DT2118 Project: Beamforming

Alejandro Marzinotto (almc@kth.se) 900102-6138

I. Introduction

Beamforming is a signal processing technique that relies on the superposition of waves caused by delayed emission/reception, it can be used in two different ways.

1) The first scenario (*forward beamforming*) involves focusing the energy that several spherical emitters produce by controlling the phase of the wave. The superposition principle will cause certain parts of space to show higher power because the waves of different emitters have constructive interference in those points, whereas it will cause other parts of space to show lower power because the waves of different emitters have destructive interference in those points.
2) The second scenario (*backward beamforming*) involves figuring out the direction from where the wave is being emitted by using the phase shift of such wave at several different receivers. By knowing the velocity of propagation of the wave and using an arrangement of properly spaced receivers, it is possible to retrieve the direction from where the wave was emitted.

In this report we present simulations of the first and second scenarios. In other words we simulate how a wifi access point would de-phase the signals sent to their antennas in order to focus the energy beam in certain directions, and we simulate an array of evenly spaced microphones to obtain the approximate position of an object that emits sound waves while moving.

We present the equations that model such simulations and the reasoning behind the beamforming techniques implemented.

There exist more complex and robust backward beamforming techniques for commercial devices such as the Microsoft Kinect. This device has an array of 4 microphones which in conjunction with their proprietary software is capable of determining with high accuracy the direction from which a sound is coming (in a 2D plane). This sound directional sensibility can be used to further separate the speech of one person from another in a robust manner if they are standing side by side in front of the Kinect.

In this report we present a hardware modification to the Kinect basic infrastructure that would allow 3D localization using 2 arrays of sensors placed in planes perpendicular to each other. We also show how a third perpendicular array of sensors could improve the detection accuracy even further.

II. Forward Beamforming

In the forward beamforming algorithm, we have a certain number of emitters (labelled s) each of which propagates a spherical wave that has the following mathematical equation.

$$Z_s = A\frac{e^{ik\sqrt{(X-s.x)^2+(Y-s.y)^2}}e^{-i(wt+s.\phi)}}{\sqrt{(X-s.x)^2+(Y-s.y)^2}} \quad (1)$$

We define the parameters v, f, λ, A, k, and ω that stand respectively for the linear velocity of propagation of the wave, the frequency of the wave in Hz, the wavelength in meters, the amplitude of the wave in meters, the wave constant (inversely proportional to the wavelength), and the angular speed of the wave.

```
v = 344;
f = 10;
lambda = v/f;
A = 1.0;
k = 2*pi/lambda;
w = k*v;
```

We also define the position of the emitters that are going to be producing the spherical waves, these are labelled s_1, s_2, s_3, s_4 for the simulation presented below and they are located at a distance 0.5λ between each other.

```
s1.pos = [0,-0.75*lambda,0];
s2.pos = [0,-0.25*lambda,0];
s3.pos = [0,+0.25*lambda,0];
s4.pos = [0,+0.75*lambda,0];
```

It is also possible to set the distances between the emitters in such a way that they are not evenly spaced. This produces slightly different interference patters as we will show below.

```
s1.pos = [0,-1.0*lambda,0];
s2.pos = [0,-0.5*lambda,0];
s3.pos = [0,+0.5*lambda,0];
s4.pos = [0,+1.0*lambda,0];
```

We dephase the signals 20% of 2π for each successive emitter. This parameter was tuned manually and different results can be achieved for different amounts of de-phase. Nonetheless, to observe a greater variety of interference patters, we introduce additional de-phase between sensors as the simulation progresses.

```
s1.phs = 0.0*2*pi;
s2.phs = 0.2*2*pi;
s3.phs = 0.4*2*pi;
s4.phs = 0.6*2*pi;
```

The actual calculation and update of the waves and their interference pattern takes place inside a for loop over the grid of the x/y plane. We sum and normalize the waves of the last 2 iterations to get a smoother looking wave of the correct amplitude.

```
[X, Y] = meshgrid(linspace(-Lx/2, Lx/2, M),
                  linspace(-Ly/2, Ly/2, M));
T = 0.5;
nt = 100;
time = linspace(0, T, nt);
phase = linspace(0, 2*pi, nt);
for iter = 1:(nt-1)
```

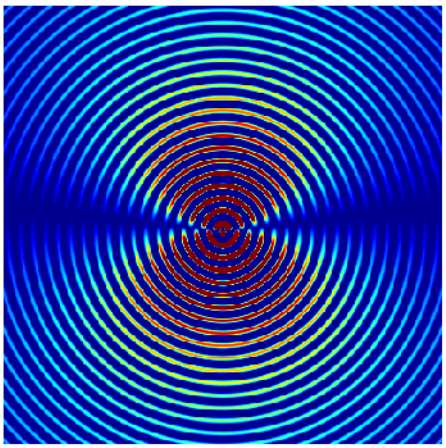

Figure 1. Forward beamforming with 2 emitters evenly spaced.

