Developing a “Sound Steerer” in Three-Dimensional Space—Project Proposal

Sarkis Ter Martirosyan

Thomas Jefferson High School for Science and Technology

Abstract

My project uses beamforming to steer sound around a room and envelop a person a private cloud of sound undetectable to anyone else. Specifically, my project investigates various beamforming algorithms—notably Delay & Sum, Pressure Matching, and Energy Difference Maximization—with both quantitative and qualitative metrics. The results of my research not only apply to beamforming in the audio domain, but also to analog signals in the more complex RF domain. In today’s world, shaving a few seconds off loading times can drastically effect a company’s bottom line. To cut loading times on mobile devices, multiple International Mobile Telecommunications-Advanced standards have been created, most recently the 5G standard. So far, all standards have used the “shotgun” approach and send the same signal to all devices within range. 5G uses beamforming to “snipe” a signal only to the relevant mobile device, allowing for faster read and write speeds. The insights gleaned from my project can help to better understand 5G and possibly improve it.

Developing a “Sound Steerer” in Three-Dimensional Space—Project Proposal

Contents

Overview

Research Questions

1. What is the best way to use a one-dimensional array of speakers to localize sound at a single “bright point” in two dimensional space?

2. What is the best way to use a two-dimensional array of speakers to localize sound at a single “bright point” in three dimensional space?

Research Goals

1. To demonstrate using purely software means that the various algorithms explored are successful in their ability to beamform sound.

2. Design and build an audio-chain using a field-programmable gate array.

3. To develop a system using mixed hardware and software to:

   (a) Convert an analog signal to an 8-bit digital signal

   (b) Pipe the 8-bit data to an FPGA

   (c) Have the FPGA use a predetermined set of multipliers to adjust the incoming signal for each speaker in the speaker array

   (d) Multiplex the data in a way that each 8-bit bus branches out to 16 speakers

   (e) De-multiplex the data, pass the 8-bit data to a bank of DACs, and amplify the data to drive the speaker array

4. To develop a system to calculate the weight vector “$\hat{\mathbf{q}}$” and transmit that vector to the FPGA for multiplication.

5. Develop a web-client to allow for a user to easily adjust the target position of the beam-formed sound signal.

Rationale

Communications

The idea of beamforming serves as one of the key features of the 5G communications scheme (Kela et al., 2016). Investigating beamforming using the easier medium of sound instead of electromagnetic waves can provide some extremely valuable insights in the behavior of near-field beamforming. Beamforming is also integral to successful communications between self-driving cars (Jones, 2016). This kind of communication is also possible to achieve on a smaller scale, for example with swarm robotics.

“Shaping” Audio

The purpose of my project is to “shape” or “steer” sound to a particular location in a free field. This creates an acoustically bright spot and an acoustically dark area (de Bruijn, n.d.). There are numerous applications for this, such as to create personal sound without headphones or to create custom volume levels for each listener in a relatively open room.

Hardware

Modules

As described above, there are five major modules in terms of hardware. They are:

1. Analog-to-Digital converter to convert an incoming analog signal to an 8-bit digital signal.
2. Processor (Raspberry Pi) to calculate the $\hat{\mathbf{q}}$ vector.
3. FPGA to map $\hat{\mathbf{q}}$ on the incoming signal. Then, on the FPGA, multiplex the data in the way described above.
4. ADC and demux unit. [1]
5. Power-amp to smooth and amplify the analog signal coming from the demux to drive a speaker. [1]

The design is shown in Figure 1.

