Sound Source Localizer

Joren Lauwers, Changping Chen
{jorenl, ccp0101}@mit.edu
6.111 Introductory Digital Systems Laboratory, Final Project
Prof. Gim P. Hom

Abstract

As our final project, we built a sound source localizer - a system that determines the position of human speech, and displays it on a two-dimensional map on a screen, using only cheap microphones and on-the-fly calculation on a Nexys4 FPGA board. The sounds picked up by the microphones are compared to determine how much time the sound spent traveling through the air, which corresponds to how far the microphones are away from each other. This report details the process of how we approached the design of this system, some technical choices and limits involved, and some mathematical background that brought us to the working set-up in Figure 1.

Goals

Our primary goal was to locate the origin of sound in a room using only microphone input. Initially, we were thinking of using three stationary microphones, and using only the signal delay between them to locate the sound, in a manner similar to what GPS does with satellites. After discussions with instructors, we proposed to first use a music playing speaker as our sound source. This setup

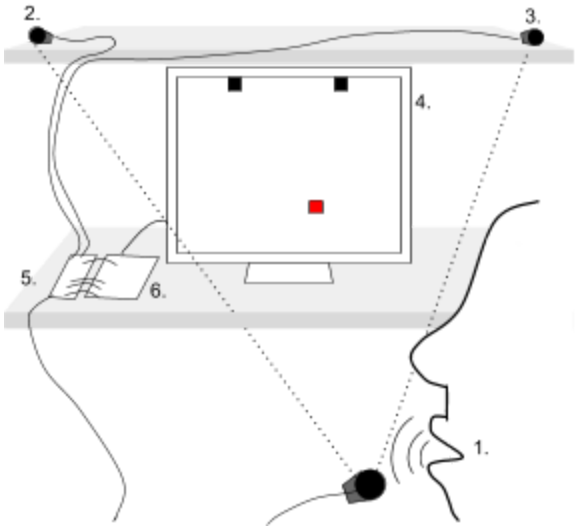


Figure 1: Schematic representation of our finalized set-up. 1. Sound source 2. Microphone A 3. Microphone B 4. VGA Display 5. Microphone preamplifiers 6. Nexys4 FPGA board

requires only two microphones, because the FPGA can determine the location of the speaker by measuring the delay between audio samples sent to the output speaker and those received from microphones. The information you get is the total travel time of the sound from point to point, not the differences between them. This makes the mathematics behind this also simpler.

We kept the three-microphone idea as our stretch goal, and eventually implemented an in-between approach: with three microphones, localizing free human speech,

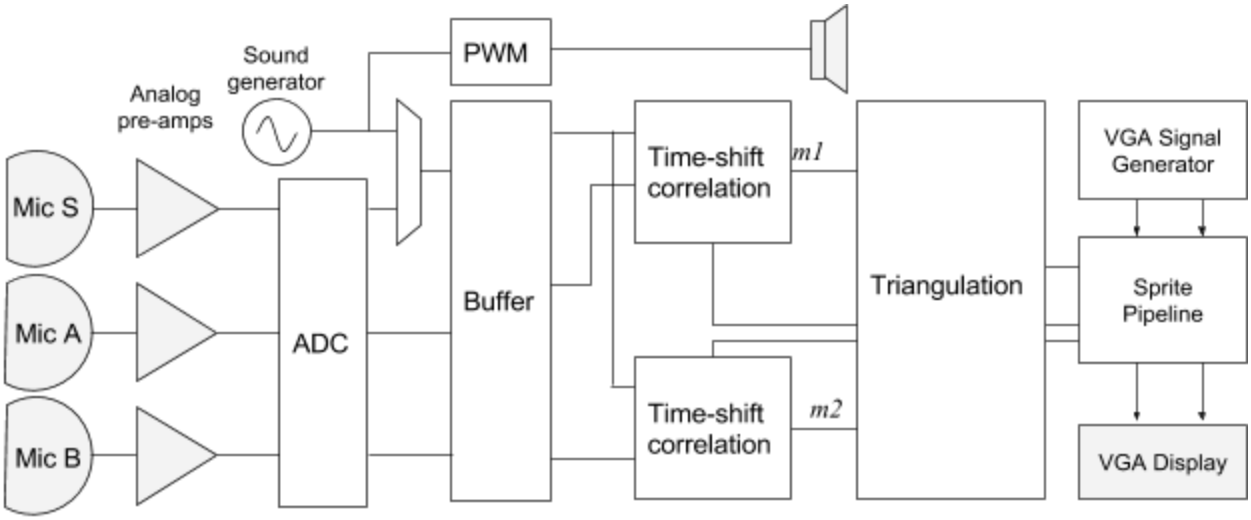


Figure 2: Block Diagram of our sound localizer design. Parts in grey are physical, parts in white are FPGA modules

but in a triangular set up, where we keep one microphone at the sound source. We find that this yields higher accuracy than with controlled output through a loudspeaker.

