UiO : Department of Informatics
University of Oslo

Mandatory exercise 3

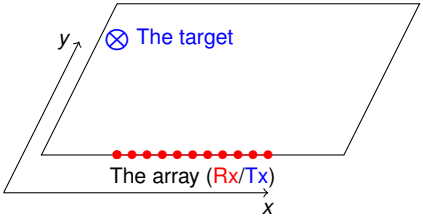
MIMO Pulse-echo imaging

Thomas Aussaguès

April 29, 2022

Table of contents

Problem statement



Pulses

- **Waveforms**: LFM upchirp and downchirps
- We use both TDMA (upchirp only) and CDMA (both up and down chirps)
- Therefore, the two **waveforms should be orthogonal** (or at least, have a low cross-correlation)

$$s_{TX,up}(t) = \exp\left\{2j\pi\left((f_c - B/2)\,t + \alpha t^2\right)\right\}\mathbb{1}_{0\leq t\leq T_p}(t)$$

$$s_{TX,down}(t) = \exp\left\{2j\pi\left((f_c + B/2)\,t - \alpha t^2\right)\right\}\mathbb{1}_{0\leq t\leq T_p}(t)$$

Parameter	f_c(kHz)	B(kHz)	f_s(kHz)	T_p(ms)
Value	10^3	10^3	2×10^2	10

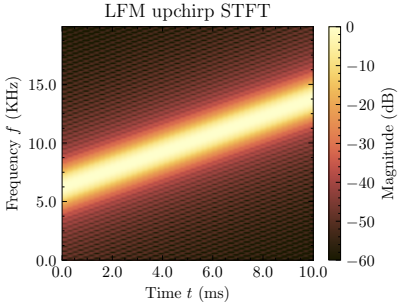


Figure: LFM upchirp

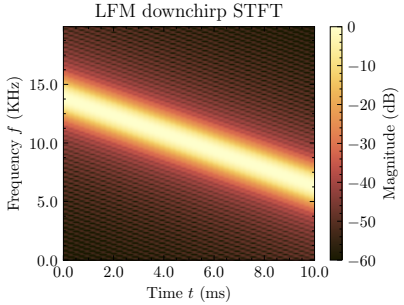


Figure: LFM downchirp

- **STFT parameters** : zero padding $= \times 4$, $L = 512$ and $D = 32$ see `utils/fourier_analysis.py`

Are the pulses truly orthogonal?

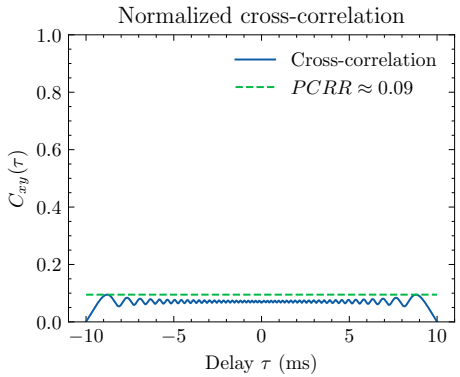


Figure: Normalized cross-correlation

- The two pulses **are not** truly orthogonal...
- **Limited cross-talk**: $PCRR \approx 0.09 \ll 1$
- $PCRR \approx 0.09 \approx \dfrac{1}{\sqrt{BT_p}} = \dfrac{1}{\sqrt{10^4 \times 10^{-2}}} = 0.1$
- The chirps **can be considered as orthogonal**
- We can **separate** signals from the left and rightmost transmitters when working with the CDMA dataset

Additional remarks:

- The pulses are **over-sampled**: $f_s = 2 \times 10^2$ kHz $\gg f_{max} = f_c + B/2 = 15$ kHz
- Except if the reflectors frequency responses have components above 15 kHz
- Moreover, up-sampling the data can **considerably improve** the time-delay estimation (using pulse-compression)
- **Which normalization**? The same as in `MATLAB`: $C_{xy} \leftarrow C_{xy}/(\|x\| \cdot \|y\|)$

