# Audio Eye

This document is a description of a concept that may allow blind people to use their ears to “see”. This would be achieved using some hardware and software. A first demo of the software has been developed on 22 December 2019 and is available at <https://github.com/mth128/AudioEye>.

The idea is to convert images and video into sound. This is not an entirely new concept, as it has already been launched using the vOICe “Seeing with sound” app, proving this idea has potential. However, the philosophy and approach of Audio Eye is slightly different. The goal of the vOICe is to generate 2D images from left to right, with a sound sample of 1 second for each image. The idea of Audio Eye is to make it possible to track light sources and/or shapes in real time speed. If it works, the blind person may be able to catch a ball.

The way this is achieved, is by using a spiral shape (named the “Eyeweb”), in which the center of the spiral has the highest resolution of the image. The center of the spiral is converted to the highest frequency audio. The further away from the center, the lower the audio frequency. Every 360 revolution in the spiral the frequency is divided by a half. Hence the sound tone is fixed for each direction. The tone on the right is set to a C of a musical instrument. Every 30 degrees clockwise, the tone is one higher. Hence having all 12 tones in a single 360 revolution.

The idea of using this spiral, is in order to mimic the function of the eye. The center of the eyes retina has the highest pixel density, while the outer edges of the retina are mostly to provide context. If anything prominent happens in these edges, the person would move their eyes toward that prominent event. The same would be required for this spiral shaped EyeWeb: in order to get more detail, the camera must be aimed at the point of interest.

## Software

The Audio Eye software is available at <https://github.com/mth128/AudioEye>. It is available under the GNU 3 license. As for December 2019 the software is a C# demo module, which asks for an image. Currently it is not yet possible to load video’s or connect a camera. The software currently consists of:

* a toolstrip (currently only containing a load button for opening an image),
* a set of variables and settings,
* a picturebox, displaying the loaded image, together with an eyeweb (see next paragraph), located at the cursor position,
* three smaller images, showing the resulting grayscale images captured by the eyeweb, in the order: left ear, mono sound, right ear,
* sound that corresponds to the captured eyeweb, as explained in this document.