As shown in Figure 1, our finalized test set-up consisted of two stationary microphones, a fixed distance apart (we used 1000mm) at eye level and a voice microphone carried by the speaker, all fed into the FPGA through external pre-amplifiers. The calculated origin of the sound picked up by the microphones is then displayed on a monitor.

Modules

As outlined in the block diagram in Figure 2, the digital part of our system consists of an Analog-to-Digital converter which reads the microphone signals, two time-shift correlation modules that continuously compute how much one signal lags behind the other, a triangulation module which computes a position based on these lag values, and a VGA display system to visually show the results. For the controlled-sound mode of operation, an SD card reader module reads waveform data from a memory card, and plays it through a Pulse-Width Modulation (PWM) based output, which is connected to an external speaker. This section goes into more detail for each of these modules.

Circuit: Audio Input Preamplifier

As described in our initial proposal, we built three identical preamplifiers for microphone signals on a single breadboard. We purchased a set of 3.5mm stereo jack terminals[1] that allow us to connect a microphone input to a breadboard, and then followed the circuit diagram for an op-amp

[1] 4 Pack CESS 3.5mm 1/8 Stereo Balanced Female Jack to AV Screw Video Balun Terminal https://www.amazon.com/CESS-Stereo-Balanced-Female-Terminal/dp/B017CBTLJK

based audio preamplifier from an online guide[2] reproduced in Figure 3.

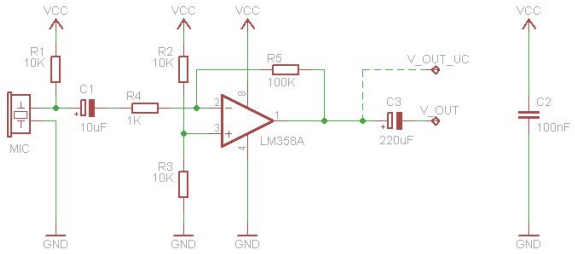


Figure 3: Microphone pre-amplifier circuit

We learned from the XADC datasheet that it internally uses a 1V reference voltage. To enable bipolar mode[3], the DC offset of our preamplification stage should be set at 0.5V, and that its output range should be $0.5V \pm 0.5V$. Therefore we added a voltage divider with $50k\Omega$ and $9k\Omega$ resistors between $3.3V$ and GND to obtain roughly $0.5V$ DC voltage, and connected it to V_{OUT}. We observed on an oscilloscope that the gain of our amplifier is appropriate for normal speech volume up to one meter away. V_{OUT} is the final output of the preamplifier, and connected to positive input pins of Nexys 4 XADC.

After we built the amplifier, we first tried using high quality stage vocal microphones. However, by design they only pick up audio in a short range (roughly 10cm in our experiment). We also suspect that their internal construction introduces unnecessary filtering and delay, so we opted to purchase three cheap condenser microphones similar to the those on the headset in our labkit.

Input Sampling with ADC

File: xadc_reader.v - author: Changping

The Artix-7 FPGA on the Nexys 4 board features two 12-bit analog-to-digital converters[4]. The two ADCs can be configured to simultaneously sample two external-input analog channels, at a sampling rate of 1MSPS. Initially we configured the XADC to sample two microphone inputs connected to $vaux3$ and $vaux11$, using Simultaneous Sampling mode, triggered by a 50 KHz tick. This ensures both channels are sampled precisely at the same time.

Ideally to implement our stretch goal of three microphones, we want to sample all three channels at the same time. However the XADC only has two cores, and as such can only sample from two channels simultaneously. We enabled its internal input multiplexer, which automatically switches to another pair of channels when the current pair of channels are acquired, sampled, and stored. We selected channel 3, 11, 2, and 10 through the IP wizard in de vivado design suite, since only these channels have physical I/O pins on the Nexys4 board. A sequence of two

[2] LM358 microphone amplifier https://lowvoltage.wordpress.com/2011/05/21/lm358-mic-amp/

[3] 7 Series FPGAs and Zynq-7000 All Programmable SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter User Guide https://www.xilinx.com/support/documentation/user_guides/ug480_7Series_XADC.pdf

[4] XA Artix-7 FPGAs Overview http://www.xilinx.com/support/documentation/data_sheets/ds197-xa-artix7-overview.pdf , http://www.xilinx.com/support/documentation/user_guides/ug480_7Series_XADC.pdf

conversions yields samples on all selected channels.

