CSE5010 Wireless Network and Mobile Computing Fall2022

Lab3

Acoustic Sensing Using Your Smartphone

&

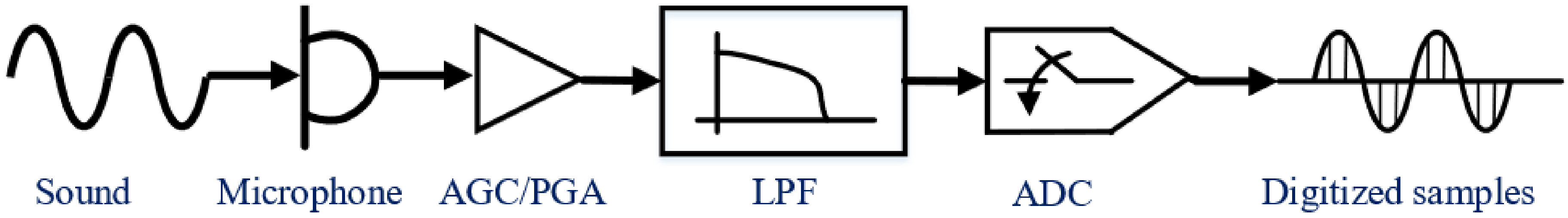
Phase-based Distance Tracking Theory

&

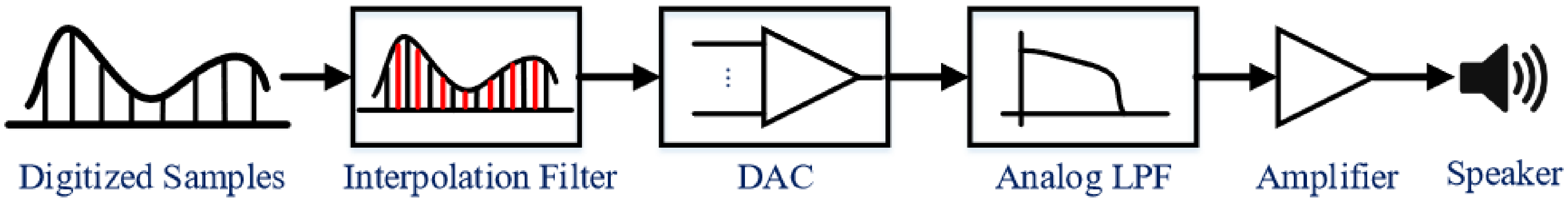
C-FMCW Based Distance Tracking

ACOUSTIC SENSING USING YOUR SMARTPHONE

Acoustic Hardware



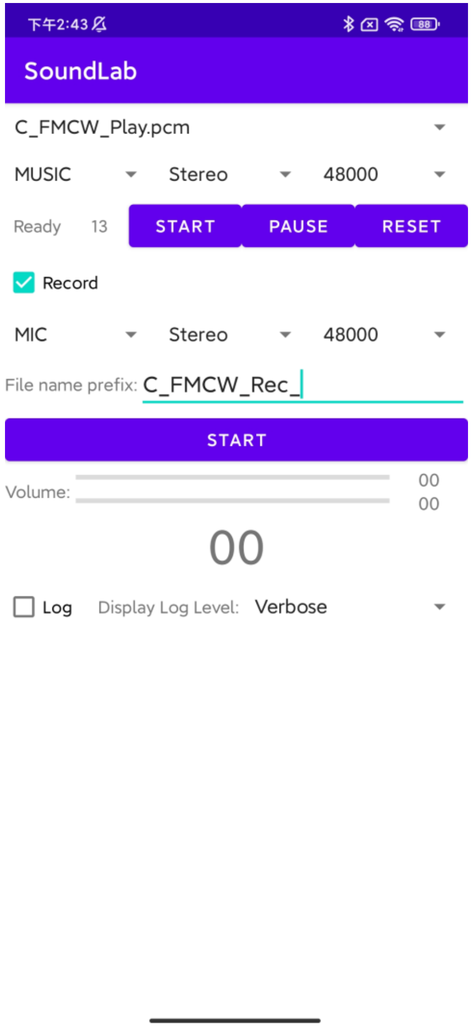
(a) Sound recording system.



(b) Sound playback system.

Fig. 2: Diagrams for typical acoustic hardware.

SoundLab for Android



GitHub link:
https://github.com/LinLin1230/SoundLab

Sampling rate: 48e3 Hz

Mono: 单声道
Stereo: 双声道

MUSIC: 上下扬声器
CALL: 听筒

PCM: Pulse-code modulation

PCM Playback

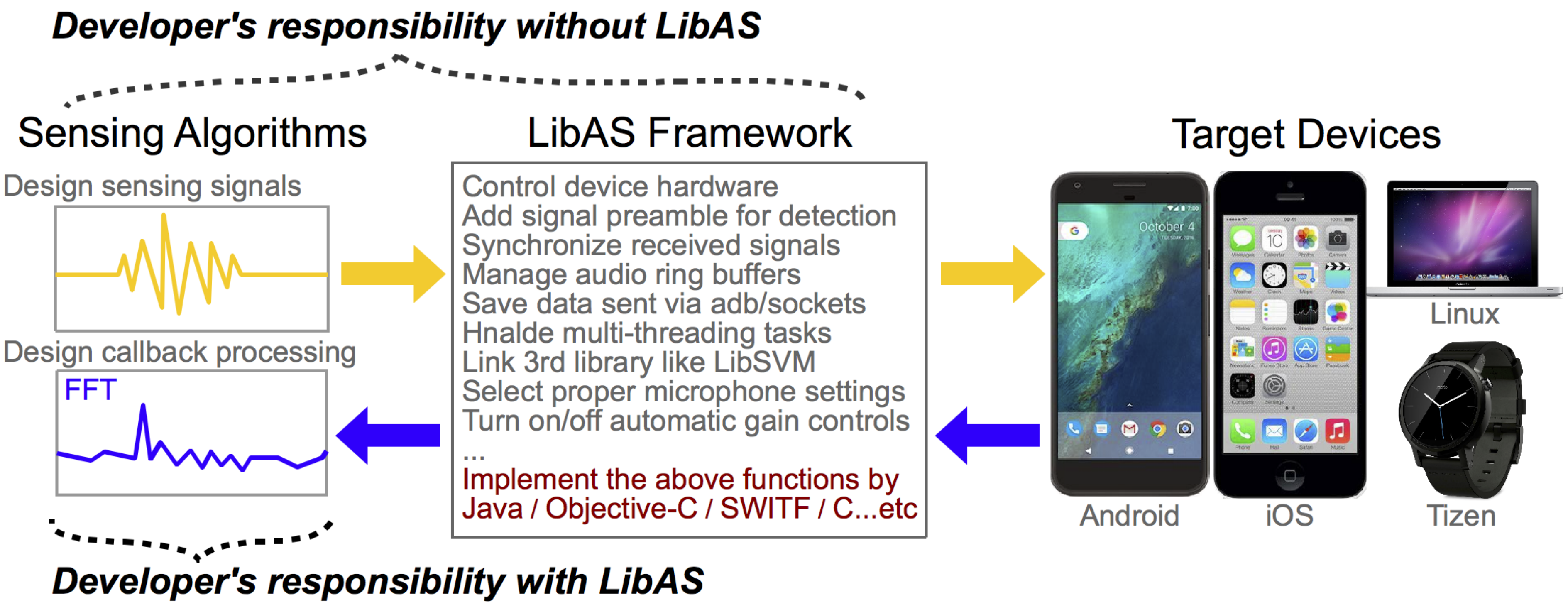
Put pcm file in the path ~/SoundLab/Playlist

PCM Recording

Recorded PCM files are stored in the path ~/SoundLab/

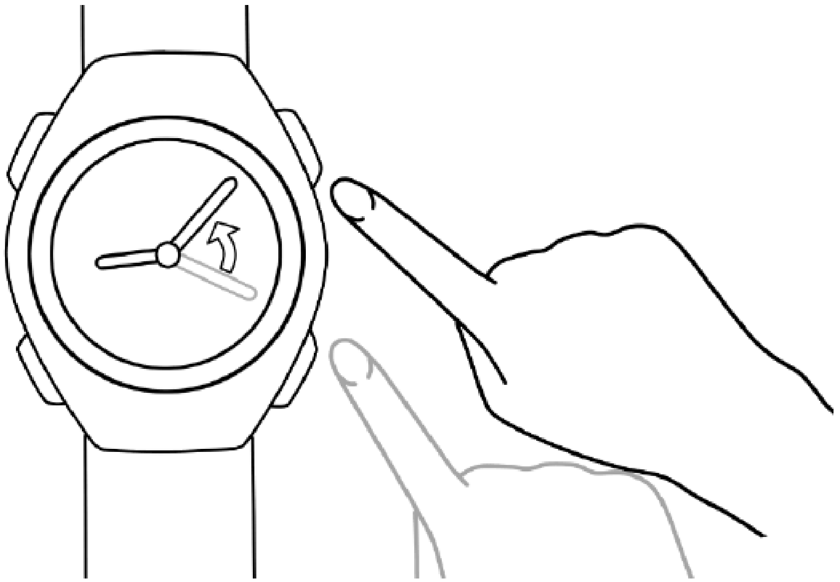
LibAS for IOS (Optional)