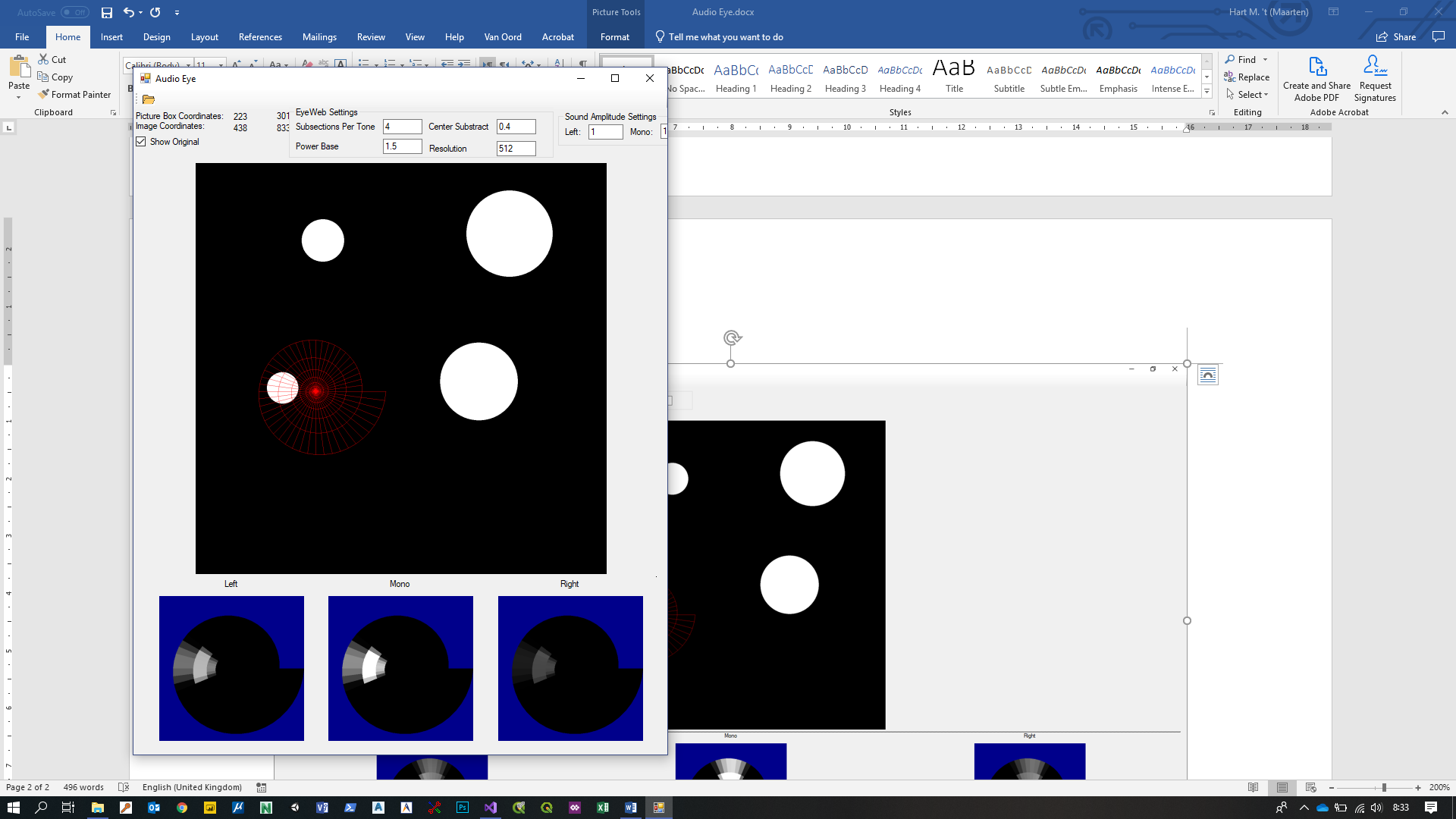