Because the XADC can sample at a rate much faster than our requirement of 50KHz, we can treat a sequence of sampled data as if they were sampled simultaneously, if we minimize the delay between the two conversions. To achieve this, we made sure to trigger another conversion immediately after the first conversion in a sequence is completed:

```
// start conversion if
// (a) there's more channel to be sampled
//     in a sequence (eoc & (~eos))
// (b) new sample needed per 50KHz

wire xadc_convst_in = (xadc_eoc &
(~xadc_eos)) | clk_sample_tick;
```

To read digital values from XADC post-conversion, we wrote a module to sequentially read its status registers at designated memory addresses. These values are fed into cross-correlation modules.

Cross-correlation

File: cross_correlation.v - author: Changping

The purpose of the correlation modules is to compare two input signals, and determine how much (in terms of number of samples, which corresponds to time) one lags behind the other.

Theory

This module calculates the cross correlation score (dot product) of two input signals when one is time shifted by a lag m over a window size of *W*. It's a common method of comparing the similarity of two time series when one lags behind another. This module is used later to calculate the time delay between when a signal is sent and the same signal is received.

The cross correlation of two inputs A and B over a finite sample window of W where B lags behind A by m samples, is given by

$$C(n,\ m) = \sum_{k=0}^{W-1} A[n+k] \cdot B[n+k+m]$$

We want to calculate the cross-correlation score for each possible lag m, and then use the lag associated with the highest score to calculate the difference in time of arrival.

Notice that $C(n,\ m)$ depends on the future value $B[n+W-1+m]$ and the current value $A[n]$, so we need to store at least $m+W$ samples, or $S = m+W$. The requirement on m is discussed in the next section.

Another implication from the equation above is that our score will be delayed by S sampling cycles, but this delay is negligible in human perception.

Example

In the example below, we choose W=5 and S=10. Suppose there exists a signal

```
0 0 0 0 0 1 2 3 4 5 4 3 2 1 0 0 0
0
```

This signal arrives at mic 1 after a delay of 2 sampling cycles, and at mic 2 after 5 sampling cycles. At one point, the sample buffer for the two mics will contain:

```
Mic 1:  2 3 4 5 4 3 2 1 0 0
```

```
Mic 2:  0 0 1 2 3 4 5 4 3 2
```

Based on this dataset, the relative delay of signal received at mic 2 compared to mic 1 is 3 cycles. If we calculate the cross correlation score for $m \in [0, 5]$, we expect the score for $m = 3$ to have be the highest.

Here:
$C(n, 3) = (2)(2) + (3)(3) + (4)(4) + (5)(5) + (4)(4)$
.
Observe that with each streaming input sample, the difference in the new and old correlation score for the same lag *m* is
$C(n, m) - C(n-1, m) =$
$$= \sum_{k=0}^{W-1} A[n+k]B[n+k+m] - \sum_{k=0}^{W-1} A[n-1+k]B[n-1+k+m]$$
$$= A[n+W-1]B[n+W-1+m] - A[n-1]B[n-1+m]$$

Therefore, in each cycle, we only need two multiplications, one subtraction and one addition to compute the next value.

Implementation

To implement the cross correlation module, initially we wanted to write a parallel module that can calculate multiple correlation scores simultaneously for many different time steps. However, that requires parallel reads from audio buffers, and requires using limited registers.

In order to increase the accuracy, we want to use a large enough window size, but we still want minimal latency. So we wrote our cross correlation module in sequential fashion, where in each cycle it issues one read from each audio buffer and computes one multiplication. This allows us to implement audio buffers using Block RAMs which has higher capacity and low latency. One caveat we encountered is that using BRAM requires minimum of two cycle delays (one cycle to program read address, and another cycle for data return). To deal with this, we set up our BRAM at the system clock rate of 100 MHz, and only perform calculation one in every four cycles, effectively reducing its frequency from 100 MHz to 25 MHz. This ensures the memory delay is hidden, and requested data appears to be immediately available on the next calculation cycle.