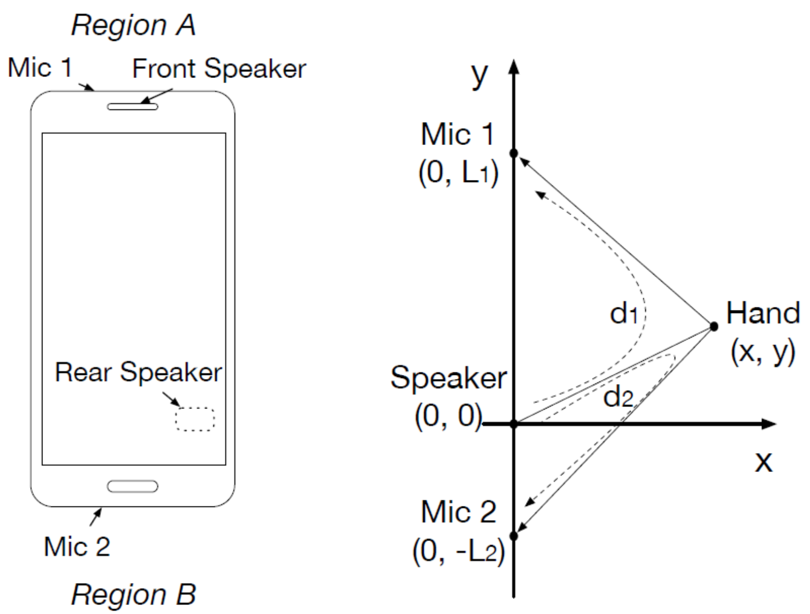
GitHub link:
https://github.com/yctung/LibAcousticSensing



PHASE-BASED DISTANCE TRACKING THEORY

Device-free Gesture Tracking



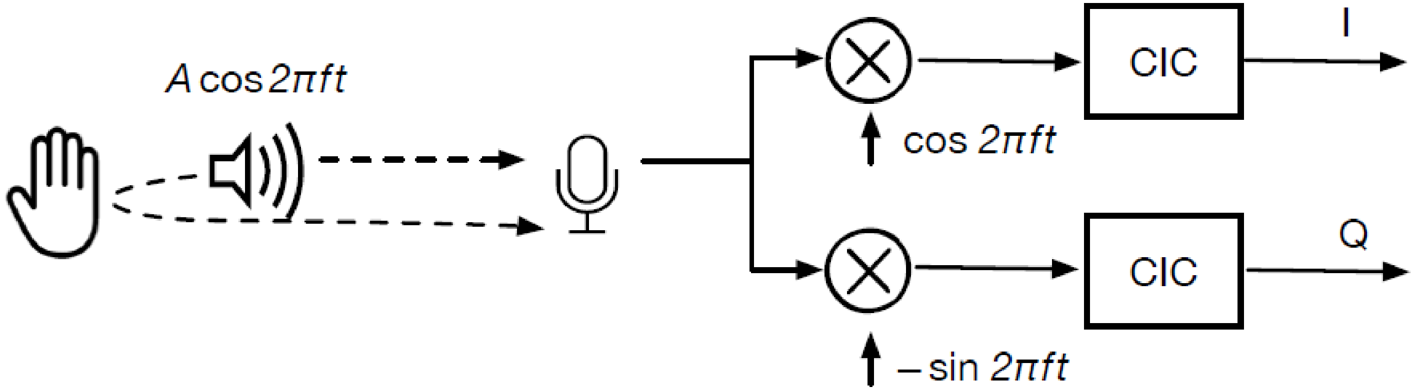


(a) Layout of Samsung S5 (b) Geometric abstraction

Figure 9: Two dimensional tracking

Acoustic signals are transmitted by speaker, reflected by hand/finger, received by microphone.
Use the link below to see a demo:
https://www.youtube.com/watch?v=gs8wMrOSY80

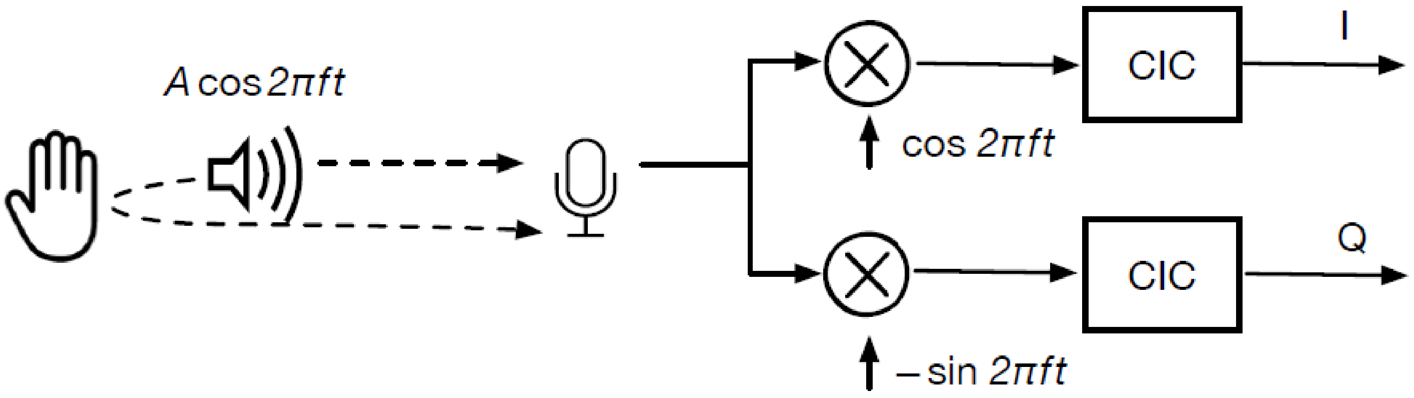
Phase-based Distance Tracking



coherent detector structure

- The sound reflected by a human hand is coherent to the sound emitted by the mobile device. They have a constant phase difference and the same frequency.
- We use a coherent detector to convert the received sound signal into a complex-valued baseband signal.

Phase-based Distance Tracking



coherent detector structure

Transmitted signal:

$$A\cos(2\pi ft)$$

Received signal (after reflection via path p):

$$2A_p' cos\left(2\pi ft - 2\pi f\frac{d_p(t)}{c} - \theta_p\right)$$

$2A_p'$ – *amplitude of the received signal*
$d_p(t)$ – *propagation distance of path p*
c – *sound speed*
θ_p – *phase caused by the hardware delay and phase inversion due to reflection*

Phase-based Distance Tracking

- Received signal (after reflection via path p):

$$2A_p' cos(2\pi f t - 2\pi f \frac{d_p(t)}{c} - \theta_p)$$

- Multiply this received signal with $cos(2\pi f t)$:

$$\cos\alpha\cos\beta = \frac{1}{2}[\cos(\alpha+\beta)+\cos(\alpha-\beta)]$$

$$2A_p' cos(2\pi f t - 2\pi f \frac{d_p(t)}{c} - \theta_p) \times cos(2\pi f t)$$

$$= A_p'(\underbrace{cos(-2\pi f \frac{d_p(t)}{c} - \theta_p)}_{\text{low-frequency part}} + \underbrace{cos(4\pi f t - 2\pi f \frac{d_p(t)}{c} - \theta_p)}_{\text{high-frequency part}})$$

- A low-pass filter is then applied, the result is the In-phase signal:

$$I_p(t) = A_p' cos(-2\pi f \frac{d_p(t)}{c} - \theta_p)$$

Quadrature signal are derived by multiply the received signal with $-sin(2\pi f t)$:

$$Q_p(t) = A_p' sin(-2\pi f \frac{d_p(t)}{c} - \theta_p)$$

$$\cos\alpha\sin\beta = \frac{1}{2}[\sin(\alpha+\beta)-\sin(\alpha-\beta)]$$

Phase-based Distance Tracking

$$I_p(t) = A'_p cos(-2\pi f \frac{d_p(t)}{c} - \theta_p)$$
$$Q_p(t) = A'_p sin(-2\pi f \frac{d_p(t)}{c} - \theta_p)$$

- Combining these two components as real and imaginary part of a complex signal, we have the complex baseband as follows:

