Introduction

In this assignment, you need to write code for reconstructing an image using the Delay-and-Sum (DAS) algorithm that is commonly used in Ultrasound image reconstruction.

NOTE: This is a very high level overview of the DAS algorithm, and the actual reconstruction process involves more detailed calculations - in particular, the aperture of the imaging system, windowing techniques to reduce distortion etc. are commonly applied in practice, but are ignored here.

Procedure

The operations involved in simulating an ultrasound transmit-receive system are broken down in a stepwise manner. In most cases, what is provided is the equations and figures, and you need to write corresponding Python code to implement the reconstruction. Finally, you will also need to apply your code to new datasets and try to reconstruct the images to identify the location of the obstacles.

There are a number of **Questions** that are marked in bold font and your final report needs to contain answers to these questions. In this document, there are also parts marked as CODE where you will need to fill in the corresponding code.

Submission

You need to create a single Python notebook or Python file that contains your code as well as answers to the questions. You also need to submit a PDF file that includes figures illustrating the outputs of the individual steps. Combine both of these into a single ZIP file in the usual manner and submit on Moodle.

Code helpers

Some example pieces of code are shown here to help you get started. However, you should create your own Python notebook and set these up properly. The documentation and setup of the notebook is part of the requirement for this assignment.

Input data

There are two text files containing samples corresponding to two locations of obstacles. You should be able to load them using the `numpy.loadtxt` function - they are matrices corresponding to 64 mics and 200 samples per mic. Reconstruct the images from these and interpret your results.

Problem setup

Sound source

Assume that you are given a single *source* of sound which is located at the coordinates (0,0) - the units are not relevant - you may think of them as metres if you wish, or any other distance unit. Similarly, time units are also not relevant, and all measurements here are relative. The source is capable of emitting a sound wave that spreads out uniformly from the source in all directions in a perfect circle.

Microphones

You have an array of `Nmics` microphones that are located on the Y-axis at x=0. The microphones are separated from each other by a distance `pitch`, so that the total vertical distance of the microphones is `Nmics * pitch` centred around (0, 0). Note that as long as `Nmics` is an even number there will be no microphone present at location (0, 0). This does not really affect the final computations, but your code should correctly implement this.

Each microphone is a perfect analog to digital converter, and *samples* the sound wave that is incident on it. This means that the output of a microphone is a sequence of numbers, corresponding to the intensity of the sound wave hitting it at any given instant. In this case, instead of giving the time between samples, we are giving the distance between samples as the parameter `dist_per_samp`. This is just for convenience - you should be able to switch between this and the time per sample easily enough.

Obstacle

There is also a single *point obstacle* present somewhere to the right (x > 0). The y-coordinate of the obstacle could be anything, but assume it is within the range covered by the locations of the microphones. This obstacle is assumed to be a perfect point scatterer: if any sound wave hits it, it will be reflected back in all directions without any loss of amplitude.

The basic setup required for these parts is shown in the code below. Some of the questions will involve changing the parameters, so use the same names as mentioned here.

```
# Main system parameters: number of mics, number of samples in time
Nmics = 8
Nsamp = 50

# Source: x,y coordinates: x: 0+, y: [-Y, +Y] where Y determined by pitch
↪  and Nmics
```

```
src = (0, 0)

# Spacing between microphones
pitch = 0.1
# proxy for sampling rate
dist_per_samp = 0.1
# Speed of sound in the medium
C = 0.5
# Time dilation factor for sinc pulse: how narrow
SincP = 5.0

# CODE Locations of microphones
mics = []

# Location of point obstacle
obstacle = (3, -1)
```

The waveform emitted by the source follows the equation here: you need to set up an appropriate time basis for computing the samples.

```
# Source sound wave - time axis and wave
# sinc wave with narrowness determined by parameter
t = 0 # CODE Nsamp time instants with spacing of dist_per_samp
def wsrc(t):
    return np.sinc(SincP*t)
```

QUESTION: The plots below show two example `sinc` pulses. How will you generate pulses that look like this? Which parameter should be changed? What effect do you think this will have on the final image?

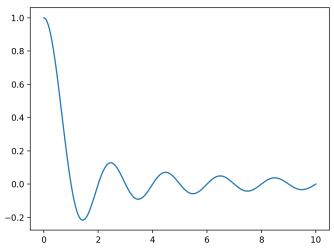

Figure 1: Sinc Pulse 1

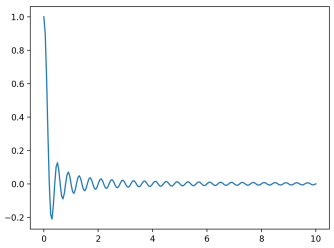

Figure 2: Sinc Pulse 2

You will also need a helper function to compute the distance traveled from the source to any microphone but with a reflection from a given point. This will be useful later as well.

```
# Distance from src to a mic after reflecting through pt
def dist(src, pt, mic):
    d1 = 0 # CODE distance from src to pt
    d2 = 0 # CODE distance from pt to mic
    return d1 + d2
```

Setup and Generating Mic Output

With the above setup, you can create the output of the microphones assuming that the source emits a wave starting from time 0. We assume there is only a single obstacle present, so each microphone will get back a reflected wave that has been delayed by an amount of time given by the time required to travel from the source to the obstacle and back to that mic.