The cross correlation module is a state machine, where the state is (op, m). op specifies the operation it's currently processing. The main two operations are OP_{SUB} and OP_{ADD}, the former subtracts from correlation score the product of the oldest samples, and the latter adds to the correlation score the product of the newest samples. Each operation is repeated for 200 distinct m. Therefore, the total number of cycles for updating scores given a new sample is roughly 400 cycles.

Each correlation module stores two streams of audio samples, and total of two correlation modules are needed. We need to store S samples in an audio stream, which is double the correlation window W (see theory section). We find the empirical limit of $W = 1024$, above which Vivado fails to compile.

The input samples have a width of 8 bits (signed), and the output scores are stored as 32 bit signed numbers.

Correlation Output Filter

File: cor_filter.v - author: Changping

Theoretically, the peak of the correlation score distribution corresponds to the true delay of a signal. However, we found that due to noise in the environment, the peak is not necessarily stable. This translates into a lot of jitter in triangulation result.

Hence, we coded a very simple filtering module with configurable parameters using switches. The filter specifies that the input delay must stay within a specified range for at least a specified amount of cycles. Any delay that fails to pass is rejected and does not reach triangulation stage. This introduces some visible level of stability.

In retrospect, a better filter would be an algorithm that tracks multiple peaks, and selects one that corresponds to previously selected peak even if there exists a peak with a higher score. This filter would be more tolerant of erroneous global peak.

Audio Output

File: pwm.v - author: Joren

The Nexys4 board has a mono audio output jack, and requires a Pulse-Width Modulation (PWM) module to play audio. The output of the PWM pin is passed through two low pass filters that smooth out the pulses of above-audible frequencies to a variable output voltage.

Since we were aiming for 8-bit audio output, we used a pulse window width of 256 clock cycles of the 100Mhz system clock, implying a PWM rate of around 390Khz, well above the audible range. The duty cycle - the amount of time along this window that the PWM output is held high - is then varied between 0 and 256. FIgure 5 gives an example of what a PWM waveform looks like.

Music Player

File: music_player.v sd_controller.v - author: Changping

The music player module plays a music from SD card. We used `sd_controller.v` file to read sectors from SD card, and connected it to a Block RAM-backed 1KB FIFO so that we can aggressively read from SD card in advance, and always maintain at least one sector (512 bytes) buffer in memory. On each sampling clock cycle, one byte is popped from FIFO and written to a audio PWM module for playback.

On a MacBook, we wrote a simple Python script to dump length and raw bytes from a 8-bit @ 44KHz *.wav* file onto an SD card as a raw block device.

Triangulation

File: triangulate.v - author: Joren

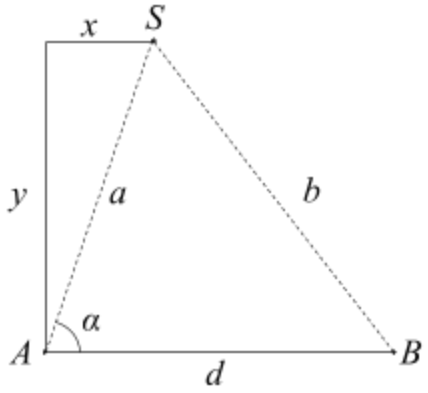


Figure 5: geometric representation of our situation

The cross correlation module computes how long a sound took to travel from the source S to the two microphones A and B. Since sound travels at a constant speed in an environment, the corresponding distances a, b can be computed using a simple scalar multiplication:

$a = c \cdot m_a$ and $b = c \cdot m_b$

Where m_a, m_b is the amount of samples the signal at A, B lags behind the source. Since we used a 50Khz sampling rate, and distance units of $1 \cdot 10^{-4} m$, we used a constant scaling factor $c = 68$, since sound travels 6.8mm each 1/50th of a second.

Given these two distances a and b, and the distance d between the two microphones A, B which is known, the triangulation module computes the cartesian coordinates x, y of the sound source. The computation relies on the following derivation of the law of cosines in the triangle $\triangle ASB$:

$$b^2 = a^2 + d^2 - 2 \cdot a \cdot d \cdot \cos(\alpha)$$
$$\Leftrightarrow x = a \cdot \cos(\alpha) = \frac{a^2 + d^2 - b^2}{2 \cdot d}$$

And from pythagoras' theorem in the right triangle $\triangle SAS'$:

$$y^2 + x^2 = a^2$$
$$\Leftrightarrow y = \sqrt{a^2 - x^2}$$

This means we can compute x,y on the FPGA without the need for trigonometric functions. It does however, require a square root operation which, unlike multiplication and addition, is not a built-in hardware operation and requires a dedicated module

Square root module

File: sqrt.v - author: Joren

Our square root implementation is based off a verilog implementation [5] of a binary search approach found online. It computes a guess for the root, by flipping up a single bit each iteration, from most significant to least significant, and computing the square g2 of that guess each time. If g2 is still less than n, the bit stays high and the guess is updated, otherwise, it stays low and the guess is not affected.