$$B_p(t) = A'_p e^{-j(2\pi f \frac{d_p(t)}{c} - \theta_p)}$$

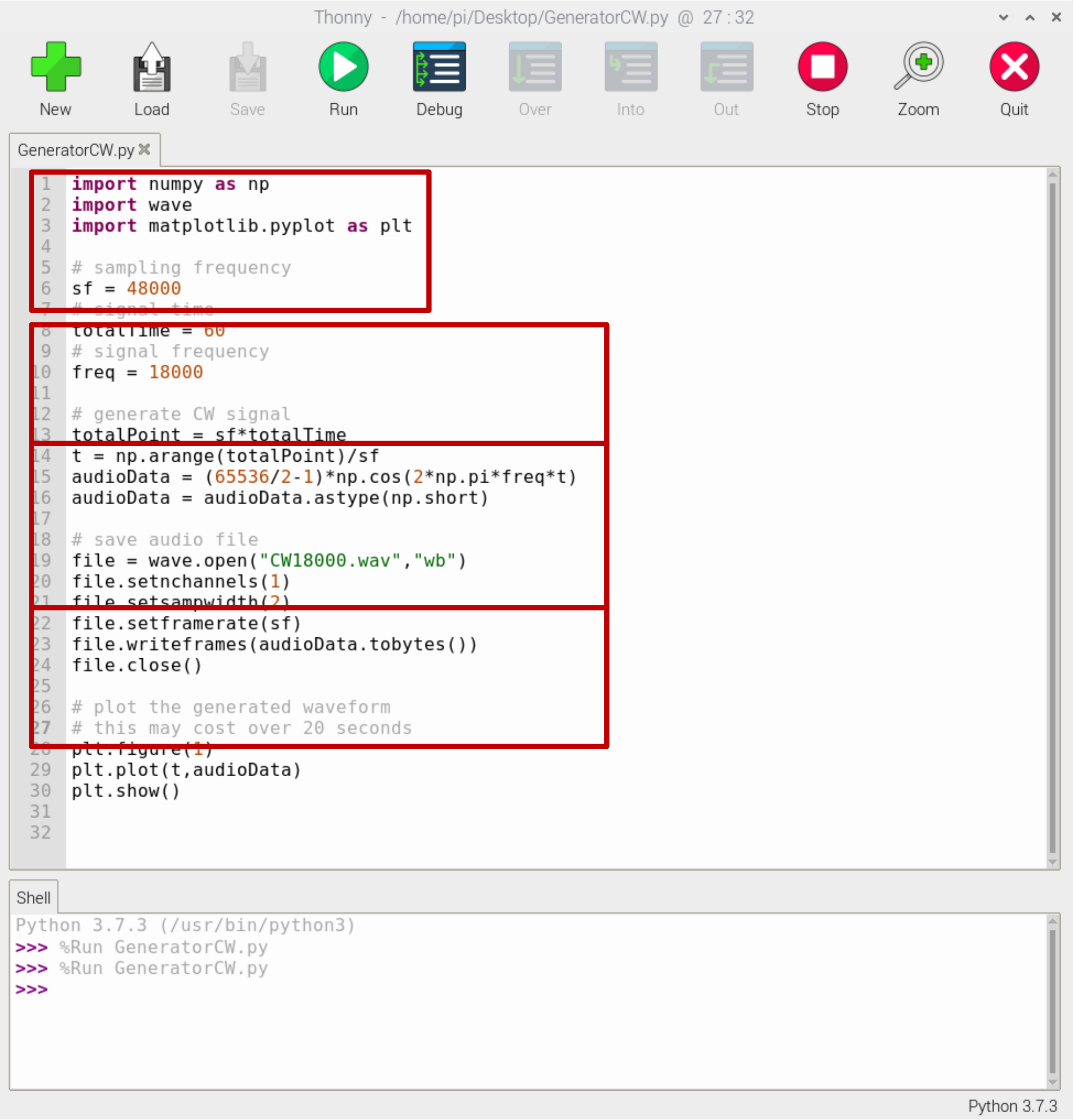
- The phase of it is:

$$\phi_p(t) = -2\pi f \frac{d_p(t)}{c} - \theta_p$$

- Note that when $d_p(t)$ changes by the amount of sound wavelength $\lambda = \frac{c}{f}$, the phase changes by 2π.

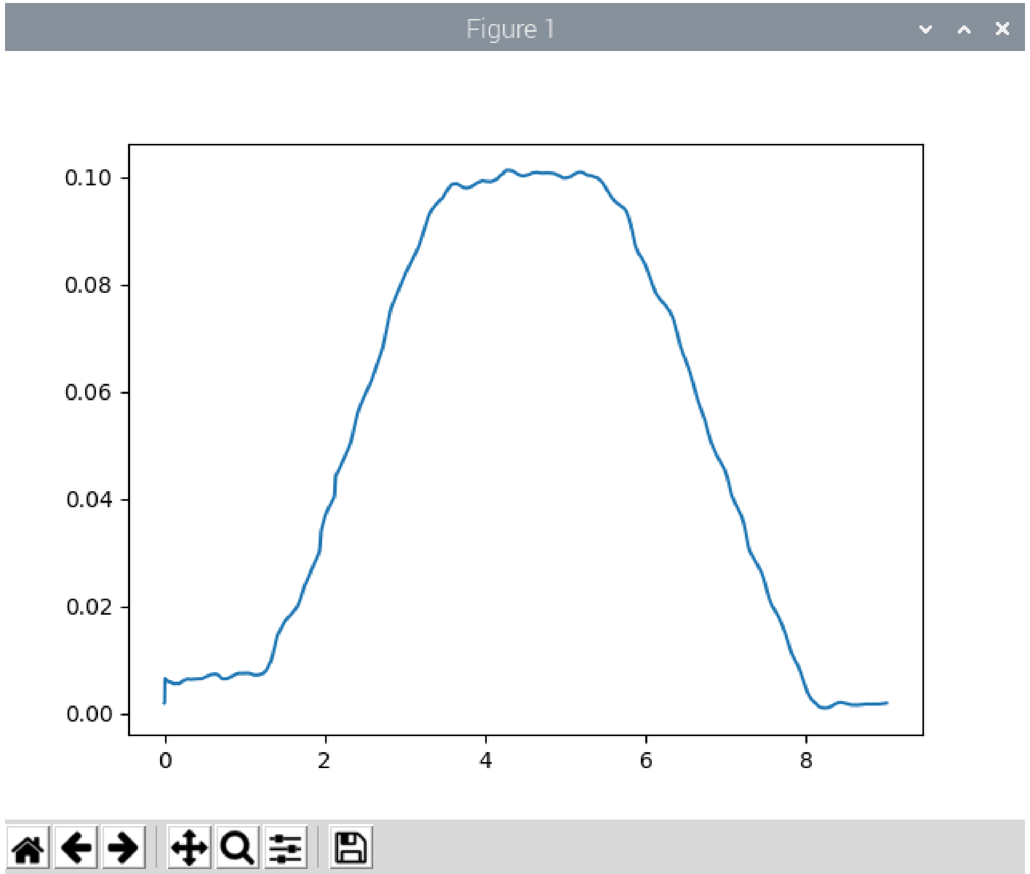
GenerateCW.py

- We use this file to generate a CW file.
- You can change “***frequency***” and “***totalTime***” to set signal frequency and signal time.



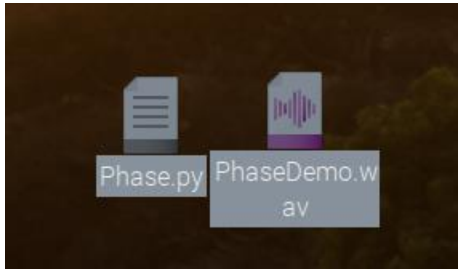
A Demo

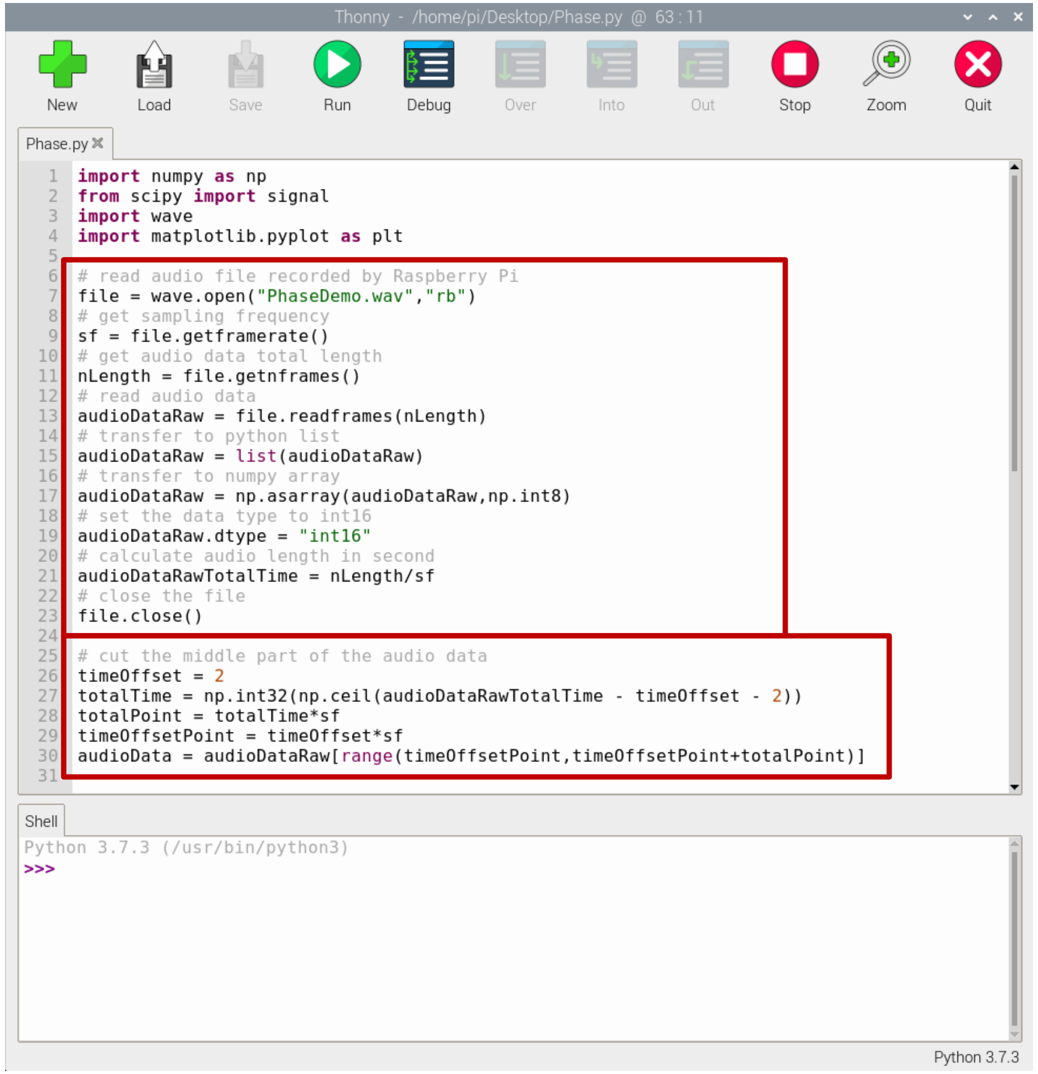
- We provide a demo to help you understand phase-based distance tracking.
- In this demo, the hand move 10 cm and move back.
- The derived distance tracking result is shown in figure.



