Bionic eye: Development of a neuromorphic system for processing visual input

A Master Thesis submitted by:

Nóra Gáspár

July, 2015

Supervisors:
Dr. Konstantin Nikolic
Centre for Bio-Inspired Technology, Department of Electrical and Electronic Engineering, Imperial College London, UK

Dr. Bart Verkerke
Faculty of Medical Sciences, University of Groningen, The Netherlands

Prof. Dr. Khosrow Mottaghy
Department of Physiology, RWTH-Aachen University, Germany

TABLE OF CONTENT

ACKNOWLEDGEMENT

Primarily, I would like to say thank you to my supervisor Dr. Konstantin Nikolic and to my co-supervisor Dr. Benjamin Evans for their support, and guidance throughout the whole project. Furthermore I'd also like to say thank you to Anish Sondhi, Mohamed El Sharkawy, Nicolas Moser, and the members of IniLabs Zurich for all of their help with technical issues. Also I'd like to say thank you to the CEMACUBE program for providing me with this unforgettable opportunity, and to both of my supervisors in my home universities Dr. Bart Verkerke, and Dr. Khosrow Mottaghy, and both of the coordinators Dr. Monika Ohler, and Dr. Irma Knevel.

Last, but not least I'd also like to say thank you to Timo Lauteslager and Francesca Troiani for all the small constructive ideas (and strawberries), and to each and every member of the Center for Bio-Inspired Technology for their warm welcome and for all of their support.

ETHICS STATEMENT

This thesis was carried out in the Center for Bio-Inspired Technology in Imperial College, London, with the aim to create a novel Visual Prosthetics system. Blindness is one of the main unresolved issues in our aging society, therefore Visual Prosthetics could improve the life quality of millions of people worldwide. However during this project, only a proof-of-concept device was developed, no human (or animal) experiments were carried out and no research data was collected. This work was not published previously, but a paper was submitted on the same topic to the IEEE Biomedical Circuits and Systems (BIOCAS) 2015 conference. Here I confirm that the work was carried out by me, and I did not use citations or external sources without listing them between the references.

I. ANALYSIS PHASE

Problem definition

Eyesight is one of the most important human sense. It is used literally in every aspect of our life, such as orientation, non-verbal communication, the ability to detect and recognize objects, and the overall perception of our surroundings. Therefore blindness is one of the most feared affliction. Unfortunately it is relatively common, and in many cases it is not curable. Sight loss can vary in severity from total blindness to minor visual impairments, caused by a number of conditions. There are estimated 285 million visually impaired people worldwide, and out of them approximately 39 million are blind. Even though the term blindness is commonly used to signify severe visual impairment, blindness is strictly defined as the state of being completely sightless in both eyes. [2]

There are several approaches to restore vision even in a limited form with visual prosthetics, such as “smart glasses” (visual augmentation), implants, or sensory substitution systems. While these devices help to improve or restore, partially or completely lost vision, their performances are still relatively poor compared to other sensory prosthetics (e.g. cochlear implants). The main reason for this slow progress is the complexity of the visual system. In order to be able to reach better performance with visual prosthetics, not just the processing algorithm, but the overall behaviour of the system needs to mimic the behaviour of the human eye as closely as possible. In other worlds it needs to be neuromorphic. The aim of this project is to integrate a Dynamic Vision Sensor into a Retinal Prosthesis system, creating a low-power, neuromorphic system with multiple purposes.

Blindness, and stakeholders in the problem

There are several illnesses, injuries or congenital diseases causing blindness. The cause effects the possibility and the method to restore eyesight with a visual prosthesis, depending if the nerves are fully developed, and if they are intact. Causes can be broadly categorised based on the type and the onset of the disease. We can differentiate two categories as 1) optical blindness and 2) neural blindness; or as a) congenital and b) acquired variants.

OPTICAL/NEURAL BLINDNESS

In cases of optical blindness, cloudiness of the eyeball, either in the cornea or the lens (i.e. cataract), prevents clear images from reaching the back of the eye. Various forms of optical blindness are all treatable and mostly preventable. [3] Common causes of neural blindness include Age-Related Macular Degeneration (AMD), Retinitis Pigmentosa (RP), diabetic retinopathy, traumatic injury (causing detached retina), vascular diseases involving the retina or optic nerve, hereditary diseases, and tumors. The most common later onset illnesses in the developed countries causing blindness can be seen on the figure below.

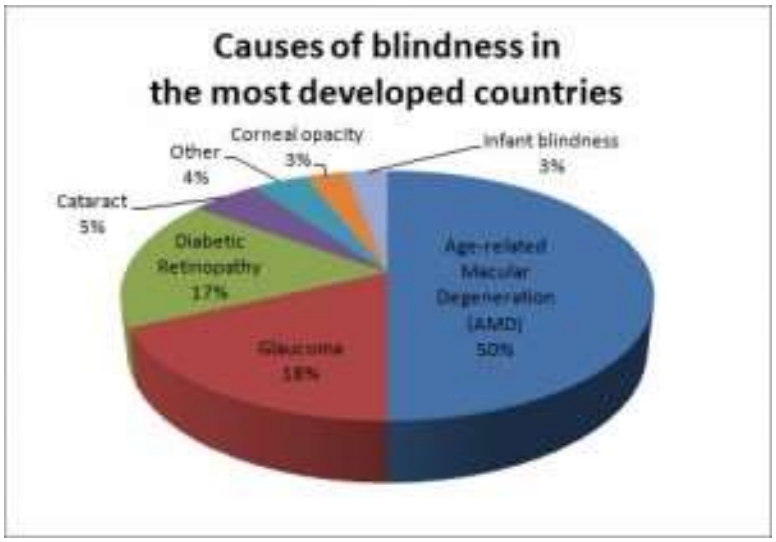


Figure I-1: Causes of blindness in the developed world[4]

While diabetic neuropathy, Cataract, and Glaucoma can all be treated with laser, or operation, until now there is no cure for Age-related Macular Degeneration and Retinitis Pigmentosa. However people suffering from these two illnesses seem to be ideal candidates for retinal implants. Age related Macular Degeneration (AMD) is a painless degenerative disease that leads to gradual loss of central vision. It usually affects both eyes, but the speed at which it progresses can vary from eye to eye. It is the leading cause of visual impairment in the UK, affecting up to 500,000 people to some degree. It is estimated that one in every 10 people over 65 have some degree of AMD.[5] Retinitis Pigmentosa is the most common hereditary progressive retinal degeneration disease. It leads to gradual loss of vision, from periphery gradually constricting towards the centre.[6] In both cases the decline is gradual over years, and only effects the retina, while the optic pathway stays intact.

COGENITAL/ACQUIRED BLINDNESS

When speaking about restoring vision, it is important to note that congenitally blind people have some "advantages" over their later-onset counterparts. Children with congenital blindness (or who went blind at a very young age) probably went to specialized schools, attended trainings, and are proficient in using assistive tools, such as Braille, guide dogs, reading programs, white sticks etc. Also their intact senses, such as hearing perform remarkably well. They might have established an independent lifestyle, where most of them can work, study, do sports or do any other everyday activity independently, with specialized tools and with minor help from their environment.

When a visual prosthesis is applied, the brain needs to learn to interpret the new flood of information. Patients who have no previous memories of vision might have serious problems integrating different perspectives over time into a coherent scene. A famous example of this problem is based on a case study from 1963 [7], where a congenitally blind patient received a corneal graft at the age of 52. The procedure “miraculously” restored his vision, however after the operations he became depressed, and finally he ended his own life. Therefore restoring vision after being deprived of visual input for their entire life, might cause more harm to these users than help.

On the other hand, people who lost their eyesight at a later stage of their life due to a disease have to face many problems with adjusting to this new lifestyle. In many cases these people become dependent on their family, or professional caretakers. This is a great problem, especially because 90% of visual impaired patients live in low income settings, and because most of blind people (82%) are at the age of 50 or above [7]. For them, and for their families even restoring a fraction of their lost vision might be a great relieve, allowing them to regain more independence.

Physiology of human eye and the visual pathway

The eye allows us to see and interpret shapes, colours, and dimensions of objects in the world by processing the light they reflect or emit, and translating it to meaningful visual scenes in the brain. This complex process can be seen as a chain of sub-processes. The eyeball is responsible for creating a focused, clear image on the retina. The retina is responsible to convert light signals to nerve signals, and to forward them to the brain where the signals are processed. [8] In order to be able to design an efficient neuromorphic visual prosthesis, the basic neurophysiological principles of the eye and the image forming processes need to be understood and simulated.

THE EYEBALL

The reflected light first enters through the cornea, then progresses through the pupil to the retina. As the eye has a curved shape, and as the light rays bend when passing from one transparent medium into another (if the speed of light differs in the two media), the transparent media of the eye function as a biconvex lens. The image formed by eye's lens system is inverted (upside-down), reversed (right-left) and it is smaller than the object viewed. Therefore, the temporal (left) hemifield of the left eye is projected onto the nasal (right) half of the left eye's retina and the nasal (left) hemifield of right eye is projected onto temporal (right) half of the right eye's retina. [9]

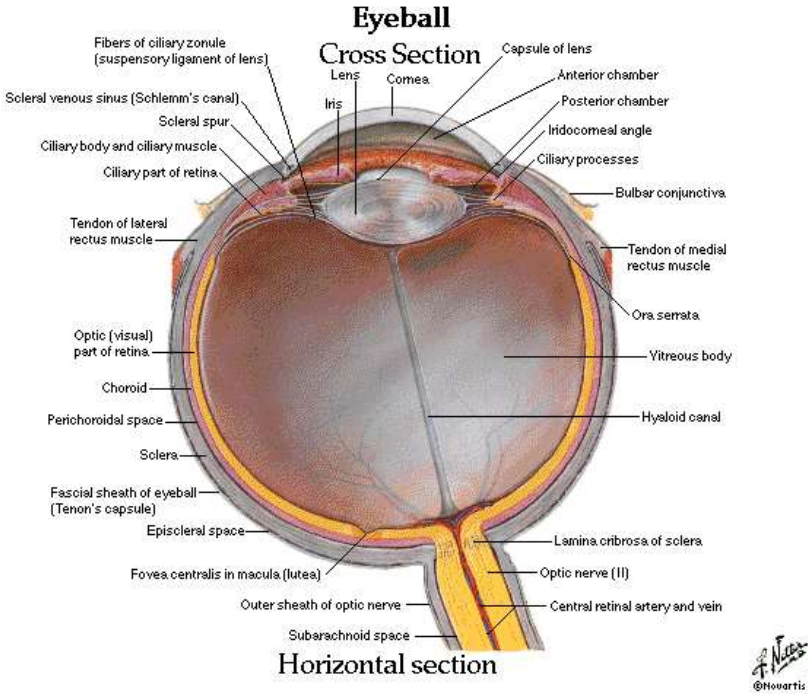


Figure -I-2 Anatomy of the human eye [10]

RETINA

The light sensitive retina forms the innermost layer of the eye, having two main layers, the retinal pigment epithelium, functioning as a separating layer, and the neural retina, where receptors and neurons can be found. This structure converts light signal to nerve signals and forwards them to the brain. The retinal pigment epithelium forms an important part of vision, as there are dark pigments within this layer. Their function is to absorb light passing through the receptor layer in order to reduce light scatter and image distortion within the eye.

The neural retina consist of three layers of nerve cells, each of them separated by a layer containing synapses. It is built up by at least five different types of neurons: the photoreceptors (rods and cones), horizontal cells, bipolar cells, amacrine cells and ganglion cells. The eye is developed in such a backward fashion, that the light first have to pass through all the layers in order to stimulate the receptor cells that are placed in the back of the retina. The layers in front of the receptors are fairly transparent and don't blur the image. Visual information can be transmitted from the receptors through the retina to the brain through the direct or through the indirect pathway.

The direct pathway includes only photoreceptor cells, bipolar cells and ganglion cells. As only one or few receptors feed to a bipolar cell, and only one or few bipolar cell feeds to a ganglion cell, this system is highly specific, and compact. On the other hand the indirect pathway also includes a horizontal cell between the receptor and the bipolar cell, and/or an amacrine cell between the bipolar cell and the ganglion cell. This way it is a more diffuse system.

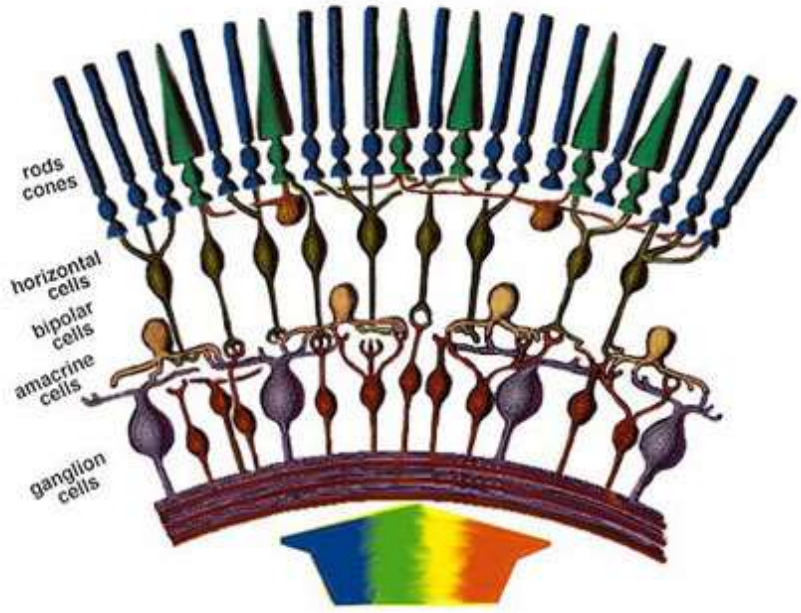


Figure-I-3: layers of the neural retina[10]

PHOTORECEPTORS

The mammalian retina has a remarkable feature to stay operational in a very wide range of light intensities. (The transition from night to day brings about a nearly 50 billion-fold change.[11]) This is achieved by using two distinct types of photoreceptors with different light sensitivities and operational ranges. The two types of photoreceptors are rods and cones. While there are only about 6.5 to 7 million cones, there are about 120 to 130 million rods in each eye. The number of rods and cones vary remarkably in different parts of the retinal surface. In the very centre, where our fine-detail vision is the best, we have only cones. This area is called fovea.

The visual process starts when light causes a chemical reaction with the photoreceptor proteins: "iodopsin" in cones, and "rhodopsin" in rods. Iodopsin is activated in photopic (bright) conditions, while rhodopsin is activated in scotopic (dark) conditions. This way rods are responsible for recognizing movements and shapes in a dark environment with high sensitivity, but low acuity, and cones are responsible for colour vision in a bright environment with lower sensitivity, but bigger acuity.

BIPOLAR CELLS

The task of bipolar cells is to receive input from photoreceptors and horizontal cells (one or few per bipolar cell), and feed their output to a retinal ganglion cell. Several bipolar cells feed to one ganglion cell. Rods and cones are feeding separately to their respective bipolar cell. Bipolar cells process visual signals through integration of analogue signals (synaptic currents). They come in two fundamentally different forms: ON and OFF cells, depending if they are hyperpolarized by central illumination (OFF) or depolarized by central illumination (ON).[8]

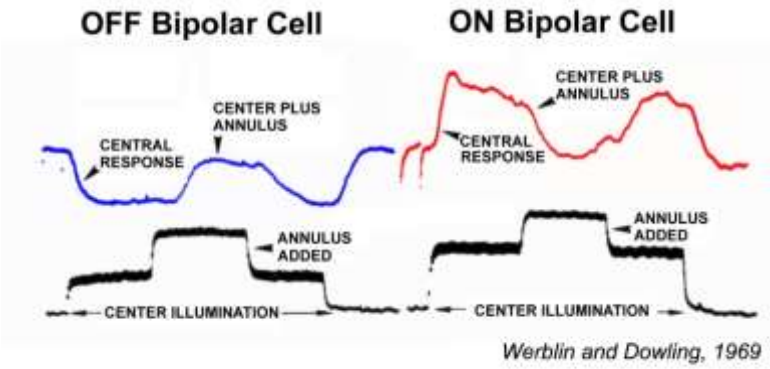


Figure I-4: OFF and ON bipolar cells[12]

HORIZONTAL CELLS

These large cells link receptors and bipolar cells by relatively long connections that run parallel to the retinal layers. They only take part in the indirect pathway. Their processes make close contact with the terminals of many photoreceptors distributed over an area that is wide compared with the area directly feeding a single bipolar cell.

AMACRINE CELLS

Similarly to horizontal cells, amacrine cells link bipolar cells and ganglion cells by parallel connections. They also only take part in the indirect pathway. There is a wide variety in their types and their functions, many of them are unknown yet.

GANGLION CELLS

Ganglion cells are placed in the third, innermost layer of the neural retina. They are the first neurons in the process that respond with an action potential. [13] Their axons pass across the surface of the retina, collect in a bundle at the optic disc, and leave the eye to form the optic nerve. As several receptor cells feed into a bipolar cell, and several bipolar cells feed into a ganglion cell, there is an approximate 125:1 ratio between the number of receptor cells and the number of ganglion cells. This way there are "only" 1 million ganglion cells in each eyes. Similarly to the bipolar cells, there are ON and OFF ganglion cells. The two parallel pathways of ON and OFF ganglion cells are not just physiologically, but even anatomically separated, and they only merge in the primary visual cortex.

VISUAL PATHWAY FROM THE EYE TO THE BRAIN

The optic nerve is a continuation of the axons of the ganglion cells in the retina. There are approximately 1.1 million nerve cells in each optic nerve. In the optic chiasm, the optic nerve fibres originating from the nasal half of each retina cross over to the other side, but the nerve fibres originating in the temporal retina do not cross over. From there, the nerve fibres become the optic tract, passing through the thalamus and turning into the optic radiation until they reach the visual cortex in the occipital lobe at the back of the brain. This is where the visual centre of the brain is located.

