    plt.show()
    return
```

function called like this
Sig is the data file, Nx, and Ny are dimensions, time_range is the time over the total recording, and max_range is the distance
imagesc(Sig,Nx,Ny-2,time_range,max_range)