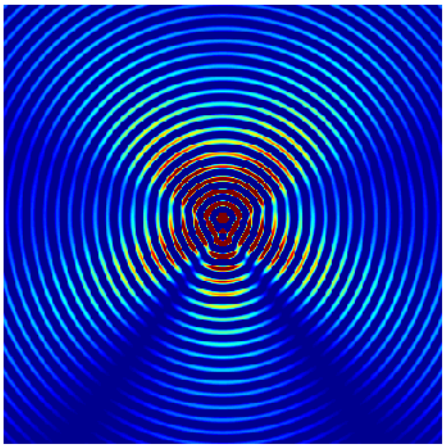

Figure 2. Forward beamforming with 3 emitters unevenly spaced.

```
    t = time(iter);
    Z1 = A*exp(1i*k*sqrt((X-s1.pos(1)).^2
                         +(Y-s1.pos(2)).^2))* ...
           exp(-1i*(w*t + s1.phs + 0*phase(iter)))
           ./sqrt((X-s1.pos(1)).^2+(Y-s1.pos(2)).^2);

    Z2 = A*exp(1i*k*sqrt((X-s2.pos(1)).^2
                         +(Y-s2.pos(2)).^2))* ...
           exp(-1i*(w*t + s2.phs + 0.25*phase(iter)))
           ./sqrt((X-s2.pos(1)).^2+(Y-s2.pos(2)).^2);

    Z3 = A*exp(1i*k*sqrt((X-s3.pos(1)).^2
                         +(Y-s3.pos(2)).^2))* ...
           exp(-1i*(w*t + s3.phs + 0.5*phase(iter)))
           ./sqrt((X-s3.pos(1)).^2+(Y-s3.pos(2)).^2);

    Z4 = A*exp(1i*k*sqrt((X-s4.pos(1)).^2
                         +(Y-s4.pos(2)).^2))* ...
           exp(-1i*(w*t + s4.phs + 1*phase(iter)))
           ./sqrt((X-s4.pos(1)).^2+(Y-s4.pos(2)).^2);


    average_Z = average_Z + real(Z1) + real(Z2)
                          + real(Z3) + real(Z4);
    average_Z = average_Z / norm(average_Z);
end
```

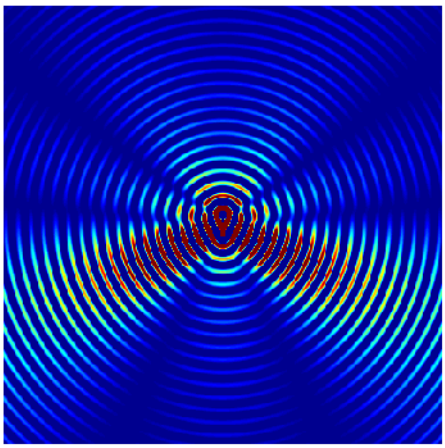

Figure 3. Forward beamforming with 4 emitters evenly spaced.

Briefly presenting some of the results obtained in simulation for the interference patters formed using different arrays with different parameters we have the following figures: Fig. 1, Fig. 2, Fig. 3, Fig. 4, Fig. 5.

As one may see from the experimental results, it is possible to create a wide variety of shapes: stars with 4 corners, stars with 5 corners, etc. It is also possible to focus the energy in 2 different locations (in front and behind, or side by side). We point out that all these experiments were carried out using a linear array of 4 emitters and more complex shapes can be created by using a bi-dimensional array of sensors, for example in the figures: Fig. 6 and Fig. 7.

As one can see the directionality of the beams is increased by adding one more dimension to the grid of sensors and this idea could carry over into 3 dimensional arrays of sensors.