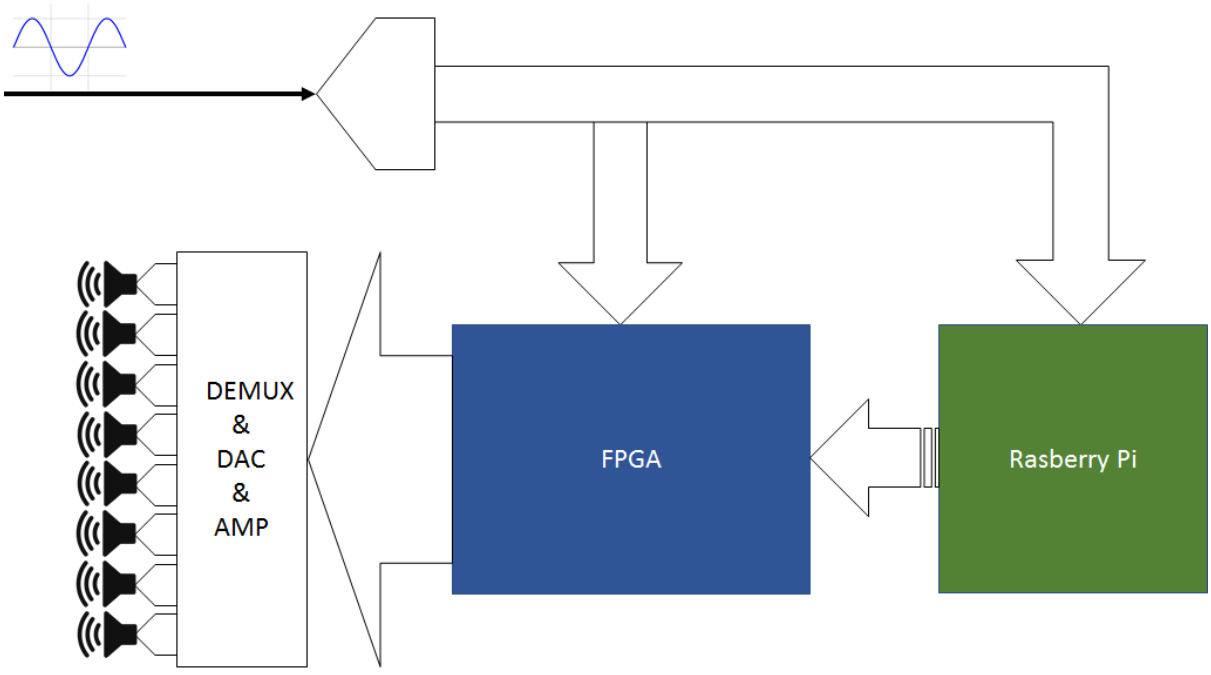


Figure 1. Block Diagram of Hardware

[1] The output module is better described in my first quarter schematic and engineering notes

Why the Raspberry Pi?

I believe that a Raspberry Pi is the optimal tool for the calculation of the $\hat{\mathbf{q}}$ vector because of the following features.

1. The ability to pre-process $\hat{\mathbf{q}}_f$ for each frequency specified by an FFT of S samples and store those values in a Python “pickle” file for quick accession at a later time. The time gains from using this method would be a huge boon to achieving near real-time digital signal processing.

2. The Raspberry Pi, with its more advanced and high-level architecture, allows for more rapid development. While the processes executed on the Raspberry Pi are technically possible on an FPGA (notably, the operation of the Fast Fourier Transform and the calculating of $\hat{\mathbf{q}}$), the code to accomplish the same task is vastly simpler on the Raspberry Pi, thanks to Python and its libraries.

3. Thanks to using Python as the base language for digital signal processing, I can take advantage of agile development to test a multitude of algorithms in rapid succession.

Software

There are two main software components: Python on the Raspberry Pi and Verilog on the FPGA.

Python

There are a number of tasks that will be accomplished on the Raspberry Pi. They are:

1. Calculating the $\mathbf{Q}$ matrix[2] and “pickling”[3] it.
2. Parsing incoming analog data using the “pyaudio” library.
3. Performing operations[4] on the parsed data using the “numpy” library.
4. Output the calculated $\hat{\mathbf{q}}$ vector to the FPGA via serial using the “serial” library.

The decision to eschew MATLAB was not easily made. But, because MATLAB (a *programming language*) accomplishes the same task as numpy (a *Python library*), and the DSP is more iterative than parallel, a processor makes sense. Being able to simply pre-process and recall (using “pickle”) the $\mathbf{Q}$ matrix rather than recalculating it anew each iteration is also a huge boon to accomplishing near-real time performance. While a ROM could technically be used, the cost (in terms of pin-usage and needless extraneous circuitry) is too much[5].

---

[2] The $\mathbf{Q}$ matrix is an array of N_{FFT} $\hat{\mathbf{q}}_f$ column vectors.

[3] Pickling, its benefits, and why I chose to include as an important part of my DSP system, is explained in my quarter one engineering notes.

[4] Those operations are performing an fft on the incoming data and using the fft find a weighted average of the $\mathbf{Q}$ matrix. This gives the $\hat{\mathbf{q}}$ vector.

[5] Specifically, the cost is

$n_{real} \times n_{imag} \times n_\theta \times n_\phi \times N_{FFT} \times N_{speakers} = 2 \times 2 \times 180 \times 180 \times 1024 \times 16 = 2123366400 \approx 2^{31}$. That means that the address line from the FPGA alone would occupy nearly all the pmod headers on the FPGA to achieve the same operating speed as the proposed block diagram. Note that the calculation of this cost in my first quarter engineering notes was incorrect, as I forgot to include N_{FFT}.

FPGA

There are also a number of important tasks that the FPGA is responsible for. They are:

1. Inputting and parsing the incoming serial data from the Raspberry Pi.
2. Mapping the recieved $\hat{\mathbf{q}}$ vector on the incoming audio signal.
3. Multiplexing the data and outputting that digital signal to the demux/output unit (described above).