Phase Demo

- There are two files:
  - “***Phase.py***” and “***PhaseDemo.wav***”



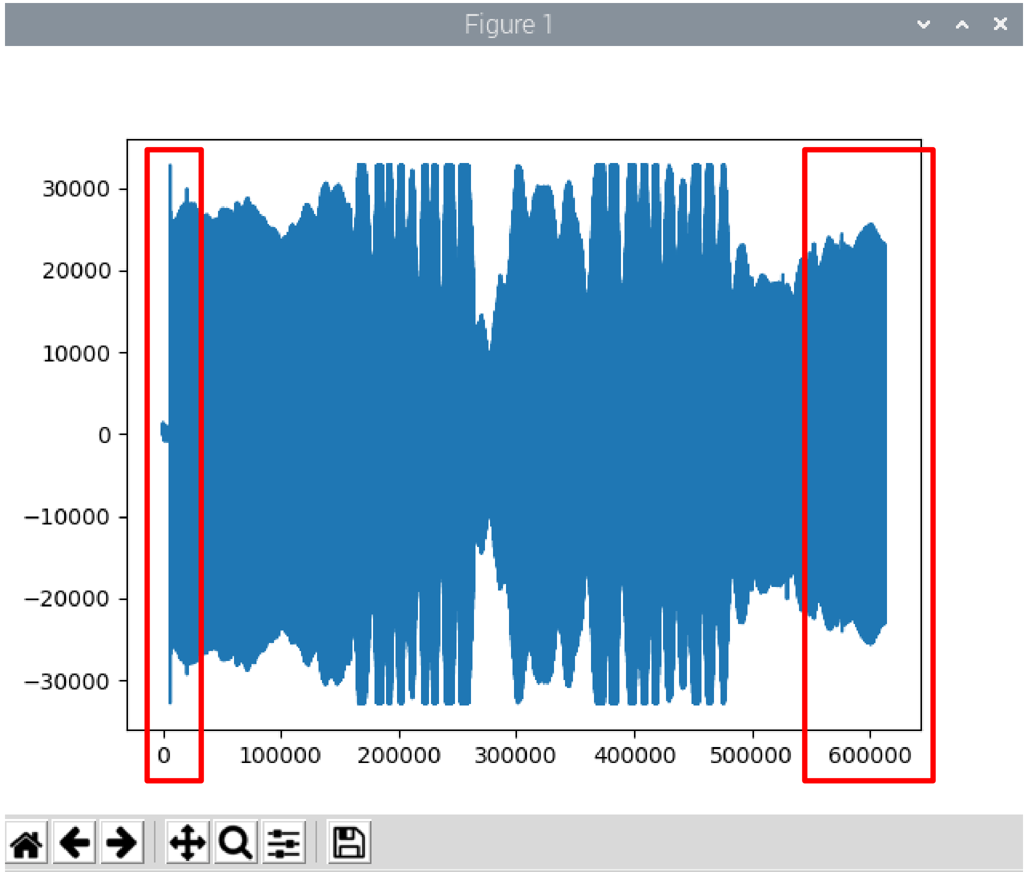


Read Audio File

- We first read the data file and convert data type.

```
# read audio file recorded by Raspberry Pi
file = wave.open("PhaseDemo.wav","rb")
# get sampling frequency
sf = file.getframerate()
# get audio data total length
nLength = file.getnframes()
# read audio data
audioDataRaw = file.readframes(nLength)
# transfer to python list
audioDataRaw = list(audioDataRaw)
# transfer to numpy array
audioDataRaw = np.asarray(audioDataRaw,np.int8)
# set the data type to int16
audioDataRaw.dtype = "int16"
# calculate audio length in second
audioDataRawTotalTime = nLength/sf
# close the file
file.close()
```

audioDataRaw

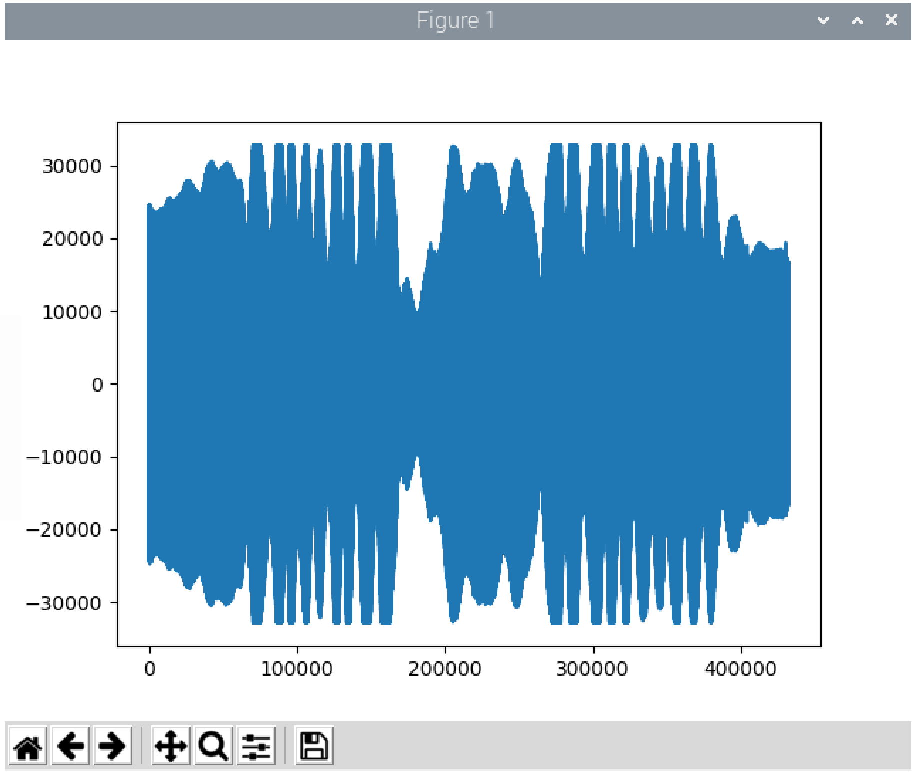


Get Middle Part of Audio Data

- The start and end of the experiment may be unstable. Thus we use the middle part of the audio file.

```
# cut the middle part of the audio data
timeOffset = 2
totalTime = np.int32(np.ceil(audioDataRawTotalTime - timeOffset - 2))
totalPoint = totalTime*sf
timeOffsetPoint = timeOffset*sf
audioData = audioDataRaw[range(timeOffsetPoint,timeOffsetPoint+totalPoint)]
```

audioData



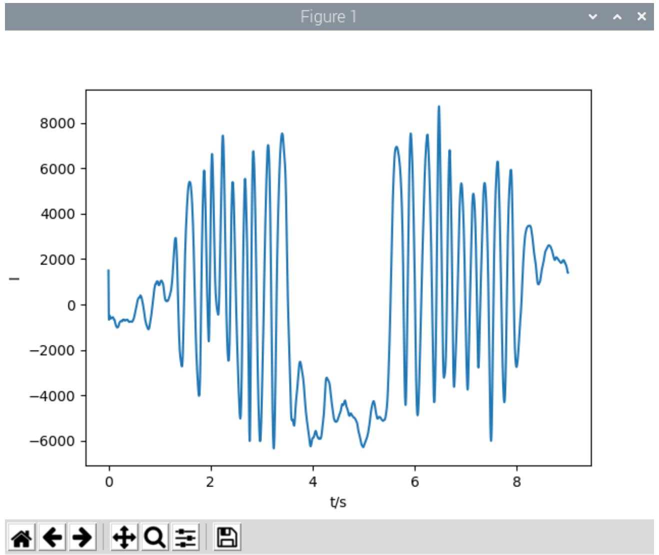
Recall the Key Steps

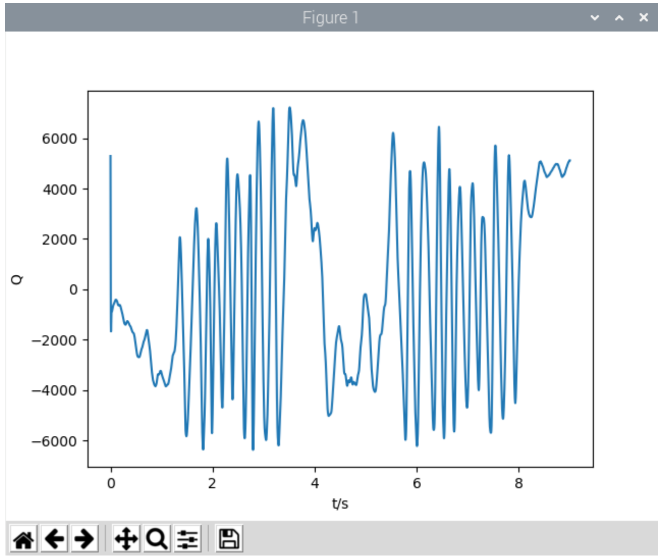
1. Derive In-phase signal I.
   - Multiply received signal with $cos(2\pi f t)$
   - Apply a low-pass filter
2. Derive Quadrature signal Q.
   - Multiply received signal with $-sin(2\pi f t)$
   - Apply a low-pass filter
3. Calculate the phase using I and Q.
4. Convert phase change to distance change.