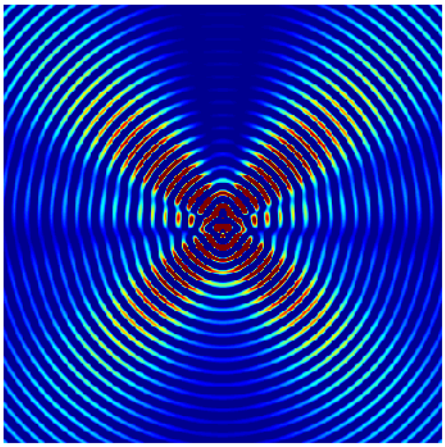

Figure 4. Forward beamforming with 4 emitters unevenly spaced.

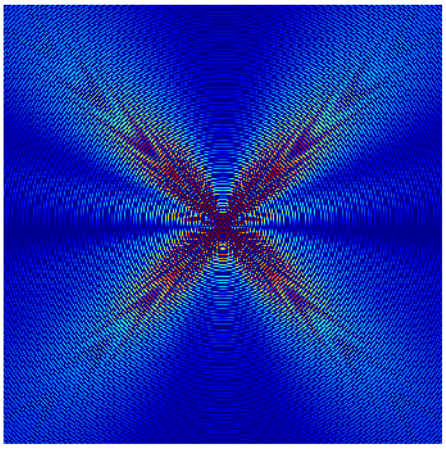

Figure 5. Forward beamforming with 4 emitters unevenly spaced with 4 times more frequency (shorter wavelength).

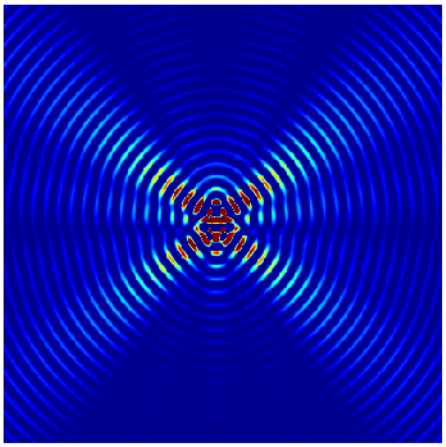

Figure 6. Forward beamforming with 4x2 emitters unevenly spaced with 2 columns spaced 1λ.

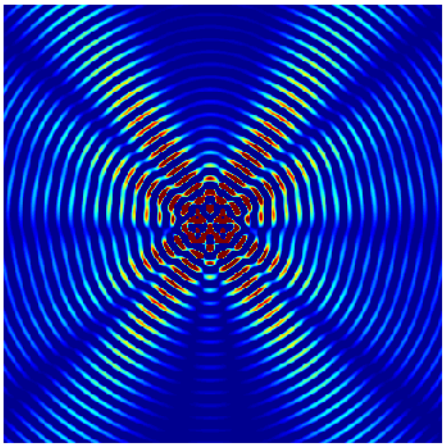

Figure 7. Forward beamforming with 4x2 emitters unevenly spaced with 2 columns spaced 2λ.

III. Backward Beamforming

In the backward beamforming algorithm, we have a certain number of sensors each of which receives a spherical wave that has the same mathematical equation as in (1) and is emitted by an object that has unknown position and velocity.

Using the delay in reception of the wave at different sensors we can triangulate the source of the sound / wave in a an approximate fashion.

For greater simulation simplicity, we don't model the intensity of the wave at each point in space for the calculation of the backward beamforming. Instead we model only the peak of the wave as a circle of increasing radius.

This means that we throw away information about the intensity of the wave which is not reliable to be used in the calculation of the backward beamforming problem.

The reason for this is because the wave attenuation at a given sensor can be due to many factors: distance between emitter/sensor, environmental disturbances, reflections, etc. For this reason we don't use the intensity of the beam but rather the time difference of arrival (delay) between the reception of the wave at different sensors.

The backward beamforming algorithm described below is custom made to solve the problem in a computationally efficient way rather than providing the best estimate of the angle to the emitter / position of the emitter.

Broadly speaking the algorithm works as follows; we assume that we know the wave propagation speed in the medium that is being considered and that such speed is approximately constant. We take pairs of sensors and we study the delay in the wave reception between them.