Algorithms

In my project, I have decided to investigate two beamforming algorithms: Delay and Sum (DAS) and Pressure Matching (PM). The following equations are based on Olivieri et al. (2016).

Math Foundations

The relationship between L sources and M control points is

$$\mathbf{p}(\omega) = \mathbf{Z}(\omega)\mathbf{q}(\omega)s(\omega) \tag{1}$$

where $\mathbf{p}(\omega)$ are the sound pressure at the M control points, $\mathbf{q}(\omega)$ is the vector of weights for the L sources, $\mathbf{Z}(\omega)$ is the transfer matrix, and $s(\omega)$ is set to 1 at each angular velocity ω.

The transfer matrix $\mathbf{Z}$ is

$$\mathbf{Z}_{M\times L}(\omega) = \begin{bmatrix} Z(\mathbf{x}_1, \mathbf{y}_1, \omega) & \dots & Z(\mathbf{x}_1, \mathbf{y}_L, \omega) \\ \vdots & \ddots & \vdots \\ Z(\mathbf{x}_M, \mathbf{y}_1, \omega) & \dots & Z(\mathbf{x}_M, \mathbf{y}_L, \omega) \end{bmatrix} \tag{2}$$

Where the transfer function is defined as

$$Z(\mathbf{x}_m, \mathbf{y}_\ell, \omega) = \frac{e^{-j\frac{\omega}{c}||\mathbf{x}_m - \mathbf{y}_\ell||}}{4\Pi \,||\, \mathbf{x}_m - \mathbf{y}_\ell \,||} \tag{3}$$

This is all based on Euler's Formula $e^{j\theta} = \cos\theta + j\sin\theta$

For simplicity's sake, ω will be omitted in all further equations.

Math for DAS

The equation that gives us the $\mathbf{q}$ using the DAS method is

$$\mathbf{q}_{DAS} = \mathbf{\Gamma}\mathbf{z}_B^* \tag{4}$$

Where $[\,\cdot\,]^*$ is the operation of complex conjugate. The vector $\mathbf{z}_B$ is the transfer vector for the singular bright point and the matrix $\mathbf{\Gamma}$ is defined as

$$\mathbf{\Gamma} = \begin{bmatrix} \gamma_1 & 0 & \dots & 0 \\ 0 & \gamma_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \gamma_L \end{bmatrix} \tag{5}$$

Where $\gamma_\ell = \frac{16\pi^2 ||\mathbf{x}_B - \mathbf{y}_\ell||^2}{L}$.

Math for PM

The equation that allows us to derive $\mathbf{q}$ using the PM method is

$$\min_{\mathbf{q}} J_{PM} = \min_{\mathbf{q}} (\mathbf{e}_{PM}^H \mathbf{e}_{PM} + \beta_{PM} E_q) \tag{6}$$

Where $\mathbf{e}_{PM} = \hat{\mathbf{p}} - \mathbf{Z}\mathbf{q}$. The Tikhonov regularization parameter β_{PM} is given by gradually increasing β_{PM} until this inequality is satisfied

$$E_{\mathbf{q}} = \mathbf{q}^H \mathbf{q} \leq E_{max} \tag{7}$$

The final relationship used to calculate the vector $\mathbf{q}_{PM}$ is

$$\mathbf{q}_{PM} = (\mathbf{Z}^H \mathbf{Z} + \beta_{PM} \mathbf{I})^{-1} \mathbf{Z}^H \hat{\mathbf{p}} \tag{8}$$

Discoveries Thus Far

From tinkering with the algorithms in a Jupyter notebook, I’ve elucidated some of the intricacies of the two algorithms. Notably:

1. $\mathbf{q}_{DAS}$ is much, much faster to compute than $\mathbf{q}_{PM}$
2. PM has much better directive performance than DAS
3. both algorithms perform better at higher frequencies and worse at lower frequencies

The following figures highlight points 2 and 3:

Figure 2. Delay Sum

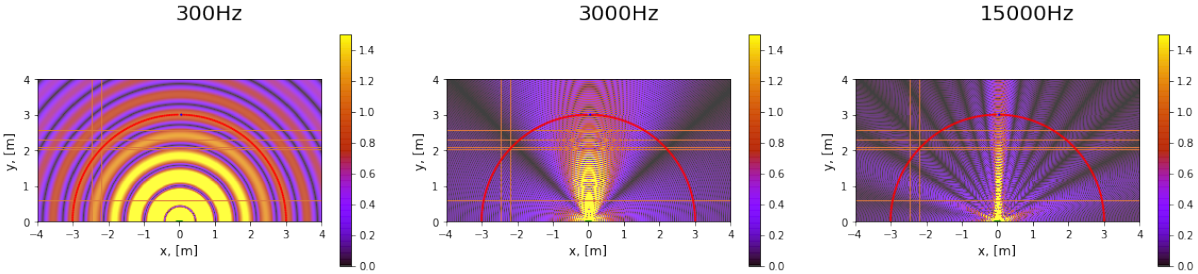
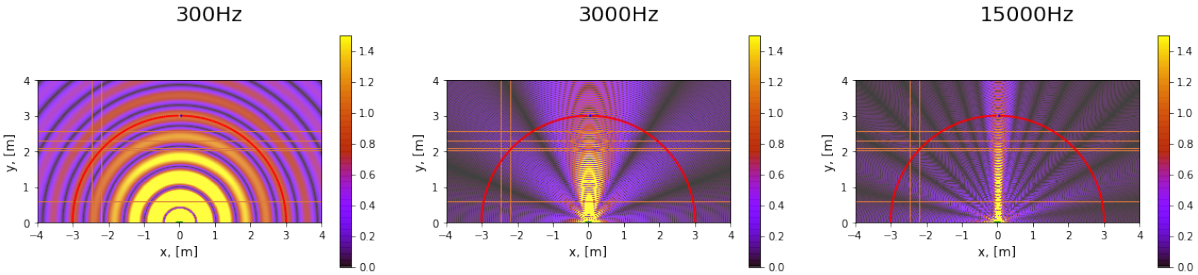


Figure 3. Pressure Matching



Planned Schedule

Schedule

The following is a tentative schedule that is subject to change, and will likely change based on a host of extraneous factors not under my control. The schedule is based on month and includes August+September and October.

1. August+September : Take care of lab service; begin research.
2. October : Investigate pertinent literature and play around with relevant algorithms; brainstorm ideas; make and finalize a block diagram; create research plan and project proposal.
3. November : Design and build a full audio chain (everything but the Raspberry Pi).
4. December : Work on having the Raspberry Pi perform an FFT, calculate $\hat{\mathbf{q}}$, and transmit $\hat{\mathbf{q}}$ to the FPGA.
5. January+$\frac{1}{2}$February : Built-in buffer time; iron-out bugs.

6. $\frac{1}{2}$February+March : Expand the speaker array to include 24 speakers in a 3×8 configuration.

7. April : Create a web client to specify to the Raspberry Pi where to steer audio; create website to showcase lab research

8. May : Finish everything; present

9. June : Graduate!

Tangibles

These are timely, reachable goals to ensure that my project is finished in a timely manner. Perhaps they could be called my personal SMARTR Goals, which will ensure that my research is on track to completion in June.

1. August+September : Make the electronics lab look great.

2. October : Demonstration of all the algorithms, their drawbacks and advantages

3. November : Play Rick Astley’s *Never Gonna Give You Up* and hear the audio intelligibly.

4. December :

    - Have the Raspberry Pi successfully corroborate the results of synthetic testing.
    - Have the Raspberry Pi compute the $\hat{\mathbf{q}}$ vector in near real-time to Rick Astley’s *Never Gonna Give You Up*.
    - Have the FPGA succesfully recieve and decode the $\hat{\mathbf{q}}$ vector from the Raspberry Pi.
    - Beamform Rick Astley’s *Never Gonna Give You Up* successfully using a 1×8-speaker array.

5. January : See October & November & December.

6. February : Nothing new.
7. March : Beamform Rick Astley’s *Never Gonna Give You Up* successfully using a 3×8-speaker array.
8. April :
    - Be able to successfully beamform Rick Astley’s *Never Gonna Give You Up* to any location on a hemisphere with a radius of 3 meters using a web client.
    - Create a website to showcase my year-long research.
9. May : Have everything working up to spec and present my research at TJ STAR.
10. June : Graduate!

References

de Bruijn, W. (n.d.). *Making all the right noises: Shaping sound with audio beamforming.* Retrieved from https://www.mathworks.com/company/newsletters/articles/making-all-the-right-noises-shaping-sound-with-audio-beamforming.html

Jones, W. D. (2016, May). *How should a self-driving car tell you to take the wheel?* IEEE. Retrieved from https://spectrum.ieee.org/cars-that-think/transportation/self-driving/how-should-a-selfdriving-car-tell-you-it-needs-you-to-take-over

Kela, P., Costa, M., Turkka, J., Koivisto, M., Werner, J., Hakkarainen, A., ... Leppanen, K. (2016). Location based beamforming in 5g ultra-dense networks. In *Vehicular technology conference (vtc-fall), 2016 ieee 84th* (pp. 1–7).

Olivieri, F., Fazi, F. M., Nelson, P. A., Shin, M., Fontana, S., & Yue, L. (2016). Theoretical and experimental comparative analysis of beamforming methods for loudspeaker arrays under given performance constraints. *Journal of Sound and Vibration*, *373*(Supplement C), 302 - 324. Retrieved from http://www.sciencedirect.com/science/article/pii/S0022460X16002340 doi: https://doi.org/10.1016/j.jsv.2016.03.005