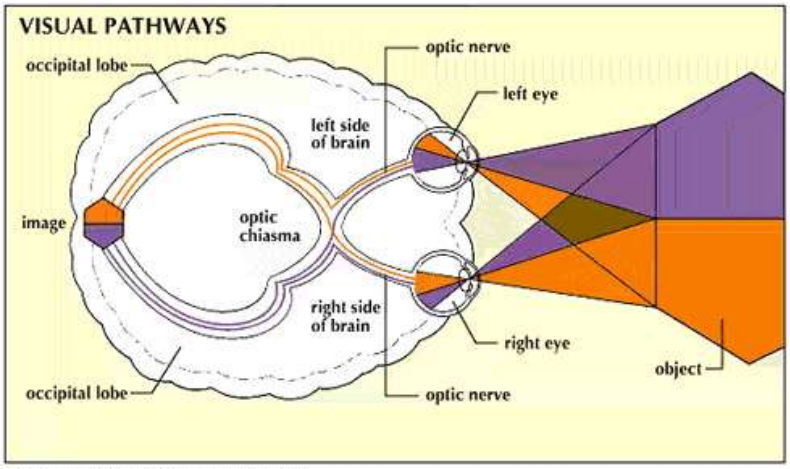


Figure I-5: Visual Pathway illustration[10]

RECEPTIVE FIELD

Receptive field of a neuron is the region over which we can stimulate the cell. Additionally to the spatial dimension, the term receptive field also includes a temporal dimension. The spatiotemporal receptive field describes the relation between the spatial region of visual space where neuronal responses are evoked and the temporal course of the response. This is especially important in the case of the direction selective responses. On the retina both bipolar and ganglion cells have a receptive field. Retinal ganglion cells located at the centre of vision, in the fovea, have the smallest, and those located in the visual periphery have the largest receptive fields. This explains the phenomenon of having poor spatial resolution in the periphery when fixating on a point. The receptive field can be subdivided to two parts: centre, and surroundings.

Additionally, two types of retinal ganglion cells can be defined the 'ON'-centre and the 'OFF'-centre type of cells (similarly to the case of bipolar cells), depending if the centre of the receptive field would give an ON or an OFF response. The two system works parallel, but completely distinct from each other (both physiologically, and anatomically). Both are completely covering the visual field.

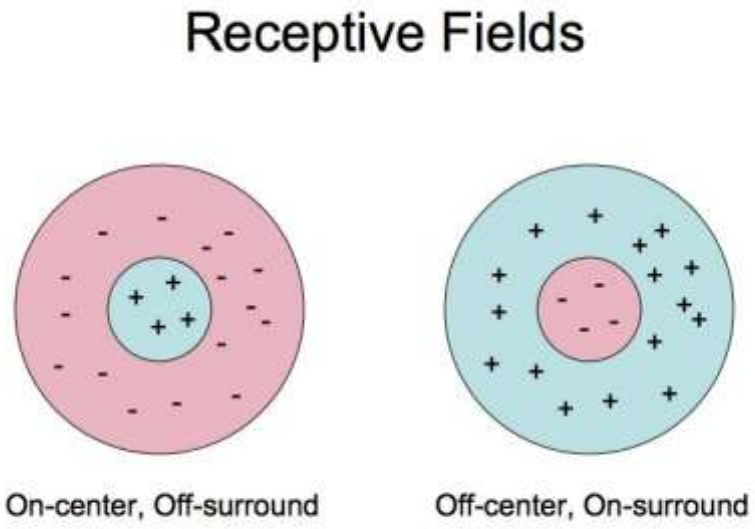


Figure I-6: receptive fields [13]

Visual prosthetics overview

One of the first researches on the effect of electrical stimulations of the optic nerve derives back to the 18th century. In 1755, French physician and scientist Charles Leroy discharged the static electricity from a Leyden jar into a blind patient's body using a wire attached to the head above the eye, and one to the leg. The patient who has been blind for 3 months then, described the feeling like if a flame was passing downwards in front of his eyes.[14] Nowadays, 250 years later, blindness is still a primary unresolved issue of the modern society, and visual implants are aiming to work in a slightly similar way, than Dr. Leroy's jar did. They provide visual information to patients by stimulating the healthy neurons in the retina, in the optic nerve, or directly in the brain's visual areas, with various neural impulses. The different designs of implants are named according to their locations (i.e., cortical, optic nerve, subretinal, and epiretinal).

GENERAL FUNCTIONAL ANALYSIS OF VISUAL PROSTHETICS

Even though each device differs in many ways, the general system of visual prosthetics are very similar. There is a visual sensor (camera) capturing the scene and transmitting the information to an image processing unit, which transforms the signal and forward it to a stimulator. Figure I-7 describes the functional analysis of the general system. Explanation of the signs can be found in the appendix.

Depending on the device, the stimulator might have very different implementations. In many cases it is an implant along the visual pathways within the central nervous system, but there are other alternative non-invasive techniques too, called sensory substitution devices (SSD). While in case of implants, the stimulating signal is always electric current, in case of non-invasive devices it can also be voice, (enhanced) light, or vibrotactile stimulation as well. Both visual implants and non-invasive prosthetic devices have advantages and disadvantages as well. In the followings the most popular available implants and SSD devices will be introduced and their performance will be compared.

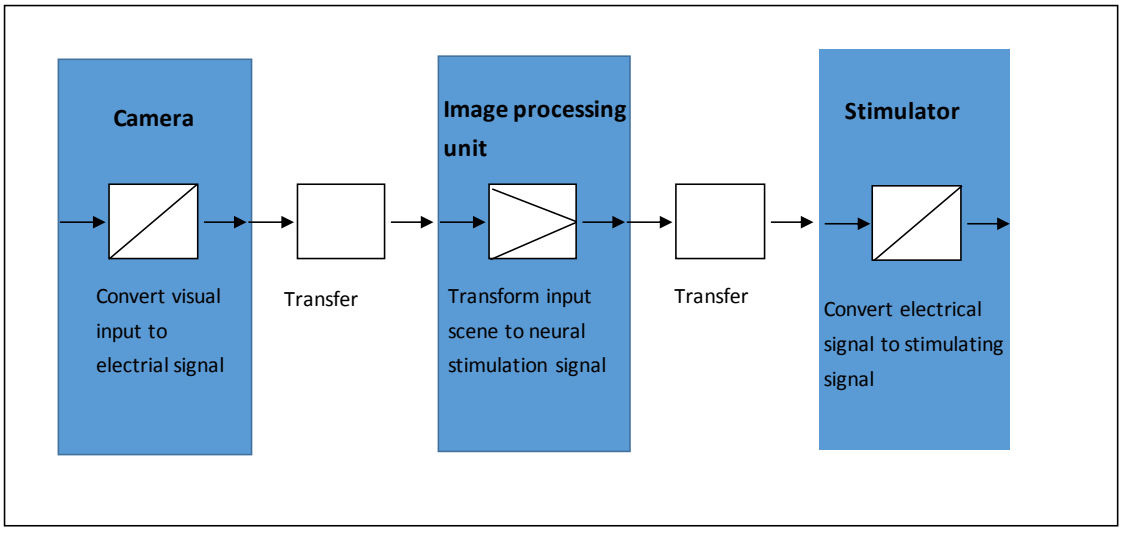


Figure I-7: Functional analysis of Visual Prosthetics

IMPLANTS

Visual implants can be placed in several different locations throughout the optic pathway, however the most commonly used invasive visual prostheses are retinal implants. There are several approaches in terms of placement of the electrode array, such as subretinal [15], epiretinal[16] and suprachoroidal or intrascleral implants [14] (see Figure I-8).

In the subretinal approach, electrodes are placed between the retinal pigment epithelium and the retina, where they stimulate the non-spiking inner retinal neurons. Suprachoroidal implants are placed in a less invasive location, between the vascular choroid and the outer sclera. Both subretinal and suprachoroidal implants utilize retinal processing network from the bipolar cells down to the ganglion cells to the brain, preserving the eye's normal processing structure. Epiretinal devices are placed on the anterior surface of the retina, completely bypassing the retina, and directly stimulation the ganglion cells.

There are clinical studies proving that different retinal implants can help restoring some functions of the visual system, such as light perception, recognition of object, and in some cases even reading letters.[17] However resolution of retinal implants are limited due to the heat dissipation caused by the injected current. Up until now the number of electrodes successfully implanted are only 60-1500. [15, 18]

Additionally, these devices are only efficient in case of specific illnesses, namely in case of later-onset photoreceptor diseases such as Retinitis Pigmentosa, or Age-Related Macular Degeneration.[19]

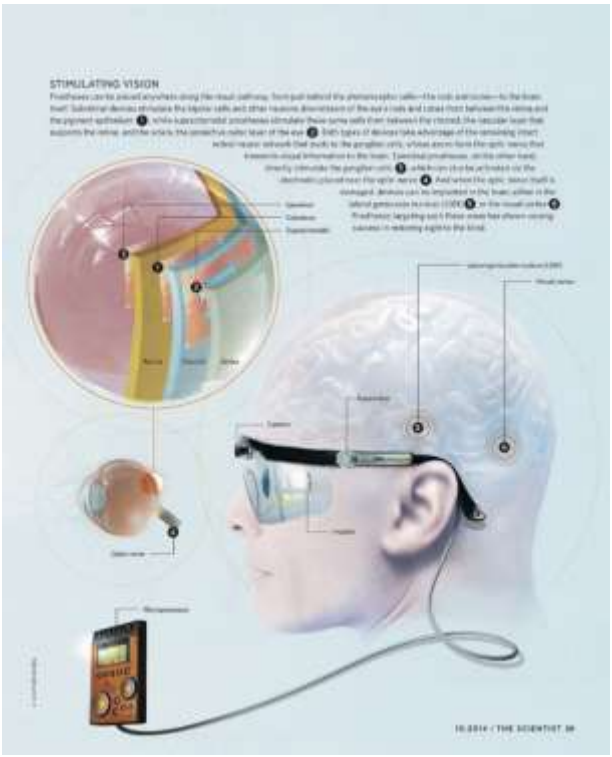


Figure I-8 Possible stimulator locations[20]

Cortical implant devices are bypassing the optical nerve, and with different stimulation methods, directly stimulate the visual cortex. One of the main advantages of such systems are that they can be used by a wide range of blind users. In the followings some of the most well-known examples of implant systems will be introduced.

Dobelle cortical implant [21]

Since 1968 Dr. Dobelle and his team has been working on an artificial vision system where the stimulator device is an electrode array implanted in the occipital lobe. They were the first to successfully implant such electrodes in 1970-72. The man on Figure I-9 was the first person successfully using a portable system. He was able to be placed in a room, recognize and retrieve a black ski cap from a far wall, turn around, recognize a mannequin, walk over and place the cap on its head. This led to his recognition in the Guinness Book of Records in 2001. Since then their system has developed further in many ways, and currently the implant system is commercially available in Portugal.

Figure I-9: The first patient using a portable system[21]

THE ARGUS II [16]

The Argus II Retinal Prosthesis System is the world's first approved retinal implant intended to restore some functional vision. It is approved for use in the United States (FDA) and European Economic Area (CE Mark) and is available in some European countries. So far with this system users can read letters, or recognize objects, but the restored vision is still very limited.

The system consist of an implanted part, and an external part. The epiretinal implant includes an antenna, an electronics case, and an electrode array consisting of 60 electrodes. The additional external equipment includes a camera, a video processing unit (VPU), and an antenna built into the glasses (see figures below). The captured scene is processed by the VPU, then transmitted to the electrode array in the implant. The implant stimulates the ganglion cells in the retina, creating the perception of patterns of light.

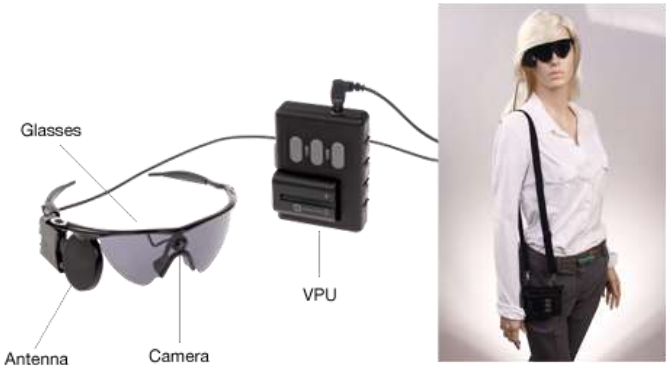


Figure-I-10: Argus II external parts

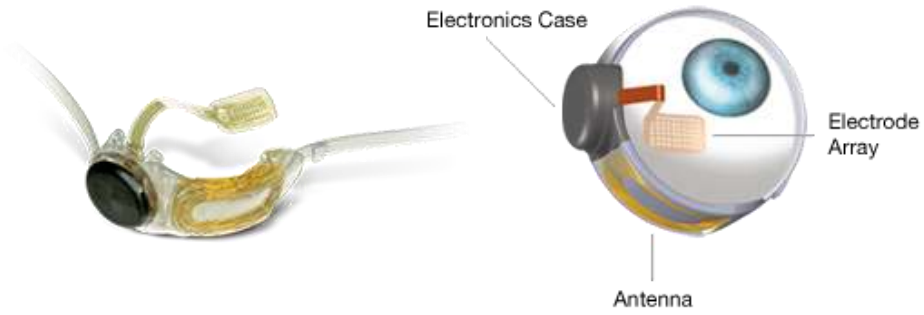


Figure-I-11 Argus II implanted device

Alpha-IMS

The Alpha-IMS [22] is a novel subretinal implant developed in a collaboration between several German universities. The system has an outstanding resolution (1500 pixels), and unlike other systems, it has no front-end. The implant itself contains the CMOS camera, the processing unit, and the stimulator as well.

A non-invasive approach: Optogenetic Prosthesis

While retinal implants stimulate nerves through small electric currents, another interesting approach is being researched in Imperial College London, based on the discovery that neurons can be photostimulated via genetic incorporation of artificial opsins. An optogenetic retinal prosthesis uses opsins (e.g. Channelrhodopsin-2) to render neural cells sensitive to light such that light can then be used to modulate their activity. Therefore, instead of an invasive electrode array, a non-invasive external LED array can be used as the stimulator.[23, 24]

NON-INVASIVE SOLUTIONS - SENSORY SUBSTITUTION DEVICES (SSD)

While implants might be the long term solution to help incurable blindness in the future, there are alternative possibilities, and their performance is just as good, or even better than the current possible performance of implants. SSD systems "bypass" damaged nerves by using other senses (voice, touch) to transmit the information. Effectiveness of these devices mostly depend on neural plasticity, the capability of the brain to "rewire". In the followings some of the most famous SSD systems will be introduced.

vOICe system

The vOICe, auditory SSD system aims to replace visual information with audial one. Theoretically the acuity can reach really high resolution, but interpreting the signals requires extensive training. However experienced users are able to retrieve detailed visual information at a much higher resolution than with any other visual prosthesis. Remarkably, proficient users are also able to perform outstanding results on the Snellen-test, even passing the WHO blindness threshold. [25]

The device consist of camera mounted in the glasses, a processing unit (a smartphone), and stereo headphones. The images are converted into "soundscapes" using a predictable algorithm that can be seen on Figure I-12. In the algorithm every pixel has three attribute: 1) y coordinate: represented by the tone, 2) brightness level: represented by the volume level, and 3) x coordinate: represented with timing. The system scans through the frame from left to right, and gives a ticking noise at the end of the frame to let the user know that a new frame is coming.

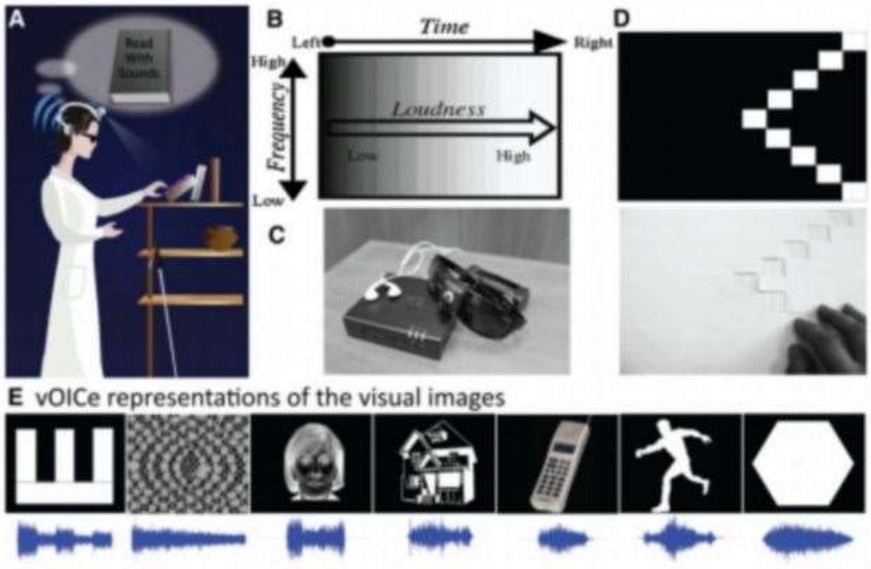


Figure I-12 vOICe algorithm and examples[26]

BrainPort® V100

The BrainPort device translates information from a digital video camera to the user's tongue, using gentle electrical stimulation. The system consists of a stamp–sized array of 400 electrodes placed on the top surface of the tongue (see figure below), a camera affixed to a pair of sunglasses and a hand-held controller unit. With training, completely blind users learn to interpret the images on their tongue. [27]

Figure I-13: Tongue stimulator[27]

CONCLUSION

There are plenty of different approaches in the field of sight restoration. Every approach have advantages and disadvantages as well. At the current state of the art, implants can only restore a very restricted vision, on a cost of a very expensive invasive surgery, and they can be applied only to a restricted group of users. On the other hand, in a long term perspective implants have the big advantage that the restored vision is more close to real vision, and the system would not interfere with other senses. Therefore despite of all the disadvantages, further developments would be beneficial.

On the other hand Sensory substitution devices, have a lot of advantages in terms of usability and efficiency. They are cost efficient, non-invasive, and they are available for all kind of users. They could be especially beneficial for congenitally blind users, and for users who could not afford an implant surgery.