When determining whether the next bit, at position i, of the current guess g should be high, it is easy to compute the square of this potential better guess:

---

[5] https://github.com/tchoi8/verilog/blob/master/examples/sqrt.vl

$(g')^2$
$= (g + (1 << i))^2$
$= g^2 + (1 << i)^2 + 2 \cdot g \cdot (1 << i)$
$= g^2 + (1 << 2i) + (g << 1 << i)$
If $(g')^2 < n$ then $g \leftarrow g'$

Note that g^2 was computed in the previous iteration of the algorithm, or is 0 in the first iteration. All other operations involved are simple additions and bit shifts.

Since the square of an n-bit number has at most n/2 significant digits, the algorithm starts with $i = n/2$, decreases i by one each iteration, and consistently computes the root of an n-bit unsigned integer in n/2 cycles.

Note that, when computing the square root of a number that is not a perfect square, this algorithm will return the floor of the square root.

Theoretical accuracy

All computation modules store distance values as 16-bit integers. The squares of these quantities are stored in 32-bit registers. Given the nature of our set-up, x needs to be a signed coordinate, since it is negative for all points S left of the main stationary microphone. The y coordinate is always positive.

Since we're only doing integer arithmetic, we picked our unit of distance to be $1 \cdot 10^{-4} m$, and in theory, the calculations could triangulate coordinates with this accuracy. It is worth reconsidering this choice of units to allow for larger maximum coordinates, since there is a margin of error introduced by the cross-correlation module that is much higher, as sound travels $68 \cdot 10^{-4} m$ per sample.

Display

File: vga.v, - author: Joren

In order to understand the results of our real-time cross-correlation and triangulation calculations, we included a representation of the setup, with its microphones and the location of the sound source. We also added some real-time insight into the results of the calculations beyond just the final x,y coordinate.

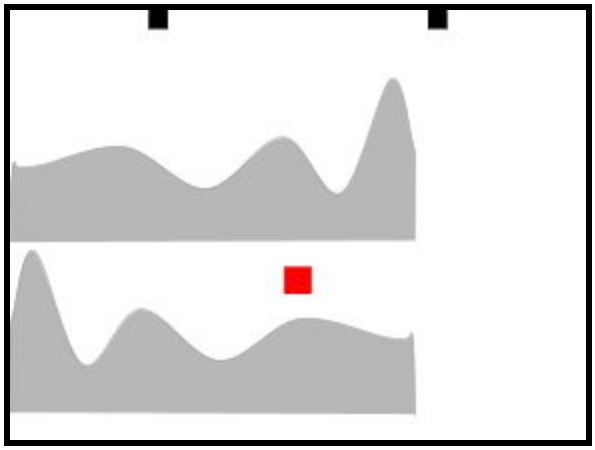


Figure 6: Example of screen layout with two correlation waveforms, indicators for two stationary microphones (black) and one speaker (red)

VGA Display Signals

Our vga module is based on the one provided with lab4 on the course website[6], but with some minor modifications. The vga module generates the proper hsync, vsync and red, blue, green video signals. While we built a working xvga module (1024x768

[6] web.mit.edu/6.111/www/f2016/handouts/labs/labkit_lab4.v

resolution), we intentionally decided to stick with the default 640x480 resolution with a 60Hz refresh rate, since that allows for a 25Mhz pixel clock[7], easily obtained by running the 100Mhz system clock through a quarter divider. Larger resolutions would require a separate video clock domain. This allows components that work in the visual system, like waveform renderers and sprites, to operate on the same clock domain as the triangulation and audio sampling modules with which they have to interact.

Screen layout

We implemented a simple sprite pipeline, where a pixel signal encoding the color corresponding to a current x,y position on the screen is passed on between sprites. Where the video chip on the 6.111 labkit supports 24bit color resolution with 8 bit for each channel, the built in ADC's tied to the Nexys4 VGA port are limited and only support 4 bit for each channel, giving us a 12 bit color resolution.