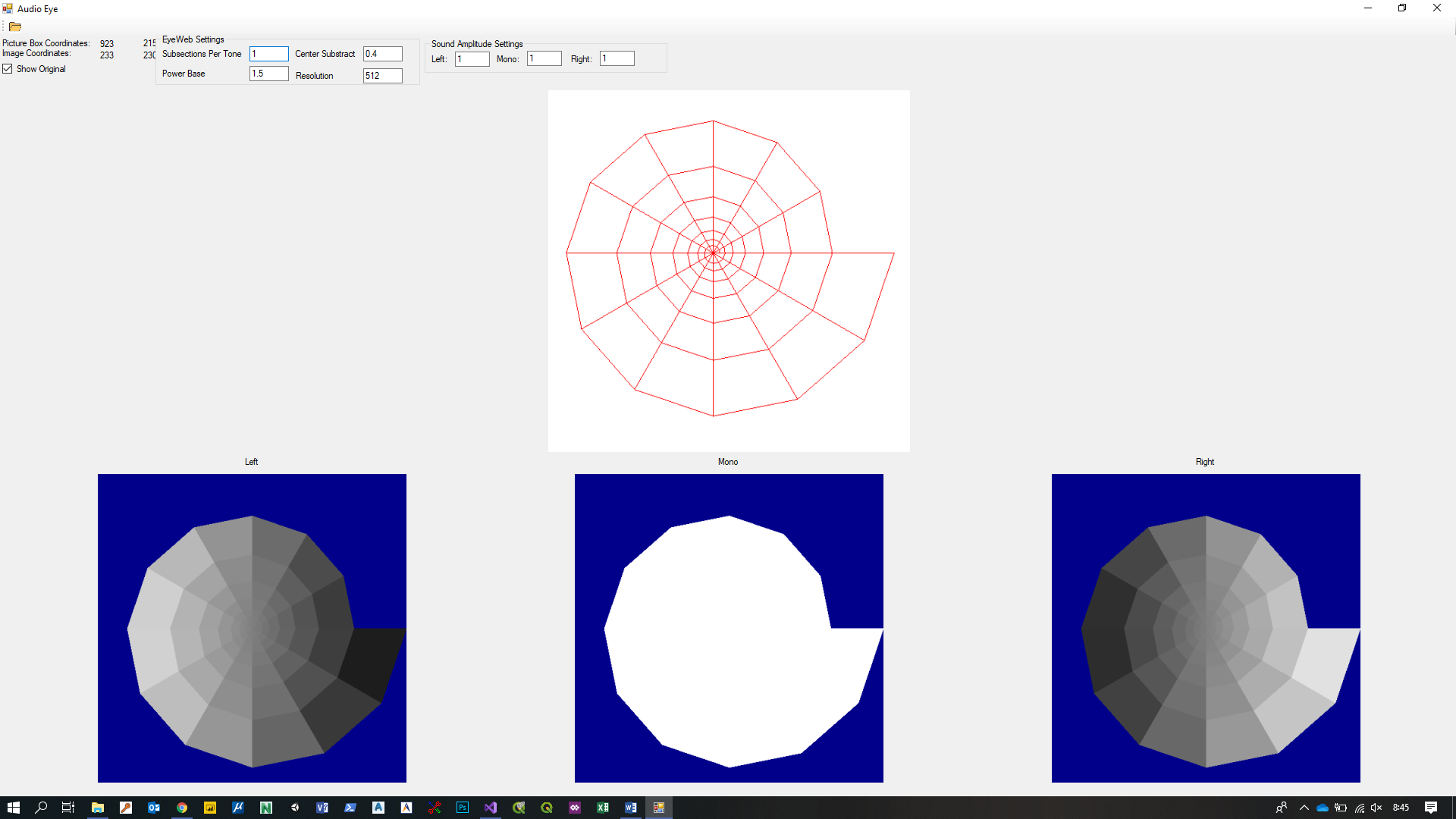