In order to further develop both implant and SSD systems, the focus from the improvement of the stimulator side needs to be shifted to the focus on the camera and image processing side. Most of the current systems have a common disadvantage: they use a conventional frame based camera as their front-end system, and their image processing is not neuromorphic.

Neuromorphic image processing

In recent years, a new approach to engineering and computer vision research was developed, that is based on algorithms and techniques inspired from, and mimicking the principles of information processing of the nervous system. The most relevant example of these new developments is the Dynamic Vision Sensor (DVS).

CONVENVENTIONAL CAMERA VS. DYNAMIC VISION SENSOR

A conventional cameras see the world as a series of frames, containing a lot of redundant information, which causes waste of time, memory and computational power. On the other hand the DVS camera recognises, and transmits relative changes on pixel level, asynchronously, and independently. While a conventional camera recognises and transmits discrete intensity levels, the DVS camera (similarly to the retinal ganglion cells) recognises ON and OFF events depending if the light intensity increases or decreases on the given pixel. The recognised event is transmitted with only 15 μs latency, in a so called address-event form. (AE)

Apart from being power and memory efficient, the DVS camera would provide extra benefits in a retinal prosthesis system with its neuromorphic behaviour. Not just that it could avoid constant stimulation caused by large uninformative areas, but even more importantly, it would make it simpler to target ON and OFF retinal pathways separately. Even though the camera's current resolution is only 128x128 pixels, it is already more than enough for being used as a front-end camera for any of the currently available retinal prosthesis system.

Previous research carried out in our department

Before the goals of this project can be set, it is important to point out that this thesis is built on several previous student projects carried out in this department on the topic of retinal prosthesis and the possible usages of the DVS camera. The implemented system is a continuation of the project carried out by Anish Sondhi, with the title *"Interfacing an embedded DVS (eDVS) to a microcontroller and an LED matrix"*[1].

Requirements and goal setting

Based on the previous chapter, certain wishes and requirements were deducted. A low-power, real-time, neuromorphic system is required, that is portable, easy to use, convenient for the proposed application, and widely available. Considering these requirements, the overall goal of the design project was defined with the following two points:

1. ***Development of a low power, real time, neuromorphic front-end system for visual prosthetics, using an eDVS camera (both implants and SSD devices)***
2. ***Development of a low power, real time, neuromorphic Visual to Audio Sensory substitution system using an eDVS camera***

II. APPLIED ALGORITHMS AND NEURON MODELS

When speaking about neuromorphic image processing we are aiming to send signals to the stimulator that are as close to what they would have received from a healthy retina, as possible. In order to be able to reproduce the behaviour of a retinal ganglion cell, certain simplified mathematical models need to be applied. The simplest neuronal model, the "Integrate and Fire" [28] model was developed by Lapicque in 1907, but it is still widely used today.

INTEGRATE AND FIRE NEURONAL MODEL

Lapicque modelled the neuron using an electric circuit consisting of a parallel capacitor and resistor, representing the capacitance and leakage resistance of the cell membrane. When the membrane capacitor was charged to certain threshold potential, an action potential would have been generated and the capacitor would have discharged, resetting the membrane potential to zero. The model can be mathematically described with the following equation:

$$I(t) = C_m * \frac{dV_m(t)}{dt}$$

While we note that more complex models, such as leaky integrate and fire model including a refractory period would be more accurate, in our system only the simplified version of this algorithm was used. Introducing more complex algorithms might be a topic of future work.

LINEAR RECEPTIVE FIELD MODEL

In the simplified linear model ganglion cells act like shift-invariant linear systems, where the input is stimulus contrast, and the output is the firing rate. They obey the rule of additivity, scalarity and shift invariance, as they give the same response, when stimulated with the same visual stimulus again later in time. Also this model assumes, that there are ganglion cells with similar receptive fields located at different retinal positions. [13]

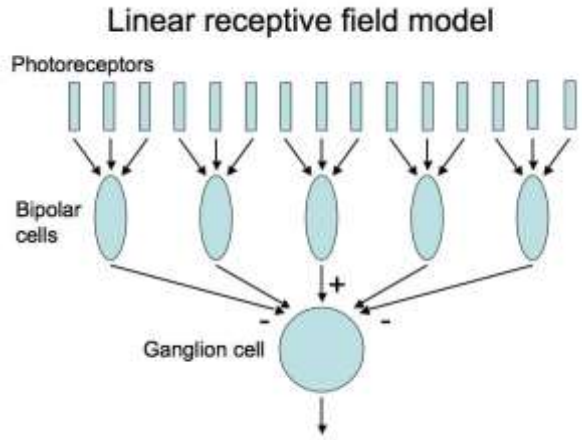
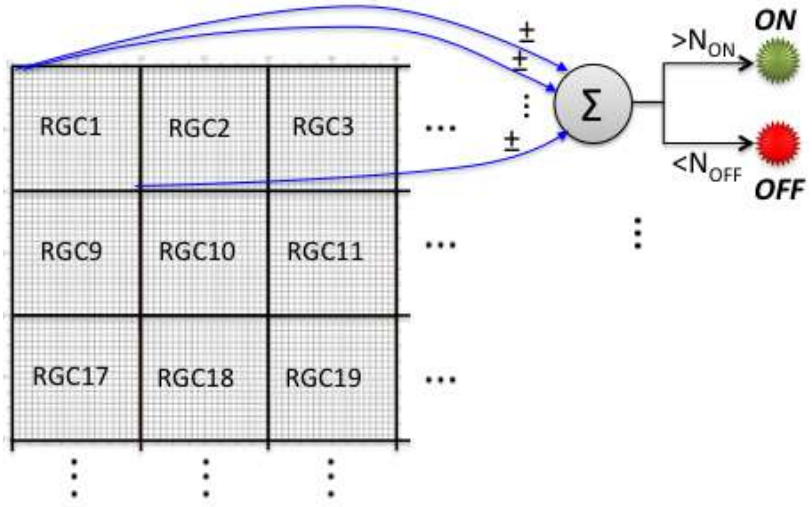


Figure II-1: Block diagram of the linear receptive field model[13]

Image processing algorithm

Due to the already neuromorphic behaviour of the eDVS camera, a relatively simple algorithm can be used on the processing unit: We consider the input to each LED to be a retinal ganglion cell (RGC), and we apply the simplified version of the Integrate and Fire model and the Linear Receptive field model. The 128x128 address matrix of the eDVS events is mapped to the 8x8 representation of the LED matrix, by discarding the last four bits of each eDVS event addresses. In this way each RGC has an effective "receptive field" of 16x16 (=256) DVS pixels. The 'synaptic inputs' are ON and OFF events, quantified as +1 and -1, summated from the addresses that correspond to the receptive field of each RGC[29]. If a certain threshold is reached (N_{ON}/N_{OFF}) the RGC spikes, which is represented with the activation of the corresponding LED. Threshold levels are adjusted in a way, that only those events can be outputted that reach 90% of the maximum counter value. This way the system can adapt to different light conditions and setups. Additionally cut-off ratios can be adjusted with an external button, and there is a fixed minimum threshold applied, eliminating most of the noise. Different LED colours are representing ON and OFF events at different threshold settings. (Green and yellow for ON, and red and blue for OFF).



II-2"Receptive field" grids of individual RGCs. Each RGC receives input from 16x16 eDVS pixels. Events from the receptive field of each RGC are summated and if they reach the threshold for either ON or OFF events, the corresponding LED is lit up. [1]

Visual to Audio mapping

Once an RGC is activated with the previously described algorithm, it needs to be converted to audio output in a way that it is easy to interpret, and takes advantages of the most important features provided by the eDVS camera, such as being real-time, and work asynchronously. The two main requirements towards the algorithm are the followings:

1. ***To maintain the real-time behavior, the user need to "hear the whole frame at once" without delay.***
2. ***The activation/deactivation of sounds needs to be as asynchronous as possible***

While the Integrate and Fire spiking neuron model remains the same, instead of an 8x8 RGC matrix, for simplification reasons here we are using only a 4x4 matrix, and an extra two steps algorithm was developed in order to meet all the aforementioned requirements, and resolve possible issues.

MAPPING

The 4x4 frame is mapped into two (left and right) 4x4 matrices in a way that each Y value is represented with a discrete frequency level, and each X value is represented with a discrete volume weight (see figures below). In order to make the stereo output more comprehensible, the frequency levels at the two ears are slightly different, but they are in a close range. Theoretically in the left ear musical note C3, C4, C5 and C6, and in the right ear A2, A3, A4 and A5 can be heard, but the synthetized PWM signals are not exactly these notes.

The volume weighting in the two ears are inverted (see below), this way the user can literally hear how an object moves from the left to the right. In the following figure the two different mapping can be seen in the left and in the right ear. V0, V1, V2 and V3 represent volume weights, while F0, F1, F2 and F3 represent frequency levels.

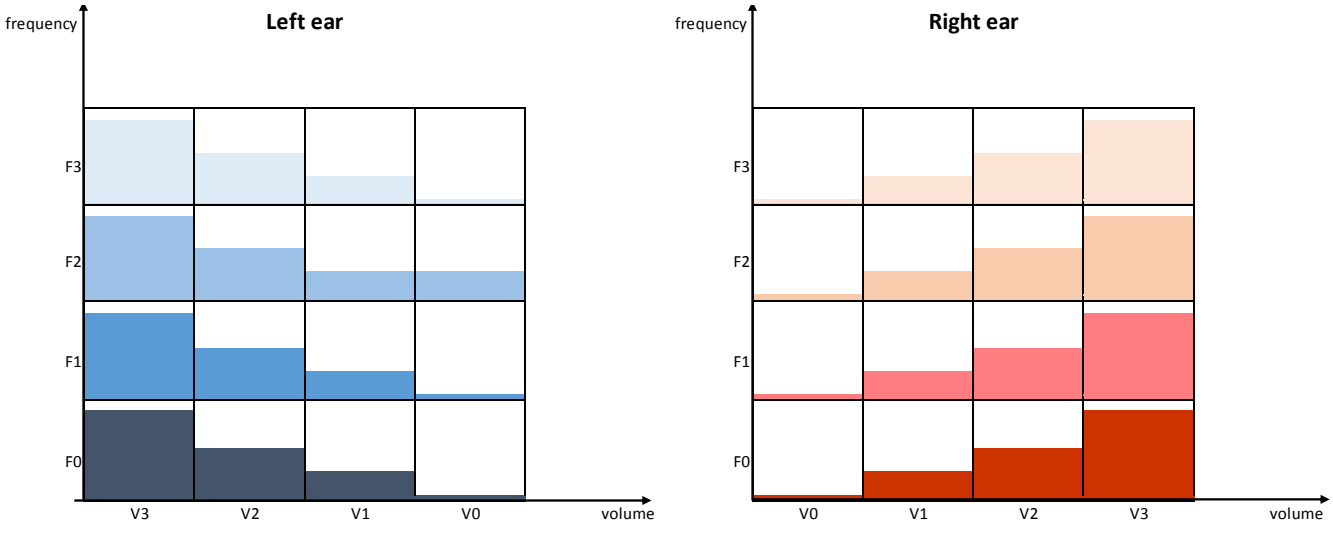


Figure II-3 a&b: Volume weight-Frequency mapping in the right and the left ear

SUMMATION

Once the frequency and volume weight of each pixels are defined in both ears, the algorithm simply sums up volume weights for both ears separately, corresponding to the active pixels. This way the user can hear two polyphonic tones (one in each ear), with each frequency on a different volume level. With the summation, there are many possible volume couples that can be the result in the two ears. In order to pair each possible frame an individual tone, while keeping the system as simple as possible, an appropriate weighting need to be applied on the volumes.

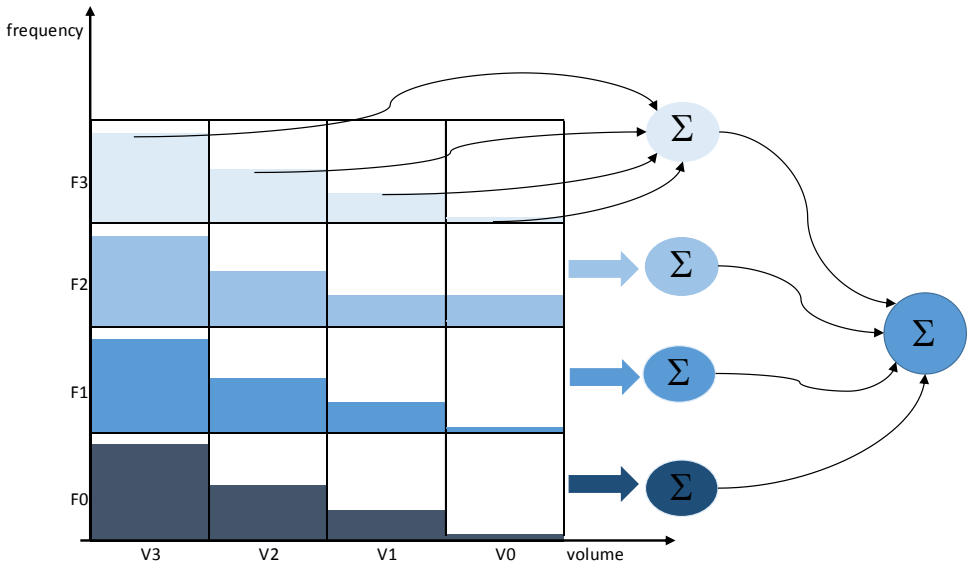


Figure II-4: Frequency summation

WEIGHTING

While intuitively it would be logical to use the most simple weighting where V0=0, V1=1, V2=2, V3=3, it would cause overlaps in activation patterns for different frames. In order to avoid these problems, the following weighting was applied V0=0, V1=1, V2=2, V3=4 (see Figure II-5). With this weighting there is no overlap, and still only 7 discrete volume levels need to be applied (plus 0=silence), which can be easily interpreted by the user. Figure II-5 presents a simulation of all possible activity pattern options for F3. The frames highlighted in green are those frames that would cause overlap if V3 would equal 3. The same pattern would apply to other frequencies as well.

While the volume weighting algorithm makes it possible in theory to differentiate between activation patterns, it is important to note that the usability of the system depends on the appropriate choice of volume levels physically, as well as on the appropriate length of activation periods, and on the proper choice of frequencies as well. Further usability research would be required in order to determine these values, but in the current setup the volume levels were arbitrary, and the length of activation periods were defined according to the refreshing rate of the SPI interface.

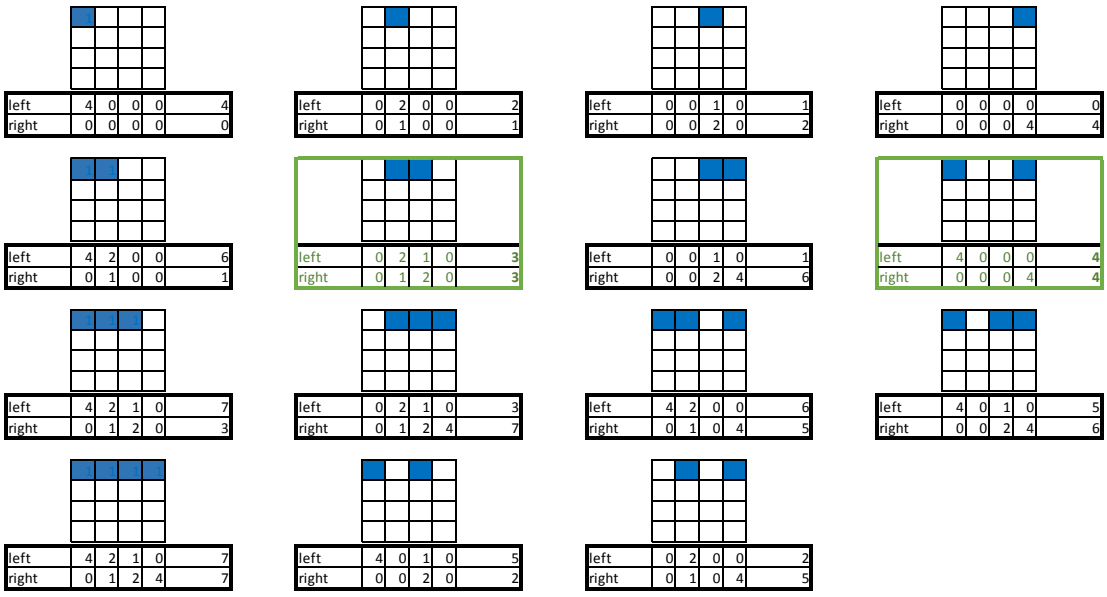


Figure II-5: volume weighting: V1=0, V2=1, V3=2, V4=4

EXAMPLE ACTIVATION PATTERN

On Figure II-6 the algorithm is represented through an example frame, which is a symmetrical smiley face. On Figure II-7 the result of the summation can be seen.

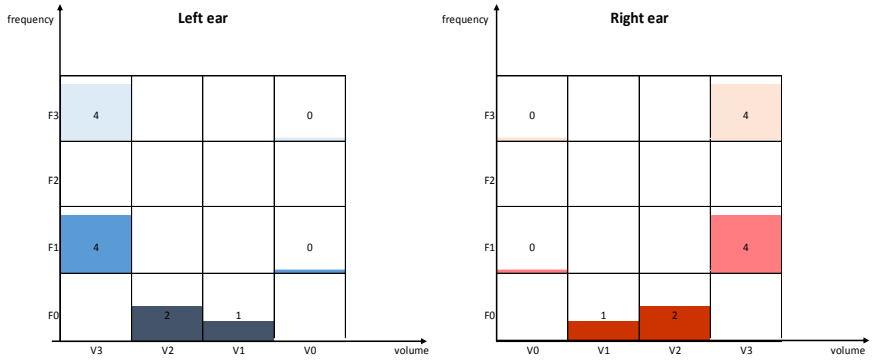


Figure II-6: Example frame

Figure II-7: Resulting volume levels

III. SYSTEM DESIGN AND IMPLEMENTATION

The system design and implementation is also based on previous works carried out by Imperial College students. [1] Initially the system consisted of three devices. The eDVS (embedded Dynamic Vision Sensor[30]), a PIC18 developer board which is the processing unit, and an 8x8 LED matrix representing the stimulator (see Figure III-1). The second setup, the SSD audio system works as an addition to the original eDVS→PIC board→LED-matrix design, as the voices could not be tested without the visual output on the LED-matrix. The third setup applies the same algorithms, but uses an Android device as the processing and stimulator unit.