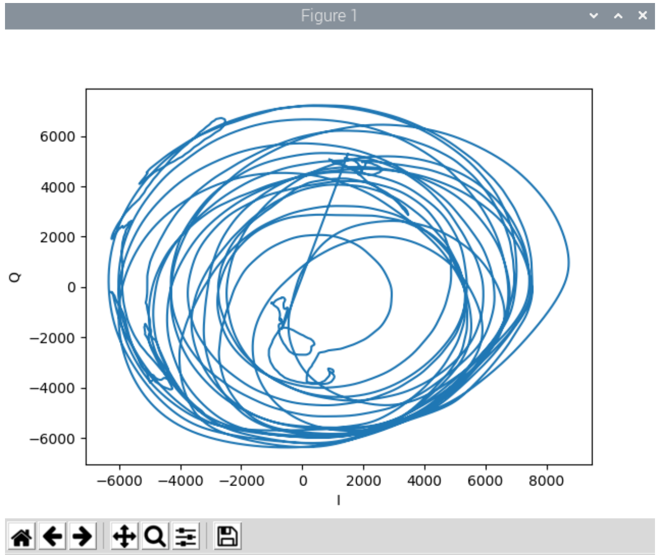
```
# set frequency
freq = 18000
# calculate time t
t = np.arange(totalPoint)/sf
# get cos and -sin used in demodulation
signalCos = np.cos(2*np.pi*freq*t)
signalSin = -np.sin(2*np.pi*freq*t)
# get a butterworth filter
b, a = signal.butter(3, 50/(sf/2), 'lowpass')
# multiply received signal (audioData) and demodulation signal, also apply the filter
signalI = signal.filtfilt(b,a,audioData*signalCos)
signalQ = signal.filtfilt(b,a,audioData*signalSin)
# remove static vector
signalI = signalI - np.mean(signalI)
signalQ = signalQ - np.mean(signalQ)
# calculate the phase angle
phase = np.arctan(signalQ/signalI)
# unwrap the phase angle
phase = np.unwrap(phase*2)/2
# calculate the wave length
waveLength = 342/freq
# calculate distance
distance = phase/2/np.pi*waveLength/2
```

Calculate I and Q

```
# set frequency
freq = 18000
# calculate time t
t = np.arange(totalPoint)/sf
# get cos and -sin used in demodulation
signalCos = np.cos(2*np.pi*freq*t)
signalSin = -np.sin(2*np.pi*freq*t)
# get a butterworth filter
b, a = signal.butter(3, 50/(sf/2), 'lowpass')
# multiply received signal (audioData) and demodulation signal, also apply the filter
signalI = signal.filtfilt(b,a,audioData*signalCos)
signalQ = signal.filtfilt(b,a,audioData*signalSin)
```

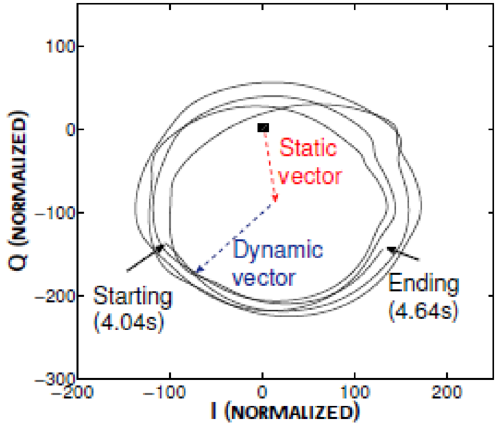






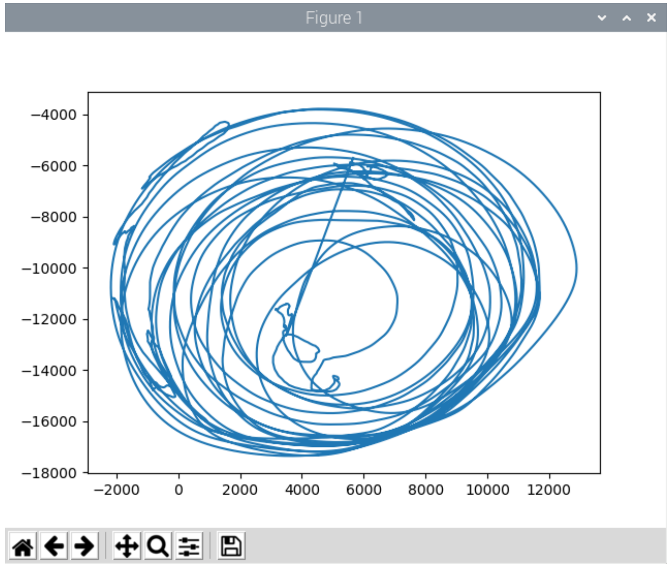
Remove Static Vector

```
# remove static vector
signalI = signalI - np.mean(signalI)
signalQ = signalQ - np.mean(signalQ)
```

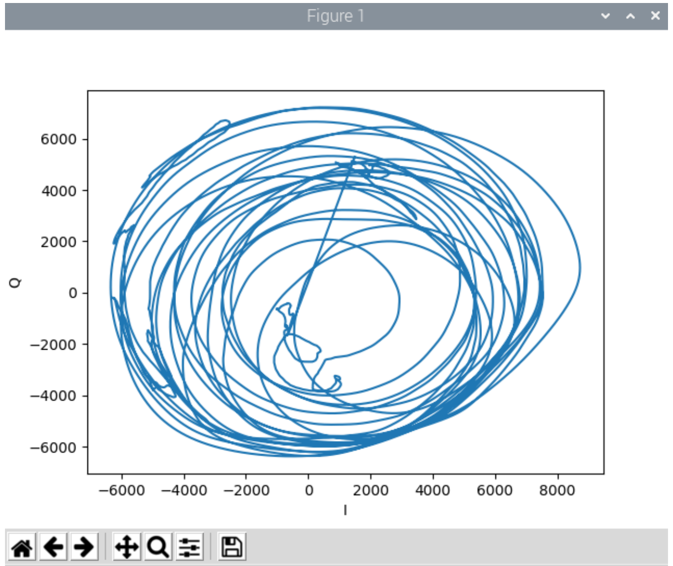


(b) Complex I/Q traces

Before removal



after removal

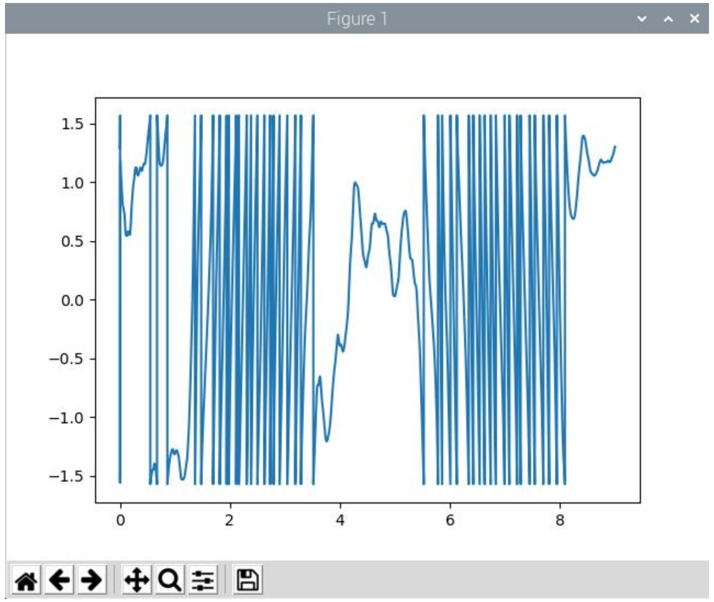


Derive Phase Angle and Unwrap

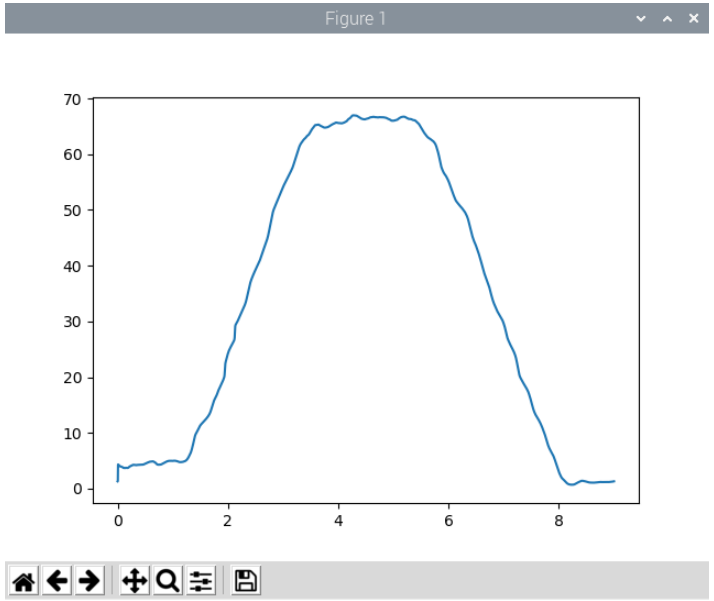
- Unwrap to solve the problem of boundary $0/2\pi$ $(-\pi/\pi)$.

```
# calculate the phase angle
phase = np.arctan(signalQ/signalI)
# unwrap the phase angle
phase = np.unwrap(phase*2)/2
```

