Sound Localisation

Keshav Saravanan, EE23B035

November 2024

In this assignment, I write a script to estimate the position of obstacles given data about how sound reflects on those obstacles.

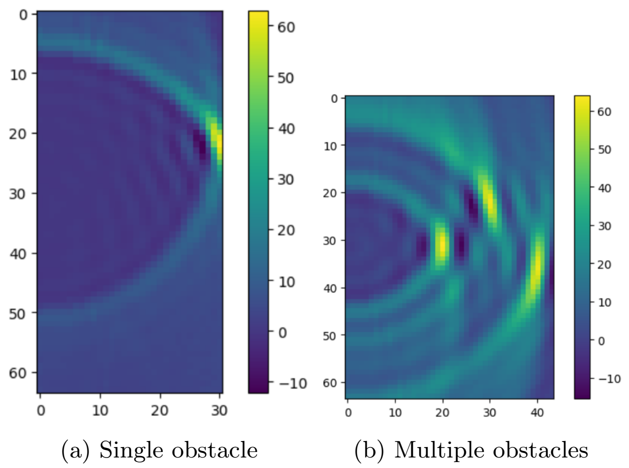
1 Running the Code

In the Jupyter Notebook - every section is dedicated to a question or task. To run one such section, ensure to run all cells under the heading, to overwrite variables that may have been written before.

2 Approach

The DAS algorith involves delaying the signal received at each mic by a certain amount, until the signals align up to give us a maximum. In this case, since our signal conveniently has only one peak - at `t = 0`, we only check if the signal sampled at that delay contributes a `1`. The delay corresponds to the travel delay of sound from the source, to the assumed obstacle, and then to each mic. If the assumed obstacle's position is not correct, the signals will not align up, and will probabilistically sum up to 0. If the assumed obstacle's position is correct, each mic contributes a `1`, which add up to yield a maxima.

3 Example Outputs



(a) Single obstacle (b) Multiple obstacles

Figure 1: Sample outputs of the DAS algorithm, for the code in the assignment

4 DAS With An Arbitrary Sound Signal

I tried this same algorithm with a sinusoidal signal, instead of a sinc wave. The algorithm works fairly well - it gives me the direction of the obstacle. However, it cannot pinpoint the exact location - this is because a sinusoidal wave has multiple peaks unlike a sinc wave, and so there are multiple peaks in the output heatmap.

5 Answers to Questions

5.1 Generating sinc pulses of different widths:

The width of the sinc pulse can be modified by changing any of the parameters `C`, `dist_per_samp`, and `SincP`. `SincP` decreases the pulse width of the sinc wave - by scaling time domain down, the frequency of the pulse increases. The same holds for `dist_per_samp` - increasing this, decreases the pulse width. On the contrary, increasing `C` increases the pulse width.

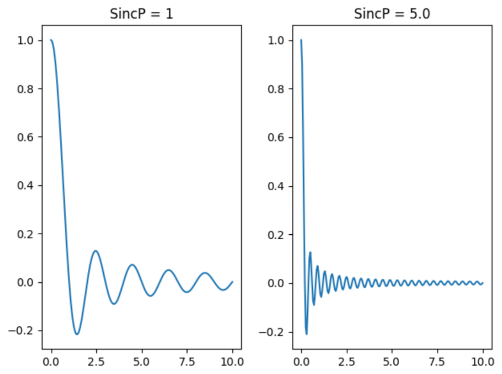


Figure 2: Varying the width of the Sinc signal

5.2 How far do we reconstruct our x-y grid to?

We do not need to reconstruct our sample space all the way up to `Nsamp * dist_per_samp / C` to find the obstacle's position, we can make do with much less. Suppose the peak value for the `i'th` mic is found at t_i, and the minimum such t_i be $(t_i)_{min}$. Then, the furthest distance we need to check for an obstacle is $(t_i)_{min}/(2C)$ - that is how far the obstacle would be from the closest mic. However in the case of multiple obstacles, we would have to take $(t_i)_{max}$ - otherwise further obstacles would not show up in the heatmap.

5.3 How does the coordinate system work?

With the example plot given, `Nmics` is 64 and `Nsamp` is 200. The peak value is reported at `(22, 30)`.

The y-axis is scaled by `pitch`, and shifted down, so that the origin is in between `Nmics/2 - 1` and `Nmics/2`. In this case, the origin is at `31.5`. The x-axis begins from the leftmost point on the image, and the scale is `dist_per_sample`.

This means, that over here, the obstacle is located at `(30 * dist_per_sample, (22 - 31.5) * pitch)`. This corresponds to `(3, -0.95)`, which is very close to our obstacle's location.

5.4 Maximum obstacle (x, y) coordinates:

The maximum time for which we are listening to input from the mic is `Nsamp * dist_per_samp / C`. That means the maximum x-coordinate for an obstacle is `Nsamp * dist_per_samp / 2` - since it will take sound double the time to travel to the object and back. If this is the case, we can be guaranteed that **atleast** one mic will pick up the reflected pulse from the obstacle in the sample time range. However, it will have to be slightly closer for a couple of reasons:

1. This distance is the worst case scenario for when the obstacle is lined up with one of the mics - if this is not the case, then the distance from the closest mic and the obstacle will be more than `Nsamp * dist_per_samp / 2` and no mic will detect it.
2. We need atleast two mics to pinpoint the location of the obstacle - so this decreases the distance ceiling even more.

The absolute maximum y-coordinate is `(Nmics / 2 - 0.5) * pitch`. This is the absolute value of the y-coordinates of the top and bottom mics - so any object with a greater absolute y-coordinate will not show up in the image.