On the display, we show:

- The position of the two stationary microphones for reference
- The moving triangulated position of the sound source
- Waveforms indicating the current correlation score distribution

Correlation distribution waveforms

File: sprite_waveform.v - author: Joren

As described in the section about the cross correlation module, it computes a cross-correlation score for all possible lag values *m*. We initially just chose the lag value corresponding to the highest correlation score as the correct answer, but this turned out to be unreliable. This is why we started experimenting with filters on our cross-correlation data.

To better visualize what is really happening, we added a waveform sprite. At the start of a VGA frame, the waveform sprites stores one run of score values from the correlation module in registers, to render on the screen.

From this, we learned that these score vs. delay distributions show several peaks. Higher pitched, periodic sounds (whistles, tones, …) produce more peaks. This makes sense, as those lag values that are shifted by a multiple of the dominating period also get high correlation scores. Had we realized this earlier, we could have made more specific filters for a steadier result.

Analysis of how this distribution changed over time was much easier to do when directly visible on the display, compared to analyzing in (non-interactively) using the Internal Logic Analyzer (ILA), mostly it was very hard to determine the evolution of the distribution with time.

---

[7] http://www-mtl.mit.edu/Courses/6.111/labkit/vga.shtml

Lessons Learned

Sampling from several channels

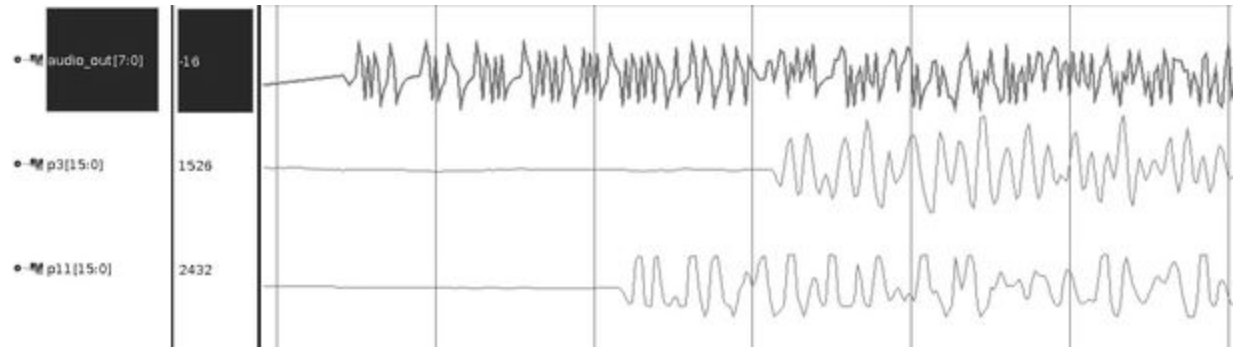
We initially expected much more trouble sampling from three microphones simultaneously. This turned out to be very simple due to the wizards built into the Vivado design suite, and the multiplexer built into the XADC.

Third microphone vs. sound output from FPGA

The localization is currently much more reliable when using a third microphone and your voice as a sound source, than when using the sound output from the FPGA. This is surprising, since the latter gives very accurate information of exactly what sound is being produced, and at what time.

One possible explanation would be that the process of converting a waveform through PWM, reproducing it through a speaker, and then picking it up using a microphone acts as a significant filter on the signal. Figure 7 compares the output wave and the recorded signals on two microphones. The sound output shown is the start of a “string pluck” sound generated by a Karplus-Strong String synthesizer, which resembles white noise and as such has a significant random high-frequency component. While the different delays between the signals are clearly visible, and the two microphone waveforms look very similar, they are significantly different from the raw output wave.

Understanding which the filtering effects at work could make location of a non-microphone sound source work just as well, but we did not have the time to look into this.



Challenges due to periodicity of sound

From the correlation score distributions, we learned that the score develops peaks for different lag values a fixed distance apart, that depends on dominant frequencies of the sound being picked up. When a sound is very periodic, the correlation module gets confused since the waveforms “align” at several places, each shifted by one period.

One solution could be to try and filter out the period component of the sound, and focus on correlating the irregularities. Another interesting solution could be to return to our initial idea of using three stationary microphones, since the difference in arrival time between two microphones that are near each other could often be less than one period of the sound. This requires changes to our correlation module (faster sampling, shorter window, …) that we did not get to experiment with.

Division of work

Joren: preamplifiers, triangulation, and VGA display modules
Changping: cross-correlation, music player, and ADC sampling modules.