Image 1: a screenshot of the AudioEye software.

### EyeWeb

The core of the software is the “EyeWeb”. The EyeWeb is a 2D spiral shape, meshed into quads. The EyeWeb is positioned over an image, and for each quad a grayscale intensity value is extracted from the image. Each quad has its own sound frequency assigned to it. The sound frequency is basically a musical note (A to G#), which defines it’s direction. The octave defines the amount of inward revolutions.



C (523.2  
 Hz)

C (261.6 Hz)

C (130.8Hz)

B (123.5Hz)

A# (116.5Hz)

A (110.0Hz)

G# (103.8Hz)

G (98.0Hz)

F# (92.5Hz)

F (87.3Hz)

E (82.4Hz)

D# (77.8Hz)

D (73.4Hz)

C# (69.3Hz)

C (65.4Hz)

Image 2: The frequencies for each full-tone quad.

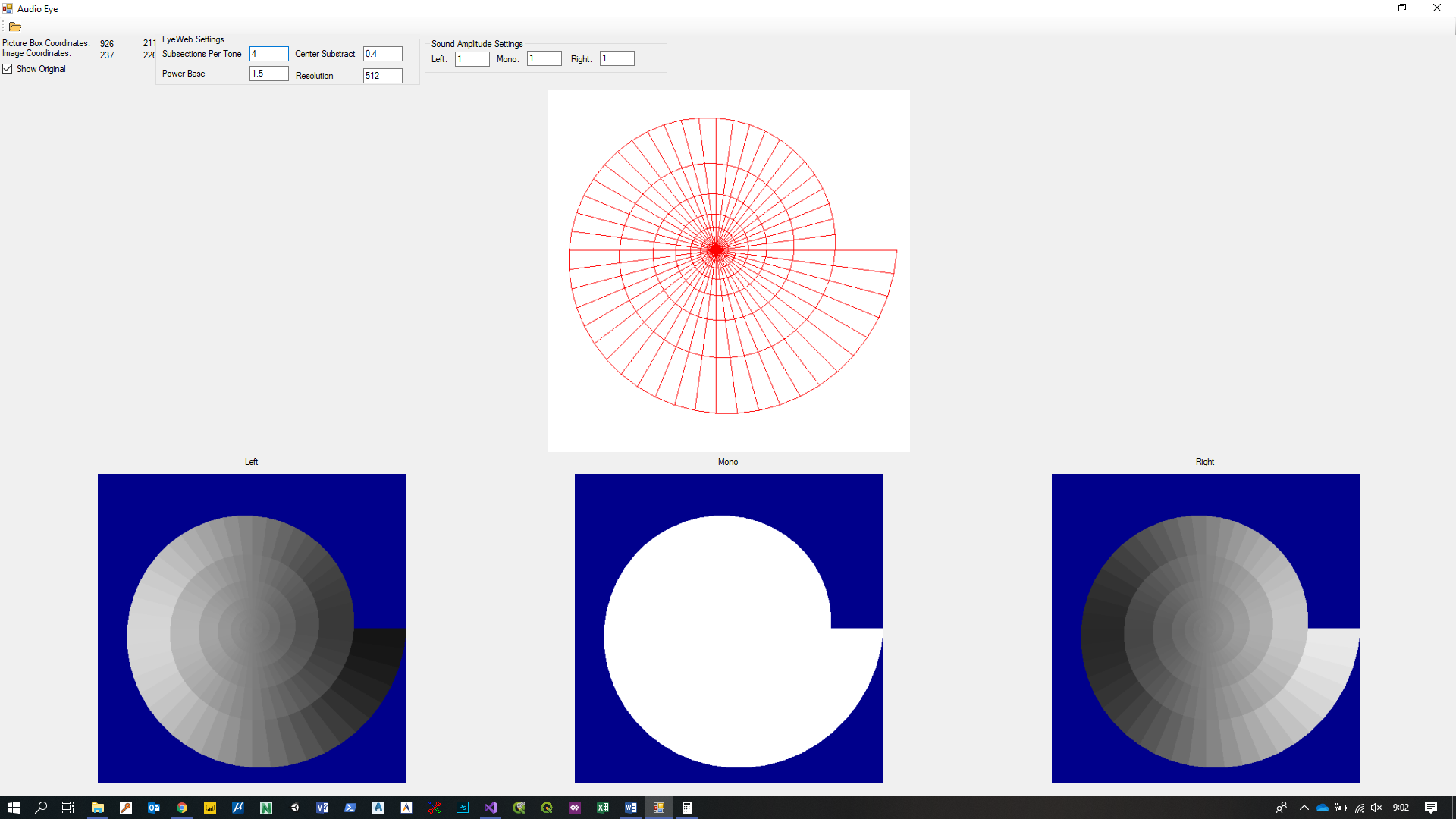


Image 3: A subdivision of the EyeWeb into 4 frequencies per tone.

The resolution can be further enhanced by subdividing the quads into smaller sections. Theoretically the web can be subdivided into an extremely high density grid, however this will be impractical for both performance and quality reasons. Performance because it will take more computer power to calculate the corresponding sound, but more important is the quality. The amount of small frequency bandwidths will eventually smear out into a vague sound, because of resonances occurring all over. 4 frequencies per musical note may be best for practical use, however further testing with other amounts is recommended.

If we consider C, octave 0 as tone 0, C# of octave 0 as tone 2, and D of octave 0 as tone 3, we can calculate the frequency for each note. We set the frequency of C octave 0 to 65.4 Hz. The frequency of each tone can be calculated using the formula:

*Frequency = 65.4 \* 2tone/12*

This formula can be used for non-integer tones as well.

### Stereo capturing

Stereo can be achieved by using two cameras. However, some stereo effect can be achieved with a single camera as well. The intensity value for a quad can be calculated by calculating the average intensity for each pixel that is within the quad. A stereo effect can be achieved by weighing the horizontal position of the pixel into the calculation. The more a pixel is positioned to the left, the more it contributes to the intensity of the left image capture, and the lesser it contributes to the intensity of the right image capture. Of course, if a pixel is positioned more to the right, the opposite occurs. Applying this for each pixel rather than for each complete quad, contributes noticeably to the stereo effect.

### Playing with contrasts.

## Hardware

This chapter describes the optimal hardware. A simpler version with simpler hardware should be possible for testing purposes. The minimal hardware should be a camera and a speaker, which is standard available in a mobile phone.

### (Sun)glasses mounted double Camera

### Eye Tracking device

### Headphones