Pulses

```python
def lfm_pulse(B: float, f_c: float,T_p: float, fs: float) -> np.array :

  '''This function computes and returns int(f_s*T_p) samples of a Linear Frequency Modulated (LFM).
  Note that if you want an UP LFM pulse, you need to use a positive bandwidth B. For a DOWN LFM pulse,
  use a negative one.

  Inputs:
  - B: float, bandwidth in Hertz (Hz)
  - f_c: float, pulse central frequency in Hertz (Hz)
  - T_p: float, pulse length in seconds (s)
  - f_s: float, pulse sampling frequency in Hertz (Hz)

  Output:
  - pulse: np.array (dtyp = 'complex'), LFM pulse'''

  '''First, we compute the chirp rate alpha (in 1/s^2) definded as the bandwidth divided by the pulse length
  alpha = B/T_p.'''
  alpha = B / T_p

  '''Then, we create a time np.array from t = 0 s to t = T_p with a time sampling interval of 1/fs which corresponds
  to a total number of points of int(Tp * fs).'''
  time = np.linspace(0, T_p, int(T_p * fs))

  '''We allocate space for the pulse array. Note that this array must be a complex array!'''
  pulse = np.zeros((int(T_p * fs) + 1), dtype= 'complex')

  '''Finally, we compute the pulse using the LFM pulse formula:
   pulse[k] = exp(2j * pi * ( (fc - B / 2) * t + alpha * t ** 2 / 2) )'''
  pulse = np.exp(2 * 1j * np.pi * ((f_c - B/2) * time + alpha * time ** 2 / 2))

  '''We return the pulse array.'''

  return pulse
```

Listing 1: LFM pulse generator

Code

- Given an input pulse (`ping`, $x[k]$) and its echo (`echo`, $y[k]$), the output of the **match filter** (the compressed pulse) is given by:

$$z[n] = C_{yx}[n] = y * x[n]$$

- Note that we use the mode `'same'` inside `np.correlate`: the cross-correlation has the same size the output (or the maximum size of the inputs if they differ)

```
1
2  def run_pulse_compression(ping : np.array, echo : np.array) -> np.array:
3
4    '''This function returns a pulse compressed version of the input ping echo
5    using 'ping' as reference.
6
7    Inputs:
8    - ping: np.array (dtype = 'complex'), array of the transmitted signal
9    - echo: np.array (dtype = 'complex'), array of the received echo
10
11   Output:
12   - pulse_compressed_signal: np.array (dtype = 'complex'), array of the pulse compressed singal'''
13
14   '''We run the match filter in the time domain using numpy's correlate function.'''
15   pulse_compressed_signal = np.correlate(echo, ping, mode = 'same')
16   '''Normalization step: we normalize the cross-correlation by the product of the ping and the echo norms.'''
17   pulse_compressed_signal /= (np.linalg.norm(ping) * np.linalg.norm(echo))
18
19   '''We return the normalized cross-correlation'''
20
21   return pulse_compressed_signal
22
```

Listing 2: Pulse compression code

Pulse compression for TDMA data

- We apply pulse compression for the TDMA data $\implies$ the **reference signal** is the LFM UP chirp

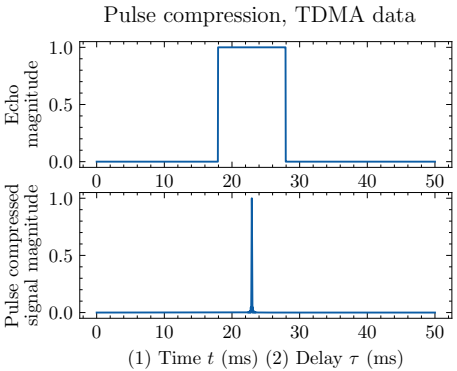


Figure: Pulse compression, TDMA data, $N_{tx} = 1$, $N_{rx} = 4$

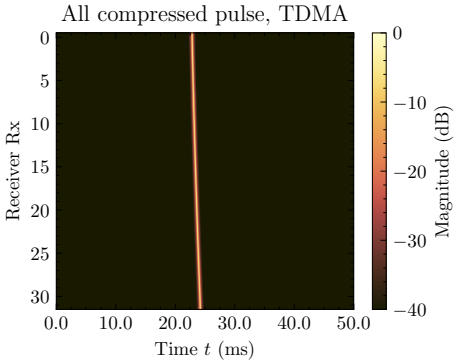


Figure: Pulse compression for $N_{tx} = 1$, and all receivers, TDMA data

- All the data / energy is **compressed** in the **shortest possible time extent**
- The match filter output is flat outside the peak (for CDMA data, this is not the case)
- When we run pulse compression for all the 32 receivers, we observe that the delay corresponding to the maximum increases: DOA or near-field (limit $d_f = \frac{D^2}{2\lambda}$)

Theoretical and practical time resolutions

- The match filter acts as a cross-correlation between the chirp and its echo
- Assuming that both the reflector and the medium **dot not alter the waveform**, the echo is a delayed version of the chirp
- Therefore, there is only one time delay such that the echo perfectly overlaps the chirp
- This explain why the output signal has a peak and is flat elsewhere
- All the data / energy is compressed in the shortest possible time extent. Both signals contain the same amount of data about the scene
- **What it the the shortest possible time extent?**
- **Theoretical time resolution:** $\delta t_t = \frac{1}{B} = 0.10$ ms
- **Practical time-resolution:** $\delta t_m = 0.13$ ms
- Perfectly symmetric sinc pattern
- Note that there is no AWGN here!