Without unwrap



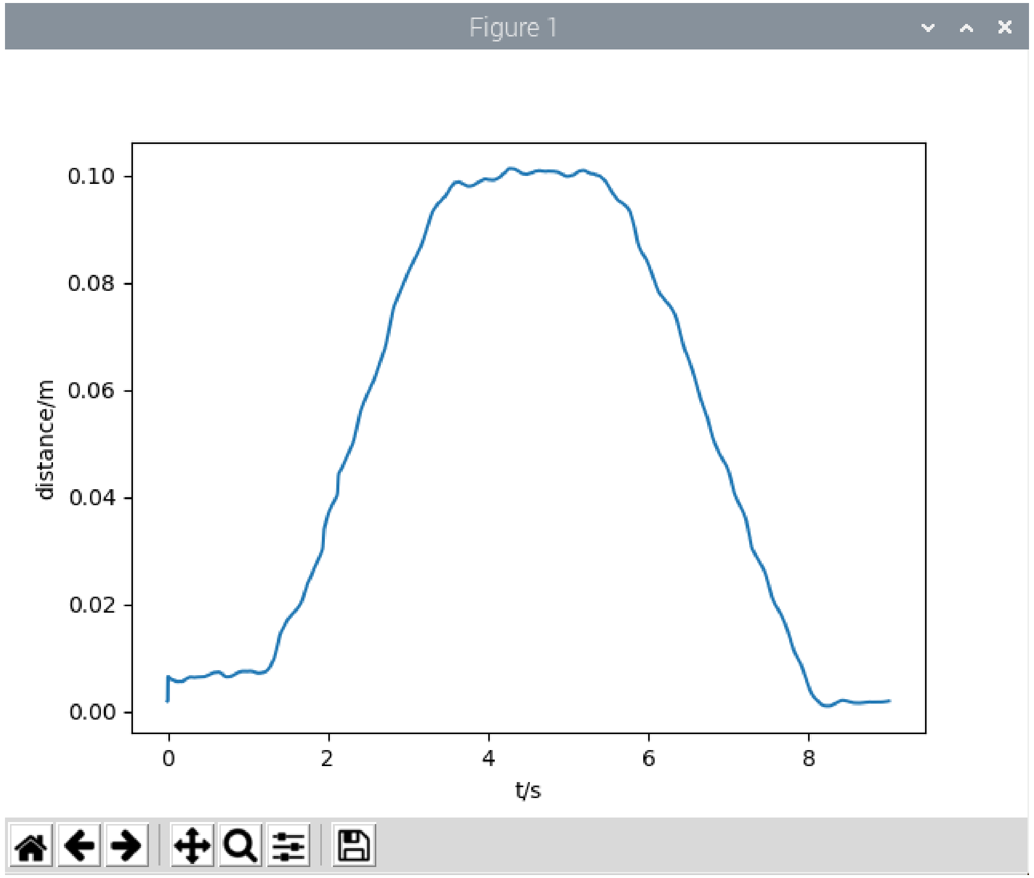
With unwrap



Calculate Distance by Phase

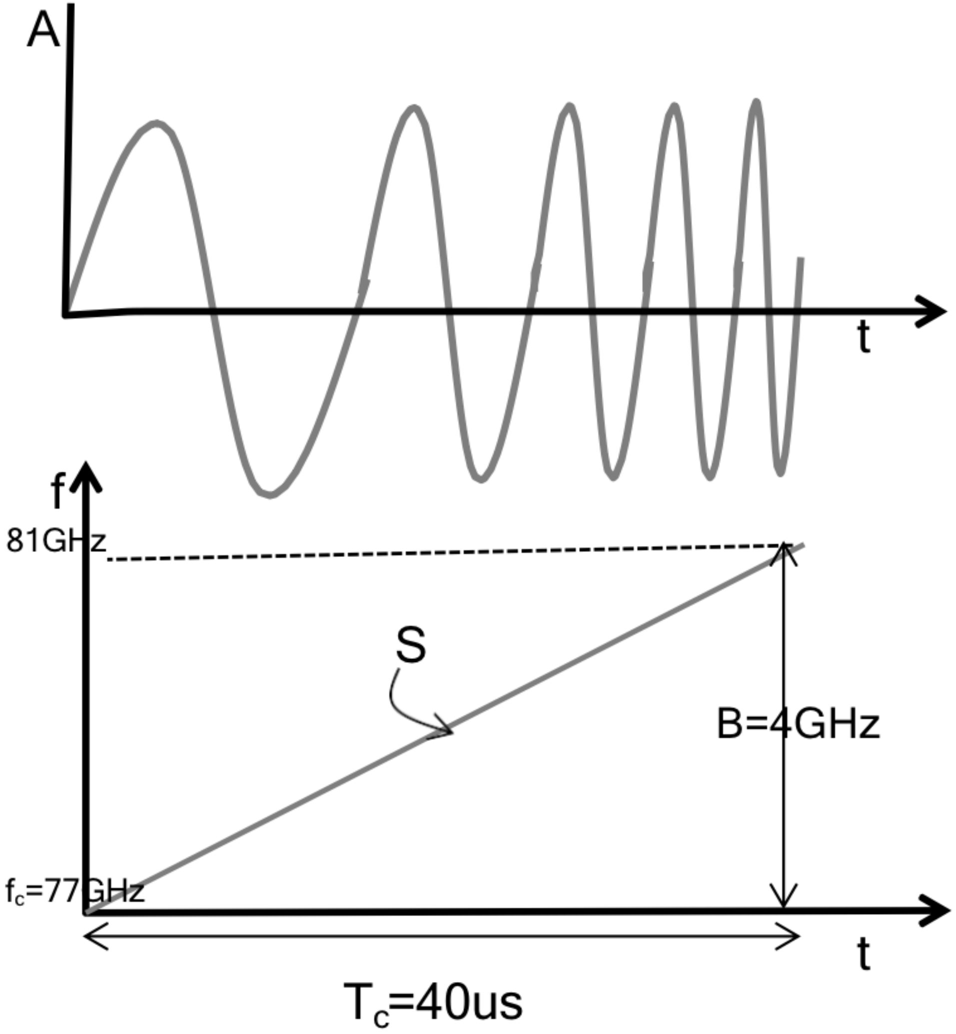
- 2π change in phase is distance change of a wavelength.

```
# calculate the wave length
waveLength = 342/freq
# calculate distance
distance = phase/2/np.pi*waveLength/2
```



CORRELATION BASED FREQUENCY MODULATED CONTINUOUS WAVE METHOD (C-FMCW) BASED DISTANCE TRACKING

What is a chirp?



An FMCW radar transmits a signal called a “chirp”. A chirp is a sinusoid whose frequency increases linearly with time, as shown in the Amplitude vs time (or ‘A-t’ plot) here.

- A frequency vs time plot (or ‘f-t plot’) is a convenient way to represent a chirp.
- A chirp is characterized by a start frequency (f_c), Bandwidth(B) and duration (T_c).
- The Slope (S) of the chirp defines the rate at which the chirp ramps up. In this example the chirp is sweeping a bandwidth of 4GHz in 40us which corresponds to a Slope of 100MHz/us