1. Setup : Front-end system

This system applies the above mentioned image processing algorithm, and outputs the results only on the 8x8 LED-matrix. The communication between the camera and the PIC board is carried out through RS-232 protocol, and the communication between the microcontroller and the LED-matrix is carried out through SPI protocol.

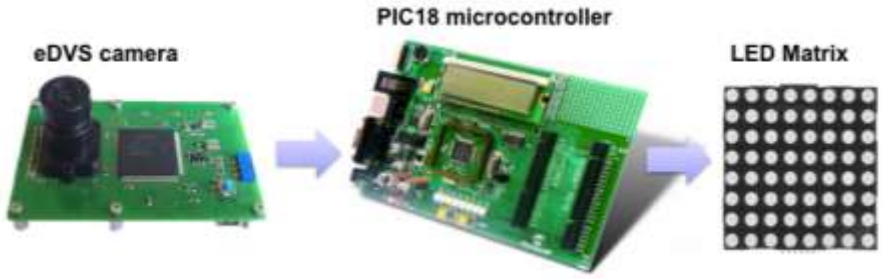


Figure III-1: Elements of the first setup[1]

COMMUNICATION PROTOCOLS

The camera is connected with the microcontroller using the UART2 communication port on the developer board, with the baud rate of 115200 bits/s. Even though the camera would be capable of faster communication (up to 4 Mbit/s), due to the restricted capabilities of the PIC board, we had to reduce the baud rate. Even though this caused significantly less events to be received, it does not affect the overall performance of the system. The LED matrix is connected to the microcontroller using the SPI2 port of the developer board.

EDVS CAMERA

The eDVS records data from the visual scene and sends pairs of characters (events) to the microcontroller in a continuous manner. Each event consists of a 2-byte (16-bit) word, where the bits arrive in a predefined order (see Figure III-2). The first byte consist of a synchronization bit and the 7 bit y-address. It is followed by the second byte consisting of one bit polarity (ON event: 0, OFF event: 1), and a 7 bits *x*-address. These addresses represent the event in a 128x128 frame.

In order to synchronize the events and avoid coupling the first event's x address with the second event's y address (see Figure III-2), the MSB bit of every y address needs to be checked, and if necessary, the stream should be "shifted" with a byte.

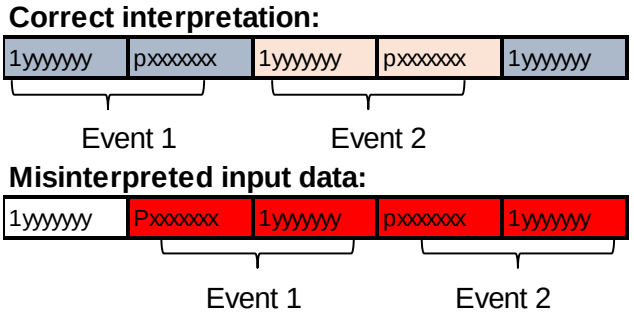


Figure III-2: eDVS data interpretation

PIC18 DEVELOPMENT BOARD

For the processing unit, the PICDEM™ PIC18 Explorer Demonstration Board[31] was used. This contains an 80 PIN PIC18F8722 microcontroller that is mounted on the development board. The LCD screen, the 8 LED pins, and the buttons were used for debugging purposes, but they are not used in the final solution. The microcontroller is programmed in C language with the MPLAB software development IDE using the CCS C compiler[32, 33]. The pin arrangement of the board is summed up on Table 1. The RxD and the TxD pins are used for UART communication, while the SCLK, /CS, and MOSI pins are used for SPI communication. Both the eDVS and the LED-matrix are powered from the PIC board. The board itself can be powered through the VIN, and the GND pins with a single 9V battery using a regulator circuit described in Setup 2, or with its own power supply.

Connection to	Pin	Purpose
DC input	VIN	+5V input from the custom power supply
	GND	Ground from the custom power supply
eDVS	+5V DC input	Vcc
	RG1	RxD (UART)
	RG2	TxD (UART)
	GND	GND
LED-matrix	+5V DC input	Vcc
	RD6	SCLK (SPI clock)
	RD1	/CS (SPI chip select)
	RD4	MOSI (data from microcontroller)
	GND	GND

Table 1: PIC18 pin arrangement

Handling incoming events

The microcontroller handles each incoming event-byte in an RS-232 interrupt function. Each event is processed right in the interrupt, this way avoiding excess memory usage or loss of events. When a new byte arrives, the algorithm checks if it is a valid X or Y byte, then stores the character in a union of characters and bit fields. By including both characters and bit-fields in the same union, events could be stored as characters but read as bits to extract the addresses and polarity directly inside the interrupt without the need for additional processing. As each RGC has a 16x16 pixels receptive field, the last four bits of each incoming address can be discarded, and only the three most significant bits are used for defining indexes in an 8x8 counter array. Depending on the polarity, these counters are increased or reduced with each event. At the end of each acquisition period, thresholding is applied, and the new frame is sent out through the SPI interface. The processing algorithm of incoming events can be seen on the following flowchart:

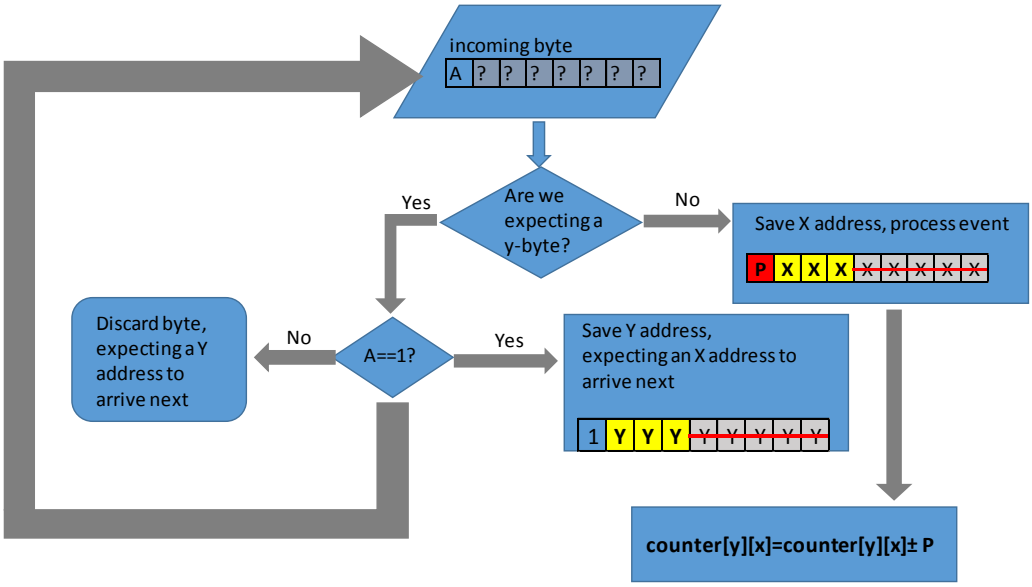


Figure III-3: Incoming byte processing algorithm

SPI communication to the LED matrix

The driver on the LED matrix takes input for a whole frame, (i.e. for all 64 LEDs), at once, therefore we have to send 64 bytes of data (8 bits per LED) consecutively, even if that requires just switching on one LED. An SPI communication cycle to the LED-matrix consist of the following steps:

- CS pulled low, communication initiated
- 0.5 ms delay to allow the system to start the communication
- Send the reset character (0x26) to reset the LED matrix
- 64*8 bit to define the RGB colour of each LED in the frame
- CS set high, communication is over

The time to transfer the data from the microcontroller to the LED matrix depends on the maximum possible clock frequency of the SPI protocol, which in this case is 125 kHz, therefore it takes the system approximately 5ms to send out a new frame. Theoretically it would be more efficient to use an address-event (AE) representation based output system, than to send out whole frames, however in this particular system, due to the aforementioned 0.5 ms delay, the output would be significantly slower if all events were to be sent separately. Therefore, even though an AE representation is used to receive and process the input, whole frames are sent to the LED matrix as an output. Since the LED matrix only serves to illustrate a real stimulator interface and as the method of outputting events does not really affect the processing method, we have decided to accept these limitations, and use the 5ms SPI communication time as an acquisition time for the incoming events. Similarly this is not an issue for the SSD, since the upper limit of the display is 200 frames per second – significantly faster than a normal display.

Processing pipeline

The system's processing pipeline can be seen on Figure III-4. The three main steps: 1) event acquisition and thresholding, 2) SPI communication between the microcontroller and the LED matrix, and 3) display on LED matrix; happen in parallel, each of them taking the same time. This time also defines the shortest possible acquisition period, which is in our model 5ms (one frame refreshing period), due to the aforementioned limitations of the slow SPI communication. We note however, that more biologically accurate time windows would be closer to the RGCs memory, which is in the region of 200–300 μs [34].

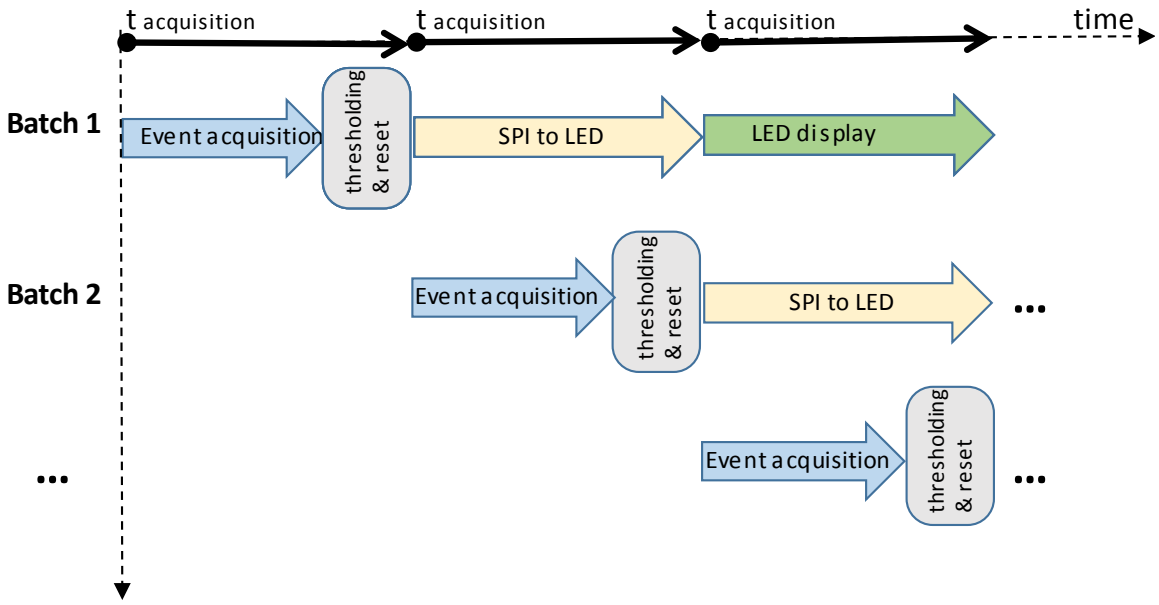


Figure III-4: Processing pipeline. Each cycle consists of collecting and processing events received from DVS, thresholding them in the microcontroller and displaying the stimulation patterns on the LED array.

LED-MATRIX

In order to visualize the output of the processing algorithm, an 8x8 tri-colour (RGB) LED matrix was used from SparkFun [35]. Each LED represents an output of an RGC. The colour of each LED is defined by an 8 bit RGB colour value. Colour pairs are used to indicate different events, with green, or yellow corresponding to ON type and red or blue corresponding to OFF type. While green and red represents lower, yellow and blue represent higher threshold levels. The device comes with its own driver circuitry (including an AVR Atmel Mega328P microcontroller and three 8-bit shift registers), which is mounted on the back of the matrix. The default firmware takes input via the SPI interface.

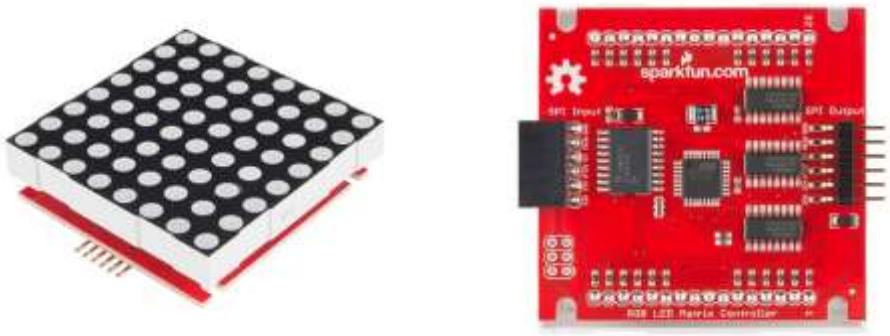


Figure III-5: SparkFun LED-matrix[35]

2. Setup: Audio system

The sensory to audio SSD system works as an addition to the first setup, this way the tones can be heard, while the output can still be seen on the LED-matrix. The additional system consist of a fixed 8 channel audio input, a volume controller unit, and a pair of stereo headphones. It communicates with the microcontroller through an SPI interface, using the SPI1 port of the developer board. In order to keep the system asynchronous, instead of daisy chaining, each volume controller chip have a different chip select pin.

Each audio channel plays one constant tone. With the help of the 4 (stereo) volume controller chips, the volume of each tones can be controlled simultaneously and independently. Correspondingly to their orientations (left/right), the outputs are summed up, and the resulting stereo output is connected to a stereo headphone. The block diagram of the new system can be seen on the figure below.

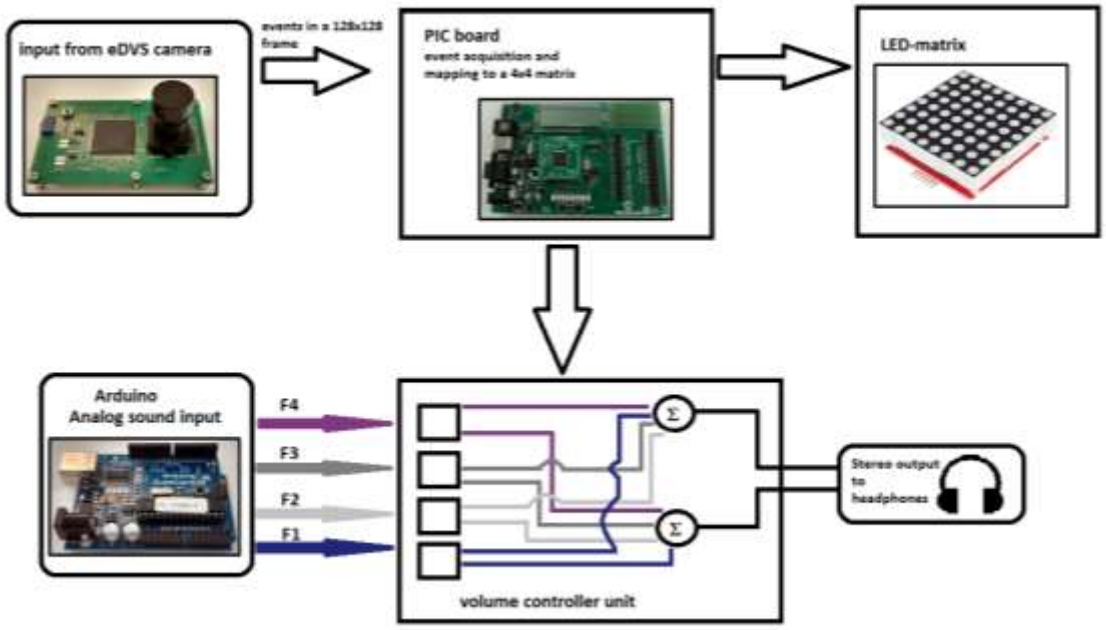


Figure III-6 block diagram of the second setup

AUDIO CHANNELS

The 8 fixed audio input were created using Pulse Width Modulated (PWM) signals with different frequencies and duty cycles. 6 inputs are supplied from the PWM outputs of an Arduino Duemilanove [36], and two of them are supplied from the PWM outputs of the PIC development board. This arrangement was necessary, as the Arduino board only has 6 PWM output pins. In the current solution two sets of 4 tones are created (representing four frequency levels, in both ears). Ideally these tones would be sine waves with frequencies of the musical tone A2, A3, A4, A5 and C3, C4, C5, C6, or any other predefined musical tones. Unfortunately, at the current implementation, even though it is possible to interpret the sounds, they are less “enjoyable”, as they are just 4 pairs of square waves ranging from low to higher pitches. Future

implementations might include Programmable Sound Generator (PSG)[37] circuits instead of the PWM signals. Pin arrangement of all the 8 PWM inputs can be seen in Table 2. Frequency levels are interpreted in a way, that for example F0R represents the right ear input for the lowest frequency.

Device	Frequency level	Pin
Arduino	F0 R	PWM 3
	F2 R	PWM 5
	F3 R	PWM 6
	F1L	PWM 9
	F0 L	PWM 10
	F1 R	PWM 11
PIC	F3 L	RC1
	F2 L	RC2

Table 2: Pin arrangement of PWM inputs

VOLUME CONTROLLER UNIT

In order to independently and simultaneously control all the 8 audio channels, four PGA2311[38] audio amplifiers were used. Each audio amplifier has two channells, representing the left and the right ears, and their volumes can be set independently through the SPI interface. The value 0 mutes the system, while the value 255 represents the loudest possible output. After the volume levels are set, two summing amplifiers sums the different frequencies together, and outputs the result to a stereo headphone. The block diagram of the unit can be seen below, and the connection diagram of the whole circuit is attached in the appendix.