Knowing how fast the wave travels we can establish the following border conditions:

1) if the emitter is aligned with the axis defined by the 2 sensors the delay in the reception of the signal will be maximal and will correspond to $\text{max\_delay}_{1-2} = \text{norm}(s_1.pos - s_2.pos, 2)/v$. Where v is the linear propagation velocity of the wave and the norm between the positions of the sensors is simply the euclidean distance. The delay is calculated as the amount of time it would take the wave to travel from one sensor to the other if it was produced by an emitter aligned to the sensors (worst case scenario, or maximum delay).
2) if the emitter is equidistant to both sensors, i.e. if it lies in the line that is perpendicular to the axis of the 2 sensors and crosses the axis exactly at the point between the sensors. Then, the expected delay in the reception of the wave is zero seconds. One can use these two border conditions to establish a linear or non-linear interpolation scheme which approximates the direction from the incoming wave for all other delays that are between 0 seconds and max_delay_{1-2} seconds.

The sensor that first receives the wave will determine if the axis is positive or negative yielding a coverage radius that goes between 0 to $\pi/2$ and between 0 to $-\pi/2$ (right hand side in Cartesian coordinate space). The angles on the second and third quadrants (negative cosine) are impossible to distinguish between the symmetric angles found in the first and fourth

quadrants. This means that an emitter located at 0 degrees will produce the same delays as one located at 180 degrees.

To solve this problem robustly one would need another array of sensors laying on an axis that is perpendicular to the axis defined by the first set of sensors. Given our case study where the first set of sensors is located along the Y axis, this second axis corresponds to the X axis and it would be able to distinguish between symmetric beams incoming from left and right, i.e. angles α, β such that $\sin(\alpha) = \sin(\beta)$, e.g. 0 degrees and 180 degrees.

In these experiments we use the same definitions as before for the wave parameters, however, now the elements s_1, s_2, s_3, s_4 are no longer wave emitters but sensors and we only deal with the case where they are uniformly distributed.

```
s1.pos = [0,-0.75*lambda,0];
s2.pos = [0,-0.25*lambda,0];
s3.pos = [0,+0.25*lambda,0];
s4.pos = [0,+0.75*lambda,0];
```

Arrangements where the distance between sensors is large enough allow us to create multiple directional vectors for the incoming beam that when intersected give an approximate x, y position of the emitter.

The velocity of the wave is of critical importance for the calculation of the maximum delay expected and therefore necessary for our interpolation.

```
v = 344;
f = 10;
lambda = v/f;
A = 1.0;
k = 2*pi/lambda;
w = k*v;
```

The phases of each sensor are now calculated inside the main simulation loop unlike before where they were set beforehand to be 20% higher on each new sensor.

In the following code we update the position of the emitter using a circular trajectory, we then calculate how long it would take for the wave to reach each of the emitters based on the distance and the assumption that the wave speed is constant. We then calculate the delays of each sensor with respect to the first sensor (s_1).

This yields some positive and some negative delays depending on which was the actual sensor that first received the wave. To account for this fact we obtain the minimum of all delays and we subtract it from the set of 4 delays in order to normalize them (make them ≥ 0). Lastly, for each pair of sensors we calculate the proportion of the observed delay with respect to the maximum delay expected for the wave speed v, i.e. $\text{max\_delay}_{1-2} = \text{norm}(s_1.pos - s_2.pos, 2)/v$.

```
[X, Y]=meshgrid(linspace(-Lx/2, Lx/2, M),
                linspace(-Ly/2, Ly/2, M));
T = 0.5;
nt = 100;
time = linspace(0, T, nt);

for iter = 1:(nt-1)
    t = time(iter);
    em.pos = [500*cos(pi/4*t+pi/2),
              300*sin(pi/4*t+pi/2), 0];
    scatter3(s1.pos(1), s1.pos(2), s1.pos(3),'blue');
    scatter3(s2.pos(1), s2.pos(2), s2.pos(3),'blue');
    scatter3(s3.pos(1), s3.pos(2), s3.pos(3),'blue');
    scatter3(s4.pos(1), s4.pos(2), s4.pos(3),'blue');
    scatter3(em.pos(1), em.pos(2), em.pos(3),'red');

    s1.d = norm(s1.pos - em.pos, 2);
    s2.d = norm(s2.pos - em.pos, 2);
    s3.d = norm(s3.pos - em.pos, 2);
    s4.d = norm(s4.pos - em.pos, 2);

    power = 0.1;
    s1.t = (s1.d / v) + 1.0*wgn(1,1,power);
    s2.t = (s2.d / v) + 0.0*wgn(1,1,power);
    s3.t = (s3.d / v) + 0.0*wgn(1,1,power);
    s4.t = (s4.d / v) + 0.0*wgn(1,1,power);

    s1.delay = 0.0;
    s2.delay = s2.t - s1.t;
    s3.delay = s3.t - s1.t;
    s4.delay = s4.t - s1.t;

    first_delay = min([s1.delay, s2.delay,...
                       s3.delay, s4.delay]);

    s1.delay = s1.delay - first_delay;
    s2.delay = s2.delay - first_delay;
    s3.delay = s3.delay - first_delay;
    s4.delay = s4.delay - first_delay;

    length = 200;
    % sensor 1/2 pair
    max_delay_1_2 = norm(s1.pos - s2.pos, 2) / v;
    rel_delay_1_2 = max([s1.delay, s2.delay])
                  - min([s1.delay, s2.delay]);
    proportion_1_2 = pi/2* rel_delay_1_2 / ...
                     max_delay_1_2;
    if (s2.delay > s1.delay)
        proportion_1_2 = -1*proportion_1_2;
    end
    vector_1_2_o = (s1.pos + s2.pos) / 2;
    vector_1_2_v = vector_1_2_o + length * ...
        [cos(proportion_1_2),sin(proportion_1_2), 0];

    arrowStarts = [vector_1_2_o;vector_2_3_o;
                                vector_3_4_o];
    arrowEnds = [vector_1_2_v;vector_2_3_v;
                              vector_3_4_v];
    arrow(arrowStarts,arrowEnds);
end
```