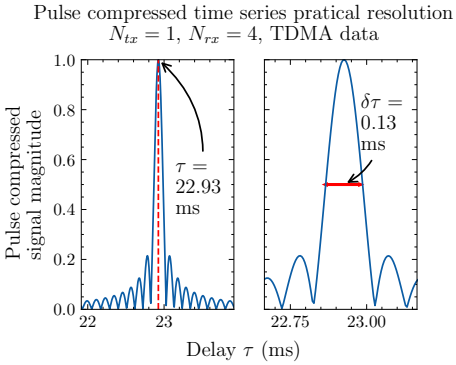


Figure: Pulse compression resolution for $N_{tx} = 1 N_{rx} = 4$, TDMA data

Theoretical resolutions

- **Virtual array:** constructed by placing a virtual element at the between a Tx/Rx pairs for all possible pairs

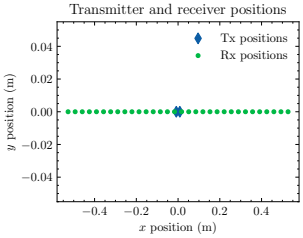


Figure: Physical array

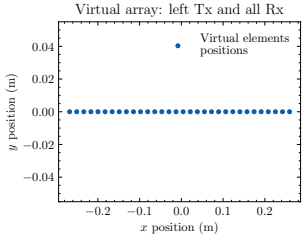


Figure: Virtual array (using only the leftmost transmitter and all receivers)

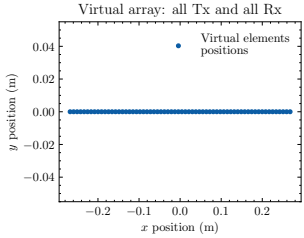


Figure: Virtual array (using all receivers and transmitters)

- When using two transmitters instead of one, we double the number of virtual elements ($N = N_{tx} N_{rx}$)
- Is the virtual 'well built'? The element spacing d **should be less or equal** to $\lambda/2$ to avoid **aliasing**
- Lowest wavelength: $\lambda = \dfrac{c}{f_c + B/2} = 22.7$ mm and $\lambda/2 = 11.3$ mm
- **With only one Tx and all receivers:** $d = 17.0$ mm $> \dfrac{\lambda}{2}$ ✗
- **With all Tx and all Rx:** $d = 8.5$ mm $< \dfrac{\lambda}{2}$ ✓

Theoretical resolutions

- **Lateral resolution** (in rad): $\delta\beta = \dfrac{\lambda}{2L_{SA}}$
- Which frequency? We use the minimum frequency $f_{min} = B + \frac{f_c}{2} = 15$ kHz $\leftrightarrow$ 22.7 mm to obtain **the best achievable resolution**
- $\delta\beta = \dfrac{22.7 \times 10^{-3}}{2 \times 1.054} = 1.08 \times 10^{-2}$ rad
- **Along-track** resolution at a range of $R = 4$ m: $\delta x = R\delta\beta = 4 \times 1.08 \times 10^{-2} = 43.1$ mm
- **Cross-track** resolution: $\delta y = \dfrac{c}{2B} = \dfrac{340}{2 \times 10 \times 10^3} = 17$ mm

