Humanoid Sensors and Actuators

Tutorial 5 – Acoustic Sensors and Signal Processing

Group 4

Binkun Huang Luca Hu Liangyu Chen

Summer Semester 2025

1 Sound Source Localisation

R.1.1 (3 pts)

Find the expression to determine the azimuth angle of a sound source for a system with two microphones. Derive the equations shown in the slides of Lecture 3 step by step.

Assuming two microphones separated by distance ℓ, sound speed c, and arrival time difference Δt, we have the path difference:

$$\Delta d = c \cdot \Delta t$$

Geometrically, this path difference relates to the azimuth angle θ as follows:

$$\Delta d = \ell \sin(\theta)$$

Combining both equations, we get:

$$c \cdot \Delta t = \ell \sin(\theta) \quad \Rightarrow \quad \theta = \arcsin\left(\frac{c\,\Delta t}{\ell}\right)$$

R.1.2 (3 pts)

Find the expression to determine the velocity of a target from the pulse duration difference of a radar sensor. Derive the equations shown in the slides of Lecture 3 step by step.

Given two consecutive radar pulses with measured round-trip durations τ_n and τ_{n+1}, the distances to the target are:

$$R_n = \frac{c\,\tau_n}{2}, \quad R_{n+1} = \frac{c\,\tau_{n+1}}{2}$$

The target velocity v is calculated from the distance difference divided by the pulse repetition interval T_r:

$$v = \frac{R_{n+1} - R_n}{T_r} = \frac{\frac{c\,\tau_{n+1}}{2} - \frac{c\,\tau_n}{2}}{T_r} = \frac{c\,(\tau_{n+1} - \tau_n)}{2\,T_r}$$

R.1.3 (1 pt)

How can we measure the distance to a target?

We measure the distance using the time-of-flight (ToF) principle, calculating the travel time Δt of a pulse reflected from the target:

$$d = \frac{c\,\Delta t}{2}$$

R.1.4 (1 pt)

How can we measure the speed of a moving target?

We measure speed using the Doppler frequency shift by observing the change in reflected frequency Δf:

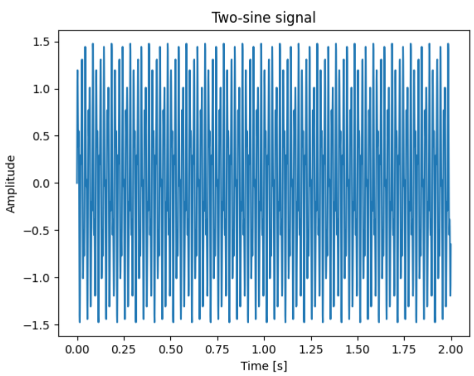
$$v = \frac{c\,\Delta f}{2\,f_0}$$

Alternatively, the speed can also be measured by the change in distance Δd over a known time interval Δt:

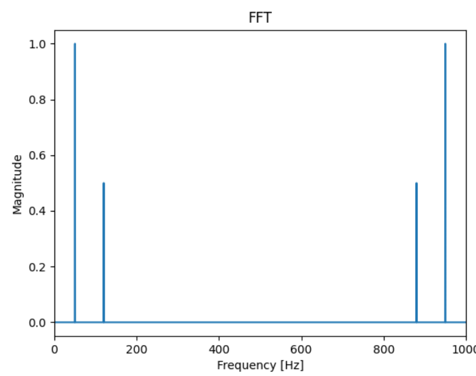
$$v = \frac{\Delta d}{\Delta t}$$

2 Fast Fourier Transform

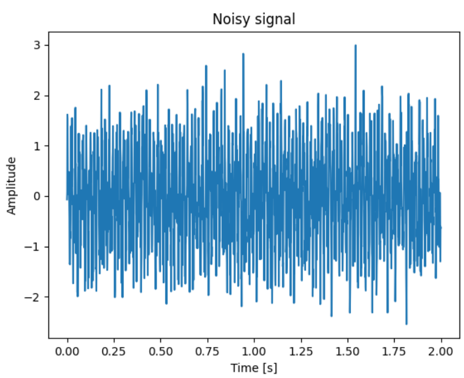
Task results (T.2.1 – T.2.4)



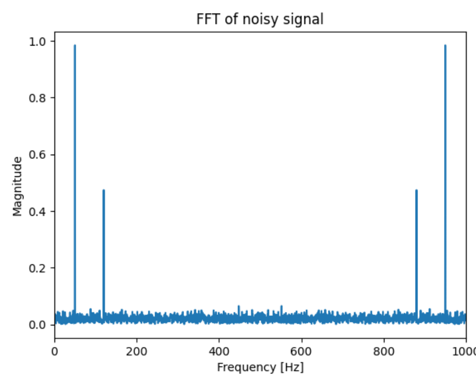
(a) Two-sine signal (T.2.1)



(b) FFT of clean signal (T.2.2)



(c) Signal with Gaussian noise (T.2.3)



(d) FFT of noisy signal (T.2.4)

R.2.1 (2 pts)

Is it possible to implement FFT to an online data streaming? Why?