Using the combined result for several pairs of sensors and given that they are appropriately distanced from each other. It's possible to project the directionality vectors obtained through beamforming and get an approximate distance to the emitter. This method of finding the x, y position of the emitter with respect to the array of sensors becomes unreliable if the sensors are placed too close together because they would yield almost parallel directionality vectors.

As one may see from the experimental results the directionality of the beam works accurately for small angles and gets worse as we get closer to 45 degrees, It regains its high accuracy for angles close to 90 degrees as expected. This could be because we are using a linear interpolation to solve our problem when in reality the function that governs the reception delay of the wave's physical propagation is non-linear.

We also notice that the estimation of the x, y position of the emitter (green star) is always closer than in reality (red circle). We have tried for different distance between the sensors (blue circles) but the constant overestimation of the closeness of the

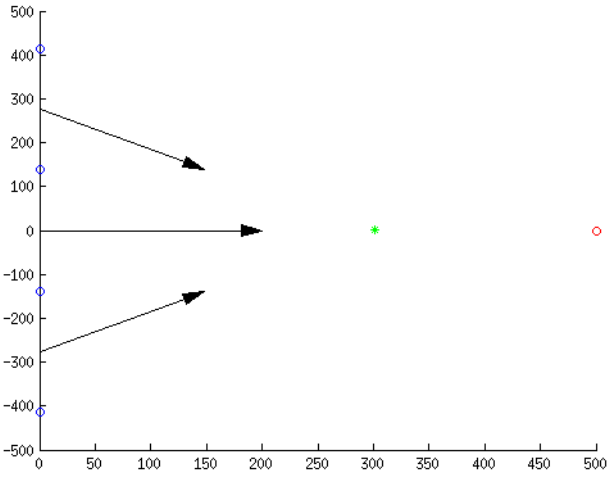

Figure 8. Backward beamforming with 4 sensors placed at large distance between each other (8λ) with the emitter at 0 degrees (good accuracy).

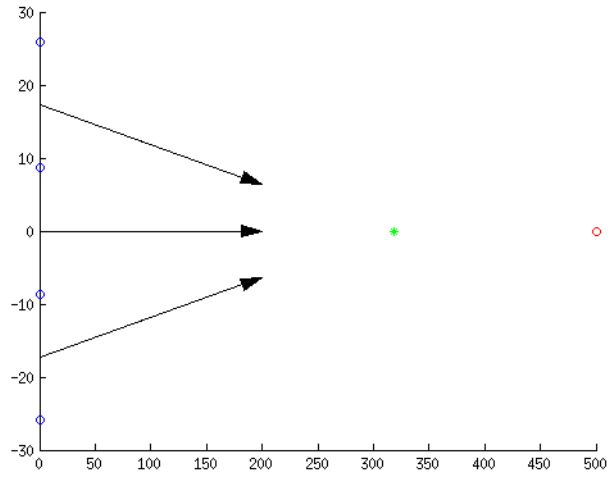

Figure 9. Backward beamforming with 4 sensors placed at short distance between each other (0.5λ) with the emitter at 0 degrees (good accuracy).

emitter is still there. For the results refer to the figures: Fig. 8, Fig. 9, Fig. 10, Fig. 11.