An example of the output corresponding to the given parameters is shown here for the following parameters:

- `Nmics` = 64
- `Nsamp` = 200
- C = 2.0
- Obstacle location = `(3, -1)`

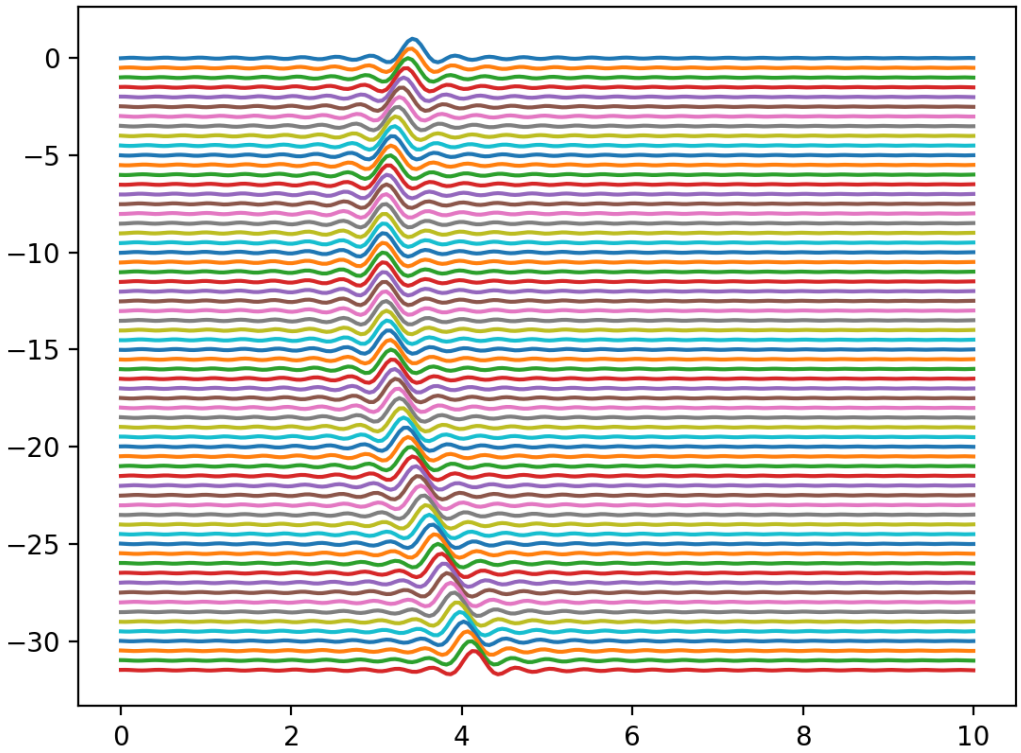

Figure 3: Reflected waves received at the mic array

The same can also be visualized as a heatmap as below:

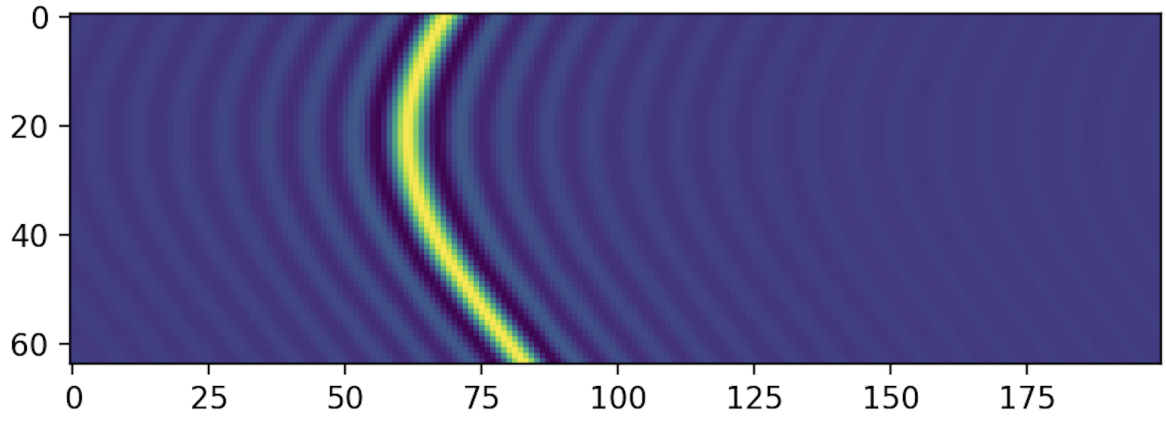

Figure 4: Heatmap of reflected waves

(Ignore the x- and y- coordinates in the plots).