FFT cannot directly process samples one-by-one in real-time. It needs data segments of fixed length (usually a power of two). Thus, FFT can only be applied to streaming data by dividing it into windows or segments.

R.2.2 (2 pts)

How can you use FFT in signal processing?

FFT transforms signals from the time domain into the frequency domain. This is useful for analyzing frequency components, identifying dominant frequencies, filtering noise, and extracting spectral features.

R.2.3 (2 pts)

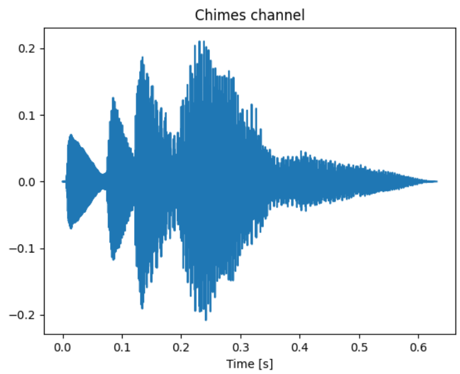
Deliver the code used to generate the signals and plots?

The code used is located in `scripts/task2_fft.py`, which calls utility functions from `src/signal_generation.py` and `src/fft_utils.py`. A simplified excerpt is shown below:

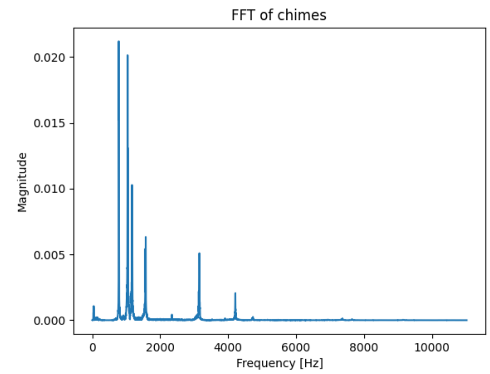
```
from src.signal_generation import sum_sine_waves, add_noise
from src.fft_utils import compute_fft
import matplotlib.pyplot as plt

fs = 1000
duration = 2.0
components = [(50, 1.0), (120, 0.5)]
t, signal = sum_sine_waves(components, fs, duration)

freqs, mag = compute_fft(signal, fs, full_range=True)

noisy_signal = add_noise(signal, 0.5)
freqs_n, mag_n = compute_fft(noisy_signal, fs, full_range=True)
...
```

3 Audio Correlation

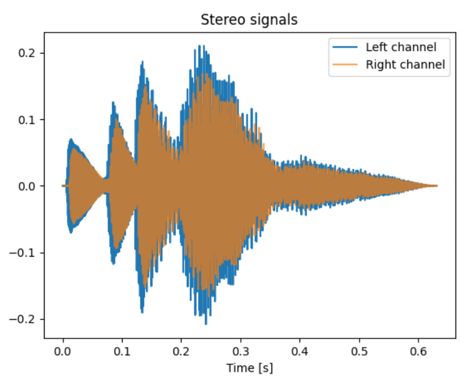
Task results (T.3.1 – T.3.8)



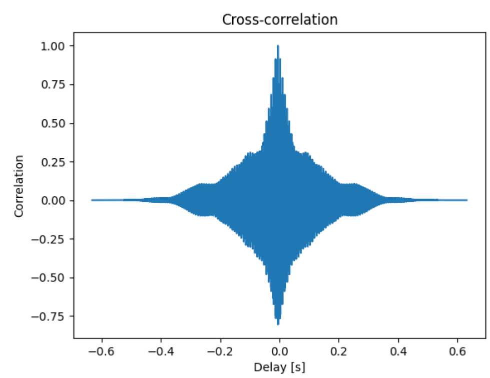
(a) Isolated channel (T.3.1)



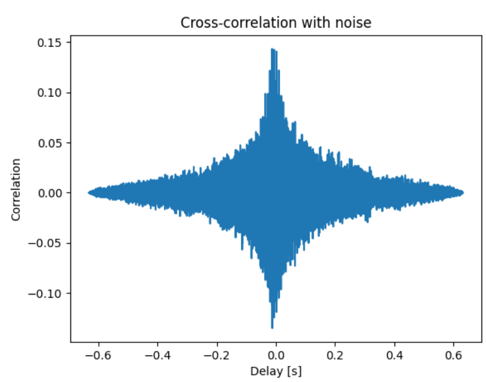
(b) FFT (T.3.2)



(c) Stereo pair (T.3.3/3.4)



(d) Cross-correlation (T.3.6)



(e) Cross-correlation with noise (T.3.8)

R.3.1 (2 pts)

Is it possible to implement the cross-correlation to an online data streaming? Why?