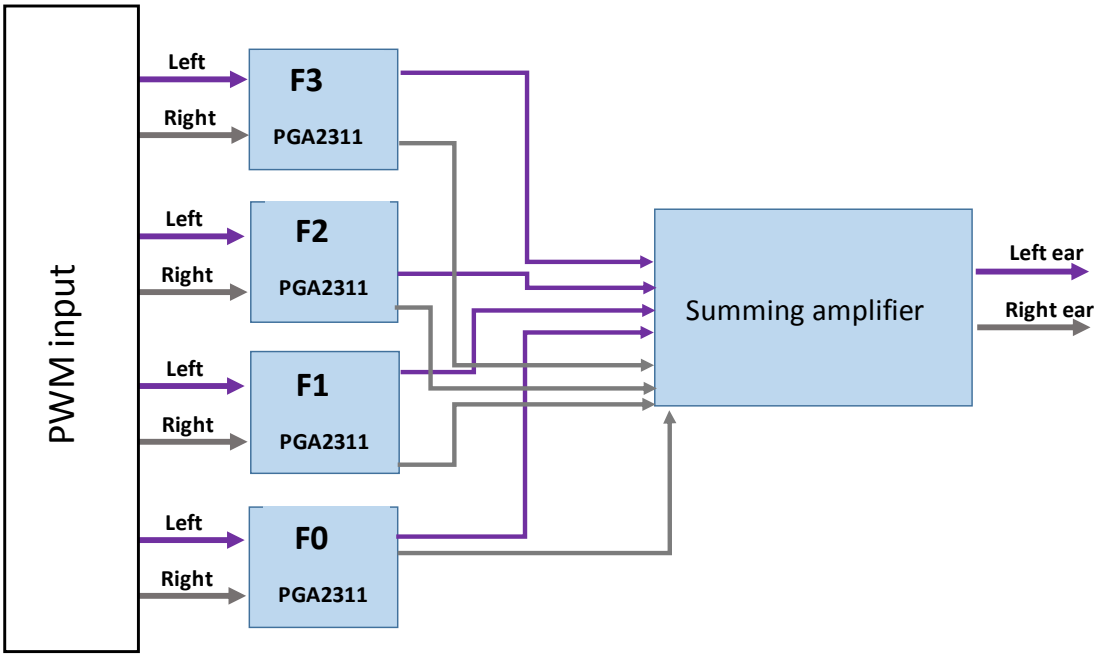
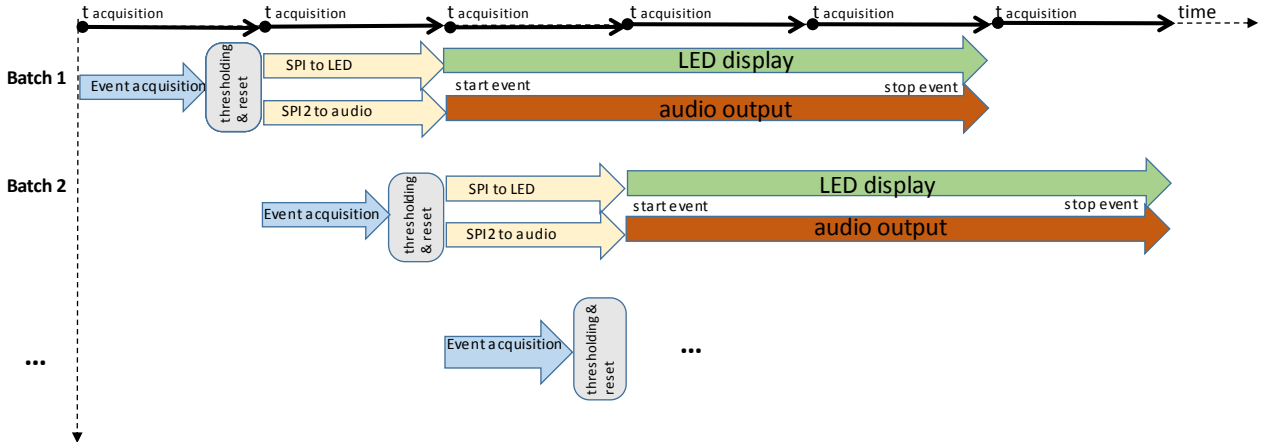


Figure III-7: Block diagram of the volume controller unit

PIC18 BOARD

In order to maintain the asynchronous behaviour of the system, it is necessary to keep the event acquisition period as low as possible, while it is also important to find an optimal length for each tone that would allow the user to interpret it. In order to meet both of these requirements, each activated tone remains active for four acquisition periods, and if another volume weight in the same frequency level gets activated, the new one is simply added to the old one. The new processing pipeline can be seen on the following figure. The complete pin arrangement can be seen on Table 3.



III-8: Processing pipeline of Setup 2

Connection	Pin	Purpose
DC input	VIN	+5V input from the custom power supply
	GND	Ground from the custom power supply
eDVS	+5V DC input	Vcc
	RG1	RxD
	RG2	TxD
	GND	GND
Volume controller unit	RA0	/MUTE
	RC5	SDI (data from microcontroller)
	RC3	SCLK
	RA3	/CS for F1
	RH3	/CS for F2
	RH5	/CS for F3
	RH7	/CS for F4
LED-matrix	+5V DC input	Vcc
	RD6	SCLK
	RD1	/CS
	RD4	MOSI (data from microcontroller)
	GND	GND
Audio input	RC1	PWM input to F1 right side
	RC2	PWM input to F2 left side

Table 3: Pin arrangement of Setup 2

POWERING UP THE SYSTEM

As all the amplifiers require dual power supply of -5V and +5V, powering up the system requires a relatively complicated setup. A 12V power supply is applied as the only power source. This 12V is divided with a voltage divider consisting two 100 kOhm resistors. The middle level (6V) is used as a virtual ground, while 12V is used as the positive side, and the ground is used as the negative side. An operational amplifier is applied as a buffer circuit, and its output is fed to two regulators. The LM7805 regulator takes the +6V difference as an input, and outputs +5V, while LM7905 regulates from -6V to -5V. The PIC microcontroller, the Arduino, the LED-matrix, the eDVS, and the positive side of the amplifiers are all supplied from the +5V side of the circuit, while only the negative side of the amplifiers are supplied from the -5V side.

In everyday usage there are only two possible setups for the system 1) when the camera is attached and it works as a standalone system, 2) when the camera is detached, and the input comes from Matlab to test certain functionalities. When the camera is attached to the system, the positive side draws 0.23 A, which means 1.15 W power consumption, when it is detached, the system draws 0.1A, which means 0.5W power consumption. On the other hand the negative side draws only 4mA in both cases. This imbalanced loading caused dysfunction in the voltage divider, which caused dysfunction in the whole system. In order to roughly balance the system out, an appropriate resistor needed to be parallel connected on the negative side. For case 2) 2 pieces of 120 Ohm resistors were parallel connected, and for case 1) another 3 of them can be added with a switch. (see Figure III-9) The reason why several parallel resistors were applied instead of one with a smaller resistance, was to avoid overheating. It is important to point out that these resistors are not balancing the system perfectly, but they bring both voltages into an operational range, which is adequate for this current proof-of concept system.

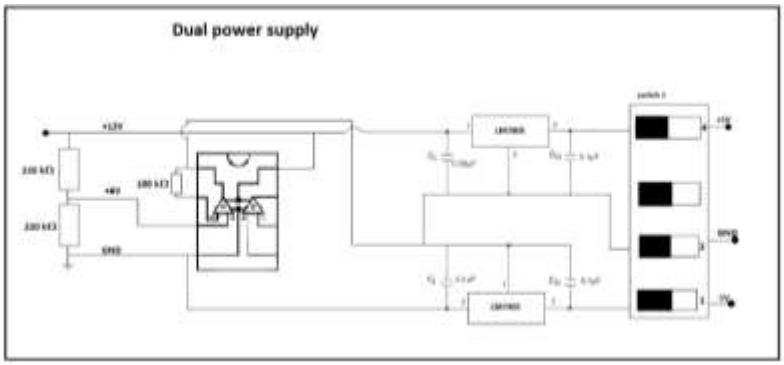


Figure III-9: Dual power supply regulating circuit

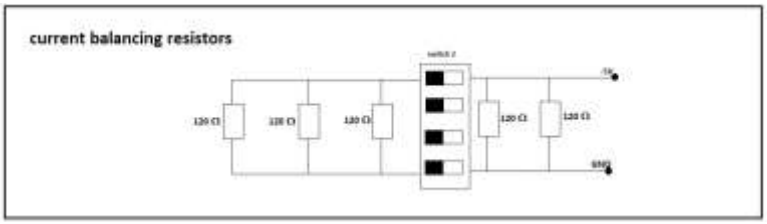


Figure III-10: Connection diagram of the current balancing resistors

3. Setup: SounDVS Android application

Apart from the previously described hardware setup, a more robust, flexible and cheap solution was created as well. The application is based on the open source, publicly available AndroideDVS android application[39], and it was named SounDVS. The physical connection between the camera and the phone is carried out through connecting a mini-USB B to USB-B cable with an USB A to micro-USB B.

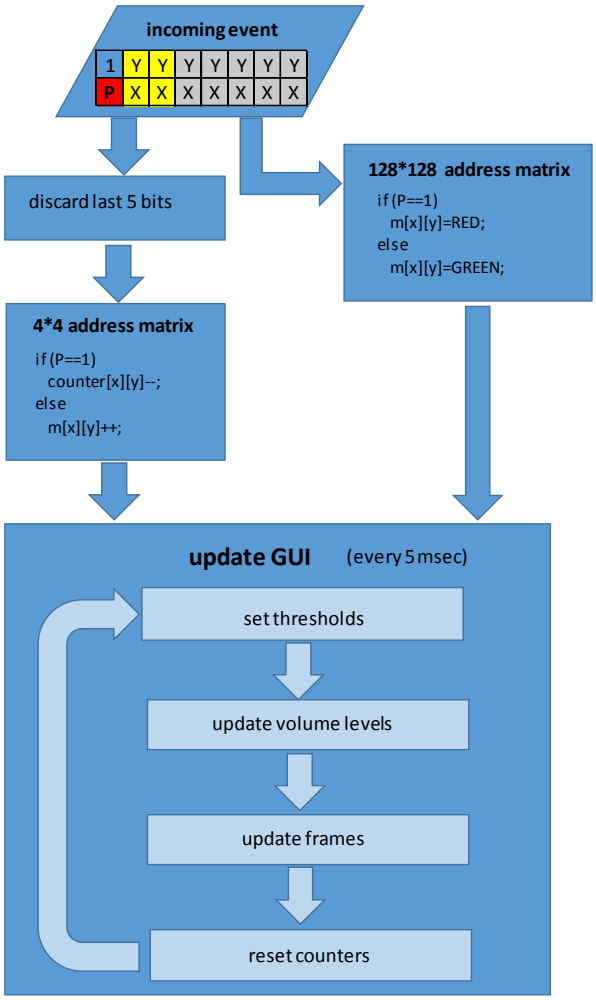
III-11: Setup 3: SounDVS android application

ANDROIDEDVS

The original AndroideDVS application takes input through the microUSB port using Java libraries provided by FTDI. When both bytes of an address arrive, the event handler function adds RGB values of red or green (depending on the polarity of the event) to an array corresponding to a 128*128 address matrix. In every 40 millisecond, the GUI is updated based on this matrix, and all the colours in the matrix are reset to black.

NEW PROCESSING PIPELINE

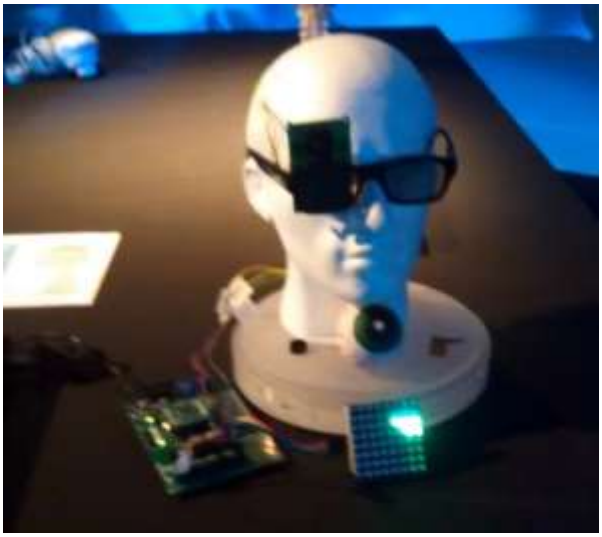
In general, the new application adds the two previously described algorithms (image processing, and image-to sound mapping) to the existing application. It outputs on the screen the 128*128 pixels address-event output from the eDVS, as well as the 4*4 result of the processing algorithm, and the stereo sound output. Similarly to the second setup, the audio inputs are constant (four stereo audio files, that are played continuously in a loop), and the algorithm is only changing their volume levels based on the activation pattern. The flowchart of the processing algorithm can be seen on the figure below.



III-12: SounDVS algorithm

IV. RESULTS

During this thesis project, the two goals, to create a front end of a retinal implant, and to create a sensory substitution system were both carried out successfully. All three setups can be seen on the attached YouTube video[40]. In March 2015, the first setup was exhibited on the "You have been upgraded-Human Enhancement Festival" in the Science Museum in London (see Figure IV-1).



IV-1: Setup presented in the Science Museum exhibition

Testing

In order to test the system, a number of simple visual inputs were created, and they were compared with the output of the camera, with the output on the LED matrix, and later with the resulting sounds. However, due to lack of time, the second and the third setup was not tested as extensively as the first one. The simple stimulation patterns were white shapes on a black background. All of them were created with MATLAB™, and displayed on a 21.5-inch LED screen of a PC. As an initial step, the DVS128 and the eDVS output was visualised and compared with the open source jAERViewer software (Java Address Event Representation Viewer)[33].

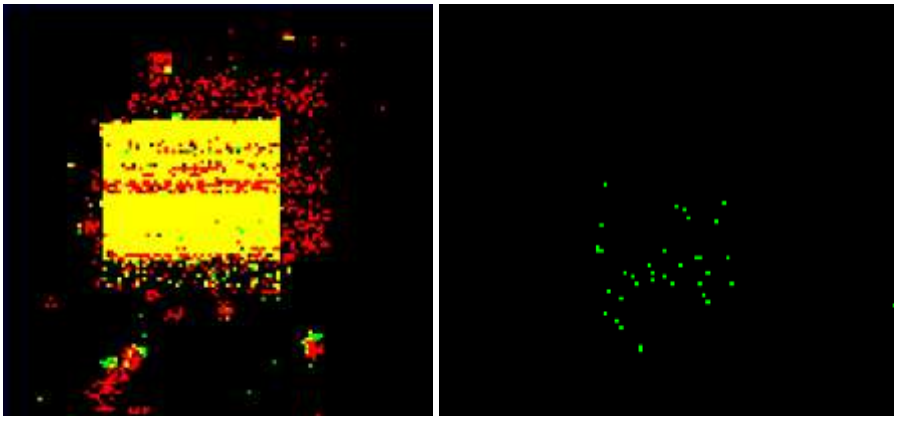


Figure IV-2 a&b: DVS128 output with 4Mbit/s baud rate vs. eDVS output with 115200 bit/s baud rate

Due to the reduced baud rate, there were remarkably less events received from the eDVS, than from the DVS128 (see Figure IV-3). We note this difference, however this did not make the LED output less comprehensible. For the DVS we used a higher sensitivity– hence the denser appearance of events. With a lower sensitivity for the DVS we would see only the edges of the objects. On the eDVS we used only ON events (green dots in jAER representation).

In every case we have tested several different scenarios: the pattern object was static, blinking (changing colour from black to white), moving, or shaking (small movements). As a general result, we have found that directions of movement were easily recognizable in every case. Shaking objects were easier to recognise and produced less noise than in the case of blinking and static patterns. In the case of static objects, after a short period of time, the pattern was not recognisable on the output, only occasional noise was visible. This is expected behaviour since we use a DVS camera, which registers temporal contrasts.

Figure IV-3: Test setup

On the following figures the stimulation pattern, and the resulting LED output patterns are shown for a selection of input patterns. The pictures were not recorded simultaneously, thus the stimulation, and the LED output picture slightly differ in each case.