DAS algorithm

- The presented algorithm is a simplified version of the one I used
- Please see `utils/parallel_DAS_beamforming.py` for the code used to generate the following images
- This code uses parallel computing which significantly improve the computation time of the image!

```python
def DAS_imaging_TDMA(grid_config : dict, rx_positions : np.array, tx_positions : np.array, tdma_data : np.array, B :
    float, fc : float,c : float, T_p : float, N_t : float, fs : float) -> np.array:

  x_values = np.arange(grid_config['x_min'],  grid_config['x_max'], grid_config['x_step'])
  n_x = len(x_values)
  y_values = np.arange(grid_config['y_min'],  grid_config['y_max'], grid_config['y_step'])
  n_y = len(y_values)
  image = np.zeros((n_x, n_y), dtype = 'complex')
  up_pulse = lfm_pulse(B = B, f_c = fc, T_p = T_p, fs = fs)
  N_rx = len(rx_positions)
  N_tx = len(tx_positions)
  for n_rx in tqdm(range(N_rx)):
    for n_tx in range(N_tx):
      echo = tdma_data[:, n_rx, n_tx]
      compressed_pulse = run_pulse_compression(ping = up_pulse, echo = echo)
      tmp_image = np.zeros_like(image, dtype = 'complex')
      tx_pos = tx_positions[n_tx]
      rx_pos = rx_positions[n_rx]
      for j in range(n_x):
        for k in range(n_y):
          pixel_x_pos = x_values[j]
          pixel_y_pos = y_values[k]
          tx_to_pixel_distance = np.sqrt((pixel_x_pos - tx_pos) ** 2 + pixel_y_pos ** 2)
          rx_to_pixel_distance = np.sqrt((pixel_x_pos - rx_pos) ** 2 + pixel_y_pos ** 2)
          time_delay = (tx_to_pixel_distance + rx_to_pixel_distance) / c
          if index_delay < N_t:
            tmp_image[j, k] += compressed_pulse[index_delay]
      image += tmp_image
return x_values, y_values, image.T
```

Listing 3: Pulse compression code

Experimental setup

- To image the scene, we use the following grid:
  - x-axis: $[-5, 5]$ m with a resolution of 10 mm (10^3 points)
  - y-axis: $[0, 5]$ m with a resolution of 5 mm (10^3 points)
- Impossible to know if the target lies between 0 and 5 m or 0 and -5 m because of the array symmetry along the x-axis
- Given the pulse compressed time series, we expect to find only one reflectors, in the upper left part of the image
- Thereby, we assume that the target is located between 0 and 5 m
- Since the is no noise, the point scatterer will be clearly distinguishable from the background (no energy)

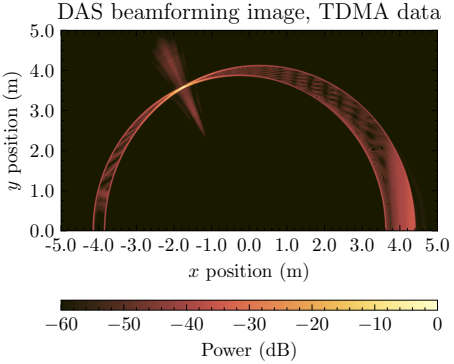


Figure: TDMA dataset, all Tx/Rx

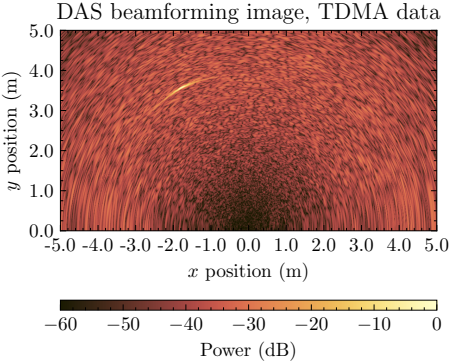


Figure: TDMA dataset, all Tx/Rx + noise

DAS on TDMA dataset

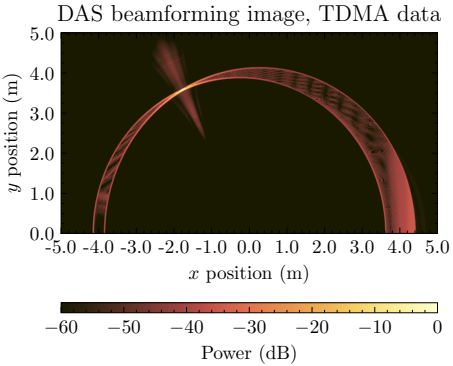


Figure: TDMA dataset, all Tx/Rx

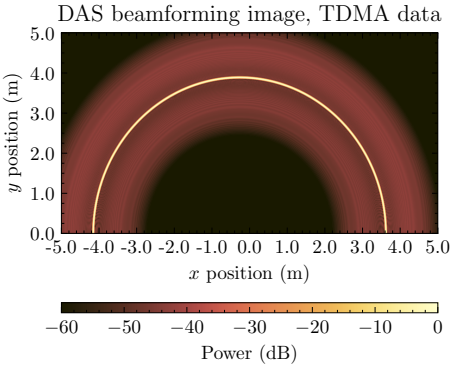


Figure: TDMA dataset, one transmit sequence (Tx 0 $\rightarrow$ Rx 0)