No. Classical cross-correlation needs the complete sequences to evaluate every possible lag, so it is not a true sample-by-sample real-time algorithm. In streaming, the future samples are still unknown; therefore the full correlation map cannot be produced without buffering a window of past *and* future data.

R.3.2 (2 pts)

If you answered "no" to R.3.1, how would you work around to use it to identify the interaural time delay?

Use a short sliding window whose length only covers plausible interaural delays (e.g. ± 1 ms). Update the window each time new samples arrive and compute the correlation (or GCC-PHAT) inside that window. The delay corresponding to the peak in each window gives an online estimate of the interaural time delay with acceptable latency.

R.3.3 (4 pts)

Deliver the code to generate the signals and the plots (T.3.1 – T.3.8).

The implementation is in `scripts/task3_correlation.py`; it relies on helper functions in `src/correlation_utils.py`, `src/fft_utils.py` and `src/signal_generation.py`. Key lines are:

```
data, fs = sf.read("data/chimes.wav")
mono = data[:, 0]              # T.3.1
freqs, mag = compute_fft(mono, fs)  # T.3.2

delay_samples = int(0.005 * fs)
left  = mono
right = np.roll(mono, delay_samples) * 0.8  # T.3.3

lags,  corr  = cross_correlation(left, right)       # T.3.6
left_n  = add_noise(left, 0.1)
right_n = add_noise(right, 0.1)
lags_n, corr_n = cross_correlation(left_n, right_n) # T.3.8
...
```

4 Signal Filtering

Task results (T.4.1 – T.4.12)