CODE

Write code for generating samples corresponding to each mic for a given obstacle position and parameters as given above. Your code should be fully parametrized so that you can change and experiment with the parameters later.

Delay-and-Sum algorithm

The main idea behind the DAS algorithm is the following: if the distance to each of the mics was the same, then each mic would have received exactly the same signal. But since the distance is different, we will get delayed versions of the signal at each mic.

Therefore, by inverting this observation, we can expect that if we delay each of the mic signals by exactly the amount that we expect at each point on a grid, we may be able to align the signals such that they will add up to a maximum at the point where the obstacle is present, whereas all other points will have some amplitude that is not as high.

The basic idea behind this is shown in the figure here.

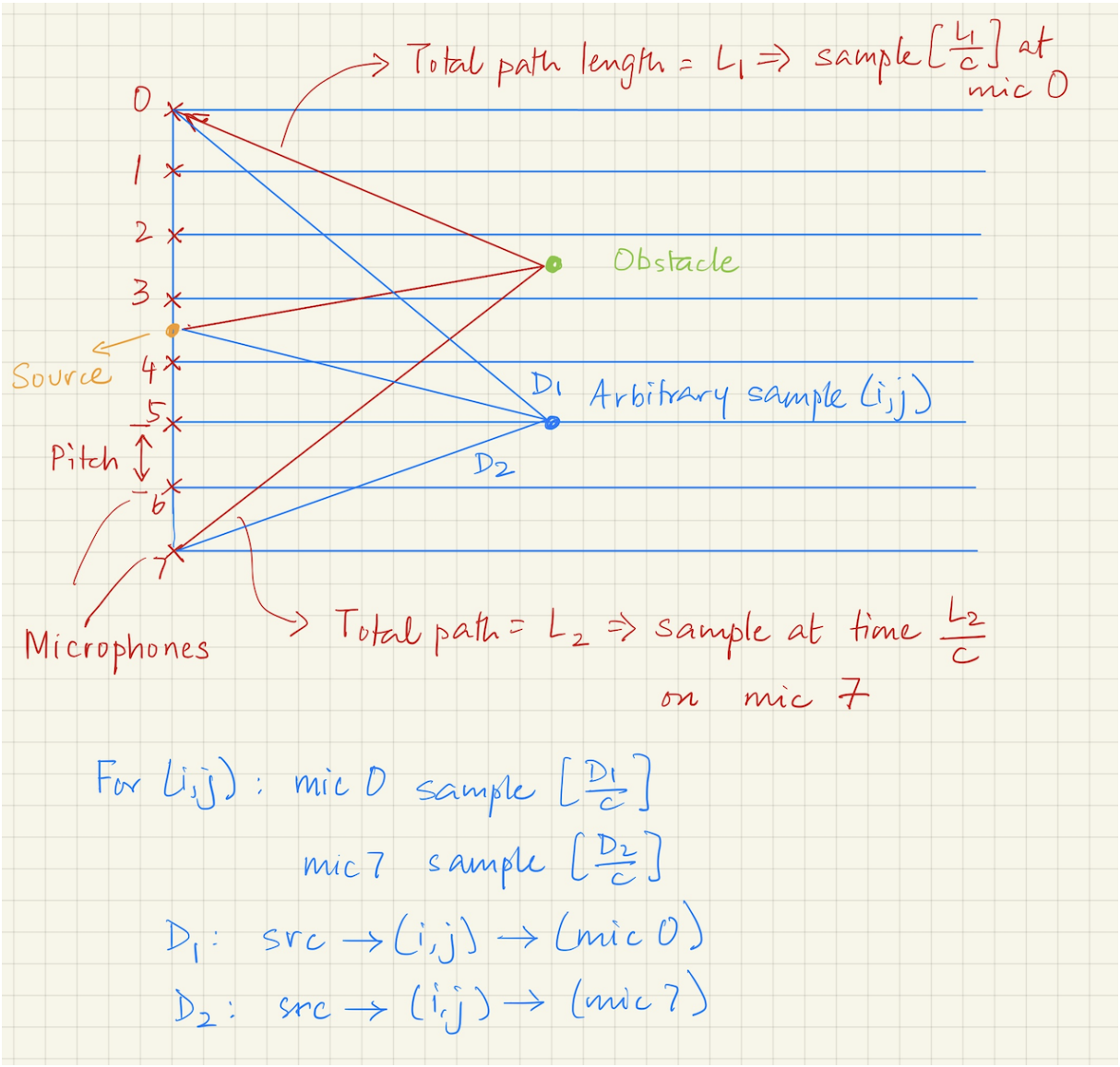

Figure 5: Basic Delay and Sum approach

As can be seen, what this means is that your algorithm should be as follows:

- Construct an (X, Y) grid of points where the Y-axis corresponds to the microphone locations, and the X-axis corresponds to time samples. The simple solution would seem to indicate that we go up to `Nsamp`.
  - **QUESTION** - Does it make sense to reconstruct up to `Nsamp`? What value is more reasonable as an upper limit for the x-axis here?
- For each point (i, j) on this grid, we can assume that the source signal reflects and goes back to

each of the `Nmics` microphones. If there is an obstacle at (i, j), this would be a strong signal, but if there is no obstacle, this would be a small or non-existent signal.

- Find the total time delay from src to (i, j) to each microphone, and pick up the corresponding points from the actual mic samples. Add up all these values (if the corresponding delay lies within the sampled range) to get the estimated value at (i, j).
- Repeat across all grid points to get a reconstructed image.

An example reconstructed image for the parameters shown above is given here:

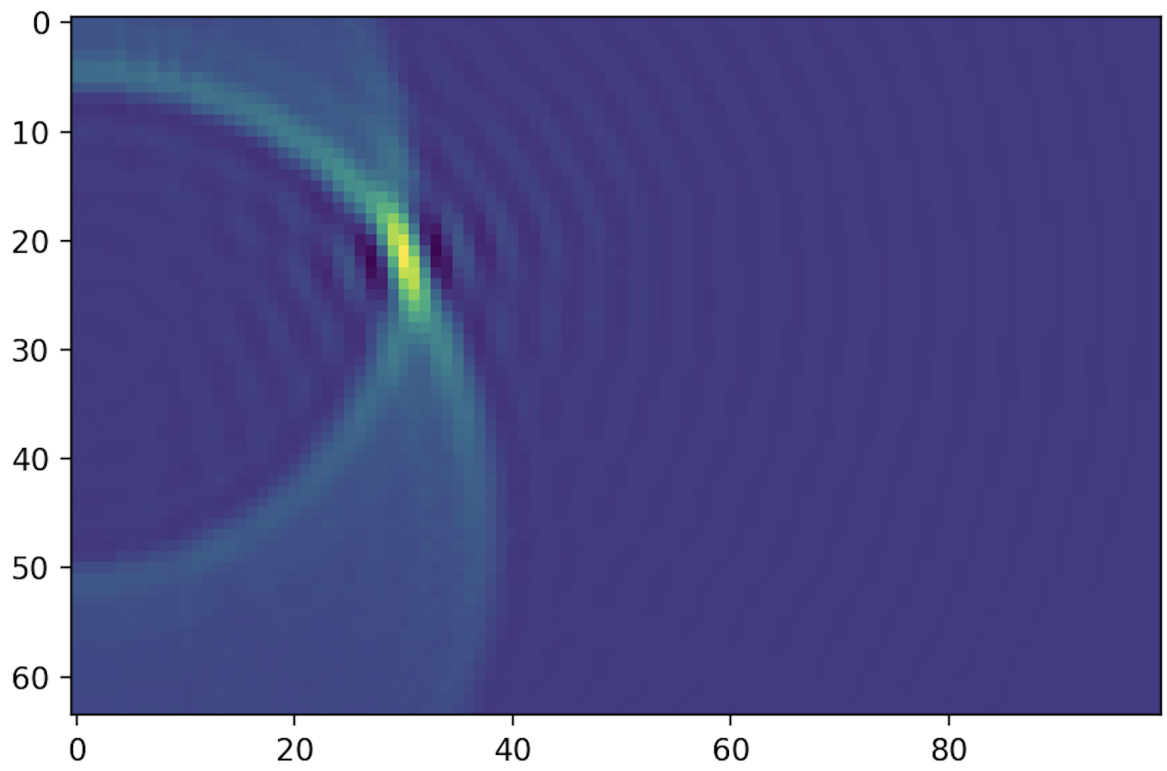

Figure 6: Reconstructed image

CODE

Implement the code required to implement the DAS algorithm and reconstruct the image from the data that you have generated previously.

QUESTIONS

- The (x, y) coordinates corresponding to the maximum amplitude (yellow colour) is approximately (30, 22). Explain why this is the correct expected position for the given obstacle.
- What is the maximum obstacle x- and y- coordinate that you can use and still have an image reconstructed?
- What happens if C is different - if C is decreased it looks like the image becomes sharper. Can you explain why intuitively?
- What happens if `Nmics` is increased or decreased? Do the experiments with `Nmics` = [8, 32, 64] and `Nsamp` = [50, 100, 200] (all combinations). Attach the resulting images.