When it comes to how the error in the absolute distance between the estimated point and the real point evolve for different angles of the emitter with respect to the array of sensors we have the plot in Fig. 12.

When it comes to how the error in the angle between the estimated direction and the real direction evolve for different angles of the emitter with respect to the array of sensors we have the plot in Fig. 13.

IV. Kinect Evaluation

In this last section we review the beamforming algorithm that comes with the Kinect. We try to dig into the source code of the platform to understand how it works, unfortunately the results were not very promising as they have a proprietary pre-compiled function that retrieves the angle without showing the developer how.

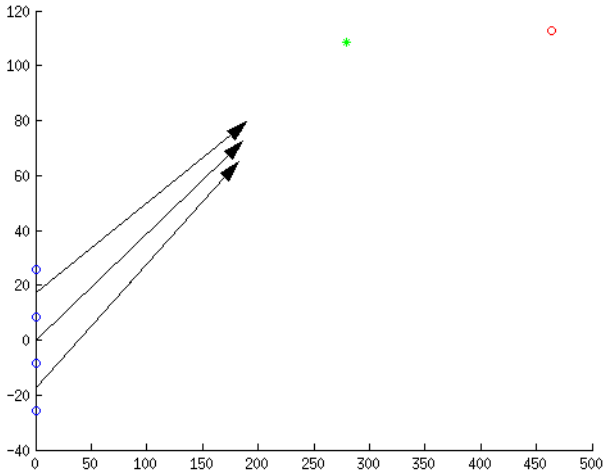

Figure 10. Backward beamforming with 4 sensors placed at short distance between each other (0.5λ) with the emitter at 45 degrees (bad accuracy).

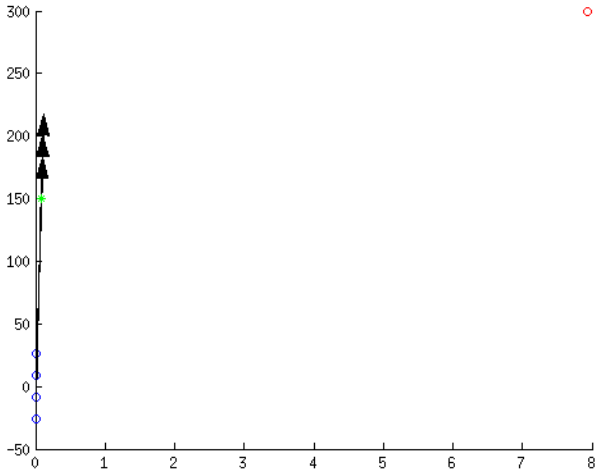

Figure 11. Backward beamforming with 4 sensors placed at short distance between each other (0.5λ) with the emitter at 90 degrees (good accuracy). Notice that even though the error appears big it's only 8 meters in x direction from a starting position of $x = 500$ meters

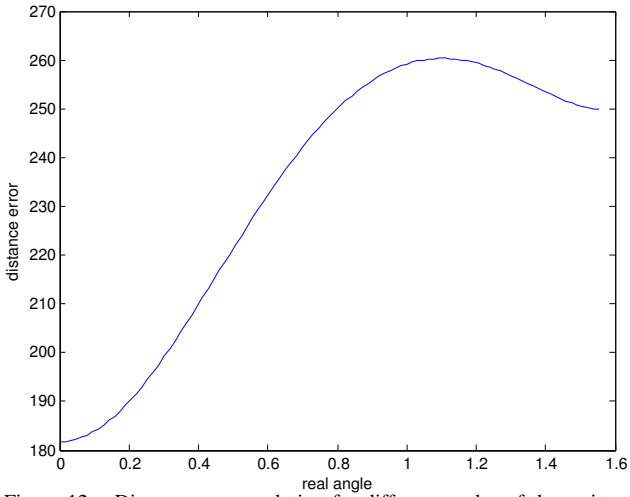

Figure 12. Distance error evolution for different angles of the emitter.