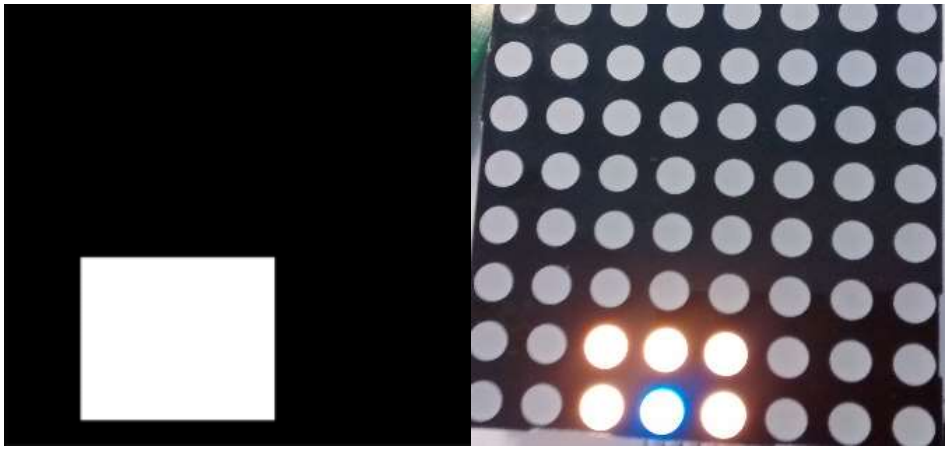


Figure IV-4 a&b: Matlab stimulation of a moving box, and the result on the LED matrix

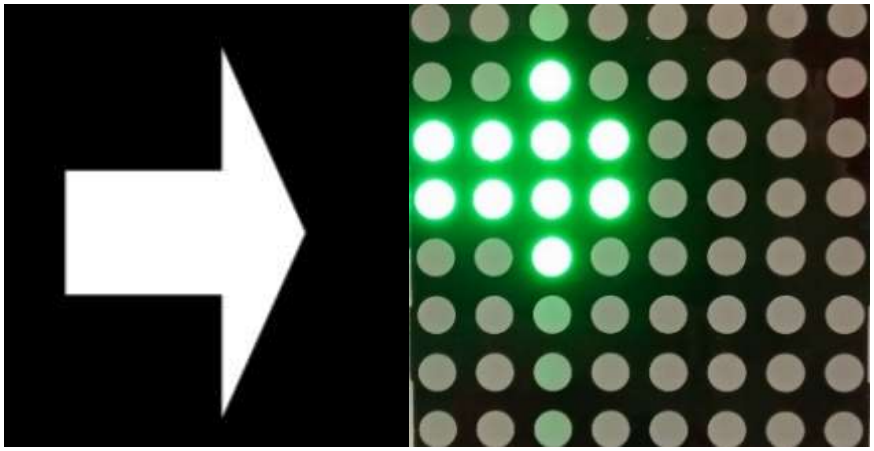


Figure IV-5 a&b: an arrow moving from left to right

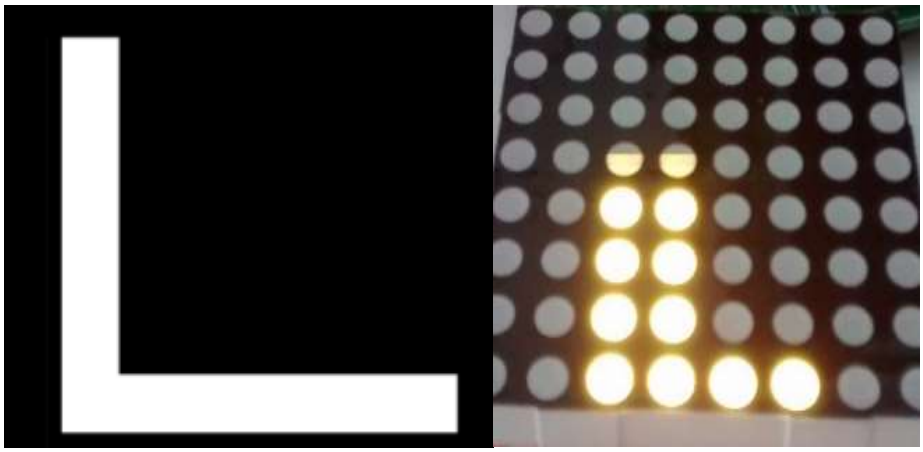


Figure IV-6 a&b: A blinking letter L

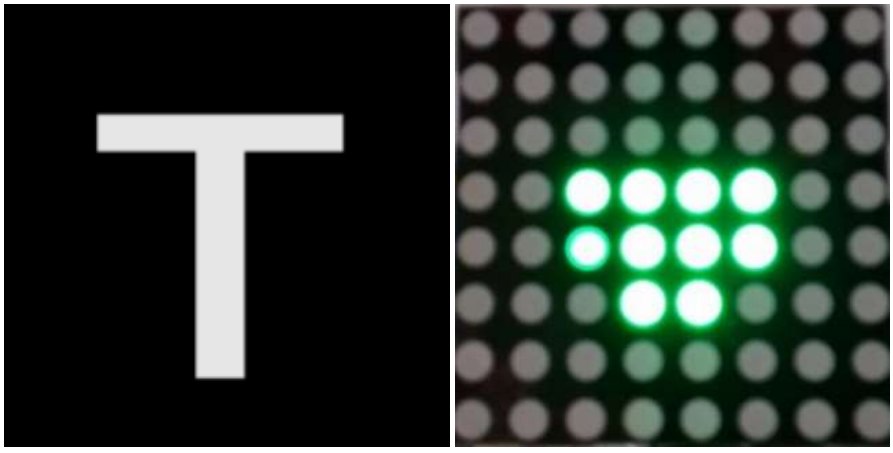


Figure IV-7 a&b: A blinking letter T

V. DISCUSSION

Using a DVS camera in any visual prosthesis system enables many new design choices, and creates opportunities for several new approaches in development. Both the front-end and the SSD system benefits from its neuromorphic behaviour, but in a slightly different way. On the other hand both systems have certain disadvantages deriving from this setup.

Advantages and disadvantages of the front-end system

***Advantages**:* Primarily, by only responding to temporal contrasts, the power consumption of the system is reduced (to 23 mW) and thereby confers several important properties for a retinal prosthesis or a vision augmentation system. It has the potential to create a much more efficient (longer time between recharging), safer and more accurate system for the restoration of vision, than any of the already available solutions.

Furthermore, one of the main barriers to improving resolution for retinal implants stems from the risk of damage to the healthy cells of the retina, through receiving too much current from the implant's electrode array. An event based-system reduces this risk through only injecting current upon event detection. For example, static images with large, uninformative bright areas will not create constant stimulation. Conversely, for a fixed risk level (current envelope) larger electrode arrays can be implanted since they will be active for less of the time. Additionally, when applied for retinal implants, not only the risk of damage to the eye is reduced, but the outputs of such systems are also more likely to be properly processed and interpreted by downstream neural pathways. This should make any rehabilitation and training period needed for learning how to use such a DVS-based implant swifter and less difficult.

Apart from retinal implants, with a specialised electrode array, the system can be used as a front-end of an Optogenetic Prosthesis, or of a visual augmentation system that could be used by visually impaired, but not blind users, as part of a "smart glass".

Disadvantages: Due to the photoreceptor circuit, the eDVS records some noise, which cannot be entirely eliminated through the integrate-and-fire neuron model. This sometimes leads to erroneous spikes. Additionally, due to the constraints on the acquisition time, only a very simple integrate-and-fire model can be implemented on this system, highlighting the event-processing algorithm as an important area for improvement in future work. However, in general, most of the encountered limitations can be easily rectified with further development in terms of larger memory chips, a faster processing unit, and with more accurate neuromorphic models.

Advantages and disadvantages of the SSD system compared to existing SSD's

***Advantages**:* While there are many existing solutions for visual to auditory sensory substitution, this system represents an entirely novel approach, with taking advantage of the features provided by the DVS camera: real-time processing, and asynchronous behaviour. Thanks to these features, it is it possible for users to track movements, as the user can actually here if an object moves from left to right, or from bottom to top. Also the simple algorithm makes it possible for users to interpret a simple output with practically no training period. More complex scenes would still require training, but it would still be significantly shorter.

Disadvantages: Blind people heavily rely on their intact senses, especially on their hearing. Using an SSD system would make users to at least partially give up on this sense, as it requires headphones. Therefore it would only be advisable for a blind person to use an SSD system if it leads to significant improvements in their performance. At the current state of this project, only directions of movements can be recognised, and only on a 4x4 address matrix. Also further research would be required to determine those volume levels and those frequency levels that are the easiest to be distinguished and that leads to optimal user experience. Conclusively, the system in its current stage does not give enough information for the users to replace existing SSD systems, but it represents a promising new approach. Further development would be necessary to increase resolution, and for improving user experience.

Standalone Hardware solution vs. Android application

Both the second and the third setup have certain advantages over the other in different cases. A standalone hardware system would make it possible to integrate the whole system (camera, processing unit, and stimulator) into the glasses, making it very simple, comfortable, and easy to use for blind users. Additionally it would not depend on the phone's battery, and it would not interfere with other applications on the device. On the other hand, a hardware system is less flexible, as it is more complicated to increase resolution, or change settings. Also, as each element of it needs to be purchased individually, it would be more expensive, and each change would result in additional costs. On the contrary, each smartphone contains high performance CPU's, and are able to generate the required sounds. It is a flexible, scalable, and cheap solution. However in everyday usage it would cause issues because of the short battery life, because it would disable all other applications, and because it would require a long cable running from the camera to the phone, if the phone is placed in the user's pocket.

Conclusively, an android application seems to be a good solution for creating a proof-of concept prototype, and to develop it further, until the point where most of the usability issues are solved, and the system performs as required. However in long term perspective, after the extensive testing and optimisation on the phone is finished, a device built into the glasses would benefit the user more.

VI. CONCLUSION

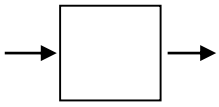
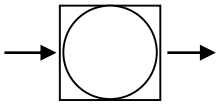
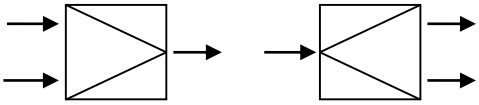
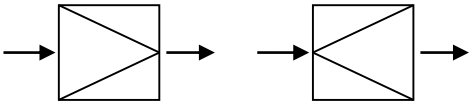
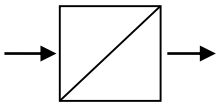
During this project a proof-of-concept for a low-power, real time front end system for retinal implant and a novel sensory substitution system was demonstrated. Both systems uses neuromorphic hardware with several desirable retinomorphic properties, and both systems are relatively easy to use and to interpret.

The proposed front-end system can also be a front end of several other devices too, such as optogenetic prosthetics, or visual augmentation systems. Even though both systems in their current stages are relatively simple, many further developments are possible, such as implementing other neuromorphic models, and increasing resolution.

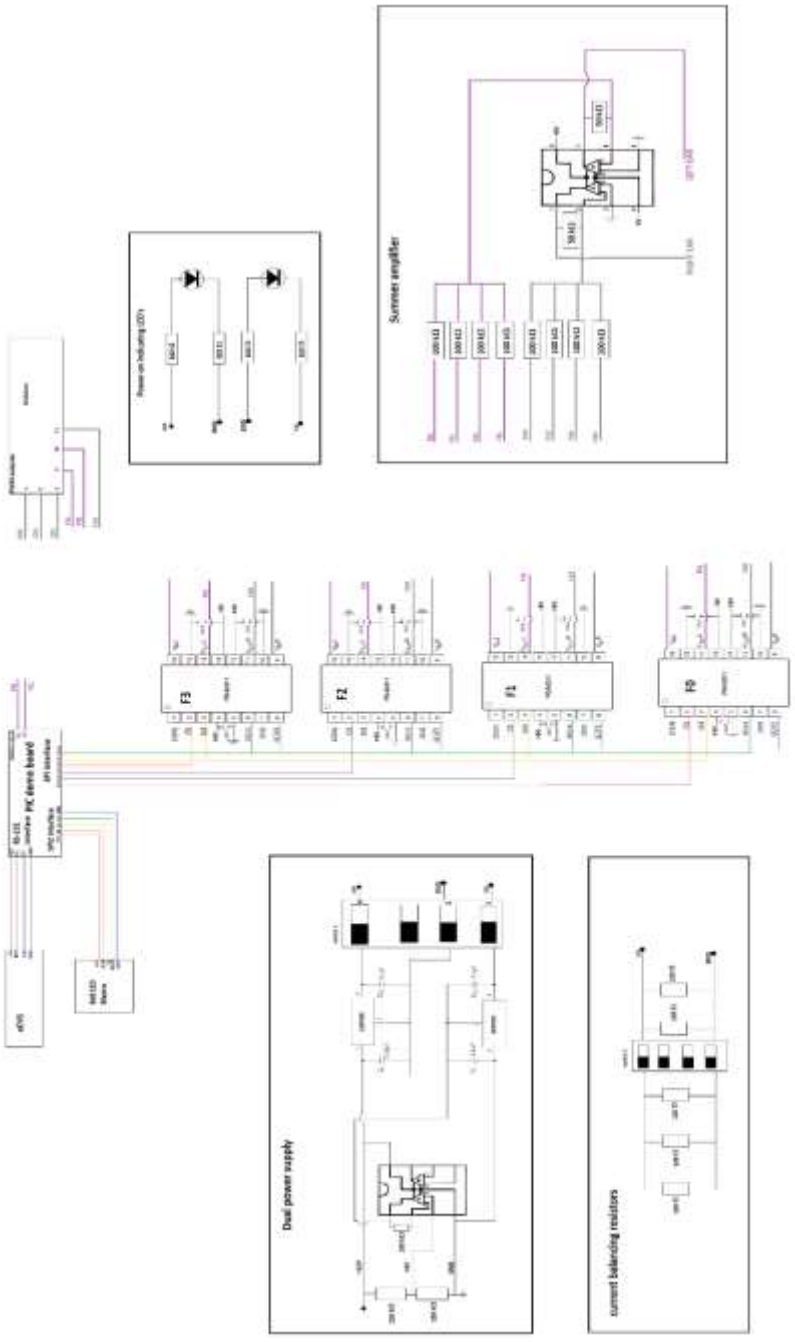
The sensory substitution system represents a novel approach in comparison to other visual to audio SSD's, as it works in real-time, and asynchronously, making it possible for users to track movements. Even though the current 4x4 resolution might not be informative enough to effectively improve visual understanding, with further development in terms of resolution and usability, it could be a promising approach.

APPENDIX

Functional analysis signs

Function	Verbal meaning	Symbol
Transport	Transport, conduct, move, pump, relay	
Store	Store, keep, hold, memorise	
Connect (and separate)	Add, print, mix, connect, stir (cut, distil, scrape, read, saw, distribute)	
Transform	Flatten, grind, parse, translate, step-down, adapt	
Convert	Drive, power, time, use, control, generate, convert, burn	

Volume controller system schematics



PIC source codes

functions for communication with the camera (edvs.c)

```
#use delay(clock=8000000)
//setup the UART connection between PC and micro-controller xmit: PIN_G1,
rcv: PIN_G2
#use rs232(baud=115200, UART2, parity=N, bits=8)

void camera_start(){
//  puts("!L1\n"); // to test if the communication works
    puts("E+\n");
}
void camera_stop(){
    puts("E-\n");
}
```

functions for communication with the LED matrix (led.c)

```
//SPI pins that are used:      MOSI: PIN_D4, SCLK: PIN_D6, CS: PIN_D1
#define LED_CS PIN_D1
#define MATRIX_SIZE 8
//colour codes
#define RED 0xE0
#define GREEN 0x1C
#define BLACK 0x00
#define YELLOW 0xF0
#define PINK 0xE3
#define ORANGE 0xFC
#define WHITE 0xFF
#define BLUE 0x0F

void led_matrix_reset(){
    output_low(LED_CS);
    delay_us(500);
    spi_write2(0x26); //sending reset byte
    output_high(LED_CS);
}
//sending out a reset (1 byte), and then a whole frame (64 byte)
void led_matrix_frame(char frame[]){
    int i=0;
    output_low(LED_CS);
        delay_us(500);
        spi_write2(0x26);
        for (i=0;i<MATRIX_SIZE*MATRIX_SIZE;i++)
         spi_write2(frame[i]);
    output_high(LED_CS);
}
/*initialisation function:
    1) Sets up SPI communication
    2) Turns on [0,0] as pink, [0,1] as orange, and [1,0] as white-->
Initialises directions
    3) Stops, and leaves the LEDs on for 1 second */
```

```
void led_matrix_init(){
    int m_size=64;
    int i=0;
    char led_colors[MATRIX_SIZE*MATRIX_SIZE];
    //setup spi2 connection
    setup_spi2(SPI_MASTER|SPI_L_TO_H|SPI_CLK_DIV_64|SPI_XMIT_L_TO_H);
    led_matrix_reset();
    delay_ms(200);

    for (i=0;i<m_size;i++)
        led_colors[i]=BLACK;

    led_colors[0]=PINK; // pink
    led_colors[1]=ORANGE; //orange
    led_colors[8]=WHITE; //white
    led_matrix_frame(led_colors);
    delay_ms(1000);
}

// test function that sends a smiley face on the LED matrix
void smiley(){
    int i=0;
    char led_colors[64];
    for (i=0;i<64;i++)
        led_colors[i]=YELLOW;
    led_colors[10]=BLACK;
    led_colors[18]=BLACK;
    led_colors[26]=BLACK;
    led_colors[13]=BLACK;
    led_colors[21]=BLACK;
    led_colors[29]=BLACK;
    led_colors[46]=BLACK;
    led_colors[53]=BLACK;
    led_colors[52]=BLACK;
    led_colors[51]=BLACK;
    led_colors[50]=BLACK;
    led_colors[41]=BLACK;
    led_matrix_frame(led_colors);
}
```

functions for communication with the audio system (AE_audio.c)

```
#use delay(clock=8000000)

//sclk: RC3, data: RC5 --> common for all
#define CS_1 PIN_A3    //chip select for F0
#define CS_2 PIN_H3    //chip select for F1
#define CS_3 PIN_H5    //chip select for F2
#define CS_4 PIN_H7    //chip select for F3
#define MUTE PIN_A0    //hardware mute: common
#define CHANNELS 4

int8 volumes[8]={0, 120, 130, 140, 150, 170, 190, 220};

//software mute function
void mute_all(){
    output_low(CS_1);
    output_low(CS_2);
    output_low(CS_3);
    output_low(CS_4);
    spi_write(0);
    spi_write(0);
    output_high(CS_1);
    output_high(CS_2);
    output_high(CS_3);
    output_high(CS_4);
}
/* changes the volume of one of the four volume controllers, y: choose
controller, left/right: volume level at the left/right ear-->number between 0
and 7!*/
void write_channel(int y, int left, int right){
    switch (y){
        case 0:
        output_low(CS_1);
            spi_write(volumes[left]);
            spi_write(volumes[right]);
        output_high(CS_1);
        break;
        case 1:
        output_low(CS_2);
            spi_write(volumes[left]);
            spi_write(volumes[right]);
        output_high(CS_2);
        break;
        case 2:
        output_low(CS_3);
            spi_write(volumes[left]);
            spi_write(volumes[right]);
        output_high(CS_3);
        break;
        case 3:
        output_low(CS_4);
            spi_write(volumes[left]);
```

```
            spi_write(volumes[right]);
        output_high(CS_4);
        break;
    }
}
//hardware mute function
void mute_hardware(int time){
    output_low(CS_1);
    output_low(CS_2);
    output_low(CS_3);
    output_low(CS_4);
    output_low(MUTE);
    delay_ms(time);
    output_high(MUTE);
    output_high(CS_1);
    output_high(CS_2);
    output_high(CS_3);
    output_high(CS_4);
}
//activates all sounds (after each other) for half second
void test_all(){
    int i=0;
    for (i=0; i<CHANNELS; i++){
        write_channel(i, 6, 0);
        delay_ms(500);
        write_channel(i, 0, 6);
        delay_ms(500);
        write_channel(i, 0, 0);
    }
    mute_all();
}
/* initializes the two PWM signals for F1R, F0L initializes the SPI
communication to the audio system sets all /CS and /MUTE signals to high*/
void audio_init(){
    //PWM initialisation
    output_low(PIN_C2);   // Set CCP2 output low
    output_low(PIN_C1);   // Set CCP1 output low
    setup_ccp1(CCP_PWM);  // Configure CCP1 as a PWM ------F1R
    setup_ccp2(CCP_PWM);  // Configure CCP2 as a PWM ------F0L
    setup_timer_2(T2_DIV_BY_16, 255, 1);
    set_pwm2_duty(30);
    set_pwm1_duty(130);
    //SPI initialization
    setup_spi(SPI_MASTER|SPI_L_TO_H|SPI_XMIT_L_TO_H|SPI_CLK_DIV_16|SPI_SCK_IDL
E_LOW);
    output_high(MUTE);
    output_high(CS_1);
    output_high(CS_2);
    output_high(CS_3);
    output_high(CS_4);
    test_all();
}
```