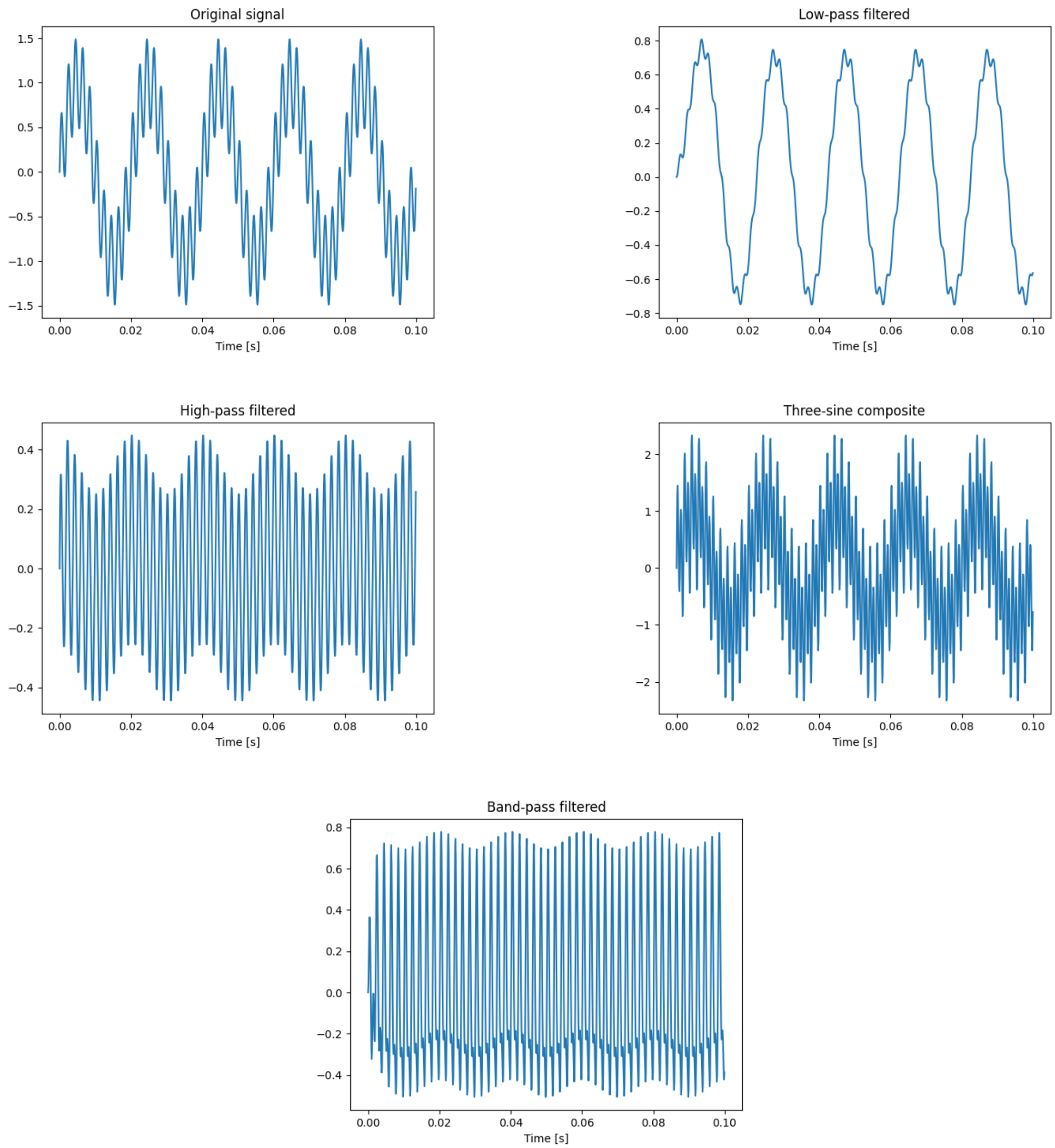


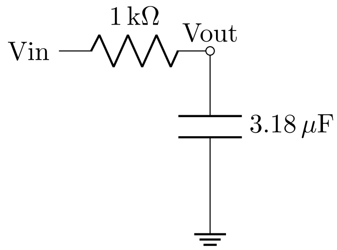
Figure 3: Filtering outputs for Task 4.

R.4.1 Detail the design process of the filter in T.4.3. Draw the required circuit and calculate the value for the components step by step.

A first–order passive low-pass was built with a series resistor and a shunt capacitor. Selecting the default value used in the code (`rc_lowpass(50)`), R is fixed to $1\,\mathrm{k}\Omega$ and

$$C = \frac{1}{2\pi f_c R} = \frac{1}{2\pi\,(50)\,(1\,000)} \approx 3.18\,\mu\mathrm{F}.$$

The -3 dB corner frequency is verified by $f_c = 1/(2\pi RC)$. The circuit is: input $\rightarrow R \rightarrow$ node; node $\rightarrow$ $C \rightarrow$ ground; node is the output.



R.4.2 Detail the design process of the filter in T.4.4. Derive the equation to implement the filter step by step on a data stream.

In the script a 1-st-order digital Butterworth low-pass is generated by `butter_lowpass(50, fs=10 000, order=1)`. The bilinear transform maps the analog prototype $H(s) = \frac{\omega_c}{s+\omega_c}$ to $H(z) = \frac{b_0+b_1 z^{-1}}{1+a_1 z^{-1}}$. Scipy returns

$$y[n] = b_0\, x[n] + b_1\, x[n-1] - a_1\, y[n-1].$$

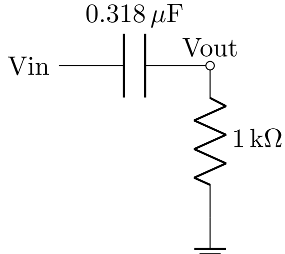
With $f_s = 10\,000$ Hz and $f_c = 50$ Hz the coefficients are $b_0 = b_1 \approx 0.0155,\ a_1 \approx -0.969$ (see `b,a` in the code). This equation is applied sample-by-sample using `scipy.signal.lfilter`.

R.4.3 Detail the design process of the filter in T.4.6. Draw the required circuit and calculate the value for the components step by step.

For a passive high-pass the positions of R and C are swapped. Using `rc_highpass(500)` the script again fixes $R = 1\,\mathrm{k}\Omega$:

$$C = \frac{1}{2\pi f_c R} = \frac{1}{2\pi\,(500)\,(1\,000)} \approx 0.32\,\mu\mathrm{F}.$$

Circuit: input $\to C \to$ node; node $\to R \to$ ground; node is the output.



R.4.4 Detail the design process of the filter in T.4.7. Derive the equation to implement the filter step by step on a data stream.