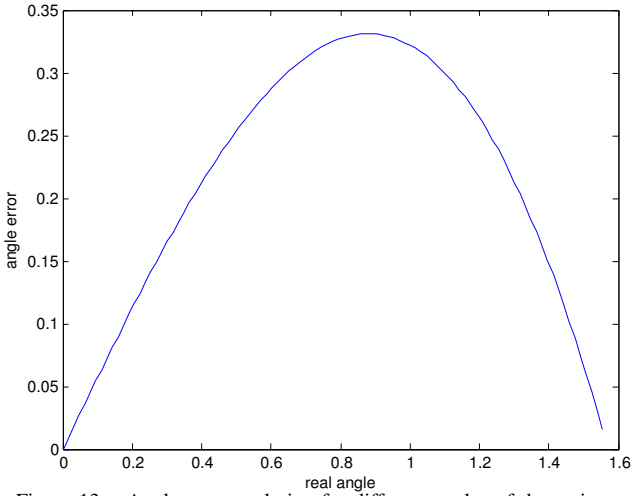

Figure 13. Angle error evolution for different angles of the emitter.

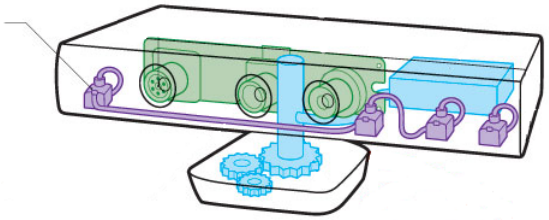

Figure 14. Kinect diagram showing in purple the microphone array (4 in total).

Some of the data relevant about the Kinect is that it has a microphone array that captures audio at 24-bit resolution. The microphones are placed unevenly across the device as shown by the purple devices in Fig. 14.

The kinect microphone array of sensors in conjunction with the Microsoft developer kit for Kinect allows the following tasks:

1) High-quality audio capture.
2) Focus on audio coming from a particular direction with beamforming.
3) Identification of the direction of audio sources.
4) Improved speech recognition as a result of audio capture and beamforming.
5) Raw voice data access.

By studying the source code provided in the examples we have reached the following scripts: code1.cpp, code2.cpp, code3.cpp and code4.cpp. Located inside the "code" directory of the report folder hierarchy. These code snippets show sections that are relevant for the beamforming and audio extraction done by the Kinect.

Importantly one may notice the following functions from code1.cpp:

```
m_pNuiAudioSource->GetBeam(&beamAngle);
m_pNuiAudioSource->GetPosition(&sourceAngle,&sourceConfidence);
// Convert angles to degrees and set values in audio panel
m_pAudioPanel->SetBeam(static_cast<float>
              ((180.0 * beamAngle) / M_PI));
m_pAudioPanel->SetSoundSource(static_cast<float>
               ((180.0 * sourceAngle) /
                M_PI), static_cast<float>(sourceConfidence));
```

that take care of performing the beam angle extraction along with the confidence. The algorithm of the Kinect does use the strength of the signal received unlike our implementation of beamforming in simulation.

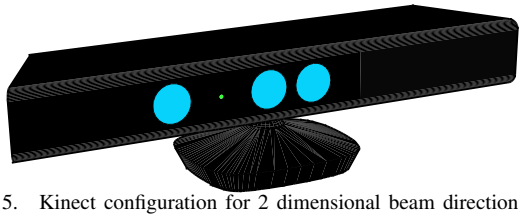

Figure 15. Kinect configuration for 2 dimensional beam direction recognition.

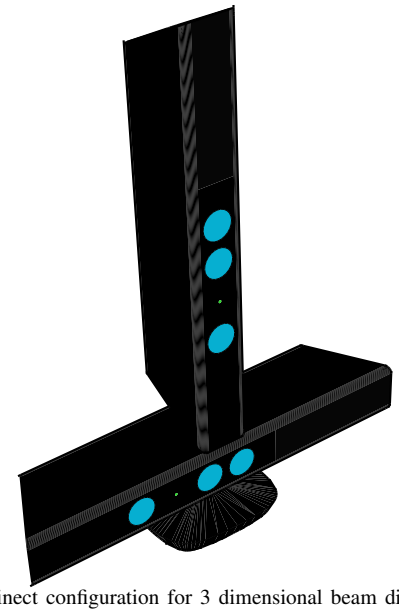

Figure 16. Kinect configuration for 3 dimensional beam direction recognition.

The extraction of data from the Kinect is done through an audio re-sample as shown in code3.cpp.

```
DWORD bytesAvailable = framesAvailable * _MixFrameSize;
// Process input to resampler
hr = ProcessResamplerInput(pData, bytesAvailable, flags);
if (SUCCEEDED(hr))
{
  DWORD bytesWritten;
  // Process output from resampler
  hr = ProcessResamplerOutput(&bytesWritten);
  if (SUCCEEDED(hr))
  {
      //  Audio capture was successful,
      // so bump the capture buffer pointer.
      _BytesCaptured += bytesWritten;
  }
}
```

The standard configuration of the Kinect that we used to run the sample code is as shown in Fig. 15.