Main PIC file for setup 1 (front_end.c)

```
#include <18F8722.h>
#include <stdlib.h>
#include "edvs.c" // eDVS control functions
#include "led.c"  // 8x8 LED matrix functions
#fuses INTRC, NOPROTECT // select internal oscillator

#define BTN1 (PIN_B0)
#define BTN2 (PIN_A5)

//threshold levels
int8 threshold[5]={1, 8, 10, 12, 15};
//corresponding colors for positive thresholds
char colors[5]={0x1C, 0xF0, 0xE3, 0xFC, 0xFF};

// initialising the union Events
union Events
{
    char twoBytes[2];
    struct{     // setting up bit field
         unsigned int Ylower : 4;   // bit field reads LSB first!!!!
         unsigned int Yhigher : 3;
         unsigned int Yflag : 1;
         unsigned int Xlower : 4;
         unsigned int Xhigher : 3;
          unsigned int Intensity : 1;
    }bitEvents;
};

//----------global variables--------------------

int1 timer_flag=0;      // 1 when there is a timer overflow
int1 x_byte=0;   //flag indicating if the NEXT byte is going to be an x addr.
union Events myEvents;          // instantiation of the Events union
int8 threshold_counter=0;       // counter to indicate which threshold level is
used (increased at button_interrupt)
signed int led_count[8][8] = {0};// 8*8 counter array for the incoming events

//----------functions ----------------------------
void simple_coordinates(union Events e);
void timer_init();
void output_matrix();

//------------------- interrupts--------------------
//button interrupt to circulate the threshold level
#INT_EXT
void button_interrupt(){
    if (threshold_counter<4)
        threshold_counter++;
    else
        threshold_counter=0;
}
```

```
// timer interrupt
#int_timer1
void timer1_isr() {
    timer_flag=1;
}

// RS232 incoming byte interrupt
#int_rda
void serial_isr() {
    output_high(PIN_D0);
    myEvents.twoBytes[x_byte]= getc();//saves the byte to 0 or 1 location
    if(!x_byte)//If it's a y byte: We need to check if the stream is aligned
    {
        //if it's a valid y event: let's it pass through, next one will be an
        x, so x_byte=1
        //if not: next one will be a y again, so x_byte=0
        x_byte=myEvents.bitEvents.Yflag;
    }
    else//if this is the x_byte: We have received the whole event, it can be
processed
    {
        simple_coordinates(myEvents);
        x_byte=0;
    }
}

//timer initialisation
void timer_init(){
    enable_interrupts(int_timer1);
    setup_timer_1(T1_INTERNAL | T1_DIV_BY_1);
    set_timer1(0);
}
//function to process an event-->discards last 4 bits, add/deduce intensity
to/from the counter
void simple_coordinates(union Events e){
    signed int intensity;
    intensity=(myEvents.bitEvents.Intensity==0)?1:-1; // check if ON or OFF
event  ON: 0, OFF: 1
    led_count[e.bitEvents.Yhigher][e.bitEvents.Xhigher]=led_count[e.bitEvents.
Yhigher][e.bitEvents.Xhigher]+intensity;
}

//outputs the whole matrix
void output_matrix(){
    int i=0;   //y address
    int j=0;   //x address
    int matrix[MATRIX_SIZE*MATRIX_SIZE]; //going to be the outpu
        for (i=0; i<MATRIX_SIZE; i++){
            for (j=0; j<MATRIX_SIZE; j++){
            //default: black color
            matrix[i*MATRIX_SIZE+j] = BLACK;
```

```
            //thresholding
            if (led_count[i][j] > threshold[threshold_counter])
                matrix[i*MATRIX_SIZE+j] =colors[threshold_counter];
            if (led_count[i][j] < (-1*threshold[threshold_counter]))
                matrix[i*MATRIX_SIZE+j] = BLUE;
            //resets the counter
            led_count[i][j]=0;
         }
        }
        //send the frame to the LED_matrix
        led_matrix_frame(matrix);
}

void main(){
    //enabling interrupt functions
    //if running from camera: rda2, if running from matlab: rda
    enable_interrupts(int_rda);
    enable_interrupts(global);
    enable_interrupts(int_ext);
    setup_oscillator(OSC_8MHZ|OSC_INTRC);
    //initialising
    timer_init();
    led_matrix_init();
    smiley();
    camera_start(); //send command to eDVS to start streaming

    while (true){
            if (timer_flag){
                output_matrix();
                timer_flag=0;
            }
    }
}
```

Main PIC function for setup 2 (audio_SSD.c)

```
#include <18F8722.h>
#include <stdlib.h>
#include "AE_audio.c"          // functions controlling the audio system
#include "led.c"               // functions controlling the led matrix
#include "edvs.c"              // functions controlling the camera
#fuses INTRC, NOPROTECT        // select internal oscillator

#define TIMER_START 4          //defines how long will an event last
#define THRESHOLD 1            //defines threshold for RGC's

//constants volume weight mapping
const int left[4]={0, 1, 2, 4};
const int right[4]={4, 2, 1, 0};

// initialising the union Events
union Events
{
    char twoBytes[2];
    struct{                          // setting up bit field
        unsigned int Ylower : 5;   // bit field reads LSB first!!!!
        unsigned int Yhigher : 2;
        unsigned int Yflag : 1;
        unsigned int Xlower : 5;
        unsigned int Xhigher : 2;
        unsigned int Intensity : 1;
    }bitEvents;
};
//4*2 array for current volume weight levels--- [][0]: left---[][1]:right
int8 frequencies[4][2]={};

//both of the arrays are organised as [x][y] !!!
int32 led_count[4][4]={};      //4*4 counter array for incoming events
int8 event_timer[4][4]={};     //4*4 counter array for activating/deactivating
RGC's

int1 x_byte=0;    // flag indicating if the NEXT byte is going to be an x
address
union Events myEvents;         // instantiation of the Events union

int16 overflow_counter=0;      //counter for timer2
int1 timer_flag=0;                    //flag for timer 1

//variables for testing
int8 stepy=0;
int1 init_flag=0;
int option_counter=0;
//------------ list of functions ----------------------------------------
// processing functions
    void init();
    void startEvent(int y, int x);
    void stopEvent(int y, int x);
```

```
    void set_timers();
    void timer_init();
    void timer2_init();
//old functions:
    void output_matrix();
    void simple_coordinates(union Events e);
// testing functions
    void all_options_x();//shows all possible activation patterns in the x
direction, goes through all frequencies 1 by 1
    void all_options_sum(); //shows all possible activation patterns in x
direction in all 4 frequencies paralelly direction
    void steps_x();//steps through all the frequencies/volumes 1 by 1
    void smiley_4_4(); // activates a smiley face
    void special(int stepy, int counter);
    void stepper(int stepy, int counter);
    void count_up(int stepy, int counter);
    void count_down(int stepy, int counter);

// ----------------interrupts-----------------------------------------
//RS232 incoming byte interrupt
#int_rda2
void serial_isr(){
    myEvents.twoBytes[x_byte]= getc(); //saves the byte to 0 or 1 location
    if(!x_byte) //If it's a y byte: We need to check if the stream is aligned
    {
        //if it's a valid y event: let's it pass through, next one will be an
        x, so x_byte=1
        //if not: next one will be a y again, so x_byte=0
        x_byte=myEvents.bitEvents.Yflag;
    }
    else//if this is the x_byte: We have received the whole event, it can be
processed
    {
        simple_coordinates(myEvents);
        x_byte=0;
    }
}

//button interrupt: starts running the test functions
#INT_EXT
void button_interrupt(){
    init_flag=~init_flag;
}

//timer 1 interrupt: used in every case to output the new frame
#int_timer1
void timer1_isr() {
    timer_flag=1;
}

//timer 2 interrupt: used for testing outputs
#int_timer2
void timer2_isr() {
```

```
    overflow_counter++;
    if (overflow_counter%256==0){
    if (init_flag)
        steps_x();
    }
}
//--------------------------- functions ------------------------------------
void steps_x(){
    if (option_counter<4){
        stepper(stepy, option_counter);
        option_counter++;
        }
    else{
        option_counter=0;
        if (stepy<3){
        stepy++;
        }
        else{
        stepy=0;
        }
    }
}
void smiley_4_4(){
    event_timer[3][0]=TIMER_START;
    event_timer[3][3]=TIMER_START;
    event_timer[1][0]=TIMER_START;
    event_timer[1][3]=TIMER_START;
    event_timer[0][1]=TIMER_START;
    event_timer[0][2]=TIMER_START;
}

void all_options_x(){
    if (option_counter<18){
        if (option_counter<4){
        stepper(stepy, option_counter);
        }
        else if(option_counter<8){
        count_up(stepy, (option_counter-4));
        }
        else if(option_counter<12){
        count_down(stepy, (option_counter-8));
        }
        else {
        special(stepy, option_counter);
        }
        option_counter++;
    }
    else{
        option_counter=0;
        if (stepy<3){
        stepy++;
        }
        else{
```

```
        stepy=0;
        }
    }
}
void all_options_sum(){
    if (option_counter<18){
        if (option_counter<4){
        stepper(0, option_counter);
        stepper(1, option_counter);
        stepper(2, option_counter);
        stepper(3, option_counter);
        }
        else if(option_counter<8){
        count_up(0, (option_counter-4));
        count_up(1, (option_counter-4));
        count_up(2, (option_counter-4));
        count_up(3, (option_counter-4));
        }
        else if(option_counter<12){
        count_down(0, (option_counter-8));
            count_down(1, (option_counter-8));
            count_down(2, (option_counter-8));
            count_down(3, (option_counter-8));
        }
        else {
        special(0, option_counter);
        special(1, option_counter);
        special(2, option_counter);
        special(3, option_counter);
        }
        option_counter++;
    }
    else{
        option_counter=0;
    }
}
void special(int stepy, int counter){
    switch (counter){
    case 12:
        event_timer[0][stepy]=TIMER_START;
        event_timer[2][stepy]=TIMER_START;
        break;
    case 13:
        event_timer[0][stepy]=TIMER_START;
        event_timer[2][stepy]=TIMER_START;
        event_timer[3][stepy]=TIMER_START;

        break;
    case 14:
        event_timer[0][stepy]=TIMER_START;
        event_timer[3][stepy]=TIMER_START;
        break;
    case 15:
```

```
            event_timer[0][stepy]=TIMER_START;
            event_timer[1][stepy]=TIMER_START;
            event_timer[3][stepy]=TIMER_START;
            break;
        case 16:
            event_timer[1][stepy]=TIMER_START;
            event_timer[2][stepy]=TIMER_START;
            break;
        case 17:
            event_timer[1][stepy]=TIMER_START;
            event_timer[3][stepy]=TIMER_START;
            break;
        }
}
void stepper(int stepy, int counter){
    event_timer[counter][stepy]=TIMER_START;
}
void count_up(int stepy, int counter){
    int i=0;
    for (i=0; i<counter; i++){
        event_timer[i][stepy]=TIMER_START;
    }
    event_timer[counter][stepy]=TIMER_START;
}
void count_down(int stepy, int counter){
    int i=0;
    for (i=0; i<counter; i++){
        event_timer[3-i][stepy]=TIMER_START;
    }
    event_timer[3-counter][stepy]=TIMER_START;
}

//initalising the counters
void init(){
    int i=0;
    int j=0;
    for (i=0; i<4; i++){
        for (j=0; j<2; j++){
         frequencies[i][j]=0;
        }
    }
    for (i=0; i<4; i++){
        for (j=0; j<4; j++){
        event_timer[i][j]=0;
        led_count[i][j]=0;
        }
    }
}
void startEvent(int y, int x){
//  when an RGC is activated: the volume of this event is added to the already
activated volume
    //left ear
    frequencies[y][0]=frequencies[y][0]+left[x];
```

```
    //right ear
    frequencies[y][1]=frequencies[y][1]+right[x];
    write_channel(y, frequencies[y][0], frequencies[y][1]);
}
void stopEvent(int y, int x){
    //  when an RGC is deactivated: the volume of this event is deducted from
the already activated volume
    //left ear
    frequencies[y][0]=frequencies[y][0]-left[x];
    //right ear
    frequencies[y][1]=frequencies[y][1]-right[x];
    write_channel(y, frequencies[y][0], frequencies[y][1]);
}
//iterates down--> if reaches 1: event stops
void set_timers(){
    int x=0;
    int y=0;
    for (x=0; x<4; x++){
        for (y=0; y<4; y++){
        if (event_timer[x][y]==TIMER_START){
            startEvent(y,x);
        }
        if (event_timer[x][y]>1){
                event_timer[x][y]--;
        }
        if (event_timer[x][y]==1){
            stopEvent(y,x);
            event_timer[x][y]=0;
        }
        }
    }
}
void timer_init(){
    enable_interrupts(int_timer1);
    setup_timer_1(T1_INTERNAL | T1_DIV_BY_8);
    set_timer1(0);
}
void timer2_init(){
    enable_interrupts(int_timer2);
    setup_timer_2(T2_DIV_BY_16, 255,1);
    set_timer2(0);
}
//function to process an event-->discards last 5 bits, add/deduce intensity
to/from the counter
void simple_coordinates(union Events e){
    signed int intensity;
    intensity=(myEvents.bitEvents.Intensity==0)?1:-1; //check if ON or OFF
    led_count[e.bitEvents.Xhigher][e.bitEvents.Yhigher]=
        led_count[e.bitEvents.Xhigher][e.bitEvents.Yhigher]+intensity;
}

void output_matrix(){
    int i=0;   //y address
```

```
    int j=0;   //x address
    int matrix[MATRIX_SIZE*MATRIX_SIZE];      //going to be the output
    for (i=0; i<MATRIX_SIZE; i++){
        for (j=0; j<MATRIX_SIZE; j++){
            matrix[i*MATRIX_SIZE+j] = BLACK; //default: all are deactivated
        }
    }
    for (i=0; i<4; i++){
        for (j=0; j<4; j++){
    //thresholding: if it's higher than the threshold level: RGC is activated
            if (led_count[i][j] > THRESHOLD){
                event_timer[i][j]=TIMER_START;
                led_count[i][j]=0;
            }
    // if the RGC is active: corresponding 4 LEDs are yellow (because the 8x8
matrix is used with a 4x4 mapping)
            if (event_timer[i][j] > 1){
                matrix[(i*2)*MATRIX_SIZE+(j*2)] = YELLOW;
                matrix[(i*2)*MATRIX_SIZE+(j*2)+1] = YELLOW;
                matrix[(i*2+1)*MATRIX_SIZE+(j*2)] = YELLOW;
                matrix[(i*2+1)*MATRIX_SIZE+(j*2)+1] = YELLOW;
            }
        }
    }
    led_matrix_frame(matrix); //outputs frame
}
void main()
{
    //enabling interrupts
    enable_interrupts(global);
    enable_interrupts(int_ext);
    enable_interrupts(int_rda2);
    setup_oscillator(OSC_8MHZ|OSC_INTRC);
    //initialising
    init();
    led_matrix_init();
    timer_init();
    audio_init();
    timer2_init();
    //sending command to eDVS to start streaming
    camera_start();
    while (true){
        if(timer_flag){
         timer_flag=0;
         output_matrix(); //output the currently active RGC's
         set_timers(); //iterates down the counters

        }
    }
}
```

Arduino source code

```
void setup() {
    // Pin 3: F3L (highest pitch)
    pinMode(3, OUTPUT);
    //Pin 11: F2L
    pinMode(11, OUTPUT);
    //Pin 10: F3R (highest)
    pinMode(10, OUTPUT);
    ////Pin 9: F2R
    pinMode(9, OUTPUT);
    //Pin 5: F1L
    pinMode(5, OUTPUT);
    //Pin 6: F0R
    pinMode(6, OUTPUT);

//The six pins are controlled by 3 different timers, with different settings
//In order to gain better control over their performance then just setting
the duty cycle with AnalogWrite()

//setting the Timer/Counter Control Registers for each timer
// All of these bit settings are based on the documentation:
http://www.atmel.com/images/doc2545.pdf
    TCCR2A = _BV(COM2A0) | _BV(COM2B1)| _BV(WGM21) | _BV(WGM20);
    TCCR2B = _BV(WGM22) | _BV(CS22)| _BV(CS20);

    TCCR1A = _BV(COM1A0) | _BV(COM1B1)| _BV(WGM10);
    TCCR1B = _BV(WGM13)| _BV(CS11)| _BV(CS10);

    TCCR0A = _BV(COM0A0) | _BV(COM0B1)| _BV(WGM00);
    TCCR0B = _BV(WGM02)| _BV(CS01)| _BV(CS00);

     ICR1=255;
//setting the output compare registers (settings are not precise, but
approximately create sounds in the range that can be interpreted

     OCR2A = 65;    // pin 11: OC2A
     OCR2B= 55;     // pin 3: OC2B
     OCR1A= 90;     // pin 9: OC1A
     OCR1B = 45;    // pin 10: OC1B
     OCR0A=220;     // pin 6: OC0A
     OCR0B = 180;   // pin 5: OC0B
}
void loop() {}
```