- Reflector position: $(x, y) =$.
- The image shows the superposition of the pulse compresses delayed series. So, the brighter a pixel is, the better is the position estimate
- Note that the energy is spread in the reflector direction
- With one transmit: we have one arc-circle with maximum brightness. Since we use only one Rx/Tx couple, all the pixel which such that $r_r + r_t = d$ while have the same delay. This leads to the arc-circle pattern

Resolution

Resolution (in mm)	Theoretical	Measured
Along-track	43.1	47.0
Cross-track	17.0	25.5

DAS on CDMA dataset

How to deal with CDMA data?

- We don't know which transmitter is used for the transmit sequences $\longleftrightarrow$ we don't if the used waveform is an up or downchirp
- Since the two waveforms are orthogonal, the pulse compression step will return a peak if we use the correct chirp and 0 if not (upchirp in the example)
- Using the linearity of the cross-correlation, we can modify the pulse compression step in the following way:

$$C_{echo,chirps}(\tau) = C_{echo,upchirp+downchirp}(\tau) = \underbrace{C_{echo,upchirp}(\tau)}_{\approx 1} + \underbrace{C_{echo,downchirp}(\tau)}_{\approx PCRR \approx 0.09} \tag{1}$$

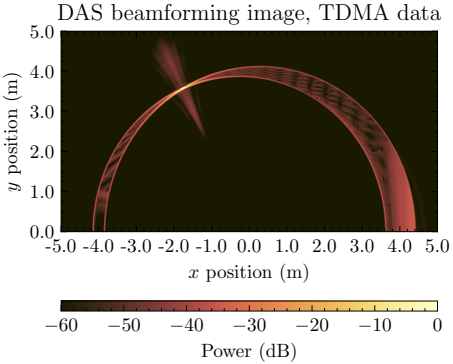


Figure: TDMA dataset

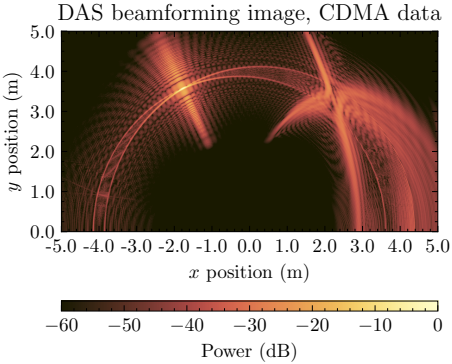


Figure: CDMA dataset

DAS on CDMA dataset

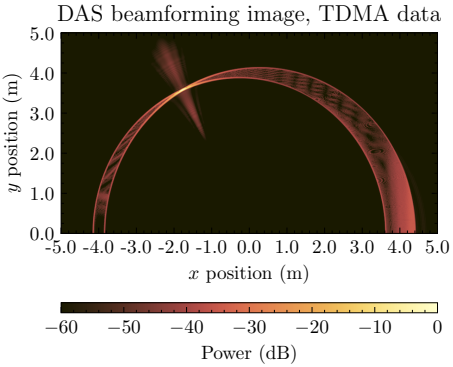


Figure: TDMA dataset

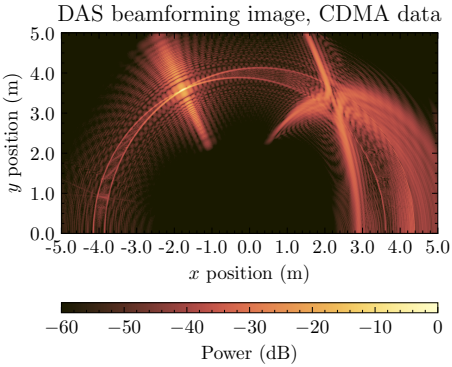


Figure: CDMA dataset

- Using CDMA data leads to the same position estimate
- Nevertheless, one can notice the circle pattern is wider than for TDMA dataset
- This side lobe is caused by the fact that the waveforms are not truly orthogonal
- Moreover, the is a wide side lobe at $(3, 3)$ m
- Pollution level
- Resolution

UiO : Department of Informatics
University of Oslo

Thomas Aussaguès

Mandatory exercise 3

MIMO Pulse-echo imaging