In order to use Kinects to get the beam direction in 3 dimensions it's needed to put them perpendicular to one another as shown in Fig. 16.

This configuration will yield, using the demonstration software, 2 angles: one for the azimuth and another for the elevation (also called altitude). Using simple math these angles can be translated into other representations such as the Cartesian one. For the meaning of these angles refer to Fig. 17.

Lastly, one may be interested also in the distance to the sensor for which one can use the configuration of Fig. 18.

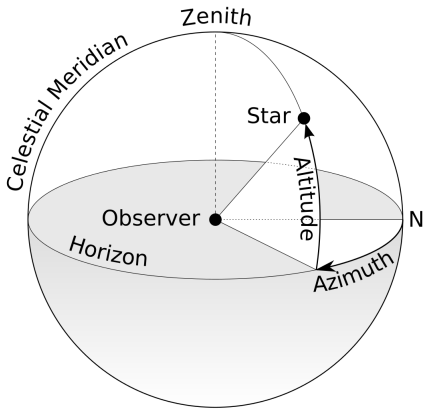

Figure 17. Azimuth and Elevation angle representation.

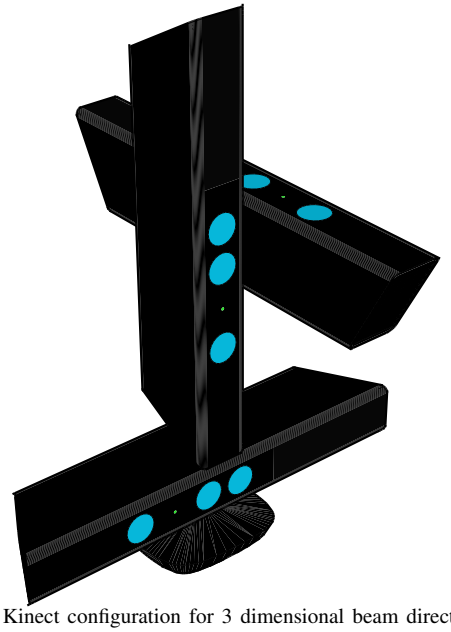

Figure 18. Kinect configuration for 3 dimensional beam direction recognition (distance to sensor and symmetric angle recognition).

This last configuration would enable not only an approximate distance to the sensor (or at least more reliable than with the method explained earlier), but also would enable the recognition in 3D to work also for angles which are behind the working area of the first 2 Kinects, i.e. it would be able to distinguish between symmetric angles with respect to the plane perpendicular to the third Kinect.

The visualization of each Kinect's extracted angles (azimuth and altitude) for the case of Fig. 16 are presented below in Fig. 19 and Fig. 20.

V. Conclusions

As concluding remarks we have shown examples of forward and backward algorithms for beamforming that enable us to: in first instance, direct our energy towards the desired direction using de-phased emitters; in second instance, retrieve the location of a certain 'speaker' and possibly to isolate it's speech from the background noise.

Beamforming techniques such as the ones presented here are not the state of the art, however they are simple enough

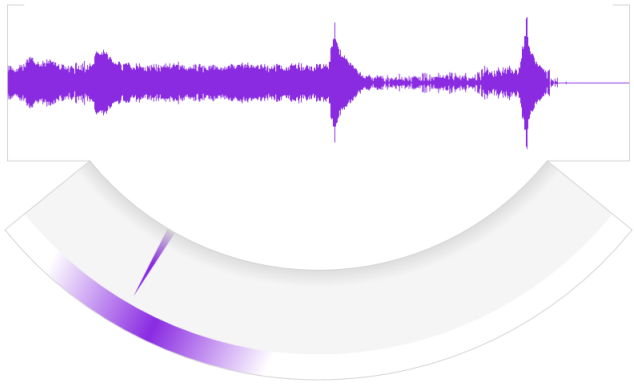

Figure 19. azimuth angle for the horizontal Kinect.

Figure 20. elevation angle for the vertical Kinect.

to understand the concept and represent a valuable testing framework for different applications. A successful integration of beamforming techniques using audio with visual RGBD data from the Kinect could yield interesting improvements when it comes to speaker localization and background noise reduction.