Most important functions for SounDVS

(MainActivity.java)

```
package carl.abr.edvs;

import java.util.ArrayList;
import android.app.Activity;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Color;
import android.media.MediaPlayer;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.view.Gravity;
import android.view.View;
import android.widget.Button;
import android.widget.FrameLayout;
import android.widget.ImageView;
import android.widget.TextView;
import android.widget.Toast;
import carl.abr.edvs.EDVS4337SerialUsbStreamProcessor.EDVS4337Event;

import com.ftdi.j2xx.D2xxManager;
import com.ftdi.j2xx.FT_Device;

public class MainActivity extends Activity
{
    //---------------- not edited parts ---------------------------------//
    static final String TAG = "main activity";
    public static D2xxManager ftD2xx = null; /** ftdi manager*/
    FT_Device ftDev = null; /** ftdi device plugged to phone*/
    public Context global_context;     /** context of activity*/
    byte stopBit = D2xxManager.FT_STOP_BITS_1;
    byte dataBit = D2xxManager.FT_DATA_BITS_8;
    byte parity = D2xxManager.FT_PARITY_NONE;
    short flowControl = D2xxManager.FT_FLOW_RTS_CTS;
    final byte XON = 0x11;    /* Resume transmission */
    final byte XOFF = 0x13;    /* Pause transmission */
    int DevCount = -1;
    int currentPortIndex = -1;
    int portIndex = 0;
    boolean bReadTheadEnable = false;
    boolean uart_configured = false;
    Thread_read the_thread;
    Handler handler;
    Runnable runnable;
    ArrayList<EDVS4337Event> the_events;
    //-------------------------------------------------------------------//

    int baudRate = 115200; //4000000;  /* baud rate  921600*/
```

```
//------------------------edited parts--------------------------//
public TextView text;
public ImageView orig_IV;
public ImageView new_IV;
int[] m = new int[128*128];      //128*128 event matrix
int[] counter_m=new int[4*4];    //4*4 event counter array
int[] current_volume=new int[8];//current volume levels (4 pitch*2 ears)
//possible volume levels
static float[] volumes={0.0f, 0.2f, 0.3f, 0.45f, 0.5f, 0.65f, 0.8f, 1f};
int init_counter=100;       //set to 0 at initialization
int sound_counter=0;        //used for counting updates
Button initbutton;
FrameLayout f_small, f_big;         //layouts
TextView textview; //shows current threshold levels
MediaPlayer[] mediaPlayer = new MediaPlayer[4];   //4 paralel audio files

@Override
protected void onCreate(Bundle savedInstanceState)
{
    global_context = this; //get context
    //arranges layout
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    f_small=(FrameLayout) findViewById(R.id.frame_small);
    f_big=(FrameLayout) findViewById(R.id.frame_big);
    orig_IV =(ImageView) findViewById(R.id.imageView_original);
    new_IV =(ImageView) findViewById(R.id.imageView_new);
    initbutton =(Button) findViewById(R.id.init);
    textview=(TextView) findViewById(R.id.text);
    initbutton.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            initbutton.setTextColor(Color.GRAY);
            init_counter=0;
        }
    });
    init_sound();//initialise the audio channels

    try {ftD2xx = D2xxManager.getInstance(this);} //get ftdi manager
    catch (D2xxManager.D2xxException e) {
    Log.e("FTDI_HT", "getInstance fail!!");}
}
//function for adapting threshold levels
protected double[] getThresholds(int[] counter_array){
    double []thresholds=new double[2];
    double max_pos=50.0;    //minimum threshold is 0.9*50=45
    double min_neg=-50.0;
    for (int i=0; i<counter_array.length; i++){
     if (counter_array[i]>=max_pos)
        max_pos=counter_array[i];
     if(counter_array[i]<=min_neg)
        min_neg=counter_array[i];
    }
```

```
        thresholds[0]=0.9*max_pos;
        thresholds[1]=0.9*min_neg;
        return thresholds;
    }
    //updates volume levels
    protected void update_sound()
    {
        int[] small_m=new int[4*4];
        int[] volume_weights={0, 1, 2, 4};
        if (init_counter<=15){
         counter_m[init_counter]=100;
         init_counter++;
         if(init_counter==15)
            initbutton.setTextColor(Color.WHITE);
        }
        double[] thresholds=getThresholds(counter_m);
        double pos_threshold= thresholds[0]; // thresholds[0];
        double neg_threshold=-20;//thresholds[1];//-20;
        for (int j=0; j<4; j++){
         current_volume[j]=0;
         current_volume[j+4]=0;
         for(int i=0; i<4; i++) {
            if (counter_m[j*4+i] > pos_threshold) {
                small_m[j*4+i] = Color.GREEN;
                //left
                current_volume[j]=current_volume[j]+volume_weights[3-i];
                //right
                current_volume[j+4]=current_volume[j+4]+volume_weights[i];
            }
            else
                small_m[j*4+i]= Color.BLACK;
         }
         textview.setText("current threshold: " + pos_threshold + ",  " +
neg_threshold);
         mediaPlayer[j].setVolume(volumes[current_volume[j]],
volumes[current_volume[j + 4]]);
        }
        Bitmap bsmall = Bitmap.createBitmap(small_m, 4, 4,
Bitmap.Config.ARGB_8888);
        Bitmap bsmall2 = Bitmap.createScaledBitmap(bsmall, 500, 500, false);
        new_IV.setImageBitmap(bsmall2);

        for(int i=0; i<counter_m.length; i++){
         counter_m[i]=0;
         small_m[i]= Color.BLACK;
        }
    }
    //updates screen in every 5 ms
    protected void update_gui()
    {
        if(sound_counter%2==0)
        update_sound();
        sound_counter++;
```

```
        Bitmap bm = Bitmap.createBitmap(m, 128, 128, Bitmap.Config.ARGB_8888);
        Bitmap bm2 = Bitmap.createScaledBitmap(bm, 500, 500, false);
        orig_IV.setImageBitmap(bm2);
        for(int i=0; i<m.length; i++){
            m[i] = Color.BLACK;
        }
        handler.postDelayed(runnable, 50);

    }
//discards the lowermost 5 bits from the addresses and add their intensity to
the counter
    public void set_events(ArrayList<EDVS4337Event> ev, String txt)
    {
        if(ev.isEmpty() == false){
            for(int i=0; i<ev.size(); i++){
                EDVS4337Event event = ev.get(i);
                int smallx=event.x>>5;
                int smally=event.y>>5;
                if(event.p == 0){
                 m[128*event.x + event.y] = Color.GREEN;
                 counter_m[4*smallx + smally]++;
                }
                else{
                 m[128*event.x + event.y] = Color.RED;
                 counter_m[4*smallx + smally]--;
                }
            }
        }
    }
    //start playing the looped media files with volume 0
    protected void init_sound(){
        //because Java sets (0,0) to top left corner
        mediaPlayer[3] = MediaPlayer.create(global_context, R.raw.f1);
        mediaPlayer[2] = MediaPlayer.create(global_context, R.raw.f2);
        mediaPlayer[1] = MediaPlayer.create(global_context, R.raw.f3);
        mediaPlayer[0] = MediaPlayer.create(global_context, R.raw.f4);

         for (int i=0; i<4; i++) {
            mediaPlayer[i].setLooping(true);
            mediaPlayer[i].setVolume(0.0f, 0.0f);
            mediaPlayer[i].setOnErrorListener(new MediaPlayer.OnErrorListener()
{
                @Override
                public boolean onError(MediaPlayer mp, int what, int extra) {
                 midToast("error in sound " + what + "!!!!" + "  extra: " +
extra, Toast.LENGTH_SHORT);
                 return false;
                }
            });
         }
    }
    //start streaming every time the camera is connected--called from onResume
    protected void start_stream(){
```

```
        if(ftDev == null || ftDev.isOpen() == false)
        {
        Log.e(TAG,"onResume - reconnect");

        createDeviceList();
        if(DevCount > 0)
        {
            connectFunction();
            setConfig();

            //send command to eDVS to start sending events
            String ss = new String("E+\n");
            byte[] text = ss.getBytes();
            sendData(text.length, text);

            //start thread that will read events
            the_thread = new Thread_read(ftDev, this);
            the_thread.start();
        }
        }
    }
    protected void onResume(){
        super.onResume();
        handler = new Handler();
        runnable = new Runnable() {
            public void run(){
                update_gui();
            }
        };
        handler.post(runnable);
        for (int i=0; i<4; i++) {
            mediaPlayer[i].start();
        }
        start_stream();
    }
    public void onDestroy() {
        if(mediaPlayer != null) {
            for (int i=0; i<4; i++) {
                mediaPlayer[i].release();
            }
            mediaPlayer = null;
        }
        super.onDestroy();
    }
//there were no changes to these functions
    void sendData(int numBytes, byte[] buffer){...}
    public void disconnectFunction(){...}
    void midToast(String str, int showTime) {...}
    void setConfig(){...}
    public void connectFunction(){...}
    public void createDeviceList(){...}
}
```

Activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="${packageName}.${activityClass}" >

    <FrameLayout
        android:id="@+id/frame_small"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:background="#ff000000"
        android:clickable="true"
        android:layout_alignParentStart="true">

        <Button
            android:id="@+id/init"
            android:text="Initialise"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:background="#ff000000"
            android:layout_gravity="center_horizontal"
            android:textColor="#fffde9ff"
            android:textSize="40dp"
            android:layout_marginTop="10dp" />
    </FrameLayout>

    <FrameLayout
        android:id="@+id/frame_big"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_below="@+id/frame_small"
        android:layout_alignParentLeft="true"
        android:background="#6f70dbe8"
        android:layout_alignParentEnd="true"
        android:layout_alignParentStart="false">

        <GridLayout
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_gravity="center|top"
            android:columnCount="1"
            android:rowCount="3">

            <ImageView
                android:id="@+id/imageView_original"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_column="0"
                android:layout_row="2"
                android:layout_gravity="center_horizontal|center_vertical"
```

```
            android:background="#c77f7f7f" />

        <ImageView
            android:id="@+id/imageView_new"
            android:layout_column="0"
            android:layout_row="1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal|center_vertical"
            android:background="#c77f7f7f" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_column="0"
            android:layout_row="0"
            android:layout_gravity="center_horizontal|center_vertical"
            android:textAppearance="?android:attr/textAppearanceLarge"
            android:text="Large Text"
            android:id="@+id/text" />
    </GridLayout>
  </FrameLayout>
</RelativeLayout>
```

REFERENCES

[1] A. Sondhi, "Bionic Eye: Development of a microcontroller interface between an event-based vision sensor and a stimulator," MSc, Electrical and Electronic Engineering Department, Imperial College London, London, UK, 2014.

[2] (1996-2015). *Blindness*. Available: http://www.medicinenet.com/blindness/article.htm

[3] P. Doyle, B. Drohan, W. Ellersick, S. Kelly, O. Mendoza, A. Priplata*, et al.*, "The Retinal Implant Project," ed: MIT Press, 2010.

[4] (posted: January 28, 2011). *What will really cause you to go blind! | Dublin Opticians | Mairead O'Leary Opticians | Blanchardstown Rathmines*. Available: http://www.maireadoleary.ie/blog/what-will-really-cause-you-to-go-blind/

[5] (2011). *Age-related macular degeneration*. Available: http://www.ncbi.nlm.nih.gov/pubmed/

[6] U. N. L. o. Medicine. (2015). *Retinitis pigmentosa*. Available: http://www.ncbi.nlm.nih.gov/pubmed/

[7] O. World Health, "Visual impairment and blindness - Factsheet," ed: World Health Organization, 2012.

[8] D. Hubel, *Eye, Brain, and Vision*, 1988.

[9] V. Dragoi and C. Tsuchitani, "Visual Processing: Eye and Retina," in *Neuroscience Online: An Electronic Textbook for the Neurosciences*, ed: Department of Neurobiology and Anatomy - The University of Texas Medical School at Houston, 2015.

[10] (1999). *PHY 3400 Image Gallery: Vision and the Eye*. Available: http://www.phys.ufl.edu/~avery/course/3400/gallery/gallery_vision.html

[11] R. F. D. Stewart A. Bloomfield, "Rod Vision: Pathways and Processing in the Mammalian Retina," vol. 20, pp. 351–384, May 2001 2001.

[12] "Bipolar Cell Pathways in the Vertebrate Retina," in *Webvision: The Organization of the Retina and Visual System [Internet]*, H. Kolb, E. Fernandez, and R. Nelson, Eds., ed Salt Lake City (UT): University of Utah Health Sciences Center.

[13] D. Heeger, "Perception Lecture Notes: Retinal Ganglion Cells," ed: Department of Psychology, New York University, 2006.

[14] "The Bionic Eye," *The Scientist Magazine,* vol. 28, 1. October 2014 2014.

[15] K. Stingl, K. U. Bartz-Schmidt, D. Besch, A. Braun, A. Bruckmann, F. Gekeler*, et al.*, "Artificial vision with wirelessly powered subretinal electronic implant alpha-IMS," 2013-04-22 2013.

[16] S. Sight. (2014). *The Argus II Retinal Prosthesis System*. Available: http://www.secondsight.com/argus-ii-rps-pr-en.html

[17] L. d. Cruz, B. F. Coley, J. Dorn, F. Merlini, E. Filley, P. Christopher*, et al.*, "The Argus II epiretinal prosthesis system allows letter and word reading and long-term function in patients with profound vision loss," *British Journal of Ophtalmology,* 2013-02-20 2013.

[18] V. Singh, A. Roy, R. Castro, K. McClure, R. Dai, R. Agrawal*, et al.*, "On the thermal elevation of a 60-electrode epiretinal prosthesis for the blind," *IEEE Trans Biomed Circuits Syst,* vol. 2, pp. 289-300, Dec 2008.

[19] K. Stingl, K. U. Bartz-Schmidt, F. Gekeler, A. Kusnyerik, H. Sachs, and E. Zrenner, "Functional outcome in subretinal electronic implants depends on foveal eccentricity," *Invest Ophthalmol Vis Sci,* vol. 54, pp. 7658-65, 2013.

[20] V. Koen, "Stimulating Vision," ed: The Scientist, 2014.

[21] W. H. Dobelle, "Artificial Vision for the Blind by Connecting a Television Camera to the Visual Cortex," *American Society of Artificial Internal Organs,* 2000.

[22] (2015). *Alpha-IMS - Technology*. Available: http://www.retina-implant.de/en/patients/technology/default.aspx

[23] P. Degenaar, N. Grossman, M. A. Memon, J. Burrone, M. Dawson, E. Drakakis*, et al.*, "Optobionic vision—a new genetically enhanced light on retinal prosthesis," *Journal of Neural Engineering,* vol. 6, p. 035007, 2009.

[24] K. Nikolic, N. Grossman, H. Yan, E. Drakakis, C. Toumazou, and P. Degenaar, "A non-invasive retinal prosthesis - testing the concept," *Conf Proc IEEE Eng Med Biol Soc,* vol. 2007, pp. 6365-8, 2007.

[25] E. Striem-Amit, M. Guendelman, and A. Amedi, "'Visual' Acuity of the Congenitally Blind Using Visual-to-Auditory Sensory Substitution," *PLoS ONE,* vol. 7, p. e33136, 2012.

[26] *voice algorithm*. Available: http://i.dailymail.co.uk/i/pix/2012/11/07/article-0-15E65E0E000005DC-289_634x419.jpg

[27] A. Arnoldussen and D. C. Fletcher, "Visual Perception for the Blind: The BrainPort Vision Device," *Retinal Physician,* vol. 9, pp. 32-34, 2015.

[28] L. F. Abbott, "Lapicque's introduction of the integrate-and-fire model neuron (1907)," *Brain Res Bull,* vol. 50, pp. 303-4, Nov-Dec 1999.

[29] M. L. Katz, K. Nikolic, and T. Delbruck, "Live demonstration: Behavioural emulation of event-based vision sensors," in *Circuits and Systems (ISCAS), 2012 IEEE International Symposium on*, 2012, pp. 736-740.

[30] iniLab. (2015). *eDVS4337 (embedded Dynamic Vision Sensor)*. Available: http://www.inilabs.com/support/edvs

[31] I. Microchip Technology, "PICDEM™ PIC18 Explorer Demonstration Board User's Guide," ed, 2008.

[32] *CCS C Compilers*. Available: http://www.ccsinfo.com/content.php?page=compilers

[33] (2015, 10 June 2015). *Java tools for Address-Event Representation (AER)*. Available: http://sourceforge.net/projects/jaer/

[34] E. J. Chichilnisky, "A simple white noise analysis of neuronal light responses," *Network: Computation in Neural Systems,* vol. 12, pp. 199-213, 2001.

[35] SparkFun_Electronics, "RGB LED Matrix."

[36] (2015). *Arduino Duemilanove*. Available: http://www.arduino.cc/en/Main/ArduinoBoardDuemilanove

[37] S. Burstein and H. General Instrument Corp., NY, USA, "A multichannel programmable sound generator IC," in *Solid-State Circuits Conference. Digest of Technical Papers. 1979 IEEE International*, 1979, pp. 218-219.

[38] "PGA2311 Volume Controller," ed: Texas Instruments, 2002.

[39] (10 May 2015). *AndroideDVS*. Available: http://neuromorphs.net/nm/wiki/AndroideDVS

[40] N. Gaspar, "Bionic eye: Neuromorphic system for Visual Prosthetics with eDVS," ed, 2015.