Range Distance Resolution

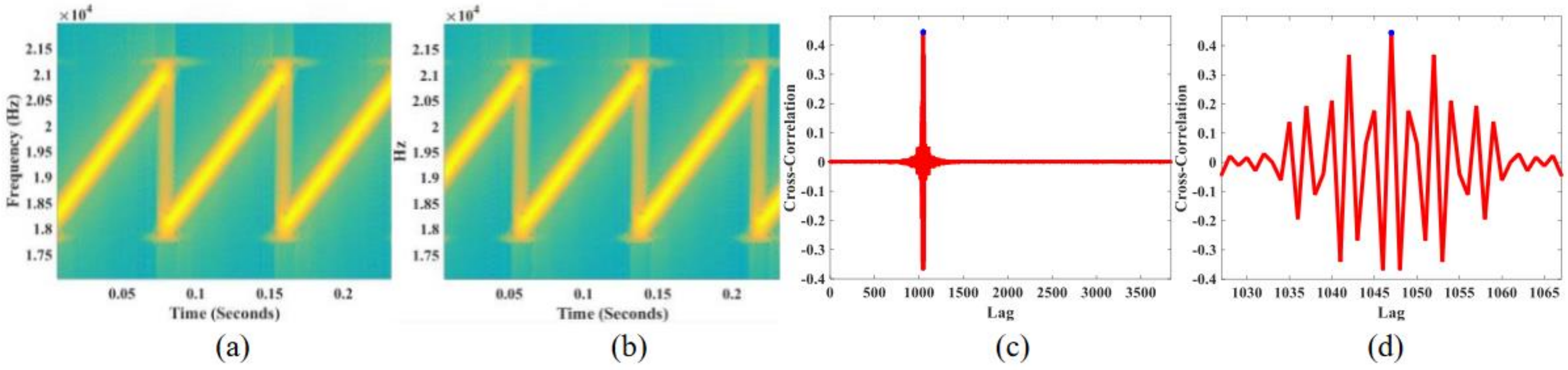
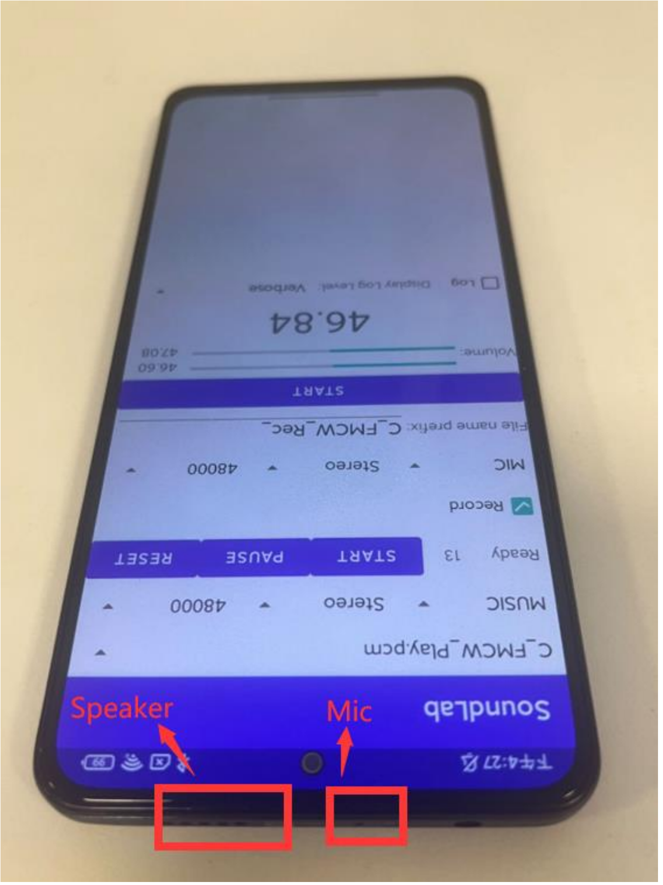


Fig. 2. Cross-correlation function of transmitted and received chirp signals with delay. (a) transmitted signal. (b)received signal with delay. (c) cross-correlation function of transmitted and received signals. (d) enlarged view round the peak of (c)

$$\delta R = \frac{C \cdot \delta Lag}{2F_s} = \frac{C}{2F_s}$$

$$\delta R = \frac{C}{2F_s} = \frac{343}{2 \times 48000} = 0.00357\ m = 0.357\ cm$$

Experimental Settings



fs = 48e3

Chirp:

- f1 = 18e3
- f2 = 22e3
- N = 256

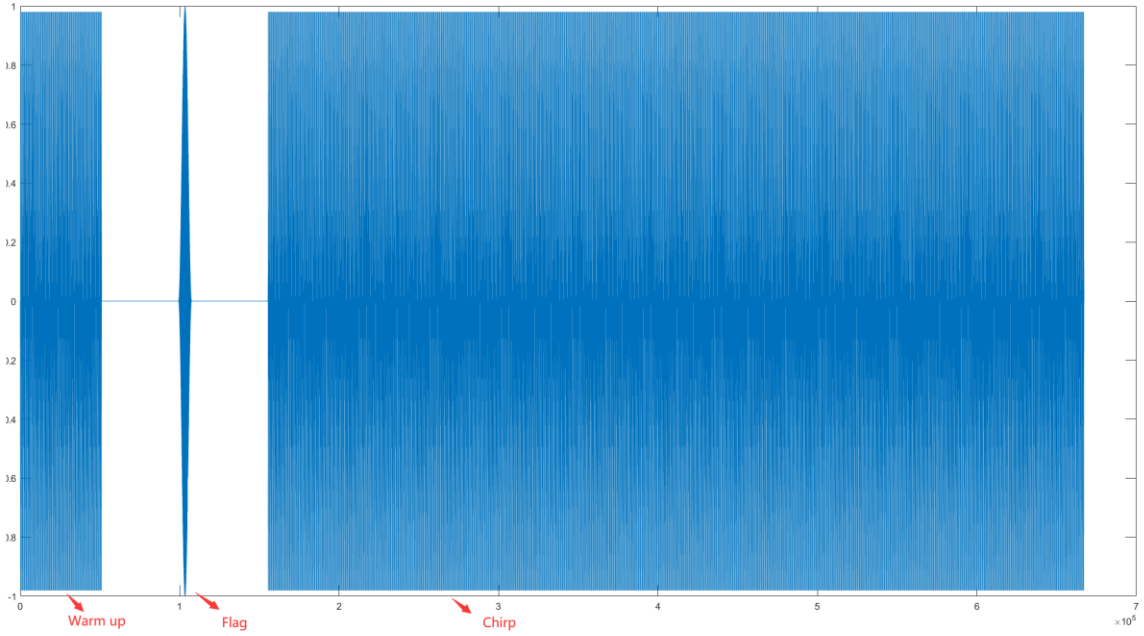
Idle time length: 256

Flag:

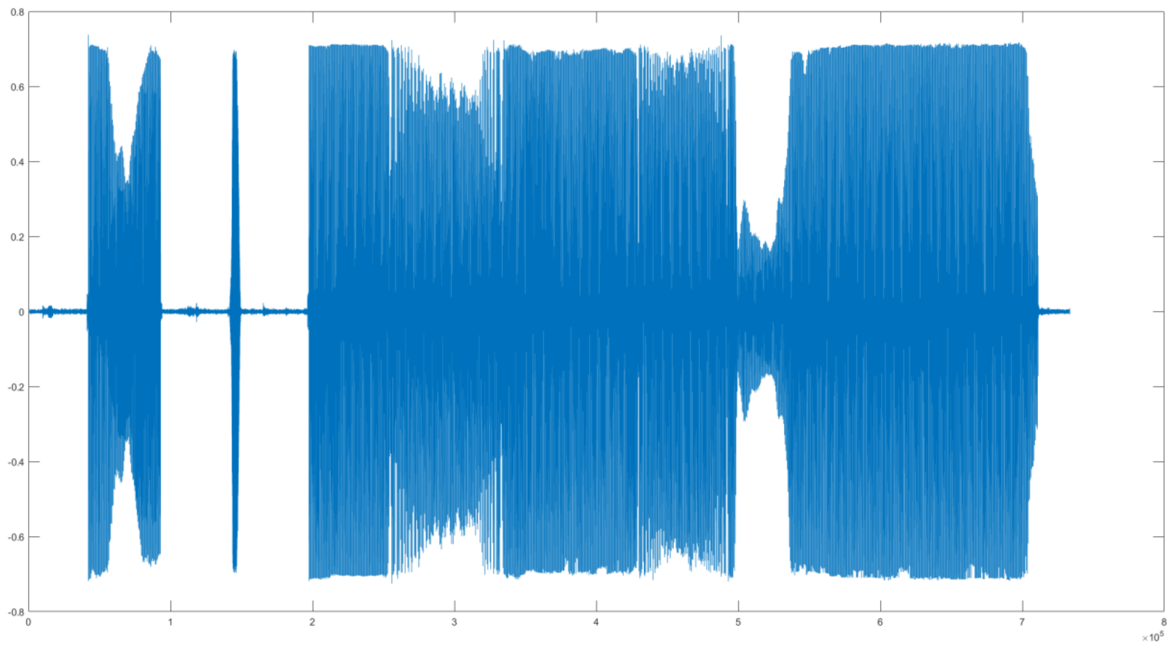
- f1 = 22e3
- f2 = 18e3
- N = 8192

Signal Transceivers

Transmitted signal

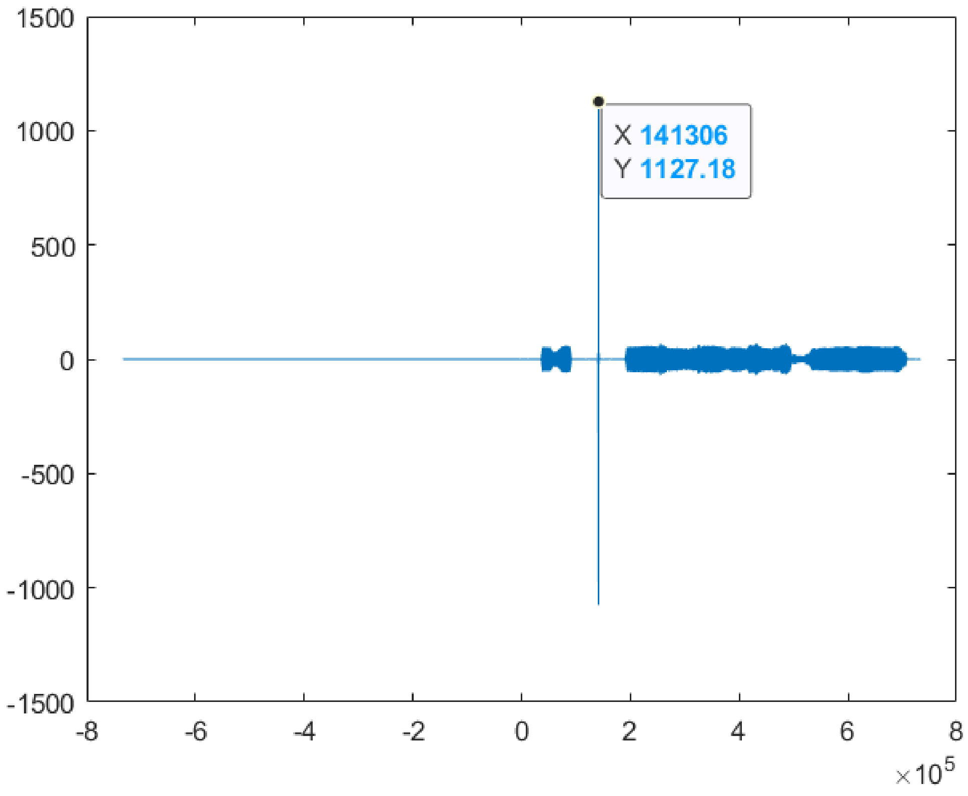


Received signal



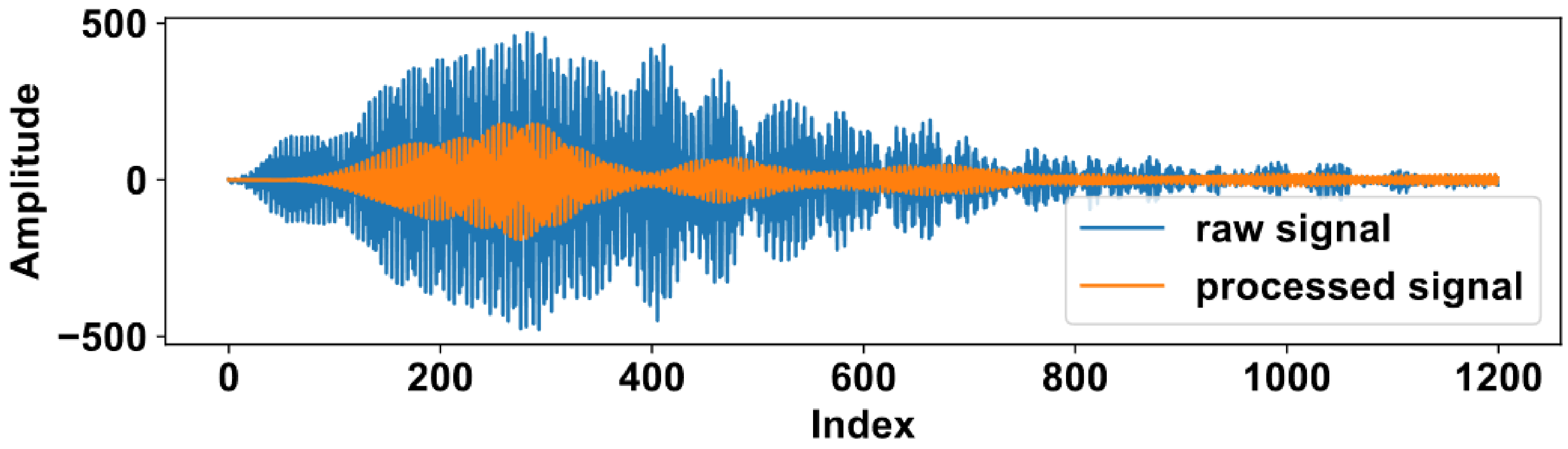
Synchronization

Synchronization through direct path propagation (xcorr).

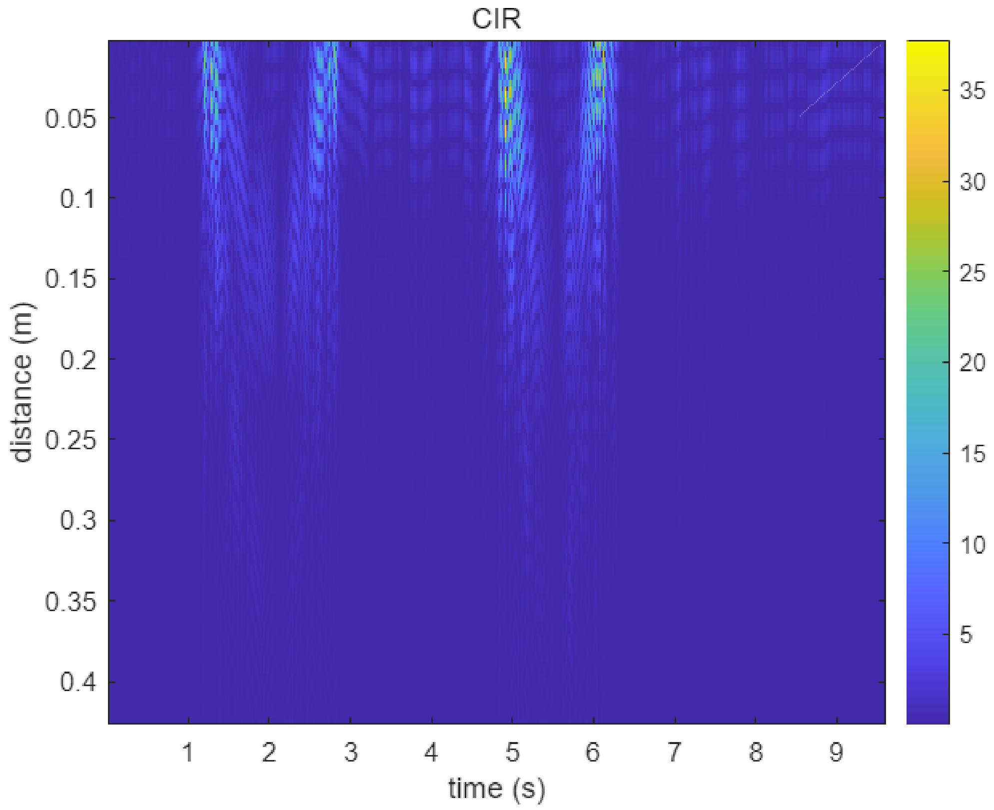


Static Elimination

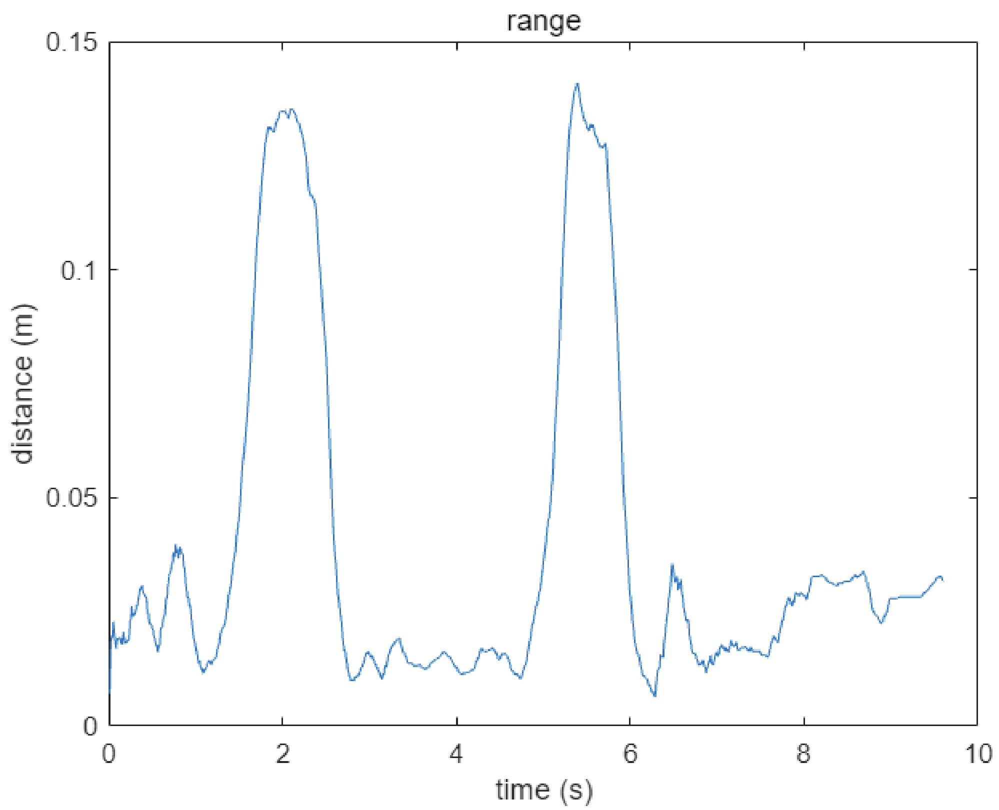
We eliminate the static components by subtracting the current chirp from its previous one.



Example



CIR
distance (m)
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
time (s)
1
2
3
4
5
6
7
8
9
35
30
25
20
15
10
5



range
distance (m)
0.15
0.1
0.05
0
time (s)
0
2
4
6
8
10

Homework

Move your hands back and forth twice, and track the distance change of your hand using C-FMCW. Draw a graph of the distance change of your hand, horizontal coordinates in s, vertical coordinates in m.

You can either collect the data yourself or use the given data C_FMCW_Rec_1.pcm

Pack your codes, figure and audio files into SID.zip. Hand in your SID.zip in bb system.

Reference

- Tung Y C, Bui D, Shin K G. Cross-platform support for rapid development of mobile acoustic sensing applications[C]//Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services. 2018: 455-467.

- Wang T, Zhang D, Zheng Y, et al. C-FMCW based contactless respiration detection using acoustic signal[J]. Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, 2018, 1(4): 1-20.