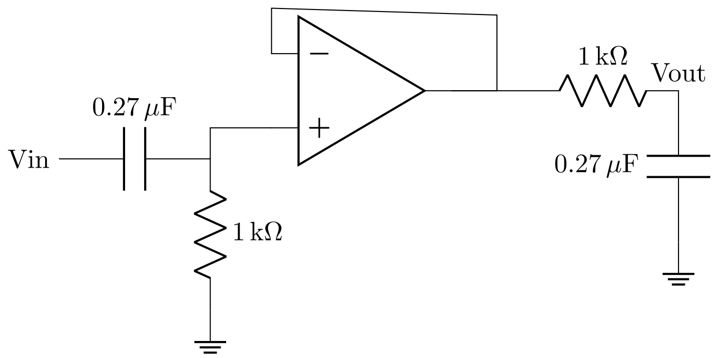
`butter_highpass(500, fs=10 000, order=1)` gives $H(z) = \frac{b_0+b_1 z^{-1}}{1+a_1 z^{-1}}$ with $b_0 \approx 0.863,\ b_1 \approx -0.863,\ a_1 \approx -0.727$. The resulting difference equation is $y[n] = b_0\, x[n] + b_1\, x[n-1] - a_1\, y[n-1]$.

R.4.5 Detail the design process of the filter in T.4.10. Draw the required circuit and calculate the value for the components step by step.

An active band-pass is formed by cascading a high-pass and a low-pass stage around an op-amp buffer. Calling `rc_bandpass(400,600)` selects the default $R = 1\,\mathrm{k}\Omega$ and computes

$$C = \frac{1}{2\pi\,(600)\,(1\,000)} \approx 0.27\,\mu\mathrm{F}.$$

Both RC sections use this R and C, giving a pass band centred near 500 Hz. The op-amp prevents interaction between stages and can provide gain.



R.4.6 Detail the design process of the filter in T.4.11. Derive the equation to implement the filter step by step on a data stream.

The digital band-pass is produced in one call: `butter_bandpass(400,600, fs=10 000, order=1)`. Scipy designs the Butterworth HP and LP sections and combines them, yielding $H(z) = \frac{b_0+b_1z^{-1}+b_2z^{-2}}{1+a_1z^{-1}+a_2z^{-2}}$ with $b_0 \approx 0.059$, $b_1 = 0$, $b_2 \approx -0.059$, $a_1 \approx -1.793$, $a_2 \approx 0.882$. Thus the difference equation is $y[n] = b_0\,x[n] + b_1\,x[n-1] + b_2\,x[n-2] - a_1\,y[n-1] - a_2\,y[n-2]$. The sample stream is filtered with `lfilter(b, a, data)`.

R.4.7 What is the difference between passive and active filters?

Passive filters use only R, L, C and cannot amplify; active filters add powered devices (op-amps, transistors) so they can buffer, amplify, and avoid large inductors.

R.4.8 What is the difference between FIR and IIR filters?

FIR filters have a finite impulse response and are always stable; IIR filters have feedback, achieve sharper roll-off with fewer coefficients, but can become unstable and introduce nonlinear phase.

R.4.9 What is the "order" of a discrete filter?

Order equals the highest delay term z^{-k} (number of poles). It determines the slope: each order adds about 20 dB/decade attenuation.

R.4.10 How could you implement a continuous-time filter in a robotic system?

Place an analog RC/active filter between the sensor output and the ADC to pre-condition the signal before sampling.

R.4.11 How could you implement a discrete-time filter in a robotic system?

Run the filter's difference equation in firmware (MCU/DSP) at the sampling rate, using fixed-point or floating-point arithmetic.

R.4.12 What are the advantages and disadvantages of analog and digital filters in robotic systems?

Analog: zero latency, no quantisation, low power; but component drift, no re-tuning in software. Digital: reconfigurable, repeatable, high order; but needs ADC/DAC, adds latency and quantisation noise.

R.4.13 Is it possible to make a 1000 Hz Low-Pass filter in a digital system with a sampling rate of 1000 Hz?

No. The Nyquist limit is 500 Hz, so a 1000 Hz cut-off is unattainable without a higher sampling rate.

R.4.14 Deliver the code to generate the signals and the plots (T.4.1 – T.4.12).

All figures were produced by `scripts/task4_filtering.py`, which calls `src/filter_design.py` and `src/signal_generation.py`. Key lines:

```
# signal generation
fs = 10000
t, signal = sum_sine_waves([(50,1.0),(500,0.5)], fs, 0.1)

# low-pass
b_lp, a_lp = butter_lowpass(50, fs, order=1)
filtered_lp = apply_filter(signal, b_lp, a_lp)

# high-pass
b_hp, a_hp = butter_highpass(500, fs, order=1)
filtered_hp = apply_filter(signal, b_hp, a_hp)

# band-pass
b_bp, a_bp = butter_bandpass(400, 600, fs, order=1)
filtered_bp = apply_filter(signal +
    sum_sine_waves([(1000,1.0)], fs, 0.1)[1], b_bp, a_bp)
    ...
```