5.5 How does C affect the heatmap?

Reducing `C` increases the sharpness of the image. This is because when `C` decreases, the width of the sinc pulse decreases - so the peak decays much faster. When we iterate over every point in the grid, the correct obstacle location will result in the value '1' being added up for each mic. Close to that correct location, the value will not be '1', but it is still likely to be some relatively large positive number. By decreasing the peak width, we reduce the magnitude of this positive number, and reduce the "spread" of the peak value in the heatmap.

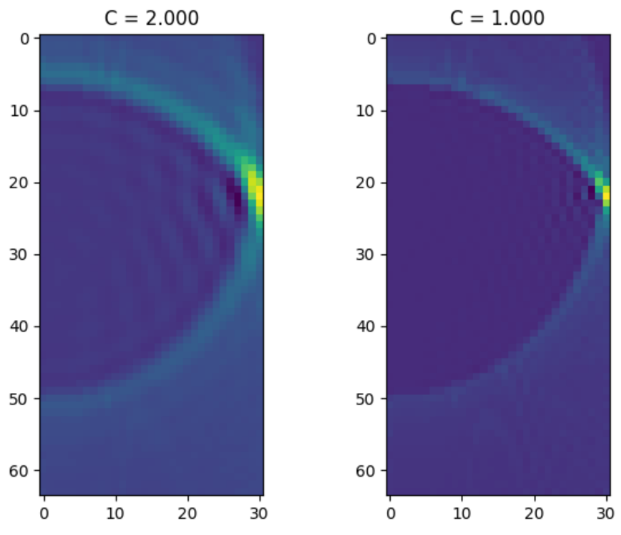


Figure 3: Varying the value of `C`

5.6 Varying Nmics and Nsamp

By varying the values of `Nmics` and `Nsamps`, the allowed bounds for the obstacle change - so I had to pick an obstacle location close to the source to ensure that all choices resulted in a detection. The obstacle's true location is `(1, -0.25)`

Decreasing `Nmics` and `Nsamps` reduces the accuracy of the DAS algorithm - this can be seen in the top-left. Increasing Nmics increases the precision of the search, since our y-axis now has more resolution. It also increases the sharpness, since summing the signal over multiple mics will probabilistically result in a value close to 0 closer to the object, than with fewer mics.

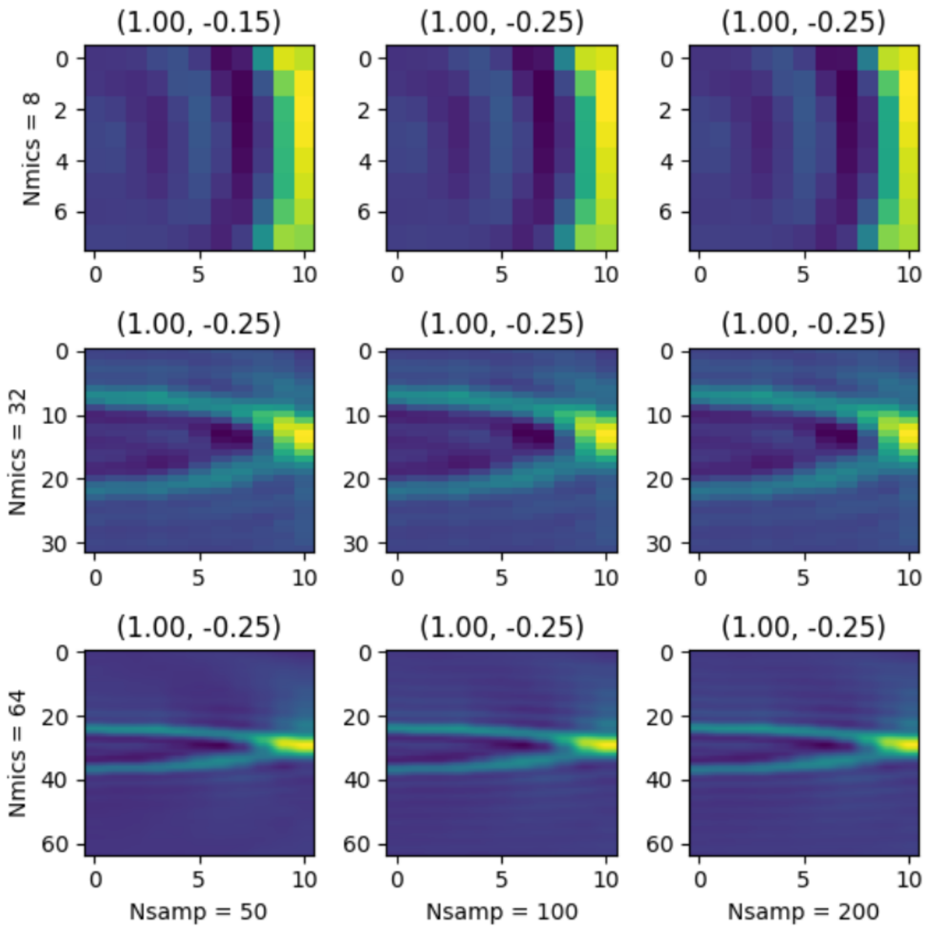


Figure 4: Varying `Nsamps` and `Nmics`

6 Findings

I learnt quite a bit about DAS, and read up about beamforming in general. I was interested in implementing this for arbitrary signals where simple sampling would not help - I'd have to find a way to compute the normalised difference between the shifted signals.

I also believe it is possible to improve this code - right now, it is limited in certain ways - we assume the y-axis is quantized according to the number of mics, and we cannot detect objects outside this y range. By modifying the code to handle such cases, it should be possible to detect objects from far away (although the accuracy will decrease as distance increases).